Contributions to Time-bounded Problem Solving Using Knowledge-based Techniques

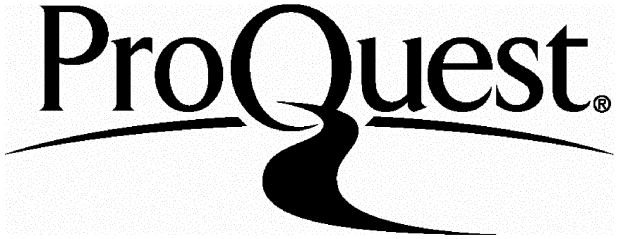
Niladri Chatterjee

A thesis submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy
of the
University of London

University College London
Department of Computer Science

August 1995



To Dear Maa & Bapi:

Shrimati Jharna Chatterjee
And
Shri Lakshmi Kanta Chatterjee

ABSTRACT

Time-bounded computations represent major challenge for knowledge-based techniques. Being primarily non-algorithmic in nature, such techniques suffer from obvious open-endedness in the sense that demands on time and other resources for a particular task cannot be predicted in advance. Consequently, efficiency of traditional knowledge-based techniques in solving time-bounded problems is not at all guaranteed. Artificial Intelligence researchers working in real-time problem solving have generally tried to avoid this difficulty by improving the speed of computation (through code optimisation or dedicated hardware) or using heuristics. However, most of these shortcuts are likely to be inappropriate or unsuitable in complicated real-time applications. Consequently, there is a need of more systematic and/or general measures.

We propose a two-fold improvement over traditional knowledge-based techniques for tackling this problem. Firstly, that a cache-based architecture should be used in choosing the best alternative approach (when there are two or more) compatible to the time constraints. This cache differs from traditional caches, used in other branches of computer science, in the sense that it can hold not just "ready to use" values but also knowledge suggesting which AI technique will be most suitable to meet a temporal demand in a given context.

The second improvement is in processing the cached knowledge itself. We propose a technique which can be called "knowledge interpolation" and which can be applied to different forms of knowledge (such as symbolic values, rules, cases) when the keys used for cache access do not make exact matches with the labels for any cell of the cache.

The research reported in this thesis comprises development of cache-based architecture and interpolation techniques, studies of their requisites and representational issues and their complementary roles in achieving time-bounded performance. Ground operations control of an airport and allocating resources for short-wave radio communications are two domains in which our proposed methods are studied.

ACKNOWLEDGEMENTS

I express my deepest thanks to my supervisor Prof. J.A.Campbell for his invaluable suggestions and support in carrying out the research. I am also thankful to him for making arrangements for realistic domain data acquisition which is crucial for conducting the experiments and without which the relevant work could not have been completed. I further acknowledge my indebtedness to him for solving various non-academic problems that occurred during my stay abroad.

I extend my thanks to all my colleagues in the department of computer science at U.C.L for their help of various natures during my period of carrying out the research. In particular, I thank John Washbrook, Dr. Elpida Keravnou (currently at University of Cyprus) and Dr. Maria Fox for their periodic evaluation of my work and their suggestions for improvements.

I would also like to thank those with whom I have discussed at length, or communicated through (e)mail about my research. These exchanges of opinions not only helped me understand the current state of the art, but also helped in deciding upon the course of this research. Among them I greatly acknowledge Prof. Manuela Veloso of C.M.U and Prof. Janet Kolodner of Georgia Institute of Technology and Prof. Agnar Aamodt of the University of Trondheim, whom I first met during the first European Workshop on CBR, at Kaiserslautern, Germany. I also express my thanks to Dr. Moonis Ali of Southwest Texas University for his appreciation and encouraging comments on my work. My thanks are extended to Dr. Christopher Owens of Yale University, Dr. Ian Watson of Salford University, and Dr. Michael Redmond of Georgia Institute of Technology for many enlightening communications.

I also express my sincere thanks to Prof. D. DuttaMajumder, Prof. B.B.Chaudhuri and Prof. S.K.Pal of Indian Statistical Institute (I.S.I.), Calcutta, who as my teachers developed my first interests in AI, without which perhaps I would never have come to this stage. I am further grateful to the former two along with Dr. S. Ray of I.S.I. for their recommendations in support of my application for a Commonwealth postgraduate scholarship.

I express my deepest gratitude to my grandmother Shrimati Tarabati Mukherjee for her blessings and good wishes. I owe a lot to my parents for their utmost care and encouragement throughout my student days which has culminated in my study for the Ph.D. I also thank my sister Indira who has taken the responsibility of looking after my ailing parents during the years of my absence from home. I am grateful to my parents-in-law Mr. K.P. Banerjee and Mrs. Bani Banerjee for being sources of encouragement. My uncles, aunts and numerous other relatives deserve my sincere thanks too. I extend my heartiest thanks to my sister-in-law Santasri, brother-in-law Mr. T.P. Chatterjee, and little niece Adrija for all their best wishes.

I also express my thanks to fellow researchers in this department (Aggeliki, Carl, Felicity, Jonathon, Mauro, to name a few) for the lively discussions I had with them on various aspects of research and thesis writing, and the help I received from them. I am also thankful to my friends Ajit, Bina-di, Chris, Davinder, John, Malpani-ji, Nikhil-da, Paul, Philip, Saadia, Sanjay, Shamim, Srirangan, Steve, Sylvia whose excellent company and cooperation at different stages of my stay in London helped me remain cheerful.

Last but not the least comes my wife Sutapa who suspended her own Ph.D, at least temporarily, to join me in London and provide me with moral support and constant encouragement.

Note: The major financial support for doing this research came from Association of Commonwealth Universities in the form of a Commonwealth Postgraduate Research Scholarship.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1.

INTRODUCTION

1.1. GENERAL BACKGROUND

As the horizon of knowledge-based applications is expanding, demands on knowledge-based technologies are also increasing. Consequently, researchers in knowledge-based technologies are facing qualitatively new challenges. One of these potential challenges emanates from time-bounded computations where a fixed time limit is prescribed, along with each new request, by which an answer should be provided.

The application of knowledge-based systems has traditionally been confined to simpler domains such as diagnosis, design, planning and configuration. Such domains can be characterised by properties such as:

- the nature of data is static, i.e. the information undergoes no change during the course of its processing;
- the control structure is monotonic, i.e. once a decision is made, it remains true throughout the course of the entire computation; and especially
- the applications are not time-critical, i.e. there is no pressure caused by limits on the computing time.

But the gamut of applications of knowledge-based techniques is fast expanding. The type of problems that present knowledge-based systems (KBS) aim to address extends from the presentation of simple, static data to complicated, dynamic-data-based problems involving asynchronous events and non-monotonic flow of control with shifting focuses of attention. Real-time domains such as satellite control, battle management, aerospace systems, communication networks, robotics and vision systems all involve the aforementioned characteristics and warrant more sophisticated treatment on the part of the system.

One fundamental requisite of real-time computations is *timeliness* [Dodhiawala 89] where a system is required to provide an answer within a given period. For systems in these domains the requests often have some associated time bounds attached to them within which a solution is necessary. Finding an answer beyond the allotted time limit may be tantamount to having no solution at all. The temporal deadline that one may expect on a specific computation can be of three different types [Dean 91]:

- absolute (called "hard" in real-time computing), having a specific time limit within which the result is needed;
- graded (or "soft"), i.e. quality of the performance depends on the completion time (e.g. the sooner one gets the result, the better).

  and

- relative, where the temporal deadline depends on the behaviour of another computation (e.g. to finish task X before task Y);

But a critical look at existing knowledge-based systems suggests that their application in domains where a time-bounded performance is desired has been relatively small. Although various attempts have been made to study the behaviour of time-dependent computations and to build systems for solving real-time problems in different domains, the success is rather limited. This observation leads one to explore the difficulties in carrying out tasks using knowledge-based techniques, when the computing time is stipulated by the imposition of certain limits.

Our work is concerned with time-bounded computations where a temporal bound (whether hard or soft) is imposed. Our cache-based system is designed on the assumption that a specific time bound, by which the answer is required, is to be supplied along with the problem specification and the cache should return the best possible answer that computed without violating the time limit.

For traditional real-time systems, where the nature of computations is algorithmic and therefore the exact demands on various computational resources are known right at the design phase of the systems, achieving timeliness is fairly straightforward. But as knowledge-based computations are open-ended and indeterministic in nature, the exact temporal requirements for a computation are not so easy to predict, and finishing a task within a small time limit is far from guaranteed. There is therefore a need for suitable techniques to be developed to ensure timeliness for time-bounded knowledge-based systems. The first aim of this research is to develop techniques that go at least some way towards answering this need.

Of course, one way of responding to the above requirement is by enhancing speed through improved technology. For the hardware, as faster processors are being developed and their speed increases from megaflops to gigaflops, the speed of computations should increase enormously, reducing the temporal requirements for the same computations considerably. Also, attempts are being made to use multiple processors and/or parallel architecture, each of which is aimed at reducing the computational time (e.g. SPHINX, a speech recognition system [Lee 88]). Hence a system that fails to meet the imposed temporal demands may appear to be successful with the same data structure and processing techniques, if much faster processor(s) replace its original ones. Similarly, improved software in the form of parallelism in rule-bases has been contemplated by AI researchers, in order to increase the speed of computations [Gupta 86]. Parallelisation of different AI search techniques (e.g. best-first search

[Kumar 88], alpha-beta pruning [Hsu 89]) have also been suggested and studied by different researchers.

But comparing a parallel computational approach on an equal footing with a sequential approach is not necessarily helpful. As noticed by Aamodt [Aamodt 91], "success or failure in developing a knowledge-based system depends upon how well expectations map to what is achievable within the given limits. The limits are, to some extent, imposed by the set of methods, techniques and tools that constitute the state-of-the art in knowledge engineering and artificial intelligence in general". We feel that the same holds equally (if not more) strongly for time-bounded knowledge-based systems.

Therefore, the direction of the research reported in this thesis is different. However fast a machine may be, there exists a limit on its speed. Thus the question arises with each occasion when needs for speedier computations appear, "should the system be changed, or should efforts be made to derive improved performance under the existing set-up?". The latter is the concern of our research.

We believe that within any computational framework, a two-way improvement can be accomplished towards the performance of a knowledge-based system:

1. designing an efficient knowledge representation scheme that enables quick access to the right piece of knowledge;
2. developing an efficient reasoning tactic to distil out quick solutions from the retrieved information.

In this research we provide answers to both the requirements by designing a cache-based architecture which provides some ready-to-use answer or solving technique that can settle a current problem; and a novel technique which we call *knowledge interpolation* that facilitates quick reasoning with the cached answers to compose a solution specifically for the problem at hand while abiding by the temporal stipulation.

1.2. OUR APPROACH

Caching is a well-known technique in computer science to speed up memory access. We suggest the same principle, here, to store answers to different problems that may occur in a given domain. However, unlike the traditional situation, the cache not only contains ready-made solutions to satisfy immediate demands, it also provides clues regarding which reasoning paradigm will be most useful for solving a problem under a given temporal stipulation. Several reasoning schemes (such as rule-based, case-based) are available within the repertoire of knowledge-based technology, each having different degrees of complexity and

consequently, differences in quality of solutions. Naturally, it is expected of a system that qualitatively better methods will be applied in solving a current problem as the available time increases. Our cache-based architecture is designed to support this expectation.

The initial idea of the cache-based system and some early results have already been reported in some of our earlier publications [Chatterjee 92, Chatterjee 93]. In this thesis we present a deeper description of the system by giving details of knowledge and its representational requirements, procedures for selecting the best indices for access and retrieval of the knowledge, and methods of carrying out different operations on the retrieved information, which are all essential in building a working system.

The main obstacle to using a cache, as described above, in a realistic domain is in the volume and varieties of the knowledge/data that ought to be stored in such a cache. A real-life domain is normally associated with an enormous number of features each having its own significance in any problem that may occur in the domain concerned, causing umpteen variations in the nature of the problems. It is unrealistic to store optimal answers to all these variations in a cache to be managed in time-bounded mode. Consequently, our suggestion is to store answers to some typical problems in the cache which will be retrieved as and when required, by matching a current problem with the prototype problem situations dealt with in the cache. Evidently, some tactic is therefore required that can adapt the cached answer(s) to render situation-specific solutions to the problems at hand. The idea of "knowledge interpolation" has been developed to meet this demand. We have named this process of selecting some intermediate position between two extremes "knowledge interpolation", recognising its superficial similarity to numerical interpolation. Resembling what happens in the domain of numerical analysis, this technique aims at deriving a quick approximation to the solution for a current problem by interpolating on the solutions suggested in the cache to some similar problems. The cache, here, will be so designed that it should be able to retrieve at least two solutions for a current request even when the characterisation of this request is imprecise, so that interpolation can be applied on them to calculate a suitable solution from those that are retrieved.

The cache-based architecture aided by the interpolation technique has certain advantages over traditional knowledge-based computing:

1. The cache provides a straightforward access to the relevant knowledge/information, and thereby obviates the need for wading through a knowledge-base in search of appropriate rules or cases (depending upon the underlying reasoning paradigm);

2. The knowledge-interpolation technique does not require to have a cached entry that is the best possible match for the current problem situation (although intuitively the closer is the match, the better should be the interpolated result). Thus, even if the cached answer(s) do not prescribe exact solutions for a given problem, the method can still work on it to produce a solution for a problem.

3. Retrieval from the cache and subsequent knowledge interpolation are both algorithmic in nature, and therefore the required computational time for solving a problem can be estimated. Such an estimation of temporal requirements is essential for time-bounded applications, although it is not normally achievable with other knowledge-based techniques.

Thus the overall architecture provides a safe footing for any attempt at solving time-bounded problems in knowledge-intensive domains.

1.2.1. Intuition Behind The Proposed Methods

The intuition of our approach finds its root in a type of reasoning that humans commonly indulge in. Humans are quite apt in switching from one mode of reasoning to another depending upon the situation, without necessarily regarding one mode as primary. For example, if someone is asked what is the best mode of transportation from location X to location Y within London, the type of answer that one expects will depend on the urgency. An instantaneous answer may be 'the Underground'. But if the replier has some more time, one will probably apply rules such as "if there is no tube station nearby then first take a bus and drop off at a certain place from where one can take the Underground"[1] - or, "if it is at the peak of the tourist season and the destination is not too far then take a taxi". However, if the replier has even more time, one will try to find any past experience with respect to either of the locations or the connecting path and may come up with an answer, say, "well, last time I travelled by that route I had a bad experience. It was about to become dark at that time. So, if the travel period is in the evening it will be better to avoid the tube". And when there is no time constraint involved one will try to see the map, distances of nearest tube stations or bus stations, and compare the merits and demerits of various possibilities to arrive at the best solution. A *cache* is a convenient structure to hold all these common (and heterogeneous) strands of human thinking.

The idea of "knowledge interpolation" also finds its root in the common human tendency of finding a solution that balances two extremes, in order to arrive at an intermediate solution. For example, consider how a human solves the problem of whether to take an umbrella on a particular day. A person normally prefers not be burdened with an extra weight; at the same time getting wet is not desirable either. It is our common experience that humans make their decisions by judging the cloud density (comparing it to some previous examples in memory), and deciding not to take an umbrella if on some past occasion it did not rain even though the density of cloud had been at least as much as it is today. Alternatively, one may even decide to take a small umbrella as an intermediate solution which does not make us feel so burdened if there is no rain and which allows some protection in the event of sporadic rain.

[1] Possible explanations behind this heuristic may be that a bus will be very slow while tube services are likely to be interrupted by equipment failures or terrorist bomb threats etc.

Although quick assessments (as above) of situations do not necessarily yield the best possible solution, human instincts for making decisions by choosing a solution that is intermediate between two extremes cannot simply be dismissed. The psychologically-based theory of "bounded rationality" [Simon 59] corroborates aspects of the aforementioned line of human decision-making as well.

But discussion of human decision-making in psychological publications revolves around the theory of maximising an expected utility (EU) [Behn 82, vonNeuman 47] where,

$$EU(A)=\sum_{i=0}^{i=n} P(E_i)*U(X_i)$$

when E_1, .. E_n are outcomes of the action A, $P(E_i)$ is the probability of the i-th outcome and X_i is the consequence of outcome E_i with a utility $U(X_i)$. Solvic [Solvic 90] has illustrated this decision making process with the simple example of whether to carry an umbrella or not on a particular day, by computing the probabilities of whether it will be sunny or raining and the utilities of (not) being burdened and (not) getting wet.

Evidently, these probabilities are worth consideration in a complete picture of informal reasoning, but they seem to be mathematically too regular to capture the essentials of what humans do in problem-solving. On the other hand, typical AI papers that deal with rapid decision-making considers different issues (e.g. indexing schemes [Kolodner 88b, Owens 89]; efficient matching, such as using dynamic similarity measurements [Leake 91], incremental similarity measurement [Veloso 91]; efficient search techniques [Rich 91]), which put more emphasis on knowledge representation and reasoning but do not represent the common human line of thinking. In our approach we have taken into consideration the common informal human problem-solving behaviour in dealing with temporally-stipulated problems and have used the caching technique for its implementation.

1.3. AIM OF THIS RESEARCH

The targets of this research are threefold:

1. Study the necessities to build a cache-based system, which include
   - designing the structure of a cache;
   - studying the issue of indexing (i.e. cues for retrieval) with respect to a cache;
     and
   - designing a control algorithm that governs the activities related to problem solving using a cache.

2. Explore a method of "knowledge interpolation". Interpolation is straightforward in numerical analysis, because a natural order exists among numerical quantities which is essential to conduct an interpolation-like act. Such a natural ordering however is absent among symbolic quantities. But a realistic domain can very well be associated with features of different types (e.g. symbolic scalars, range, sets, fuzzy descriptions). Hence designing "knowledge interpolation" necessitates:

   - designing schemes for imposition of metrics on symbols of different types and a systematic measurement of distance under each scheme;
   - identification of the interpolating variables on which interpolation is to be performed.

   We should explore various methods of conducting interpolation. Moreover, we want to design policies for knowledge interpolation in different reasoning paradigms i.e. in rule-based and case-based decision-making. We should also therefore examine different ramifications of such specialised interpolations and study the necessary requirements.

3. Finally, study the computational behaviour of an implemented cache-based system in some particular domain. Our interests are:
   - study of the qualitative improvements of solutions with successively deeper levels of the cache;
   - estimation of temporal bounds for different cache-levels;

     and
   - most importantly, studying the functional behaviour of the cache-based system in maintaining temporal stipulations and still providing the best possible solution within the given time limit.

   In chapters 8-9 we describe a working system with all the key features of caching and interpolation that has been the basis for the study.

1.4. DOMAIN(S) OF APPLICATION

The motivation of our work primarily originates from our studies on knowledge and examples in the domain of **ground operations control in an airport (AGC).** In an airport, the ground operations controller (GOC) is responsible for managing all ground operations within the airport, and solving single problems (i.e. we have not considered complex structures of multiple problems) that may come up during these operations.

A GOC normally starts the day with a schedule of flights arriving to and departing from the airport during the course of the day. A controller normally makes an a priori plan at the

beginning of the day. But such a plan needs to be modified quite often, as it is likely to be interrupted by numerous unforeseen events and unanticipated requests throughout the day. Interruptions may come from two types of source: an air traffic control centre that reports actual or predicted deviations from the target times, and an airport management centre that reports significant events in the airport that may affect the targets (e.g. failures in the system of loading or unloading baggage, fires, appearance of some obstacle on a taxiway/runway, etc.). If any such problem occurs, the GOC has to take early action to combat it, e.g. by contacting the right person who is considered to be a specialist in mitigating the problem, or modifying the basic plan of the day's operations. Thus actions to solve an immediate problem may generate secondary problems, e.g. allocation of extra time to an airline for loading baggage may create problems of availability of a gate for the next incoming flight. Although it is desirable for the GOC to attempt deep planning, in the event of some interruption, this may not be practicable in a time-critical situation. Hence the GOC may have to use an ad hoc solution in the case of processing emergencies - and may then take care of the resulting side-effects after the immediate problem is sorted out.

Although developing a full-fledged system to accomplish the task is too time-consuming as well as difficult to manage single-handedly within the time-span for one thesis (and therefore has not been contemplated as such), our studies have helped us identifying certain characteristics of real-time problem solving and developing techniques to overcome the difficulties.

The ideas generated in this work have subsequently been tested on another domain: allocating frequencies and transmitter powers (and transmitter locations where relays are recommended) for **shortwave radio communications,** where we have found that the same ideas and the approach continue to hold good. It is a common experience that transmission using shortwave radio is more prone to various types of disturbances than other transmissions e.g. medium wave, FM. However, the physics of *long-distance* transmission almost invariably calls for shortwave frequencies (between about 2.5 and 30 MHz) to be used. Our application aims at providing the frequency, power of transmitter and if necessary the location of relay point that will be ideal for a proposed transmission between two locations and at a given time of the day.

While the problems of AGC are more or less self-explanatory, the shortwave radio propagation needs detailed description of its domain in order that the problems of the domain are better understood. In chapter 8 we describe the topic in detail.

1.5. OVERVIEW OF THE THESIS

The thesis has been organised in the following way:

Chapter 2 contains an examination of real-time problems and the difficulties of solving them

in general and with knowledge-based techniques, in particular. We also present a review of the most relevant past works on time-dependent problem-solving that have used AI techniques.

Chapter 3 introduces the cache architecture and different aspects of its operational behaviour. We also present the encouraging results obtained from trying a preliminary version of the cache on numerical problems.

In chapters 4, 5, 6 and 7 we have studied different fundamental aspects of the cache-based systems from theoretical as well as implementational points of view.

Chapter 4 studies the importance and selection of indices for a cache-based system. Indexing, i.e. selection of appropriate row and column dimensions and the column headers, is essential for quick access to the stored information. This chapter investigates how suitable indexing schemes can be developed commensurate with the cache requirements.

Chapter 5 introduces the notion of "knowledge interpolation". It first investigates the implementation of two procedures, fundamental to knowledge interpolation: imposition of order on symbols and measurement of distances. Algorithms for measuring distances under varied situations have been developed. This chapter also identifies different ways of conducting interpolations and how these can be applied to different knowledge structures.

Chapter 6 deals with aspects of knowledge representation for the cache-based architecture that facilitate reasoning with the cached knowledge and accomplishing interpolation on them.

Chapter 7 describes how to reason with the cached items in order to find solutions for a current problem while observing the temporal stipulations. It also identifies the essential information that a system is required to store at various levels of the cache. In particular, it discusses three levels where the basic reasoning paradigms are in the form of (i) *default solutions,* (ii) *rules,* and (iii) *cases.*

In chapter 8 we present a description of the domain of shortwave radio propagation and the considerations for building a cache for handling problems in this domain.

In chapter 9 we describe the experiments that we have conducted and the results that we have achieved using the cache and associated interpolation scheme.

Chapter 2.

TIME-DEPENDENT PROBLEMS AND KNOWLEDGE-BASED APPROACHES

2.1. INTRODUCTION

In this chapter we first study some of the difficulties that are characteristic of time-dependent problem solving using knowledge-based methods. We then examine different approaches that have been considered by researchers in AI to deal with these difficulties. Finally we survey some of the existing real-time systems and their inadequacies in dealing with time-bounded problems.

2.2. DIFFICULTIES IN TIME-BOUNDED COMPUTATIONS USING KNOWLEDGE-BASED TECHNIQUES

The difficulties that a system may face in achieving true time-bounded performance using knowledge-based techniques can be attributed to several sources:

- Difficulties due to domain characteristics;
- Difficulties due to the temporal element;
- Difficulties due to nature of the problems;
- Difficulties due to the nature of knowledge-based computations.

2.2.1. Difficulties Due To Domain Characteristics

Time-bounded computations using knowledge-based techniques are primarily applied to real-world domains characterised generally by weak (if not intractable) domain theories. For such a domain, the relationships between the relevant concepts are rather uncertain in nature [Porter 89], e.g. causes and effects of different problems do not often follow a general fixed pattern. Naturally, for such a domain it is difficult to proclaim any particular solution to be

the best possible solution. Frequently, this difficulty is further aggravated by the fact that even problem descriptions, here, are generally incomplete and noisy, leading to additional tasks for a system to ascertain the relevance of any piece of information that it happens to receive or generate in its course of operation.

Furthermore, domains where time-bounded computations may be needed reasonably often are in general dynamic, i.e. the relevant data may change during the course of processing. In order to cope with the continuously-varying situations in a dynamic world, a system is required to inspect the problem-solving environment, so that it can determine if any problem more urgent than the one being processed currently has occurred, or if facts that its current problem-solving exercise is based on have undergone any change. In the event of any such situation a system may have to shift its focus of attention to solving the more urgent problems, leading to total abandonment or temporary suspension of its current activities (again depending upon the context).

Another important aspect of real-time domains is the nature of their interactions with the outer world. Often a real-time system needs to refer to some values that are to be computed externally and fed back to the system, or the system may need to carry out certain tasks externally, the outcome of which will be required before it can continue with its further reasoning. Interaction with the outer world has the obvious problem of resource allocation. Moreover, the system needs to monitor the resources that are available at any instant, and to reset its task priorities and resource allocations depending upon the workload, resource availability etc. - so that a smooth behaviour may be ensured despite potential perturbations in this regard. A system lacking this property may waste time and processing resources in executing less relevant tasks, or in waiting for resources that are not readily available, or even on undertaking computing tasks that are no longer required.

2.2.2. Difficulties Due To The Temporal Element

Imposition of a temporal bound on the computational activities almost invariably demands certain other properties on the part of the system. The most important of these is speed.

Speed, evidently, means the rate at which the tasks can be executed, where *task* refers to those actions that are used explicitly for solving a problem, and also other "behind the scenes" activities such as event-handling, resource allocation, interaction with the outer world. Although the importance of speed in meeting time-bounded demands is easy to appreciate, one important point to note is that speed alone cannot account for timeliness. In order to maintain time-bounded performance, the speed of the system should be complemented by some related properties:

- firstly, for a given system there can always be a problem for which the allotted time limit is too small to be respected even using the fastest available (in terms of the system concerned) method. Naturally, a complete system has to be well enough equipped to provide some sensible response for such contingencies too.

- secondly, since in a dynamic world the environment is changing continuously, even resorting to the maximum possible speed (governed primarily by the system configuration) cannot guarantee that the environment will remain unchanged during the course of execution of a certain task. There is always the possibility of occurrence of some new event that will have an influence on the current task. Such new events will act as external inputs to the system. They should be recognised as quickly as possible by the system, and their significance or criticality judged in order to take appropriate actions.

- also, the solution steps that a system will resort to should have some pre-definable or identifiable predictability as far as their consequences are concerned. The speediest tactic, if it has an unpredictable outcome, is of no practical value here.

Therefore, merely increasing the speed in calculation may not provide the right kind of treatment. The calculation procedure has to be predictable enough to ensure a relevant result within the prescribed time bounds.

2.2.3. Difficulties Due To Nature Of The Problems

The types of problems that a system is likely to address may have varying relevance to the overall functioning of the domain. Consequently, different time limits will apply in solving these problems. Therefore, timeliness does not mean that all problems are to be treated with equal urgency. Depending upon the nature of the problem, the length of the deadline may vary. Even for the same problem occurring under different circumstances a system may find different temporal bounds prescribed for the solution.

Moreover, the implication of the time bound may vary with situations. Depending upon the repercussion of any possible failure on the part of the system, time-bounded tasks have been categorised into two classes: hard and soft [Laffey 88a, Stankovic 88]. A real-time problem is *hard* when inability to meet the deadline strictly leads to a disastrous consequence. On the other hand, for a *soft* real-time task a strict time limit cannot be specified but a tentative target can be defined. Exceeding this limit does not lead to any serious consequence but may cause some inconvenience (of varying degrees, depending upon the nature of the problem and the length of delay).

Naturally, a system for time-bounded computations is not expected to have a single method for solving a particular type of problem. Rather, it is more desirable for the system to have

different tactics in its repertoire, with varying time complexities, for solving a particular type of problem. The system should have the capability of judging the implication of the prescribed time limit and should decide accordingly the problem-solving approach and also the starting and ending times of action. It is reasonable to assume that the higher is the time-complexity for a tactic, the better (qualitatively) will be the solution achieved through its application. Certainly, in the absence of any time limit a system ought to apply the best available tactic. But imposition of time bounds usually forces a compromise in quality in order to comply with the temporal demands.

2.2.4. Difficulties Due To The Nature Of Knowledge-Based Computations

The difficulties mentioned above are not particular to knowledge-based systems alone. A designer of any systems intended for time-bounded activities needs to develop strategies and tactics to tackle these obstacles. In systems characterised primarily by algorithmic actions (e.g. control of industrial processes, communication with data-recording equipment, monitoring of alarms) the nature of the possible computations and their demands on resources are clear during the design phase. Hence, most discussions of time-bounded computations in such systems concentrate on trade-offs among the different features, on the assumption that there is no difficulty in finding at least one way of achieving any of them.

On the other hand, the fundamental nature of knowledge-based computations is too open-ended to guarantee such smooth and predictable behaviour. Whether it is use of search techniques in a solution space, or application of rules under rule-based reasoning, or selecting appropriate cases from a stored case-base under case-based reasoning, the underlying principle is to keep on repeating the same procedure until a desired match is found. Hence, these techniques suffer from some inherent indeterminacy, in the sense that the exact steps that are to be carried out are not well-known beforehand. As a result, the exact demands on time and other computational resources cannot be well-estimated at the design phase of the architecture. For example, in a rule-based system it is not possible to foretell which rules are going to be fired in order to provide the answer to the next problem. Past studies of rule-based systems [Gupta 85] indicate that in a rule-based system 90% of the computational effort is spent in matching while 10% is used in conflict resolution and actual actions. Time-complexities of different heuristic search techniques do increase exponentially, and consequently, in a relatively large search space it becomes essentially impossible to predict how much computational effort will be required. Locating suitable cases in a case-base with even moderately large numbers of cases has the same difficulty. This open-endedness of knowledge-based computations is a deterrent to their straightforward application in time-bounded problem solving.

However, despite these difficulties there have been several efforts towards applying knowledge-based techniques in time-dependent problem solving. We have identified three major lines of research towards this end:

- along one line researchers have studied the temporal behaviour of time-dependent problems, in general, and have developed temporal logics to deal with importance of time as a crucial factor, its representation and relationships between events occurring at different time points/slots;
- the second line comprises development of knowledge-based techniques to cope with the temporal demands of time-bounded problems;

  and
- the third line consists of endeavours to build real-time systems pertaining to some specific domain, tuned and adjusted in accordance with the terms and needs of that domain.

As we see below, each of the above approaches has certain drawbacks as far as performance is concerned.

2.3. TEMPORAL LOGIC AND REASONING

The notion of time has long been considered as an important aspect of knowledge-based problem solving, as it has been observed by McDermott in 1982 that "No one has ever dealt with time correctly in an AI program, and there is reason to believe that doing it will change everything" [McDermott 82]. This comment sounds up to date even against the background that a considerable amount of work has been done since then in representing and reasoning with the temporal dimension of information.

Many items of work since 1982 have experienced some influence from Allen [Allen 84, Allen 85]. Time here is considered to be a convex interval represented by two points (corresponding to the beginning and end). Any event is associated with an interval during which it has occurred. Allen's work comprises exploration of the relationship between two intervals, and as many as thirteen relationships have been identified that govern the temporal relationship of two intervals. They are: *equal, before, after, overlaps, overlapped-by, starts, started-by, finishes, finished-by, during, contains, meets* and *met-with.* Logics based on these temporal relationships have also been developed.

A new interpretation of *interval-based logic* has been provided by Tsang [Tsang 87]. Instead of considering time-intervals as the basic unit of reasoning, here, *events* occurring during certain intervals have been studied as the concept of primary interest. The rationale behind this idea is that human mental experience is based on events rather than instances, and therefore two events occurring during the same time interval should be considered as two different primitives, as opposed to what Allen's interval-based logic seem to suggest. Relationships between event-based and interval-based calculi have also been established, on the basis of their dealings with time.

Allen's theory of time has subsequently been extended by Ladkin [Ladkin 86, Ladkin 87], who considered a non-convex representation of time, in the form of unions of intervals such that they comprise some convex sub-intervals with convex gaps between them. From this, a canonical model of Interval Calculus has been developed.

A different representation of time has been proposed by Shoham [Shoham 89]. Here, time is considered to be linear, described by a pair (T, <) where T is a set of time points and < is a total order on this set.

This theoretical treatment of temporal reasoning constitutes a new (post-1982) dimension in the time-dependent reasoning process. A general overview and state of the art of temporal reasoning and calculus can be found in [Vila 94]. But the work does not certainly address the time-bounded problem solving aspect of knowledge-based reasoning and is therefore outside the scope of our subsequent discussions. However, we feel that such a theoretical approach is still the closest to a formal treatment (logic-based) relevant to our work.

2.4. KNOWLEDGE-BASED TECHNIQUES FOR REAL-TIME PROBLEM SOLVING

Different techniques have been suggested so far for handling time-bounded problems using knowledge-based techniques. These include different heuristic search algorithms with a view to expediting locating a suitable solution in the solution space. Different heuristics have also been suggested to ensure other aspects of real-time performance.

2.4.1. Studies Of Different Search Techniques

The success of a knowledge-based system in the dimension of interest to us depends on how fast a suitable solution can be retrieved from its knowledge-base. Different heuristic search algorithms have been developed to achieve this objective. Some of these algorithms are quite general in their applicability while others have been attempted on some particular reasoning schemes. Also, search algorithms have been developed with the aim of meeting time-dependent demands of real-time situations. However, an in-depth look at various existing retrieval algorithms reveals that using them isolatedly does not really serve the purpose. For example:

Traditional search algorithms like depth-first, breadth-first, A* [Rich 91], iterative-deepening A* (IDA*) [Korf 85] have the limitations of exponential time complexity, along the dimension of the number of nodes of the search tree. Thus, although these algorithms are

guaranteed to find an optimal solution, their success in the real-time domain is limited only to small search spaces and unrestricted time bounds. In a comparatively large domain with strict temporal deadlines these algorithms are unlikely to produce any result.

A subsequent variation of A*, real-time A* (RTA*), [Korf 88] has been designed specifically to address this problem. This algorithm meets a temporal deadline by producing at least a partial solution within the deadline. The technique adopted here is *bounded look ahead search,* which searches to a fixed depth horizon (the depth depends upon the given deadline). Like A* , here too a cost function is used to evaluate the frontier nodes, and each move is made towards the minimum-cost value. Thus the algorithm is committed to some action within a fixed time span, and moreover, the requirement of building an entire search tree, right at the outset, is circumvented.

But the major problem with these search algorithms is that, for them to be useful, it is required to have a tree-like structure (either present explicitly or capable of being generated incrementally, i.e. from any node its successors can be generated) in which there is some guaranteed underlying understanding of the general meaning of the structures and the ratings of nodes. For a real-life domain to fit this description, one would need to have a fairly fixed and stable kind of criterion in order to control the search. But the uncertain nature of problem-solving and incompleteness in the information about the domain renders such an assumption grossly impractical, in general, in a real-life domain. Another major problem with the heuristic search algorithms is that while calculating the cost functions they consider the planning costs alone and ignore the execution cost of a plan [Shekhar 89]. Naturally, the search algorithms may spend all the available time only in searching. But in many real-life problems the execution cost is non-trivial and hence needs to be considered while planning, primarily, because of possible interactions of the system with the outer world, to cope with the demands of real-time problems. All these shortcomings of search algorithms preclude straightforward applications of the techniques in real-time domains.

Some other specialised search techniques have been developed to work in particular reasoning modes such as rule-based reasoning or case-based reasoning. But, as we see below, they have their drawbacks too, as far as applications to real-time domains are concerned.

The pioneering work to expedite searches in a rule-base comes from RETE [Forgy 82]. The efficiency of RETE can be attributed to three major sources:

- partial matching of rules' preconditions, governed by the assumption that the state description remains more or less the same between the firing of rules. It maintains a network of rule conditions and it considers only the changes in the state descriptions while it is deciding which rule to apply next;

- organisation of the rules in such a way that they share a common structure of preconditions and consequently several rules can be tested in one cycle;

- remembering of variable bindings from previous calculations and updating these with new binding information instead of computing binding consistencies from scratch each time a new condition is satisfied.

However, RETE and its subsequent improvements (such as TREAT [Miranker 87]) have been designed for production systems only, which by definition are purely rule-based. However, in most of the real-life domains compiling an comprehensive rule-base is time-consuming and difficult (if not impossible). Naturally, these algorithms can at best be a part of the overall problem-solving mechanism in most of the real-time knowledge-based problem domains. Hence, for effective design of a real-time knowledge-based system a designer needs to look for some other reasoning mechanisms as well.

Case-based reasoning (CBR) has developed in recent years as a viable alternative, as a mode of informal reasoning [Riesbeck 89, Kolodner 92a, Kolodner 93] (though not with any special reference to time-dependent problems). Instead of holding formal rules, a CBR system depends on informal knowledge structures in the form of solutions of some past cases. In order to solve a current problem a CBR system tries to locate from its case-base similar past case(s) and tailor the past solution(s) according to the need of the current situation. Naturally, as the case-base grows, searching for appropriate solutions becomes a challenging issue as well. Different search algorithms have been developed in recent years for rapid case retrieval from a case-base. But, as we observe below, all of them are based on some oversimplified assumptions regarding the domain and consequently none of them stands the test of being a foolproof solution to the real-time AI problems.

The "Case-Based Search" (CBS) algorithm [Bradtke 88], although designed for searching in a case-based reasoning environment, has its limitations with respect to application in a real-life problem. Its success depends on the design of an *index function* that divides the problem states into some equivalence classes, depending upon their distances from the goal state. Evidently, design of such an index needs considerable knowledge of the problem space and the goal state. It also requires the exact steps that take the current state into the goal. But a real-life problem domain (e.g. the AGC problem) is likely to be open in nature with incomplete or uncertain knowledge of the domain, unpredictability about presence/absence of other factors etc. Naturally, availability of required knowledge for the index is not guaranteed. Moreover, as we see later through examples, there may not be any precise goal state for a given problem. In a time-critical situation the immediate goal of the problem-solver often changes depending upon the available time. Hence deciding the equivalence of states is not easy, because here equivalence of states is measured in terms of their implications in a given context and not merely by input features. This prohibits straightforward application of CBS in time-critical problem domains.

Tree-Hash algorithms [Stottler 89] have the inherent drawbacks of oversimplification in similarity measurement. They measure similarity on the basis of exact matches of attribute values. But in real-life situations matching of cases based on attribute values may not be practicable, as various aspects of cases need to be considered and therefore massive abstraction of cases is required [Owens 88]. As a further example, Lehnert's algorithm [Lehnert 87] of searching large case-bases considers generating and searching the entire search space. Its domain of application being 'phoneme generation for pronunciation of unknown words' has the obvious advantage of a rather limited (even enumerated a priori) search space, which is not true for most real-life application domains where a case may be specified in terms of features drawn from very large sets.

The above observations suggest that efficient retrieval cannot be achieved only through superior search techniques. Instead, efforts should be made to measure similarity of the current situation to past ones in a judicious manner in order to facilitate worthwhile retrieval within a limited time. The basic limitation of existing mechanisms is that the notion of 'similarity' remains static throughout the operation. But time-critical situations deviate from this blanket assumption about similarity, where - depending upon the situation - the meaning of similarity may vary. The next section describes some of the salient influences that time-criticality may have on similarity measurement.

2.4.2. Other Heuristics And Suggestions

While speed is considered to be the foremost requirement of real-time systems, there are other aspects too. Different heuristics have so far been developed to satisfy them. For example,

Anytime Algorithms, characterised by some special features, have been suggested in dealing with time-dependent planning problems [Dean 88]. These algorithms are characterised by facilities whereby

- reasoning with such an algorithm can be suspended and resumed subsequently with little overhead;

  and

- it can be terminated at any time, yet it is guaranteed to return some answer, and the answer returned will improve as a well-behaved function of time.

Progressive reasoning [Wright 86] has been suggested to ensure timeliness. Here, inference is performed in several phases such that at each phase a more accurate solution is produced. It prescribes for a system to have several levels of reasoning, each using data that are more time-consuming to retrieve and process than the previous level. The last calculated answer is to be remembered so that as soon as the allowed time expires the system can produce a

response based strongly on that answer. Again, the idea of progressive reasoning is essentially a subset of a more generalised idea called "progressive deepening" [Winston 84], which searches each situation to successively deeper depths until the time bound is reached. Evidently, such an approach is likely to work when the problem has a character that allows this multiple-stage treatment to occur. For an arbitrary domain such an assumption is more likely to be false than true.

In order to avoid any repetition of tasks that a system may undertake while reasoning with a given problem, distinctions have been made between reasoning in the future and reasoning in the past [Long 83]. The argument presented is that a real-time system in course of its functioning may encounter some new or derived data which have influence on some decisions made in the past. In such cases treating this set of data as something entirely new should be erroneous. Instead the system should be able to backtrack and withdraw some of the conclusions (and related information) made in the past and re-execute its decision module, as it is not possible to revoke any action that has already been taken. On the other hand, any data pertinent for reasoning in the future has influence on both information and the actions to be taken, and a system can be judicious in selecting the next action based on such items of data.

2.5. DIFFERENT REAL-TIME SYSTEMS: A BRIEF OVERVIEW

There have been quite a number of efforts to build real-time systems, each having its own merits as well as limitations with respect to solutions of time-dependent problems. An exhaustive study of all of them is unnecessary; here we describe a representative set of them along with their purpose and behaviour in attempting to achieve the objective.

2.5.1. AIRPLAN

One of the earliest systems designed for time-critical decision making is called AIRPLAN, developed at Carnegie-Mellon University [Masui 83]. The purpose of the system is to assist air operations officers with the launch and recovery of aircraft from a carrier at sea. The most important aspect for this system is "graceful adaptation" in the sense that it has to recognise any changes in the environment (such as weather conditions, fuel state of an aircraft) and take appropriate action (here, to alert the officers) to combat the impending problem. AIRPLAN offers four levels of assistance to its users: display of raw data, identification of problems and rough characterisation of possible solutions, refinement of the characterisation of possible problems and identification of possible future problems.

The task of AIRPLAN has stringent time constraints in the sense that the system may discover an impending problem in a situation when there is no time to explore its implications

adequately. In order to account for this, the system has been developed in the following way.

AIRPLAN has the facility to "interrupt itself" while executing certain tasks, in order to manage its time resources more effectively. As soon as AIRPLAN receives a report it updates its display to apprise the users of the latest situation. It uses several demon[1] rules for recognising new reports and subsequently updating the display. AIRPLAN then analyses the report and computes the probable urgency, before resuming the previous task that it was engaged in. Once it finishes the latter task, it takes up the most important unanalysed report and starts working on that. When no such report exists, it tries to generate possible future problems from the current situation and finds tentative solutions in the anticipation that the problems will occur soon. AIRPLAN adapts a multistage (like progressive deepening) approach in working out solutions for current problems. At first it brands its options to solve a current problem as "bad" or "possible". Subsequently, it refines the initial conclusions and comes up with characterisations such as "good", "o.k.", "poor" and "impossible" for the available options.

2.5.2. L*STAR

The L*STAR system [Laffey 88b] has been developed for real-time telemetry analysis of data received from satellites. The major characteristic of this task is that the system needs to respond to a changing task environment involving an asynchronous flow of events and dynamically changing requirements. The responses are further stipulated by limitations on time, hardware and other resources.

A distributed architecture has been designed for this time-bounded task. The three major processes identified are: inference, data mangement and input/output.

- Inference process (IP) accounts for analysis of the data. It uses frames and time-triggered forward and backward chaining rules for this purpose.

- Data management process (DMP) is responsible for collection, compression and scaling the huge supply of data and its subsequent routing to the inference process.

- Input/output process (I/O) is used to provide an interface (consisting of a hierarchy of schematics with real-time plots) to the operator.

These three processes work independently and communicate with each other via message-passing. Their objectives are to exploit the inherent asynchrony in the overall system, and to

[1] *Demon* is, conceptually, a procedure that watches for some condition to become true and then activate an associated process [Rich 91].

maximise the overall throughput and response. The DMP acquires and compresses the incoming telemetry data, and then sends them selectively to other modules. At the initialisation, it receives messages such as: which data are necessary, which data ought to be sent and to which destinations, whether data should be smoothed, how often data need to be sent from IP and I/O processes. The IP is used to analyse the dynamic data by means of its rules, frames and various statistical procedures. Different means have been designed to invoke these rules or procedures in order to perform the time-bounded analysis of the asynchronous data. Some are temporally-driven and are tested regularly at certain fixed intervals, while others are data-driven (i.e. invoked only when changes occur in some dataset), or event-driven (invoked when a specified goal has to be achieved).

In order to achieve maximum efficiency, rules in L*STAR are compiled into an intermediate postfix format that does not require any pattern matching to occur while the system is running. All the variable bindings are resolved during compilation, and multiple rules are generated from a single one depending on the number of objects to which the variables may bind. This evidently leads to the strong likelihood of a combinatorial increase in number of the generated rules. In order to achieve a time-bounded performance the method dynamically turns off event recordings such as rule firings, procedure calls and data assertions. Even then the uncontrolled growth of the relevant rules makes this approach usable only for problems of "soft" real-time nature. Furthermore, the system cannot respond to a sudden emergency, neither can it provide any response if the time bound is not met.

2.5.3. RT-1

RT-1, a real-time knowledge processing architecture [Dodhiawala 89], has been designed to cope with real-time demands. This general architecture, too, is based on multiple reasoning modules, but here the common link between them is a shared blackboard dataspace and they communicate through signals. The modules run in parallel and asynchronously. When some change in knowledge sources or some change in the blackboard occurs, these events are signalled to the reasoning modules. A decision mechanism, event directory, is maintained to decide the modules that are appropriate (in terms of capability and interest) for a particular event and the events are routed accordingly. The entire reasoning process is event-driven. The reasoning process comprises four steps:

- *trigger step,* for recognising arrival of events and establishing relevant responses to these changes. These events may trigger various knowledge sources that are relevant to them.

  One of the primary components of a blackboard-based system is knowledge sources (KS), i.e. a set of independent modules containing system's domain-specific knowledge. Each KS is associated with a set of triggers which govern the activation of the KS. When a

[1] The other two components are: the blackboard, i.e. a shared data structure for inter-KS communications, and the control system [Rich 91].

trigger fires, it creates an "activation record" (KSAR) describing the KS that should be activated and also the specific conditions that prompt the activation [Hayes-Roth 85].

- *precondition check step,* for checking the different knowledge source activation records to determine if the context exists for execution of the action body in the corresponding knowledge source.
- *schedule step,* which applies different control heuristics to prioritise different KSARs, in order to determine goals of different reasoning modules and thence of the entire system.
- *execute step,* responsible for selecting a subset of executable KSARs and their execution.

The main feature of this architecture is prioritisation of the events. Multiple event channels are used to deal with various simultaneous events. The reasoning process in its cycle attends to the highest priority events and knowledge sources, and pays attention to lower priority ones only when there is no work to be done at the highest level.

2.5.4. TRUCKER And RUNNER

Two other time-bounded systems are TRUCKER and RUNNER, produced by a single group [Hammond 88, Hammond 89]. These rely on case-based rather than rule-based knowledge. They both apply opportunistic memory techniques to create new plans from existing plans for delivery of materials. The techniques involve switching from one set of planning techniques to another when an opportunity arises. TRUCKER controls a fleet of virtual trucks and a city map. The requests that it receives are calls with requests for picking up material from one point in the city and delivering it to another, with an attached deadline. TRUCKER assumes incomplete knowledge of the ongoing problem; hence goals change with time and computation of the best path to a goal is often intractable. The method of forming a plan, given a new request/goal, is to put this request on the agenda of one of its trucks. The module that administers the agenda tries to merge requests to improve use of time, but does so only when it finds an opportunity to satisfy one request while running over the route chosen to satisfy some prior and active request. Plans that are held for possible later use in generating routes for new requests emphasise conjunctive goals. These are usually expensive to generate ab initio, so it is better to find new conjunctive-goal plans by starting from old ones. Stored plans also carry information about their contexts of usefulness or non-usefulness, as a guide to the development of new plans.

2.5.5. Telephone Switching Network

Kopeikina et al. [Kopeikina 88] have considered case-based reasoning (CBR) as a support for a different type of traffic management: routing of calls on a public telephone switching

network. Their software architecture has three main components: indexer/matcher, selector and modifier. The indexer/matcher helps in retrieval of cases that are similar to the situation for which a solution is desired. Because of the application, it is not possible to identify a single set of attributes that is always significant in indexing. Consequently the index scheme is more like a general graph than an information-retrieval hierarchy or tree. The graph uses three types of node: index stations, which contain sets of features that characterise a problem situation best, dispatchers, which can discriminate among index stations, and dispatch ports for connections between the two. A time-limited search procedure scans the network, assigning weights to the nodes according to their relevance. This is an iterative process, except that it is terminated when a time bound is reached. The selector component then picks up the best case from among those that have achieved a high enough weighting during the indexing phase. The choice is made via two criteria: closeness of fit to the given situation, and success/failure estimation. The modifier component then adapts the best retrieved case to the new situation.

Subsequent work [Brandau 91] has suggested introduction of a set of switches to facilitate distributed control operations across multiple network loci. The representation for this scheme also captures some of the temporal and spatial structure of the network.

2.5.6. QES: Quality Expert Systems

The real-time expert system QES, developed at the Steel Resource Centre at Northwestern University [Schnelle 92], is also engaged in time-bounded computations. The purpose of the system is to help its users in assessing and maintaining the quality of the steel produced in different mills.

The quality of the production depends on various process parameters and other extraneous factors such as operating conditions, product locations etc. As the process conditions continue to change during the operation phase of the process, the quality of the product can also change. The QES is designed to monitor the production conditions and to predict possible future problems. Depending upon the nature of the suspected problem it should send appropriate messages either upstream, for resetting the working conditions, or downstream, for possible fixing of the defects in subsequent processing. The functioning of QES is extremely time-dependent, as delay in identifying defects leads to downgrading the produced steel slab or in the worst case scrapping it altogether.

The knowledge for this system has been stored in the form of rules. The rule set has been divided into five categories:

- prediction rules, for monitoring process variables and prediction of defects.

- inspection rules, for confirming (or denying) of predicted defects.
- diagnostic rules, to find causes of inspected defects.
- feed-forward rules, to generate suggestions down the process-line

  and
- feed-back rules, to tell the operator the type of defects.

In order to perform in time-bounded fashion. each rule has a scan interval value attached to it. This is the set time interval at which the particular rule will be fired. Each rule's precondition part tests for certain acceptable values for relevant variables. Each value is associated with some validity tag, describing the interval during which the value will be valid. If this time is exceeded, the processor reads a fresh value for the variable.

2.6. CONCLUSIONS

In the literature, real-time performance has been characterised not only by timeliness alone, but many other features have been identified as important for these systems, among others *asynchronous event handling, shifting focus of attention, guaranteed* (i.e. a priori predictable) *response time, graceful adaptation* (i.e. adjustability to new situations) [Ingrand 92, Dodhiawala 89]. A close look at these systems suggests that *timeliness* or ability to finish a task within a specific time-limit has not been considered as the main objective in most of them, and it has been compromised with other aspects. For example, in AIRPLAN the main attention has been given to attending to the most important problem in hand. On the other hand, in TRUCKER and RUNNER an ad hoc approach has been adopted to exploit any opportunities that may come up in order that the computing time can be minimised. In none of them is finishing the task within a given time bound guaranteed. In systems where timeliness has been considered important, either improved hardware (e.g. parallelism in RT-1) and/or domain-dependent coding approach (e.g. in L*STAR), or ad hoc methods like blind firing of rules at regular intervals (e.g. QES) have been used. None of them can be considered ideal. Ideas like progressive deepening, although this one appears very convincing, have so far had no serious reported practical implementation or study of their implementational aspects. Moreover, progressive deepening appears to need a problem structure that is tailored to it. Therefore, we see a gap between the theoretical studies and the actual implementations of particular techniques. Our work aims at reducing the gap by designing a system that is more akin to human reasoning in time-bounded situations and studying its implementational aspects in reasonably convincing detail.

Chapter 3.

CACHE-BASED ARCHITECTURE: THE FIRST LOOK

3.1. COMPUTATIONS AND CACHING

The relevant lexicant meaning of the word *cache* is a "hiding-place for treasure, stores etc." [Webster 90]. Keeping with this sense, the idea of caching has been developed in computer science as a small fast-access memory, to hold copies of relevant portions of main memory which are likely to be required soon or repeatedly. The speed of performance of a computer system primarily depends upon two of its components: CPU and main memory. For optimal performance of a system, the effective speeds of these two components are required to be at par. With the development of hardware technologies there has been a tremendous increase in CPU speed causing a manifold increase in the demands of accessing to and retrieval from memory while at the same time the size of the memory accessible to the CPU has gone up considerably. However, this development has not been matched with a proportionate increase in memory speed, and as a consequence a problem of diminishing speed in memory-bound operations has arisen. Thus computers of later generations have started suffering from a gap between the CPU and memory performances [Matick 77]. In order to overcome this discrepancy in speed the idea of a cache memory has been developed, where a buffer memory is set up to hold most recently used data for immediate access. Cache memory has been implemented commercially first in IBM 360 machines, in the year 1969 [Ralston 93]. Although primarily developed in a memory management context, similar ideas have later been transported to other areas of computer sciences. For example, many tape units are provided with a cache memory in order to facilitate easy updating with changes in corresponding system configurations.

The principal idea behind caching is "locality of reference", i.e. information is organised in a structured manner in order to facilitate quick identification of the relevant data and easy access to it. As the technique proved to be successful in solving hardware-oriented problems, it is no wonder that it was then taken as a suitable technique for storing immediately-required data in different software developments as well. Use of caching as a suitable means to ensure quick access to data has by now received much attention in different applications, particularly those involving computations with huge amounts of data. Knowledge-based applications can easily be identified as ones belonging to this category, and the notion of a cached answer that is always available for immediate use is not unfamiliar in AI. The use of *default* values in

association with frames, and storing of plans for future reuse, can be described as the caching of answers to the respective problems. The technique used in the RETE algorithm for determining an active subset of the set of rules in a production system by storing instances of patterns, has also been termed as *caching* [Laffey 88].

However, the usual treatment of cached answers in AI differs from those used in other areas of computer science. While in memory management and other tasks caching involves systematic structures (such as array, stack, bucket), the idea of default values is not familiar in those areas. On the other hand, caching in different AI applications implies a quick matching of situations (on the basis of appropriate values/patterns) and retrieval of a suitable value. The utility of appropriate data structures is not of primary importance in these areas. We combine the insights from the two sources, as a foundation for a cache-based architecture suitable for time-critical computations. But in our architecture we extend the notion of cache from a mere repertoire of immediately-usable data to a buffer storage that prescribes solutions/solving techniques of different qualities in compliance with varying temporal demands.

This chapter describes the cache-based architecture designed during the thesis project and different operations involving the cached data and gives illustrations with examples from the AGC (i.e. controlling ground operations in an airport) domain. We also compare the cache-based organisation of knowledge with existing knowledge-organisation techniques.

3.2. FUNDAMENTALS OF THE CACHE-BASED ARCHITECTURE

In a time-bounded domain it is not necessary that all situations will be of equal urgency. In practice, one can expect varying time limits imposed on solving problems in different situations. Consequently, one would expect varying solutions to the same problem depending upon the available time. When the time bound is too pressing the system may not have enough time to try any standard computation. The only possible options for these contingencies should, naturally, be some pre-computed answers. The function of the cache, in these circumstances, will therefore be to provide a ready-made solution appropriate for the current situation. On the contrary, when a temporal bound is sufficiently generous for serious computations, the system should resort to some suitable technique that will consider various aspects of the situation and compute an appropriate solution. (Obviously, we assume that different methods, with varying complexities and hence with different qualities of outcomes, are available for handling the same situations - which is generally true for many realistic applications).

The cache-based architecture proposed here has been designed to capture this expectation in dealing with situations with varying demands on time. The core of this architecture is a cache

that contains the information relevant for dealing with new requests. On one hand the cache contains the *immediate use* values for extremely time-critical situations when an almost instantaneous answer is necessary. On the other hand, for situations with less pressure on time the cache should be able to suggest the best possible problem-solving technique for providing as good a solution as practicable while obeying the imposed temporal bounds. We reckon that the two principal reasoning paradigms in knowledge-based computations, namely rule-based reasoning and case-based reasoning, can serve as the tools for solving problems when the available time permits their application. A further requirement for the cache-based system is to have capabilities that facilitate retrieval of appropriate information from the cache and meaningful reasoning with it. The basic architecture and its operations are described below.

3.2.1. Operational Scheme For The System

The fundamental requirement for the system is therefore a multi-level cache. In our illustrative scheme there are three levels in the cache: *default level, rule level* and *case level.* (This of course not to exclude other paradigms and further levels from the architecture itself). Corresponding to each level there should be some attached time values suggesting under which temporal conditions a particular level will be accessed. Apart from a pertinent rule-base and a case-base of past cases one can also expect a system to maintain auxiliary knowledge-bases of different types such as *temporal knowledge* (relevant to management of time), *action knowledge* (the different actions permissible in the domain concerned) and other relevant domain knowledge (e.g. simple semantics of items in the domain) for characterising and solving different problems. Our system should also maintain interpolation routines which are used in deriving quick solutions. The significance of these different types of knowledge will be explained in subsequent chapters.

Upon receiving users' requests, the control program should interact with the cache for retrieving relevant solving suggestions and then apply interpolations, if necessary, for designing the solution for a current problem.

A schematic diagram of the cache-based system is presented in Figure 3.1. The cache levels reflects our particular experience with applications; in general there is no requirement to have exactly three levels, or exactly the order and type of representations (except for a shallowest "default" level) shown in the figure.

3.2.2. Structure Of The Cache

The above requirements dictate an overall structure for the cache. Standard knowledge representation schemes like rules, semantic networks, scripts etc. are not well adapted to hold the entire contents of a cache. We have chosen an array-like scheme to fill the gap.

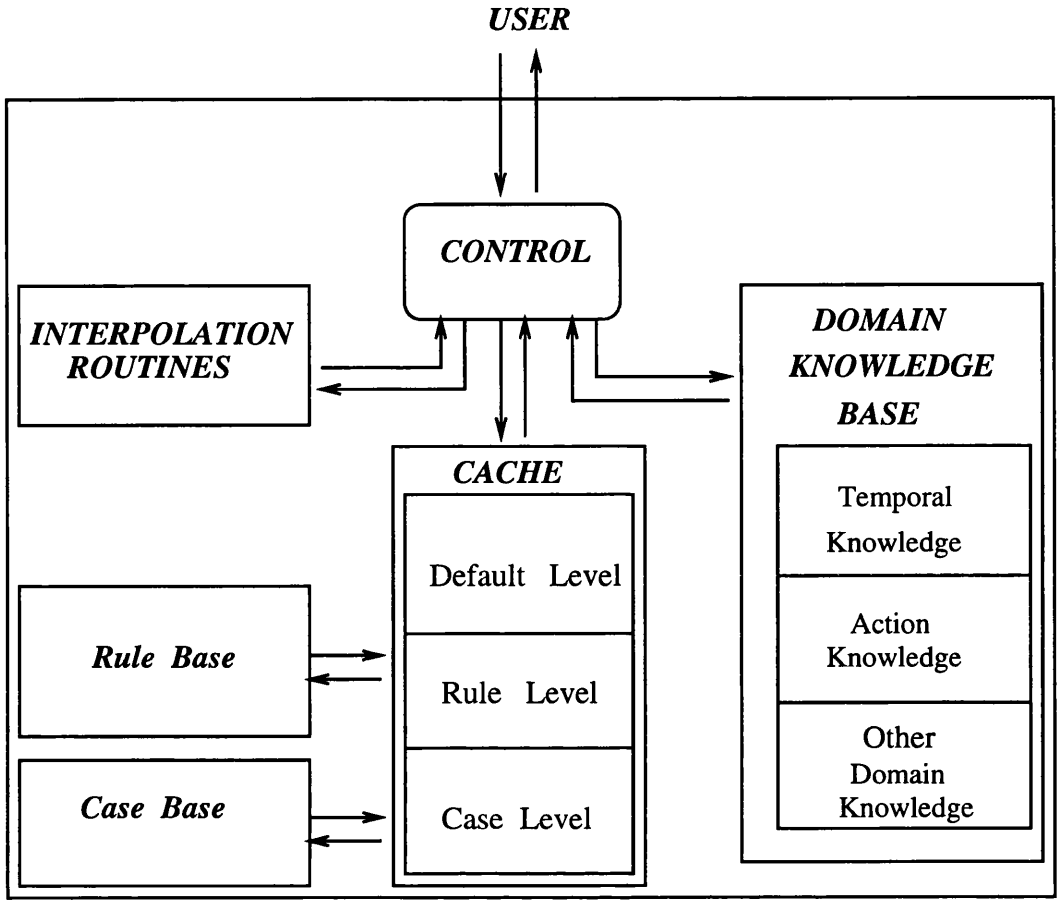


Figure 3.1: Organisation of a Cache-based System

3.2.2.1. The Basic Design

The form of cache that is simplest and easiest to understand is a 2-dimensional array or matrix $C_{i,j}$ with $i \in [1,m]$ and $j \in [1,n]$. Each row holds information on one method of solution of a given computational problem, where normally there are several methods of solving the problem. The first row refers to the method that requires the least computing resources (usually time). This is also likely to be the method whose results are of the lowest quality; for example, assignment of defaults. At the other extreme, row m should refer to the highest-quality method or result that is practicable. Computations on this row will therefore usually have the largest demands on computing resources. The column(s) can be labelled by the key coordinate(s) or value of the input. These are the values that designate a problem and thereby facilitate retrieval from the cache. Each element $C_{i,j}$ is essentially a guide towards finding the solution of a problem when the input coordinate is "closer" to j than to any other value and where the amount of the critical computing resource (time, in our example below) that is likely to be consumed in generating the answer is no larger than the amount associated with row i (but is larger than the amount for row $i-1$; $i > 1$).

Evidently, such a simplistic structure will not work well in a realistic domain as it is unreasonable to hope that each problem can be described using a single feature (represented by the column dimension). Hence for a general cache we extend the above idea to accommodate a set of features that are sufficient to represent a situation in the domain under consideration.

The cache that we propose therefore structurally resembles a multi-dimensional (say k+1) array. The first dimension (which we call *level*) carries information about the specific time limits T_i (i = *1, n*, in an *n*-level cache). Each of the dimensions 2,.. k+1 (called P_1 .. P_k, respectively) stands for some key feature that is essential in describing a problem situation. These key features should be chosen carefully so that they refer to some important (from the point of view of solving it) aspect of the problem. We discuss that issue in detail in chapter 4.

Each column dimension is then divided into several columns, each describing a typical value for the corresponding key feature. These column headers are the values that represent a given situation and facilitate information retrieval from the cache. The column headers along each dimension can be represented in many ways (e.g. using numerical values, symbols, range of values). The actual choice of the column dimensions and their partition into appropriate headers depends upon various considerations such as their relevance to the problem, or subsequent operations to be carried out, to name a few. Discussions with domain experts and systematic analysis of the domain are necessary to come out with the right set of column dimensions (i.e. key features that characterise problems) and the typical headers along each dimension. Often the final choice will be arrived at through trial and error with different sets. With respect to the AGC domain, the indices that are used in the current version of our cache have undergone quite a few changes from the ones that were used in our early versions of the cache [Chatterjee 92]. Each cell of the cache can then be referred to uniquely by (k+1)-tuples (i, v1, v2, .. vk) where each vj (j = *1, k*) is the value for the said problem along the j-th column dimension P_j and i is the relevant level of the cache. The content of this cell will be represented as $C_{i,v1,v2,..vk}$.

3.2.2.2. Basic Contents of the Cache-cells

The primary content of each cell consists of guidelines for how to tackle a problem that is described by the columns of the cache-cell (specifying the current problem). What a "guideline" should comprise will depend on the knowledge representation used in the cell's level in the cache, of course. We store solution-oriented information in a cache-cell in the form of actions that should be carried out for solving the cell-specific problem. A particular cell of the cache contains a set of actions $\{A_1, A_2, \ldots A_k\}$, depending on the level of the cell, the corresponding solving method, and also the column values. Each action A_i may in turn be broken into a sequence of subactions. The subactions can be of various types such as *prerequisites* which ought to be performed before carrying out the actual action (e.g. a prerequisite for sending a patient to a hospital is arranging a car); or *consequents* which follow from the outcome of the suggested action (e.g. a consequent for a failed request for an ambulance is to

request a friend to lend a car for sending a patient).

Contents of the *default-level* cells may be a series of unconditional instructions. These should be the most generalised actions that can be performed to mitigate a current problem even without going into its finer details. The cache cells, in this case, may contain the sequence of instructions explicitly, or if the same set of actions is repeated in many cells it is efficient from a storage point of view to hold them in some known place whose address will be stored in the relevant cache-cells. With respect to the AGC domain some examples of generic instructions are: *evacuate <location>, pass the message to <person>, control the spread of <object>;* where the relevant parameters are determined from the context, i.e. the problem description.

For a level associated with applying *rules* (and where the corresponding time available is enough to permit their application), the reasoning comprises invoking the inference engine, applying appropriate rules etc. to determine the right set of actions to be carried out. The major advantage of using the cache, here, is that the cells at this level contain pointers/references to appropriate sets of rules, so that the time required for searching rules and matching preconditions is minimised. The rules should be designed in such a way that they improve upon a basic generalised solution, by taking into account various constraints that may be relevant in a particular situation. The system needs to compute values for the key variables with the help of the features of the situation. The preconditions of the rules should test these values and qualify the actions through their postconditions.

For the *case-level* (i.e. a level where the knowledge is stored in terms of past cases) the reasoning steps may be retrieving an appropriate case, measuring the similarities and dissimilarities of the retrieved case with the current situation, adapting the old case to the new situation etc. Each cache-cell in this level should contain pointers/references to an appropriate set of cases to obviate a search through the entire case-base in choosing an appropriate case.

There are three major components of a case [Kolodner 93] (ch. 5)] that are important from a reasoner's point of view:

- features of the problem description and relationships between parts;
- constraints on these goals;
- goals that have been achieved (and how) in solving the problem.

Once the right cache-cell is identified with the help of the current problem description, the system uses the imposed constraints to examine the subtleties of the situation which govern the ultimate selection of the cases associated with the cell concerned. The actions taken in achieving the past goals then provide the requisite cues for solving the current problem.

Initial values for this information must be inserted by hand, after running a test set of problems, but these values can then be updated automatically as the cache is used in processing

further examples. For efficient functioning of the cache-based architecture one needs to store some additional information in the cache-cells. We shall discuss these issues in later chapters.

3.2.3. Illustration With An Example

To illustrate the points made above, we consider one particular problem from the AGC domain:

> *In the airport a fire has just broken out and the Ground Operations Controller needs to take appropriate action to mitigate the problem, keeping in mind that safety of people is the most important objective in the event of such a trouble.*

A cache that has been designed to deal with problems of this kind has three column dimensions:[1]

Potential-degree-of-threat, representing how dangerous the object is as far as safety is concerned. For example, a bomb, in general, is potentially more dangerous than fire;

Location-of-the-event, representing the location of the airport where the threat has appeared. This is important because the actions to be taken depend largely on the location concerned, particularly when it involves safety of passengers and/or damage of expensive items, such as aircraft;

Activity-level-of-the-airport. The steps that should be taken depend also on how busy the airport is. This is particularly important when cancellation and (re)scheduling of flights are involved.

For each of these three dimensions we have used some typical values. For illustration, the activity level of the airport can be described with the help of the flight density, i.e. the number of arrivals and departures of flights in the next hour say; and we use three different values, namely *low, normal* and *high,* to describe the ground operation controller's intuitive appreciation of a particular situation. Regarding treatment of time, we consider four different methods for solving this problem. Classical planning [Nilsson 82] is likely to give the highest-quality results, but will often take too long. This is, therefore, attached to the deepest level (4th row) of the cache and is not of interest for the present thesis. Contents of the cells in other levels are filled in according to the design choices made in the previous section (3.2.2.2). Thus we advocate a 4-dimensional cache for dealing with the AGC problems.

It is easier to understand such a structure if presented visually. But the difficulties of drawing a 4-dimensional cache compel us to rule out that idea. However, towards the end of this

[1] In chapter 7 we demonstrate reasoning with such a cache. however for simplicity the discussion there has been restricted to only the first two of the column dimensions.

chapter we discuss an example from a numerical domain, and a pictorial representation of the related cache with a single column dimension is provided there. The 4-dimensional cache for the AGC application will be apprehended better if considered as a structural extension of this one (despite the significant difference in the contents).

3.2.4. Basic Cache Operations

There are three major operations associated with a cache that ensure its most versatile and smooth functioning: **retrieval, knowledge interpolation** and **propagation.** We introduce these operations here. While 'retrieval' and 'interpolation' will be dealt with deeply in subsequent sections, the issue of 'propagation' has not been examined in any detail in this thesis.

3.2.4.1. Operation of Retrieval

By 'retrieval' we mean accessing particular cache cell(s) (by the relevant column indices) in order to pick up the instructions stored there for potential use in the given situation. The difference in the methods stored in each level makes retrieval a non-trivial issue for the designer because (as illustrated in section 1.2.1) the aspects of a situation and hence the key features that one deems to be important in solving a problem vary with the available time. The key issue therefore is to identify the most appropriate features for each level of the cache. In chapter 4 we discuss the issues concerning selection of appropriate indices and their subsequent use in measuring similarities with a given situation for a cache.

Further complications arise because in a realistic domain it is unlikely that a problem will always match uniquely with some column header along each dimension. In the event that there is no exact match, it is necessary to consider more than one cell of the cache. Additionally, the issue becomes more complicated when a particular cell that the method accesses for retrieving a solution is found to have null contents. The system should have tactics to circumvent these difficulties. Chapter 7 discusses the issue of retrieval in more detail.

3.2.4.2. Operation of Interpolation

Our initial idea of 'interpolation' has been a means of approximation to the solution on the basis of contents of adjacent non-void cache-cells in the event when a given problem does not match exactly with the column headers of any cell. However, we found later that interpolation needs to be applied even when only a single cache cell is chosen for solving a problem. This is because we have observed that a deeper-level (i.e. case and rule levels) cell of a cache is normally associated with a number of pertinent sources of information (i.e. cases or rules); and the method needs to select from among them the ones that match with a current situation. More subtle features of a situation are used to make these selections. Evidently, here too, one

may expect non-exact matches of the current problem with the cached ones and thus the possibility of a need for interpolation occurs.

In a problem-solving domain, where the purpose of the cache is to provide suitable actions to settle problems, "interpolation" can be of various types. For example, the method may interpolate between values for parameters describing features of the problem or it may interpolate between parameters of an action, or even between different actions.

However, the obvious difficulty here is that the values the system deals with are not numeric in general, so that (unlike a numeric domain) no straightforward ordering may exist between them for possible interpolation. We claim that it is possible to impose metrics on most apparently non-interpolatable features, thus facilitating the computation of a distance between two entities and thereby the application of interpolation. Chapter 5 discusses our ideas regarding interpolation-related tactics for symbolic domains.

3.2.4.3. Operation of Propagation

Since the total number of cells in a cache grows exponentially with the number of headers in each column dimension, in a a realistic domain one may expect a cache that is very large in size. It is unreasonable in these situations to assume that the cache will have non-void entries in all cells right at the outset (as there may be no previous information on the record concerning an identical or closely similar situation to the current problem). It is therefore highly desirable for improved performance of the cache-based system to upgrade its initial contents or incorporate some solution in the void cells as more problems are solved in course of time using the cache. The third operation 'propagation' aims at improving the utility of the cache, by which we mean filling the empty slots of the cache, or improving the quality of solutions stored at intermediate levels of the cache. The basic idea is that information from good-quality solutions in cells at the level of row r should be propagated upwards to improve the quality of cell contents in rows s, where $s < r$.

Typical time-critical computations do not require that all of the available computing resources be applied to responding continually to urgent inputs. It can be expected that there will be periods when some computing power can be devoted to improving the general quality of the information held in elements of the cache. If the initial state of the cache consists of low-quality values (e.g. a set of default values in row 1, and no values in other rows), resources that are not needed for urgent tasks can be used in computing or improving the values in all rows. While values in any row can be computed by techniques available within that row, the ideal situation is one in which the best value $C_{m,j1,j2...jn}$ for any set of column headers *j1, j2, .. jn* is also present in elements $C_{m-1,j1,j2,..jn}$, ... $C_{1,j1,j2,..jn}$. This ideal is never likely to be achieved in practice, because of the time-varying properties of realistic applications, and because of the difference in representational conventions between rows, but a cache system should nevertheless try to improve its cached values in the direction of the

ideal. This can be achieved by moving the values at deeper cells to the shallower cells with the same column indices. For example, a solution obtained by rigorous planning can be stored as a case, a case that is referenced often can be stored as a 'default' solution etc. (Evidently, the system needs to maintain appropriate statistics in order to facilitate effective propagation).

In general this will not be a simple overwriting of $C_{k,j1,j2,..jn}$ by $C_{m,j1,j2,..jn}$, because the different techniques used for computing within rows k and m may imply modification of the contents in level m for consistency with the conventions of row k, before the cells can receive the propagated value safely. Thus the idea of propagation in itself is a research topic related to machine learning and probably also to so-called "knowledge interchange" formats (KIF). We do not consider the issue any further in this thesis.

3.2.5. Treatment Of Time

On encountering a problem, a system designed for time-bounded performance should be able to make a plan which can be carried out within the specified time-limit. We observe that the time required for solving a problem can be computed by considering two aspects: *computational time* and *physical time.* By computational time we mean the time that is required for planning a solution (which varies along with the methods applied), while physical time is the time required for actual execution of the plan. However, it is worth noting that physical time is essentially domain-dependent and requires a human expert's advice in choosing the appropriate time-limits during the design phase of a system. An ideal cache stores against each proposed method the (updatable) average temporal requirement for its computation and also stores against each action (and/or subaction) the average time requirement for its completion, plus information on the variability. The advantage of storing these pieces of temporal information is that the system can then use them to determine the best practicable level at which to access the cache in a given situation. For example, if the average time taken by the relevant action in some level exceeds the allowed time limit, it is not worthwhile for the system to invoke that particular action or to access that level.

The same temporal information, stored against each level, can be used in different ways in different situations. For hard real-time problems the method should be such that it guarantees completion within the time bound. Naturally, the time-points T_is are to be chosen appropriately.

Let the average temporal requirement and its standard deviation for the method corresponding to the j-th level be t_j and s_j respectively. Hence one possible selection criterion for T_j will be:

$$T_j = t_j + s_j * 3$$

when the time constraint for the current problem is hard in nature. On the other hand, for soft

real-time problems it is possible to govern selection of time point T_j (j = 1,2, .. m) by:

$$T_j = t_j + s_j$$

Consequently, selection of an appropriate level in solving a current problem with allowable temporal stipulation T will vary with the nature of the problem. For hard real-time situations level *i* will be selected, such that:

$$T \geq t_i + s_i * 3$$

But for soft real-time situations the selection of *i*-th level may be governed by:

$$T \geq t_i + s_i$$

and is thereby likely to give a higher-order (in qualitative terms) technique a better chance to be employed.

We demonstrate selection of the limits for problems of hard and soft natures and evaluate the performance in section 9.3 when we discuss experimental results from the shortwave radio domain.

3.3. DESIGN CHOICES FOR THE CACHE-BASED ARCHITECTURE

Knowledge required for successful implementation of a cache-based system can be classified into two categories: *domain-specific* and *cache-specific.* Like any problem-solving system, the cache-based system too should hold knowledge on various domain-specific aspects, e.g. nature of different problems, their consequences, possible mitigating actions, side-effects (if any), degree of importance of a problem etc. Such items belong in auxiliary tables or knowledge-bases (i.e. outside the cache), and require only routine treatments in terms of acquisition and use. But, the cache-specific aspects of the knowledge deserve special attention and treatment right from the design phase of the architecture.

The most important tasks in building a cache-based architecture are:

- choosing appropriate indices for general specification of problems;
- design of interpolation for cache entries;
- identification of appropriate time points $T_i s$ and selection of suitable methods corresponding to each T_i.

The following subsections discuss various cache-specific decisions that a system designer needs to make during implementation of the architecture.

3.3.1. Choice Of Indices

One needs to decide on suitable indices to represent a current situation and also to refer to the appropriate cache-cell. However, selection of appropriate indices suffers from the following difficulties:

On one hand we find that any real-life problem has numerous contextual features [Schank 86] each having its own role within the problem description. But use of too many features not only leads to a huge cache (which may even be unmanageably large) but also increases the access time and therefore interferes with the basic purpose of the cache. On the other hand, insufficient features do not represent a situation properly. Consequently, the system has to rely to a considerable extent on its reasoning process which inevitably becomes time-consuming and the utility of the architecture becomes questionable. These two conflicting facts introduce a need to strike a balance between these two extremes and make the task of indexing rather challenging.

Another major question that needs to be addressed is how to identify or choose the column headers for the cache. The indices not only represent a current problem but also provide access to an appropriate cache-cell through their respective values. Therefore, the column indices should describe a particular situation in sufficient detail to allow access of the best column in the cache plus any relevant manipulations of that entry (e.g. interpolation) so that the situation can be judged properly and possible remedial steps can be decided upon.

Features of a problem (that lead to appropriate choices of column indices) can be divided into two categories, viz. 'raw' features, those are explicit in the description of the problem, and 'abstractions', or those that are not apparent in the problem description and must therefore be inferred. The problems of using 'raw' features for indexing are twofold:

- for most of the likely realistic problem domains, the volume of 'raw' features is prohibitively high for their straightforward use as cache indices.

- secondly, in a time-critical situation a solver may be concerned more with the implications of the problem rather than its minute details. Raw features do not necessarily provide the required insight.

Hence, use of abstractions appears more appropriate as the choice for suitable indices. The abstractions should be such that they highlight the effects and implications of a problem, so that the remedial steps can be computed easily. However, describing situations using abstractions does not guarantee a set of features that represent all possible situations that the system may expect. In practice we have found that, even for a moderately complicated domain, isolating a single set of features for representing different types of problems is not straightforward - different sets of features appear to be needed. For these situations we suggest

classifying problems into certain equivalence classes so that a common set of recovery tactics will be successful, at least in time-critical situations, for problems belonging to the same class. We then recommend maintaining more than one cache, where each is dedicated to a particular type of problem. It is easy to supplement this scheme with a decision mechanism which evaluates a new situation and refers it to the appropriate cache.

In the absence of a strong domain theory, collection of the abovementioned knowledge becomes a non-trivial task. But we feel that descriptions of past incidents in terms of cases and case-based reasoning can serve as good sources of knowledge in these situations. Past cases help in identifying different types of problems, the key features of problems, and methods and actions for solving a current problem, along with tentative ideas about time requirements and many other aspects of the domain.

In addition to basic knowledge about local effects of actions or situations (expressible by rules or inheritance hierarchies, say), a good KBS will usually contain metaknowledge regarding the particular actions that may be best in a given situation. The cache architecture is well adapted to this picture: metaknowledge can be used, for example, to group cases with similar features into clusters so that one column in the cache refers to each cluster of cases.

3.3.2. Design Of Interpolation

The basic purpose of the interpolation is to determine solutions for a current situation in the event when the designated cache-cell is empty. However, the nature of the interpolation depends on:

- *the level of the cache,* i.e. the solving method. For example, interpolation between rules is different by nature from interpolation between cases although the basic purpose may be the same. For each type of interpolation the system needs to identify the independent variables and the dependent variables on which to carry out the interpolation. While the problem features may serve as the independent variables, the dependent variables and their representation may change along with the level of the cache. A system should have capabilities to adjust its interpolation methods accordingly.

- *availability of time and other resources.* As we see later (in chapter 5), even within some particular level of the cache more than one way of interpolation can be proposed. Here too, the quality may vary, with a computationally more expensive technique providing a qualitatively better result. This can be explained better with a numerical example. In order to approximate the value of a function between two known points one can have different interpolation techniques such as linear, quadratic etc. Not only do these techniques have different computational complexities, but they differ in their basic requirements as well. While linear interpolation needs two known points for its accomplishment, quadratic interpolation requires three. Similarly, for the symbolic quantities too one can think

of different interpolation tactics. In chapter 5 we discuss in detail the tactic of knowledge interpolation with its different variations and their requirements. A cache-based system needs to have knowledge to evaluate a current situation and decide which particular method should be employed for successful interpolation.

For some types of problem, e.g. as in numerical analysis (for which an example of a computation is given in section 3.4), inputs are real-valued coordinates; therefore identification of typical column headers, matching of given problem with them and conducting subsequent interpolation are straightforward. When a problem is less well structured, as in much of AI, the coordinates may have only a partial order. Identifying right columns and performing retrieval and interpolation on the cached information then become more complex. But this does not affect the basic interpretation of the cache and primitive operations on it.

3.3.3. Identifying Different Time Limits And Corresponding Methods

Both "computational" and "physical" times (see section 3.2.5) should be considered in selecting the temporal thresholds T_is. The reason for taking physical time into account is twofold:

- dealing with a current problem involves not only designing a plan but also ensuring its proper application. If the system exceeds the allotted time limit in executing the generated plan, the whole exercise fails.

- often, at an intermediate step of designing a plan the solver requires certain values which are not readily available but are crucial to the solver to decide subsequent steps of the plan. The solver may have to collect this information from some secondary source or may depend on the outcome of some physical activity. For example, a person having a problem with a vehicle may defer any decision whatsoever until the vehicle is inspected by a mechanic and whose opinion about the nature of the problem becomes known.

Consequently, the designer should be careful in choosing methods corresponding to each time limit T_i, and hence to each level of the cache. For example, consider the following problem with respect to the AGC domain:

> *The airport authority has received an anonymous telephone call stating that a bomb has been planted in the terminal building which will explode after x minutes.*

The action that the GOC should take to deal with this problem depends on the available time. While the ideal solution is to evacuate the place and cordon it off securely and to send for appropriate authorities to find and defuse the bomb, along with rescheduling of any flight that may be affected due to this evacuation and cordoning, the modus operandi will vary with the

time in hand. Let us assume the average physical time involved in contacting the security personnel and their arrival on the spot and taking charge of the situation is 10 minutes.

Case 1: If the available time is less than 10 minutes (i.e. $x < 10$), the priority for the GOC is to ensure the safety of the passengers and therefore to effect the evacuation operation first. Although the best way of doing it to send security personnel to the spot to persuade the passengers gently to evacuate (in order to avoid any melee), the lack of time forces the GOC to make quick announcements through a microphone for evacuation (possibly aided by sending a few airport staff to take care of any untoward incident that may occur due to panic). Also, the GOC is forced to delay any flight affected by this threat indefinitely, due to the lack of time for a proper rescheduling.

Case 2: If the available time is sufficient for the security personnel to come to the spot to take charge of the situation, the GOC will simply send a few airport staff there to dissuade other passengers from entering and spend more time in rescheduling the affected flights so that the trouble causes minimum delays to the flights.

Case 3: However, if there is enough time for smooth evacuation and replanning of the flights, the GOC may try to explore the possibilities of the threat being a hoax, before taking any preventive actions which are likely to disrupt airport's normal activities badly.

The ultimate decisions regarding the temporal thresholds and corresponding methods depend largely on the particular domain. One needs to carry out a series of experiments to reach any conclusion about the temporal requirements of different related actions. Depending upon the nature of the problems, i.e. whether they are *soft* (i.e. the extent of damage in the event of a failure to maintain the temporal stipulation is not too serious) or *hard* (i.e. any failure leads to disastrous effects) one may choose to use average temporal requirements or maximum temporal requirements, respectively, as the temporal tags that determine for each cache level the appropriate scheme of activity for a given problem.

3.4. FUNCTIONING OF THE CACHE

We tested the cache-based architecture first on a numerical domain to ascertain and consolidate our ideas about this. The results of this experiments are summarised here. The numerical-analysis example has validated the overall design choices within the cache architecture, and has shown that the architecture itself is viable. This gave us the confidence to extend our ideas to encompass far more complicated AI problems.

3.4.1. An Example In Numerical Analysis

Suppose that we wish to solve the following differential equation:

$$\frac{d^2y}{dx^2} + A^2 (x - t_0)\, y = 0$$

for arbitrary values of the independent variable x in [0, 1] with the boundary conditions:

$y_0 = y(x = 0) = 0.5;$
$y'_0 = y'(x = 0) = 1;$
$t_0 = -1;$
$A = 3$

There are recognised series expansions for such problems [Conte 80], but evaluation of the series (which may have slow convergence) is likely to be time-consuming. If results are needed in a time-bounded situation, it will be useful to have other methods available that may be less accurate but that also require less time. Each method can be associated with one level in a multi-level cache. We assumed that a suitable accurate (series) method belongs to the deepest level of the cache, and devised a non-trivial demonstration of the cache architecture by considering three other methods that were more appropriate. In decreasing order of quality and demands on time, the three methods that we have chosen are 4th-order and 2nd-order Runge-Kutta and quadratic interpolation, which we have respectively associated with the fourth, third and second row of the cache. We also assume that in extreme time-pressed situations the system cannot offer anything better than a default value or a linearly interpolated result on some stored values, since no significant computation is feasible. Hence the cache in this instance has five rows, and to demonstrate the time-dependent behaviour we concentrate on the first 4 rows as described above. The columns refer to values of the independent variable x, and we divide the range of possible inputs into a convenient number of non-overlapping zones (e.g. 5), with each zone belonging to one column of the cache. Entries in each cell of the cache comprise a cue to the relevant method for obtaining the solution, a typical value of x for the corresponding column, value of y for that x, and an indication of the quality of the result for y. If there is no other way to assess quality, one can use the row index as a crude ordinal approximation. Figure 3.2 shows a schematic diagram of the cache.

To start this computation, each cell is filled in the following way. For the default-level cells, in the absence of any other information, x is taken to be the midpoint of the corresponding zone, and y is determined by the best available (series) method. For the second level, in order to facilitate quadratic interpolation, we store the x and y values corresponding to three points. For each cell these three values correspond to the lower limit, midpoint and upper limit of the values associated with the cell concerned. In the cells associated with the Runge-Kutta methods we store the required values (i.e. for x, y and y') corresponding to the lower limit of the associated range and the number of steps (100, here). [2]

[2] During actual use, the x-values in different cells in the same column may become different, due to propagation, though they will always fall within the limits defining the zone for that column).

VALUE / TIME (in ms) & METHOD	*0.0 - 0.2*	*0.2 - 0.4*	*0.4 - 0.6*	*0.6 - 0.8*	*0.8 - 1.0*
30 *#Default*	*(0.1 0.5753)*	*(0.3 0.5492)*	*(0.5 0.2795)*	*(0.7 -0.1309)*	*(0.9 -0.4609)*
150 *#2nd-Order Interpolation*	*(0.0 0.500)* *(0.1 0.5753)* *(0.2 0.5941)*	*(0.2 0.5941)* *(0.3 0.5942)* *(0.4 0.4409)*	*(0.4 0.4409)* *(0.5 0.2795)* *(0.6 0.0777)*	*(0.6 0.0777)* *(0.7 -0.131)* *(0.8 -0.3233)*	*(0.8 -0.3233)* *(0.9 -0.4607)* *(1.0 -0.522)*
400 *#2nd-Order Runge-Kutta*	*(0.0 0.500 1.00)* *100*	*(0.2 0.5941 -0.1238)* *100*	*(0.4 0.4409 -1.3801)* *100*	*(0.6 0.0777 -2.1025)* *100*	*(0.8 -0.3233 -1.6937)* *100*
650 *#4th-Order Runge-Kutta*	*(0.0 0.500 1.00)* *100*	*(0.2 0.5941 -0.1238)* *100*	*(0.4 0.4409 -1.3801)* *100*	*(0.6 0.0777 -2.1025)* *100*	*(0.8 -0.3233 -1.6937)* *100*
#Series-Method	--	--	--	--	--

Figure 3.2: Schematic Diagram of a Cache for the Numeric Problem

Key:

A pair (a b) => value of x, and corresponding value of y.

A triple (a b c) => value of x, and corresponding y and y' values.

A number n => number of steps in corresponding Runge-Kutta method.

Costs for computation (in terms of time in milliseconds) in each row are entered, after repeated running of that row's method on trial cases. (The cost information is updated by measurement of behaviour of that row during actual use. We store the cost information in the form of pairs suggesting the number of trials and the total time, as this facilitates easy updating of the information. In Figure 3.2 we have shown values that have been calculated through running each method 100 times). To test the time-bounded features of the design, requests to compute the value of y for a given value of x are presented along with time limits that fall within the times that are the "cost information" that has just been mentioned. In operation, the program that administers the cache determines the row to access from the cost information and the time bound attached to the input.

If there is an exact match between x and one of the x-values held in a cell of that row, the corresponding y-value is retrieved. If there is no such match, similar scans are made at deeper levels of the cache. If there is still no direct match, interpolation is used, at the deepest level of the cache where the row cost information indicates that the computation should end before the remaining time available expires. Alternatively, it is possible to prefer an exhaustive computation by using the method attached to cells in some deeper level, and to propagate the result upwards for use where it was first requested. These are approaches at two extremes, trading quality for immediacy in the first instance and doing the opposite in the second. One can think of mixed methods that choose one level and one approach on the basis of some combination of the two considerations. We have not done enough experiments to draw any general conclusions about mixed methods, but this is an avenue that is worth investigating in any future application where the cached information may be as homogeneous as in the numerical-analysis example.

3.4.2. Observations From Computational Experiments

In the numerical problem indicated above, we have repeated the testing of computing y for different values of x in [0,1] at steps of 0.05, after deciding on reasonable first estimates of the cost (time) and its variation, for computations at each of the three non-default (i.e. quadratic interpolation, second-order and fourth-order Runge-Kutta) levels. We have assessed the success of the tests in terms of the percentage of computations that (in retrospect) have used the best combination of retrieval, interpolation and propagation. ("Best" means that no identifiable combination would have produced a y-value of higher quality by the criteria of numerical analysis within the time bound supplied along with x). We have concentrated on the percentage of computations that have produced a result while respecting this time bound.

In the experiments, the computations have been entirely successful in choosing the best simple combination of the three basic operations.[3] Nevertheless, according to the primary consideration of respect for the time bound, performance is not perfect. Adopting adventurous

[3] However, we have not tried to define complex or hybrid trade-off criteria between cost and quality.

tactics, i.e. choosing to do a computation that is estimated to require a time close to the time bound, we found that as many as 12% of the trials exceeded their bound, and 1% of these exceeded it spectacularly (e.g. by a factor of 4 or 5). The measurements are unsatisfactory in that the readout for our system clock is in quanta of 16.6666.... milliseconds, rounded to the nearest integer, while bounds can be requested arbitrarily finely. We therefore have to allow a leeway of one such quantum in comparing measurements with time bounds. When we then try an unadventurous approach, being cautious about choosing a method of computation that approaches the bound, our recorded failure rate is about 0.5%, due entirely to spectacular failures. This suggests that the cautious tactics can in fact guarantee success, except for cases that are so peculiar that their behaviour is due not to our programs but to interference with the clock readings from other users in our time-shared system (i.e. it is likely that the times debited to user jobs and the running of system utilities are not all correct). We have seen similar effects in the more complicated knowledge-intensive tests that we report in chapter 9. Success in this experiment (simplified notwithstanding) belonging to the present chapter has encouraged us to go ahead with the knowledge-based problems.

One question that arises at this juncture is whether the caching scheme is pertinent to all available AI reasoning paradigms. Evidently, the principal requirements for building an effective system is ability to estimate the temporal requirements under the paradigms concerned. We have designed the cache and associated reasoning tactics (e.g. knowledge interpolation) such that not only a quick access to relevant set of knowledge items be ensured, an algorithmic treatment can be inflicted on the pieces of knowledge. The latter however is not guaranteed with respect to all reasoning schemes. For example, the "backward chaining" in Prolog, where finding solutions to a problem is based on the principle of backtracking. Evidently, estimation of temporal requirements in such a context has no logical justification. Hence a system designer has to be careful about the knowledge representation and reasoning paradigms that are to be employed to operate within the cache-based framework.

3.5. ADVANTAGES OF CACHING

In this section we compare the caching scheme and retrieval therefrom with other retrieval techniques that exist in AI, related to other storage schemes of information, to evaluate the caching scheme in contrast with other methods.

The issue of retrieval has been dealt with extensively in database systems. In artificial intelligence the issue of 'retrieval' is more prevalent in case-based reasoning (CBR) than any other reasoning paradigm. However, as pointed out by Kolodner [Kolodner 93, (ch. 8)], there is a subtle difference between database search and searching in CBR. While database search requires one to find an entity that matches on the keys exactly, CBR search indulges in

looking for a case or set of cases that match the current case in fulfilling the underlying purpose. Moreover, for a domain it is not guaranteed that one will find an exact match for a current situation. Hence an exact match in key features is not deemed to be essential for cases. Even a partial match serves the purpose here. From this point of view retrieval from the cache is comparable with retrieval in CBR.

In different CBR systems different techniques have been applied for case retrieval. For example, some systems (e.g PROXIMITY, GROWTH [Kibler 87], SURVER [King 88]) retrieve similar cases by scrutinising each individual case in the case-base, while in some other systems (e.g. MEDIATOR [Kolodner 89b], JULIA [Kolodner 88a], HYPO [Ashley 88, Ashley 89b]) retrieval is accomplished by using rules/heuristics on a selected subset of partially-matched cases. There are more advanced methods of retrieval, as well, which have been used in different systems. Any retrieval technique invariably involves some searching and matching processes.

However, a study of these CBR systems [Bareiss 89] suggest that generally the process of retrieval is differentiated from the measurement of similarity or matching. While similarity measurement is primarily concerned with how to make comparisons between two cases, the process of retrieval is concerned with developing systematic approaches to comparing the cases of the case-base with a current case. Thus for the retrieval procedures the order of comparison is important and not how exactly the comparison is carried out and similarity is evaluated. Retrieval is often greatly influenced by how the cases are stored in the case-base. In different CBR systems different retrieval techniques have been devised in this respect, each having its own merits and demerits. A study of these techniques helps in identifying their limitations with respect to time-bounded computations.

3.5.1. Different Existing Retrieval Techniques

Depending upon how cases are organised, two major streams of retrieval techniques can be identified:

- Flat memory organisation of cases
- Hierarchical organisation of cases.

For either type of memory, there are several different retrieval techniques in the literature.

3.5.1.1. Retrieval from Flat Memory

In a flat memory the case-base is arranged sequentially, i.e. cases are stored one after another in a linear fashion. Hence each case in the case-base has to be considered separately for similarity measurement.

Retrieval, for such a memory organisation, can be accomplished in one of the following ways:

- **Linear Search,** where each case of the case-base is compared sequentially with the current problem, using some predefined matching function, and the individual degree of matching for each case is remembered separately. The final choice is of those cases that have matched best with the current situation. CLAVIER, a system to make autoclave load designs [Barletta 89], uses this technique for retrieving past cases in order to design lay-outs for a current problem.

Two possible improvements of this simplistic scheme are:

- **Use of shallow indexing,** when indices deeper by one level are used. Here, the system isolates a set of descriptors as its indices. Each index points to those cases in the case-base that have the said descriptor in their representations. When a new situation is encountered, the set S of descriptors related to it is identified, and these descriptors are then used for retrieval purposes. Here, instead of the entire case-base only those cases are considered for matching that are pointed to by some member of the set S. SWALE, a story-analysis system to explain anomalies, [Kass 88] uses this retrieval scheme.

- **Partitioning the case-base.** Here the entire case-base is partitioned into several groups such that cases belonging to the same group are intrinsically similar. In order to make an appropriate retrieval, the system analyses a new situation to identify which particular group has the new situation as a member. A linear search, restricted only to this group, is then conducted for case retrieval. In the event of these groups becoming too large, one can use secondary indices to subdivide the concerned group further. Such partitioning reduces the search horizon and hence make quicker retrieval possible.

3.5.1.2. Hierarchical Memory Organisation

Hierarchical organisation of a case-base comprises building a tree-like structure where the root node stands for the entire case-base, and the cases are represented as leaf nodes of the tree. the intermediate nodes of the tree represent various concepts which partition the cases into some meaningful clusters. A retrieval procedure, here, involves traversing through the tree, making a decision at each level about which branch of the tree is to be followed.

Depending upon which clustering method is used and how the tree is generated, several variations of how to carry out retrieval exist:

3.5.1.2.1. *Breadth-First Graph Search in Shared-Feature Networks*

A shared-feature network clusters cases in such a way that the cases having enough common features are stored together. Each intermediate node then holds features that are shared by all its subordinate nodes. The cases that share these features are descendants from this node.

In order to retrieve a case from this tree, the general method first compares the current situation with all the nodes of the highest level of the tree. Once the best-matching node is selected, the search is restricted to its descendants only. The same procedure is repeated until a leaf node (i.e. a case) is reached.

3.5.1.2.2. *Depth-First Graph Search in Prioritised Discrimination Networks*

An alternative to the shared-feature network is a discrimination network, where each intermediate node is associated with a question about some feature of the cases in the domain. Each descendant node stands for one answer to the question asked for its ancestor, and holds those cases that comply with this answer. Each of these child nodes in turn poses another question that subdivides the answers further. This organisation continues until leaf nodes comprising cases are reached. Normally, the net is so arranged that questions about the important dimensions are posed at higher levels of the tree, so that cases matching along more important dimensions are retrieved.

This retrieval involves a depth-first search. The highest-level node poses a question regarding some feature of the current situation. The descendant node that seems to be the best match for the current situation is considered for the next step of exploration. If the case that is found in this way is not acceptable, then the system backtracks to the previous node to traverse another descendant path of the tree.

3.5.2. Cache-Retrieval vis-a-vis Existing Retrieval Techniques

Each of these major retrieval techniques has certain limitations in straightforward applications for a time-bounded computation. The flat memory organisation is inherently slow, as retrieval here involves comparison of all candidate cases (either an entire case-base or some partition) with the current situation. Although the method is capable of a better-quality solution, the task here involves unnecessary comparisons with cases, most of which are not useful for the given situation. Abstractions of some kind that help the system avoid this extra work is evidently more appropriate for the purpose, if available. An hierarchical memory organisation facilitates this approach.

Although the two fundamental hierarchical structures that have been described earlier are helpful in partitioning cases suitably for efficient retrieval, they too have disadvantages when applied to time-bounded problem-solving. These disadvantages arise from the following perspectives:

- if the solution that the system retrieves is not found suitable, one needs to redo the searching. For the structures described above, this involves backtracking and search forward along a different path. This process does not involve a fixed time, as the time taken depends on the number of levels through which the search procedure has to backtrack before proceeding along a different path. The task becomes more difficult if the system is unsure about the actual level of the hierarchy at which it made a wrong judgement. Further, any remedies to this problem involve multiple redoing of the backtracking exercise.

- use of interpolation on the tree-like structure is not straightforward, because interpolation involves identification of two solutions which differ in a single dimension. Accomplishing this task in the tree-like structure involves the same problem as stated just above.

- tree-like structures are not suitable for handling missing information. If the answers at some node of the tree are not available, then three options are left for the system: end the search process, continuing search on all descendant nodes, or search along the most likely alternative path; all of which have their own drawbacks [Porter 90]. Abandoning a search because of one missing information is the most unwanted of all possible outcomes. Continuing search over all descendant nodes makes the search equivalent to a sequential search of all cases. The purpose of the hierarchical organisation is therefore lost. The only solution left consequently is "choose some alternative path".

But the cache-based architecture is free from all these drawbacks. First of all, it is free from a time-consuming linear search of flat memory. The column indices can partition the entire solution space adequately. Finding an alternative solution in the event of one missing item of information is simpler. Given the other information that is known, makes the system search along a particular and relevant column of the cache. The array-like structure of the cache makes this search less time-consuming than it would be on the tree-like structures, as it does not require any back and forth movement along levels of the tree. Also, carrying out interpolation-like activities is simpler, because for the cache structure the solutions that are to be interpolated are overwhelmingly likely to be two consecutive entries of the cache along some column.

However, it should be noted that more advanced techniques involving parallel computations exist for the abovementioned data structures as well (such as parallel search involving flat memory, or use of redundant discriminant network involving several discrimination networks with reordering of keys etc. and a parallel search along all of them). But we exclude them from our discussion (as indicated earlier in chapter 1) and in our present version of the cache

implementation no parallelism has been incorporated even though it would have offered the promise of enhancing efficiency in meeting time-limited demands.

3.6. CONCLUSION

In this chapter we have introduced the cache-based scheme and its essential aspects. We have also compared the cache-based representation of knowledge with other existing schemes and pointed out why we have considered a cache to be more suitable for meeting temporal deadlines than the other schemes. However, a successful implementation of the above scheme also requires specific policies on the part of a system designer on various aspects of the overall architecture, which include:

1. How to choose the set of indices that facilitates purposeful retrieval from the cache.
2. How to conduct interpolation on symbols and how to identify the interpolatable quantities from a problem description and cached answers.
3. How to design such a system for a given domain, i.e. the choice of type of knowledge that one should identify for the domain concerned and the representational formalisms that one should adopt.
4. The control policies which (upon input of a problem specifications) will govern the reasoning process to arrive at a solution within the time limit.

In chapters 4, 5, 6 and 7 we shall discuss these four issues for a cache-based system.

Chapter 4.

BASICS OF THE CACHE SCHEME: CHOOSING INDICES FOR A CACHE

4.1. INTRODUCTION

Performance of the cache-based architecture, in retrieving a quick solution for a current problem, depends on its efficiency in two major aspects:

1. quick identification of an appropriate cache-cell from which relevant stored information is to be retrieved.
2. accessing subsidiary knowledge-bases (rules, cases etc.) in order to generate solutions for current problems when higher-order reasoning tactics are applied.

The cache-based architecture achieves the former with the help of the cache-columns and column-headers. Upon encountering a new problem, the system should analyse it in order to determine along each dimension the column value that represents the problem best. The particular cell of the cache, designated by the selected column-headers, can then be accessed for retrieving a solution. Evidently, the column dimensions and headers along each of them should be carefully chosen so that a given problem situation can be conveyed tacitly to the cache, with its most significant aspects represented adequately. This treatment is important for the cache, particularly for tightly time-limited situations where paucity of time will usually force the system to apply a retrieved default solution (almost) straight into a current problem. On the other hand, when the temporal stipulation is not pressing enough for the cache to use a default solution, the system may resort to using some deeper-level knowledge source (rules or cases, say) for generating solutions for a current problem. Pointers (set to relevant sections of the system's knowledge-base) from the chosen cache-cells are instrumental in retrieving matching cases or rules appropriate for a current situation.

The efficiency of the cache-based system depends significantly on the selection of the aforesaid items (i.e. column headers, pointers etc.). Following standard AI practice, we call them *indices* of the cache.[1]

[1] Ashley [Ashley 89c] defines "index" as: "an index entry points to something; it serves as a guide to facilitate reference".

4.1.1. Considerations For Indexing The Cache

Indexing has three principal uses with respect to the cache-based architecture. Any designer of the cache-based architecture needs to concentrate on these three design problems:

- first is the problem of choosing appropriate features that highlight the major aspects of a situation and hence help in isolating the column dimensions of the cache.

- second is the problem of selecting appropriate headings along each dimension to partition the dimension into meaningful groups (column headers) so that each group has its own characteristics. These column headers are also important in identifying the appropriate cell(s) from the cache (each cache cell is indexed by one column header taken from each column dimension of the cache) and each cell is distinct from every other cell in the sense that no two cache cells should have identical contents in all the levels (determined by different time limits) of the cache as the kind of knowledge representation and processing method varies with the level.

- thirdly, indices should help in retrieving solutions from different levels of the cache. For default-level cells the column headers are sufficient for retrieval. But for deeper-level cells the efficiency of retrieval depends additionally upon the pointers that address auxiliary knowledge-bases such as rules or cases highlighting the typicalities of each individual situation.

In this chapter we discuss the issues related to indexing for building a cache-based system.

4.1.2. Suitability Of Indexing

Indexing as a means of reference has been criticised by some AI researchers [Thagard 89] for several reasons, for example - overemphasising pragmatic features like goals and prediction failures; paying insufficient attention to different semantic features which are considered psychologically important; too much preprocessing etc. While these drawbacks of indexing as a tactic for referring to items of knowledge cannot be denied, we still have relied strongly on indexing for the following reasons:

1. Indexes are easy to comprehend and more appealing to common human understanding compared to other alternative approaches such as complicated networks;

2. Indexing is a well-known technique in AI. Indexes have been used thoroughly in case-based reasoning (CBR) as the means of retrieving cases from case-bases [Kolodner 93 (ch. 5)]. Even for rules, use of index methods expedites access to relevant rules through proper partioning and indexing of an entire rule-base [Rich 91].

3. Application of indexing, unlike other techniques for retrieval such as many-many match algorithms like RETE [Forgy 82] or TREAT [Miranker 87], designed for production systems; or retrieval through constraint satisfaction as in ARCS [Holyoak 89], is not restricted to one particular method of knowledge representation and reasoning ("predicate calculus" in these instances). On the contrary, the usefulness of indices over different paradigms makes indexing suitable for using in the cache environment, which involves different reasoning tactics at different levels.

4.2. INFLUENCE OF TIME-BOUNDEDNESS

We have observed that solving problems in time-bounded situations differs in many ways from other applications, because of some typical influences of time-boundedness at various stages of decision-making. Various aspects of the effect of time-boundedness are illustrated below. A system designer needs to keep these in mind while selecting indices for a cache.

4.2.1. Reprioritising Goals Depending Upon Time

Often the actions that the solver takes to settle a current problem can be resolved into several steps. But significance of these steps may vary with the available time. Consequently, priorities of these steps may become different in different situations. For example, suppose that the current problem in an AGC application is as follows:

> *The operator of an aircraft that is due to leave has reported that its pilot is sick, and unable to continue with his task.*

If the time before the take-off is quite large (one hour, say), the actions that a ground operations controller (GOC) will recommend immediately are:

1. Asking the airline company for a decision about the flight, i.e. whether it wants to maintain the scheduled time with a reserve pilot instead of the designated one; or if it wants to defer the flight. (The subsequent actions will of course depend on the reply, which the company will report to the GOC in adequate time).

2. Sending a message to the medical unit so that a medical team is moved to the aircraft for attending to the ailing pilot.

On the other hand, if the time before take-off is rather small (10 minutes, say), the controller's actions will be completely different: the GOC immediately derives the

implication of the situation, which on this occasion is that the aircraft cannot take off at its scheduled time. In a busy airport, the GOC cannot wait for the airline's decision for the flight concerned. Hence the natural action will be:

1. Defer the flight unilaterally for some considerable time (2 hours, say) and inform the airline concerned;
2. Announce the news of the deferment publicly.

(All other actions e.g. rescheduling the flight, informing the medical unit, shifting of passengers if the flight is sufficiently delayed, etc. can be taken up later in non-time-bounded mode).

This may not actually be the best possible action. For example, it could well happen that the airline can provide a substitute pilot in 45 minutes. But unilaterally, nevertheless, the controller will not defer the flight for a time period as long as 45 minutes at the beginning, as this may mean another rescheduling on the part of the controller as the probability of finding a substitute in 45 minutes is low (knowledge accrued from past experience).

The above example not only suggests that goal priorities vary with the available time limit, it also helps developing indexing tactics. There can be a host of situations when a flight has to be postponed at short notice. But it is not necessary to consider the actual reason behind the postponement, so long as the need is conveyed to the controller. From this observation it can be surmised that the relevant dimension of the cache should be something that conveys the gravity of a situation (irrespective the actual cause) based on which the GOC can decide about the postponement and its extent.

4.2.2. Maintaining Time Stipulation Is Preferred To Quality Of The Answer

Unlike traditional computer science problems, where complexity and correctness of algorithms is the primary concern [Dean 88], time-critical problem-solving puts more emphasis on maintaining the time stipulation. For most of the problem situations, no single correct answer will exist. Instead, there may be a number of different methods for solving a problem, resulting in outcomes of varying quality. Obviously any problem-solving system should strive for the best possible solution for a given situation. But in a time-critical environment often having a better (in qualitative terms) answer beyond the allotted time limit may be tantamount to not having any solution at all. For example, suppose the current situation in an airport is as follows:

> *The controller has received an anonymous telephone call stating that a bomb which is due to explode in fifteen minutes has been planted in an aircraft. The passengers of the plane have already boarded.*

Certainly, the best solution (if it works) for the controller is to contact police to check the likely reality of the possible threat, and to ask the pilot of the aircraft concerned to move to a remote corner of the airport, in order to avoid any damage in the terminal gate or other planes in the vicinity, unload all the passengers and let security personnel search the plane for the bomb and defuse it. Any search plan will obviously lead to least disturbances for other flights and least possible property and life loss. But sensing that the allotted time of fifteen minutes may not be adequate for the entire exercise, the controller may decide to unload passengers from the said aircraft as well as from those nearby as quickly as possible. Evidently this action may not be the best possible solution as it leads to delays for a number of flights which are not directly involved in the incident. But since there is a possibility of huge loss in human lives in going for the best solution, if it exceeds the stipulated time limit, the controller prefers the qualitatively poorer solution.

The cache design normally means that each successively deeper level recommends a better quality of solution. Evidently a qualitatively better solution can be achieved if all the aspects of a given problem are considered explicitly - which the temporal stipulation often precludes. Naturally, the system often has to rely on some generalised aspects of the problem, instead of its specific featural values. However, on which aspects of the problem a generalised feature will be used (and to which depth) and where a problem-specific feature is to be considered depends upon the particular level of reasoning. Therefore it is necessary to build an hierarchy of indices of problem abstractions and other relevant features, and appropriate pointers should be set from cache-cells of different levels, to access the pieces of knowledge that are appropriate to the reasoning tactic associated with a particular level.

4.2.3. Penalty Varies With Nature Of The Problem

The penalty incurred for failure to maintain the imposed temporal stipulation varies with the nature of the problem concerned. As mentioned in chapter 1, the set of time-bounded problems is divided into two categories: *hard* when failing to provide a solution within a desired time limit may prove to be potentially dangerous and *soft* when failure to abide by the temporal stipulation does not imply serious consequences. Depending upon the nature of the penalty, the solver may change priorities of certain tasks. We illustrate this with the following situation:

> *The controller is arranging the arrival of a flight to some gate (A, say), when a message arrives stating some serious problem in the terminal building where gate A is situated.*

Because of the unforeseen problem the GOC needs to find an alternative gate for the incoming plane. This can be expected to result in some delay for the arriving plane, hence some penalty.

Let us consider two possible variations of the same situation:

1. the problem is that a serious fire has broken out near the gate A.
2. there is a substantial water leakage that renders gate A totally unusable.

In the former case, there is an associated risk of a huge loss of property and possibly life. Naturally, it will be sensible on the part of the GOC to attend the task of mitigating the fire instantly and evacuating the passengers from nearby areas, to avoid any loss of life and any serious damage to property. On the other hand, in the latter case the GOC can inform the engineering unit to look for the reason behind the leakage and try to repair it. After that (in the absence of any other urgent event) the GOC should logically consider attending to the gate allocation for the incoming plane. Hence, if the delay in both cases is equal (10 minutes, say), then the penalty for the latter case should be much lower than the former one. A system designer should set appropriate indices to different problem situations to designate their importance.

All the abovementioned points should be borne in mind while developing an indexing scheme for the cache-based systems in a domain.

4.3. KEY ISSUES IN INDEXING

Despite the profuse application of indexing in different AI applications, no theory has been developed as to how to choose the best indices for a given purpose. In our work we have derived much of our insight from Case-Based Reasoning (CBR), where in different applications, due to the nature of their approach, indexing has been used thoroughly.

A CBR system functions on the principle that a case is to be chosen from the case-base in such a way that it becomes useful in solving a current problem. Naturally, the more similar (in some pre-conceived sense) is the chosen case to the current situation, the better is the solution that can be generated from it. The use of similarity that is intended here is to measure the closeness between a current situation and a past case that depicts a past problem situation and how it has been handled. Normally, a case consists of all relevant information including the background, description of the encountered problem, possible alternatives that have been tried and their outcome, the final solution and the result. A case, therefore, is naturally associated with numerous features. Not all of them, however, may be of equal importance from the representation, retrieval and reasoning points of view. Moreover, only some of these features are explicit from the case description. while some others can be inferred but many of the case features may remain unnoticed [Schank 86] by the user.

Any comparison between two situations therefore involves viewing each of them as a combination of several features, and the task of similarity measurement can be accomplished by comparing the respective features of the two situations concerned. A case-based system, in order to find a similar case to solve a current problem, needs to compare the current situation with past ones stored in the case-base. A naive way of wading through the entire case-base and comparing all the features of each case with the current situation is clearly inefficient and time-consuming. On the other hand, if all the past cases are analysed and relative merits and demerits of all their features are evaluated, the task can be simplified by identifying a set of important features which can be used as cues to probe into the case-base in isolating the cases which are most relevant for the situation. Identification of the most important subset of features for a problem domain constitutes the **indexing** problem of CBR.

In different CBR systems different techniques have been adopted for efficient indexing of cases. An in-depth study of different CBR systems suggest that the most suitable indexing scheme in a particular domain depends upon the special features of the domain and also on the nature of application. These studies help us in identifying the key issues that govern the task of indexing in an application domain.

We observe that a successful indexing scheme requires consideration of three major aspects:

- systematic analysis of the domain and its characteristics;
- identification of good features/concepts that are important in solving problems in the domain concerned and help in measuring similarities between relevant concepts;
- choice of a representative vocabulary.

4.3.1. Indexing And Domain Model

From a simplistic yet intuitive point of view, one may consider the task of *indexing* as that of assessing relative weights of different domain-related features and measuring their relevance in solving problems in the underlying domain. However, it has been observed that for most real-life domains the domain theory is not strong enough to accomplish all the tasks required for designing indices [Porter 89]. Naturally, a gap exists between the superficial features visible and the features that are to be generated. In different applications different methods for a systematic analysis of the domain have been tried. We discuss some of these methods below.

4.3.1.1. Analytical Method

In many AI applications, people have used an analytical or deep structural model of the domain from the first principles, in order to understand the causal effects of different features in a problem. Different application systems such as MYCIN [Buchanan 84], fault diagnosis

[Kobayashi 91], CASEY, a medical diagnosis system [Koton 88a, Koton 88b], litigation [Kowalski 91] etc. fall under this category.

The advantage of using an explicit analytical or structural model is that the significance of each feature concerned and its role in the overall functioning of the system is well-known to the system designer. Here, a causal chain is often used to establish various cause-effect relationships to accomplish the desired task. This helps in identifying the right set of features for possible indexing, for classifying rules, cases etc. depending upon the nature of application.

But building an exhaustive causal network requires a complete domain theory, which is often infeasible to have in most applications. For example, in a domain like AGC it is difficult to formulate any specific theory behind solving the various problems that may come up in the daily functioning of an airport. Past cases, in these situations, may provide the best help towards solving a current problem. Consequently, case-based reasoners have to analyse past cases to derive the required indices. We have identified three major ways of analysing cases to ascertain relative importance of different features for problem solving activities in a given domain: *statistical, explanation-based,* and *heuristics*.

4.3.1.2. Statistical Method

Statistical analysis of past cases can be tried in order to identify the importance of features and to assign numeric weights to them to discriminate among them. One can test different hypotheses about the importance of different features to establish their relative importance.

Another way of using statistics is to check the frequencies of certain happenings which may, in turn, influence an user's action. For example, with respect to the AGC domain, the knowledge suggests that if the fire alarm rings a proper response includes evacuation of the place and calling of a fire brigade. However, statistical analysis may indicate that in many cases the alarm is false in the sense that it is not due to fire but because of dust. This statistical analysis may change the course of action for the GOC. In the case of a fire alarm, therefore, the first action may change to ascertaining in the first place if there is a genuine fire and only then to calling for fire extinguisher/ fire brigade etc. depending upon the degree of intensity. Thus statistical analysis of past cases may help in designing solutions - and consequently, making decisions about the best set of indices as well. We have not used a statistical approach, because of the absence of stocks of data in the right form in either of the two application areas that we have considered.

4.3.1.3. Explanation-based Method

Statistical methods are not always effective because of their equal treatment of all aspects of a problem. One possible improvement over them in assigning weights to different features is to cite the past cases as precedents and to figure out from them the expected behaviour of different entities in different situations. The key concept here is *explanations*. Explanations (by domain experts) regarding failures, possible remedies, expected side-effects etc. serve as sources of useful indices for describing a situation and suggesting suitable actions.

In CBR systems explanations have been used to facilitate efficient similarity measurement. Protos [Porter 89] uses explanations provided by experts about featural equivalence to determine the relevance of various features. Overall similarity is assessed by heuristic evaluation of explanations and importance of unmatched features. GREBE [Branting 89], the law-based system for determination of workmen's compensation, also uses explanations for determining the key legal relationships and thereby assessing similarity between the new case and the precedent cases. In other domains such as fault recovery systems [Barletta 88a, Barletta 88b], explanations have been used extensively in indexing. AQUA [Ram 93] uses explanations in understanding different political phenomena and in subsequent refinement of different concepts.

4.3.1.4. Heuristics

Heuristics of many forms have been used in different systems for selecting features for indexing. Most of these heuristics aim at identifying features that are useful from certain specific points of view.

Owens [Owens 93], in order to use prior cases to explain plan failures in critical planning situations, identified four types of functional relevance to label different chunks of knowledge, called *knowledge structures*:[2]

- Failure-related - these labels capture some failure-related aspect of the central causality represented in the knowledge structure and thereby cluster together knowledge structures in which a common element plays a role in the causal description of the failure.
- Symptomatic - these labels relate an observable condition to the situation characterised by the knowledge structure, even though at the level of representation of knowledge structure no explicit causal connection is made between the observable condition and the failure.

[2] The chunks of knowledge hold a content theory of failure, recovery and repair, and a representational theory of how similarities between instances of recurring failures can be detected.

- Recovery-related - which characterise an aspect of the recovery strategy within the knowledge structure.
- Direct - which exhibit direct functional relevance to the central causality in the failure.

But such a generalised classification relies largely on having an efficient abstraction-generation mechanism available to work on the case features. And in a time-critical situation, interpreting all these abstractions to judge the relevance of a past case in solving a current problem may prove to be too time-consuming. We feel that a deeper analysis of cases and consequently a finer classification of features will be more suitable to meet the demands of time-critical problem solving.

In similar vein we look at the six types of preference heuristics with respect to PARADYME [Kolodner 89a], a case-selecting program designed to assist JULIA, a case-based meal-planner. Although these heuristics are for choosing cases from a case-base, they provide insight regarding properties that one should look for; and these can be extended to selecting indices as well. The 6 heuristics presented in this work are:

- Goal-directed preference - This is based on the principle of utility, i.e. it leans towards those cases that help in achieving certain goals of the problem solver. Naturally, it prefers those cases that share more constraints over those that share fewer.

- Salient Feature preference - This is based on the principle of focusing on features that are more important in describing the problems. Here, it means concentration on cases that share larger subsets of features.

- Specificity preference - There are some features that are more typical in specifying a situation. This heuristic suggests that it is better to choose cases that are more specific than others.

- Ease-of adaptation preference - While adapting a past solution in a current problem a solver needs to fix on certain features. This heuristic suggests that it is better to choose cases that match on easy-to-fix features.

Two other heuristics described with respect to choice of cases are:

- Frequency preference, which prefers cases that are used more frequently.
- Recency preference, which prefers cases that are used more recently.

(It may be worth mentioning that the last two heuristics are similar to some statistical studies).

Depending upon the purpose, one or more heuristics may be used to select indices.

4.3.2. Indexing And Similarity Measurement

Similarity measurement, or establishing how close the matching between two (or more) entities is, is an important aspect of modern AI. Researchers in analogical reasoning and case-based reasoning have paid considerable attention in this regard. Evidently, the issue of similarity is crucial for the cache-based system as well, because a cached answer can be considered for application only when the cache cell that holds the answer concerned is similar to the underlying problem. Similarly, when two actions are interpolated to determine a new action for a current problem, one looks for some type of similarity among the actions to be considered.

However, there is a subtle difference between similarity as viewed by analogical or case-based reasoners and when it is studied from the perspective of implementing a cache-based architecture. Traditional studies of "similarity measurement" concentrate on establishing different tactics for measuring similarities between entities; and their significance under various conditions [Kolodner 89a], [Campbell 90], [Leake 91], [Wolstencroft 93]. But with respect to the cache-based architecture their roles are not disjoint. On one hand, efficient retrieval of a solution from the cache can be accomplished by matching a current problem situation with the column headers of the cache. On the other hand, how well the similarity between two situations can be measured depends on the efficiency of the underlying indexing scheme. Thus with respect to the cache-based architecture these two issues are very closely interlinked.

However, studies of how similarity has been handled by different researchers helped us in gaining insight into the issue of indexing, which in turn has helped us in identifying features that can serve as useful indices for a cache.

One of the most elaborate works on similarity has been recorded in [Wolstencroft 93]. Here, from the point of view of analogy, similarity has been classified into 4 broad categories:

Structural, which relates to the syntactic structure of situation descriptions. It can be measured by satisfying constraints related to the syntactic nature of the candidates. For example: object is mapped with object, n-place relations with n-place relations, relationships between parts etc.

Semantic, which relates the conventional ideas on literal similarity between the objects. There are several distinct ways of measuring semantic similarity, depending upon how the mapping is done:

1. Thesaurus or dictionary meaning: two terms are called semantically similar if they correspond to similar terms as far as a dictionary or thesaurus is concerned. For example, *rain* and *shower* are similar, to an airport controller, when they appear to pose a problem for some departing flight.

2. Taxonomic meaning: when objects are arranged in an inheritance-tree like structure, descendants of the same node may be considered similar. For example, a *truck* and a *jeep* receive similar treatment from the GOC when the problem is that a vehicle is standing on a taxiway and obstructing the movement of a plane.

Thus, semantic similarity can be measured by surface features, i.e. features that are outwardly visible from the problem description. If there is a direct match in these terms, then establishing similarity is easier. However, in a problem domain involving a huge number of features direct matching may not always be possible and hierarchical trees of abstractions may need to be maintained to establish similarities.

Pragmatic: two parts are said to be pragmatically similar if they serve the same purpose in the chain of reasoning, or in other words pragmatically similar parts play similar roles in their respective contexts. Here, the main distinction is that in order to determine the pragmatic relevance of a given part the reasoner may have to go beyond the straightforward significance that is provided by the structure or semantics. For example, a *drunken pilot* is similar to a *sick pilot* as far as ability in flying a plane is concerned, although they are different in their symptoms. In similar vein, an *out-of-order jeep* on the taxiway is comparable to a *huge box* on the taxiway, as both are stationary objects posing obstructions to plane movements, although they are not apparently similar.

Organisational, which deals with organising the knowledge-base so that objects deemed to be similar by a domain expert are placed close to each other in any information storage scheme.

Our conclusion from the study of these different types of similarity is that "organisational similarity" is orthogonal to the others in nature. It is mostly relevant for organising knowledge (such as rules, cases), and not considered useful for designing indices. But the other three types contribute directly towards measuring the similarity between two situations, though not all of them are equally suitable for designing indices.

Structural similarity, although widely used in analogical reasoning, is, we feel, of limited suitability for real-life problems - because in order to solve a problem a system needs to

examine the significance of various features and also needs to infer from the problem specification (with the help of domain knowledge, of course) the implications of various features. Arriving at these decisions through a structural approach is time-consuming and may even be impossible. For example, consider two actions *sending a person with a message* and *telephone.* They have two different structures: the former having three parameters viz. *who is to be sent, to whom he is sent* and *with what message,* while the latter has two parameters: *the person to be telephoned* and *the message.* Thus structurally they are not similar, although they serve the same purpose. Moreover, default solutions, used in extreme time-critical situations, do not necessarily have a special structure. They normally involve some sequence of actions which can greatly mitigate an impending problem. Naturally, use of structural similarity in this context is of limited help. This observation precludes the use of structural similarity in determining indices for a cache-based system.

While evaluating the significance of pragmatic similarity vis-a-vis semantic similarity, we consider the following examples from the AGC context. Suppose the current problem is as follows:

> *The operators of a plane that is due to leave in 30 minutes' time have reported that its pilot is ill, and unable to continue with his task.*

Using "ill" as the cue for searching, the (semantically) most similar case that can be retrieved in a library of typical airport cases is:

Case A: *The operators of a plane that is due to leave in 45 minutes time have reported that one of the passengers is seriously ill.*

While using 'pilot' as the cue for searching the nearest case that can be retrieved, we may find the following case:

Case B: *A responsible member of the airport's staff has reported that the senior pilot of a plane due to leave in one hour seems drunk.* [3]

Evidently, neither of these two cases effectively resolves the present situation - as the main problem in the current situation is that the plane that is due for departing shortly cannot fly (because of the pilot's sickness) and will occupy the terminal gate for some extra time. Hence the GOC now has to replan for its departure (negotiating with the airline regarding how soon it can find a replacement for the pilot) and he may also have to reschedule other arrivals/ departures due to the interruption.

The most similar case from the point of view of the GOC for the cue case may therefore begin:

---

[3] As per the apparent traditions of commercial aviation, it is not good form to contradict a senior pilot - if he thinks that he is not drunk, it is difficult to persuade him otherwise or stop him from flying.

Case C: *A plane that is due to leave soon has developed some problem in its engine and requires a small repair.*

Even though it has no resemblance (at least apparently) to the current problem, by deep analysis of the situation a controller will find Case C to be most similar to it as it solves the same problem as posed in the cue case.

From this observation, we surmise that pragmatic similarities, which look at a current situation more deeply by referring to the common abstract properties of various parts, need to be considered primarily for selecting indices. Pragmatic aspects can be measured with the help of abstractions derived from the input features. Both the problem description and the background state of the domain (the airport, in our example) are required for generating these abstractions - which provides a general feel of the situation to the solver and enables quick apprehension of the nature of a current problem. Our selection of indices *potential-degree-of-threat* and *activity-level-of-the-airport,* (see section 3.2.3) is based on such pragmatic aspects of a situation. These abstractions help in identifying similarity between two situations on the basis of the actions that need to be carried out when a problem emerges.

However, in certain situations extracting such pragmatic abstractions is not straightforward. Even when they can be computed, they may not prove to be effective in measuring similarities. Comparing entities with their semantic meanings may be useful there.

For example, with respect to the AGC domain, consider two situations where the first one is that *a fire has broken out somewhere in the terminal building* and the second one says that *a powerful bomb is suspected to have been planted in the terminal building.* For both the situations the basic remedial actions will be the same, viz. evacuation of passengers, suspension of immediate flights from the gates in that building, calling appropriate agents to tackle the hazard etc. The only major difference is that while a fire brigade should be called for putting out the fire, security personnel are to be contacted for the bomb problem. Thus the same cache-cell can be used for storing solutions corresponding to the above two situations, provided they are indexed appropriately. Certainly, using raw features such as *fire* or *bomb* as column headers (because of their obvious salience in the problem specification), the desired outcome cannot be achieved efficiently enough for a time-bounded situation. Another consideration is that a real-life domain is normally associated with a large number of features. Indexing with the help of these raw features will not only lead to an explosion of vocabularies and a huge cache, but (consequently) the efficiency of the system will also be reduced. Instead, appropriate abstractions (such as potential-degree-of-threat) should be used.

4.3.3. Choice Of Indices

For the cache-based architecture, we divide the required set of indices into two different classes, depending upon their role in the overall functioning of the system:

- **primary features,** which are those that act as column headers for the cache and are essentially used in accessing the cache. Naturally, these should be those features that are capable of describing the major aspects of a current problem.

- **secondary features,** which describe the more specific aspects of a given problem. These features are used at deeper levels of the cache to refer to appropriate rules, cases etc. to facilitate better reasoning towards solving a current problem. However, it is to be noted that the selection and use of secondary features vary with the levels of the cache (as the depth of reasoning and consequently the mode of reasoning may change).

Analysis of the underlying domain must be undertaken to identify the best set of vocabulary terms for building a cache. In the absence of a clear-cut domain model, as indicated earlier, a careful analysis of a number of past cases may help a system designer to identify a set of abstractions that help in characterising the problems; and also to pin-point the key features that are crucial for the domain.

Evidently, no general theory can be produced in this regard. However, we have identified several properties that are desirable for the set of indices. They should be borne in mind while designing a cache.

4.3.3.1. Properties of Good Indices

We observe that the following aspects need to be considered while choosing indices:

- usefulness of indices in the relevant context;
- minimisation of complication (both timewise and conceptwise) in computations;
- easily understandable significance;
- provision of efficient cues for quick searching.

Usefulness of Indices

Indices should be useful in their respective context. For example, consider a problem: *there is a fire at a terminal gate in the airport.*

There are many ways to index this problem. However, the system designer should look for those indices that are useful in achieving the system's goal. For the aforesaid problem, indices such as *degree-of-threat, busy-state-of-the-location* have immediate meaning to the user for designing the immediate solution steps. These can be used as column dimensions of the cache as they represent situations better than indices such as *number-of-the-gate.* Although the latter may be useful for the AGC system at a later stage of reasoning to determine, say, if there is enough space near the affected location for possible manoeuvring of fire brigade appliances. Consequently, special features such as *number-of-the-gate* can be used as one of the pointer indices from deeper level cells of the cache, along with some related property (e.g. locational advantage/disadvantage) for selecting cases/rules from the system's repertoire. Thus we feel that "gate number" can be a suitable secondary feature, but it is not ideal for use as a primary feature to designate a column dimension.

Computationally Less Complicated

For retrieval purposes, the system needs to convert the raw features used in describing the problems into abstractions which can be used as cues for retrieval from the cache. However, if the transformation is computationally complicated, in extreme time-bounded situations it may exceed the imposed time bounds. Hence it is mandatory that the indices should be so chosen that they can be computed easily from the raw features. We found that use of tables is conducive to these computations. For example, in the AGC domain, to deal with the abstraction *potential-degree-of-threat* (PDT), we maintain a table containing experts'/our intuitive view of the amount of damage that different potentially dangerous objects may cause. The damage can be of different types: human lives, destruction of objects, functioning of the airport. We use artificial numeric values in the range [0.0, 1.0] to represent this knowledge. Figure 4.1 contains a portion of this table.

Object	PDT
bomb	0.9
fire	0.7
water	0.1
gas	0.6
dog	0.2
unattended-bag	0.7
snow	0.3

Figure 4.1: PDT-values for Different Objects in the AGC Domain

Easily Understandable Significance

Indices should be so chosen that their purpose in the problem-solving is easily understood. Since situations are matched by matching these indices, it needs to be ensured that two situations matching in these indices should have a common type of solution. For example, in order to take care of the issue of location, i.e. the region of the airport where some trouble has occurred, for the AGC cache, we had initially thought of using abstractions such as "importance-of-the-location", and wanted to use some numerical or symbolic codes (similar to those in figure 4.1) to describe the importance of various airport locations. However, we discovered later that such a scheme would not work nicely, because the significance of each location is not properly captured by such a scalar representation. For illustration, any such scheme should intuitively attach a high degree of importance to both the runway(s) and the terminal building(s), yet a controller's actions in dealing with problems in these two locations are not quite similar - occurrence of any potential threat in the terminal building almost inevitably demands a quick action for evacuation of the passengers involved; but such a step is absent in dealing with an incident affecting a runway. Hence we decided to use the names of specific locations (such as *runway, terminal building*) in our system.

Indices as Efficient Cues

The secondary indices are meant for providing cues in searching suitable auxiliary knowledge such as rules, cases etc. Naturally, the indices should be such that they offer a logical partition of such knowledge into fairly homogeneous classes. Searching with the help of these indices as cues then helps the system to isolate quickly the most useful set of information for abating a current problem. For illustration, one secondary index for the AGC domain is the controller's intuitive idea about the efficiency of management of different airlines. A GOC's action is often dependent on this view. If a particular airline requests the GOC on a particular day to postpone a departure time by 15 minutes (due to inability to load luggage into the aircraft in time), the controller's reaction is likely to be different if the airline concerned is deemed to be responsible from what it will be if the airline is not regarded as very efficient. In the former situation, the controller may defer the flight by 15 minutes as requested; but in the latter situation any decision regarding its departure time may be postponed until the airline informs the GOC that the flight is ready to take off. These decisions are normally made by using a reasoning scheme that is associated with a deeper level of the cache, and a secondary key such as *efficiency-of-the-airline* may then be useful to pick up the right case quickly.

All these above issues need to be considered for choosing indices in designing a cache-based system. Obviously the ultimate indices in any domain are coined from terms associated with the domain concerned. Therefore, a systematic analysis of the domain is in order. The task becomes easier when a structural model of the domain is available. But for the AGC (and probably in very many real-life applications), designing a complete model is almost impossible. Hence, as we indicated earlier, we have used past cases narrated by experts to facilitate the analysis of the domain. But we feel that it is difficult to identify the roles of different

features in a situation and determine their weights right at the outset. We therefore suggest a somewhat more indirect approach. Our suggestion comprises the following steps:

- Design a classification scheme to collect the set of features into some meaningful classes depending upon their roles in the overall problem.

- Once features of all the available cases have been analysed, use explanations and/or statistical methods to identify the right abstractions and also their relative weights.

- Choose on the basis of the roles and weights of different features the ones that are useful from the point of view of design of the current cache.

We have developed a feature-classification scheme that helps in understanding the significance of different features in a case. In the following section (section 4.4) we describe our scheme and how it helps in choosing indices for a cache.

4.4. ANALYSIS OF CASES: OUR APPROACH

Depending upon the role that a feature may have in a case, we classify the features involved in a case into 8 broad categories:

- **disruptive:** emerged problems within a domain that disrupt the normal flow of activities are termed disruptive features. In a problem-solving environment, this is the most fundamental feature of a case.

- **descriptive:** features that describe the the background state when the problem appears. Any intermediate state (during the process of solving) and the final state that is achieved following some remedial action (on the part of the problem solver) are members of this category.

- **imperative:** which describe the actions taken in a given situation to combat a current problem. However, in most of the occasions the actions will be decomposed into a set of subactions.

- **predictive:** given the background and the nature of the problem that emerges from it, these features can be used to infer the related consequences that the system will have to face.

- **suggestive:** these features suggest the ultimate goals that the system may try to achieve given the current situation and the time in hand.

- **explanatory:** explaining the reasons for failures (if any) in any solution that has been applied with the aim of resolving a problem.

- **obstructive:** these can be of two types. Often a solver resorts to certain actions overlooking the presence of features that do not immediately support or justify the relevant actions. Also, certain actions may produce some unintended results which are not helpful for the solver. These too are named obstructive.

- **corroborative:** these are opposite to the obstructive features. They corroborate an action of the solver by pointing to some further features which justify the action (and also point to any side-results of an action which are helpful from the solver's point of view).

We now illustrate the 8 types of features with an example from our case-base. Suppose the complete case is the following:

> A plane arriving in 15 minutes is to report at gate 20, when the controller receives a message that the flight that is due to leave gate 20 within 5 minutes has got a problem with a baggage door and therefore cannot leave until the door is repaired. The GOC immediately informs the engineering unit, which replies that its staff members are busy at this moment and cannot turn up quickly. However, they will probably turn up after 15 minutes.
>
> The controller knows from previous experience that repairing this problem should not take more than 5 minutes once the engineering people take up the task. But often they are held up with other tasks and therefore there is a chance that they will arrive at a time considerably later than the predicted 15 minutes. Also, the repairing time is not fully guaranteed to be less than 5 minutes. The controller therefore tries to look for alternative gates and finds that the state of the airport is busy, so that no other gate can be allocated immediately to the incoming plane.
>
> The GOC then looks for a possible alternative gate that will be free shortly. The schedule shows that there is an XYZ-airways flight due to leave gate 45, in 15 minutes, while IJK-airlines has a flight from gate 28 in 5 minutes. Previous experience suggests that XYZ is currently quite punctual in its departure times, with an average lateness of 5 minutes, while IJK is not reliable in this respect and is likely to come up with a last-minute request for some extra time.
>
> The GOC decides to allocate gate 45 to the incoming plane and makes a path-plan to take the incoming plane to an open space, and make it wait there till the XYZ flight has gone. Although this plan causes some delay on the part of the arriving plane, the controller prefers this plan as sticking to gate 20 involves uncertainty about the repairing of the

faulty plane while, also, gate 28 is not guaranteed to be freed. He makes the necessary announcements regarding change in terminal gate and arrival time of the incoming plane and passes the path-plan to the arriving plane's pilot.

The *disruptive* feature for this case is that there is a fault with a baggage door of a departing plane. *Descriptive* features for this case are: busy state of the airport, an incoming plane to gate 20 etc. The first *imperative* measure that the controller takes is to contact engineers to attend the faulty aircraft immediately. The fact that baggage-door repairing does not normally take more than 5 minutes corroborates his action, and consequently, is considered to be a potential *corroborative* feature. However, this local plan fails and the relevant *explanatory* feature is that engineers are currently busy. But it also generates another corroborative feature that engineers will be available after 15 minutes. The possible *predictive* feature for this action is that the GOC predicted from the problem that the incoming plane would be likely to encounter an occupied gate and therefore the hindrance should be removed. This in turn suggests that the fault should be repaired - which is the *suggestive* feature. The controller finally resorts to subsequent actions - each having its related predictive, suggestive features.[4]

The pertinent question now arises as to whether such meticulous analysis (which evidently requires a lot of effort for the knowledge engineers) is required at all. Some AI works such as classification of case features in traditional CBR systems, have followed a simpler approach. For example,

- MEDIATOR [Kolodner 89b], a system for mediating resource disputes, represents the cases with the help of three types of features: problem description, solution and outcome;

- CASEY, the heart-ailment diagnosis system [Koton 88a], on the other hand, relies on the situation description and the solution process only;

- JULIA [Hinrichs 91], the menu planner, concentrates mostly on the solution process itself, giving details of justifications and the rules and heuristics used in each step.

- KRITIK [Goel 89], a design system for electrical circuits, also pays attention to the solution process but (unlike JULIA) constructs a causal representation of the entire problem instead of stepwise justification.

- In ED [Smith 91], an example designer in lesson planning, case features have been divided into two classes: *salient features,* those which are used for selecting a case, and *descriptive features,* those which indicate the solution obtained in a case.

[4] However, it is not necessary that each case in a case-base should have all the different types of features associated with it.

In this regard, we point out first that often apparently insignificant features may become crucial in determining problem-solving tactics. For illustration, with respect to the above example from the AGC domain, the controller has decided to repair the baggage door in situ because it is known that repairing a door should not take much time (a *predictive* feature). Given everything else to be the same, had the fault been with the engine (which normally takes a significant amount of time to repair) the controller would have thought of removing the aircraft to a free location before the engineers were called. Consequently, the incoming plane would come straight to its assigned gate and the GOC would have been spared the extra planning tasks. Similarly, at a later stage of the reasoning the controller would probably prefer to allocate gate 28 to the incoming plane (from where flight IJK was due to leave in 5 minutes) if it had been assured that IJK was usually quite punctual. But its current unsatisfactory record with punctuality (which is an *obstructive* feature for the case) prompts him to choose gate 45, although it is not scheduled to be vacated for another 10 minutes.

Our argument, in this regard, is therefore that in other domains too these subtle features ought to have crucial roles in decision-making. Even though they are not explicitly presented as pertinent case features, they are unavoidable in the description of the solution steps and/or the outcome of a case as essential parts of the reasoning of a solver. In the absence of any pressure on time, identification of such finer aspects of cases as parts of the more general components (i.e. problem description, solution steps, outcome) works well. But when time is pressing, explicit identification of different features paves the way for quick decision-making.

With respect to the cache-based system the utility of this classification is twofold:

a) While dealing with cases (i.e. at a deeper level of the cache), explicit representation of features of different classes (e.g. predictive, suggestive, explanative) helps quick identification of the interpolating variables and thereby facilitates interpolation between cases. (These issues will be discussed in section 5.5.3.2).

and

b) Subtle features often have essential roles in generating abstractions, which is extremely important for determining cache indices.

In the following section (section 4.5) we describe how cache indices have been chosen for the AGC domain with the help of analysed cases.

4.5. CHOOSING INDEXING VOCABULARY

Features for indexing for a cache are of three types:

1. Column dimensions - indicating the key features that can designate a situation;
2. Column headers - along each dimension these headers are used for identifying the relevant feature value for the current problem and thus facilitating retrieval.
3. Secondary indices - which at a deeper level of reasoning are used for selecting appropriate rules or cases depending upon the level of activity as dictated by the temporal bounds.

The concern therefore is to discover which features will be the best for this objective.

Evidently, out of the 8 classes of features (discussed above) only the *descriptive, disruptive* and *imperative* ones qualify for the first two of the purposes above, as they stand for the fundamental aspects of a case. However, we observe that two situations having common descriptive features (i.e. background information) may pertain to two entirely different situations having entirely different solutions. In similar vein, two situations referring to similar types of problems (disruptive features) may end up with two different problem-solving approaches, due to the non-uniformity of their background states. On the other hand, imperative features (i.e. the solution features) have the implicit advantage that when similar actions are taken against two problems, it is due to some common effects that the two problems exercise on the functioning of the domain. This common effect is identifiable from *predictive, suggestive* and *explanatory* features of a case. Generalisation and/or abstraction features then can be generated for selecting column dimensions, and identifying appropriate column headers.

For example, consider the following problems from the AGC domain:

1. *A plane that is due to leave soon has received a threat that a bomb has been planted in it;*
2. *An escaped rabid dog is roaming freely in the passenger lounge;*
3. *Fire has broken out in the engineering unit of the airport;*
4. *An unattended box has been seen near the taxiway;*
5. *A broken electric wire is hanging in the viewing gallery;*

These problems are apparently different. But against a common backdrop of "very busy" activity level of the airport, to the ground operations controller these problems are connected through a common feature that some hindrance has occurred somewhere in the airport. Hence a GOC's actions will basically remain the same as far as these situations are concerned:

- Check the spread of the cause by calling an appropriate authority to attend the problem;
- evacuate the affected area;
- relocate any flight (arriving or departing soon) affected by the emergence of the problem.

The significant different aspects of these problems are that the authorities to be called to take care of the situation are different (e.g. fire brigade for tackling a fire; security personnel in the event of bomb or suspected luggage); or the objects to be evacuated (e.g. humans if the affected location is an aircraft or passenger lounge; aircraft if the location is runaway or taxi-way); or the extent of evacuation (e.g. if it is a bomb the distance should be 100 metres or more; but if it is live wire a circle of 5-metre radius is acceptable). Naturally, features to represent column dimensions ought to be sufficiently generalised to capture the commonality of the apparently dissimilar problems, yet expressing their subtleties.

Thus, all the above problems can be dealt with in a single cache provided the right column dimensions are chosen. In this respect we observe that the extent of evacuation depends primarily on how threatening the object is for the surroundings. Depending upon this degree of possible threat we have classified various domain-related objects into different families. In figure 4.2 we show a hierarchical organisation of the relevant objects. Within each family class further divisions can be made, depending upon difference in the treatment of the objects. One such division is the "movability" of the object, as illustrated in figure 4.2.

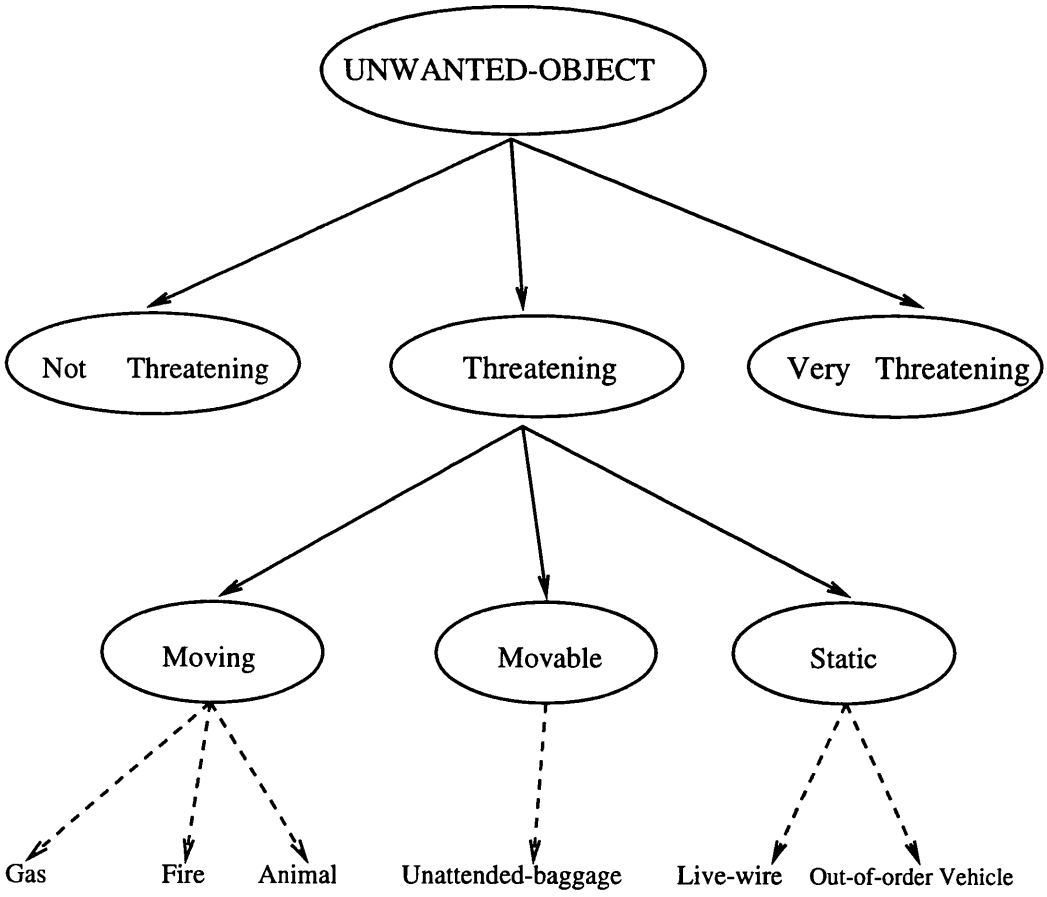


Figure 4.2: Hierarchical Organisation of Threatening Objects

Thus "potential-degree-of-threat" has been chosen as a pertinent column dimension for the said cache. In similar vein, noticing that the objects to be evacuated (and therefore the agents to be used to accomplish the evacuation) depend primarily upon the location where the problem has occurred, we have considered "location-of-the-event" to be another useful dimension. The "activity-level-of-the-airport" emerges as a further valid dimension from the considerations of what a controller will do (regarding gate allocation and change in schedule) in order to relocate the affected flights.

The next task, evidently, is to identify for each dimension the most appropriate columns. The hierarchy tree as in figure 4.2 provides a cue here. For example, with respect to the feature "potential-degree-of-threat" it suggests that the designer should look for some systematic representation of the gradation of potential threat caused by different objects. In practice there are many possible alternatives to choose from:

- Symbolic values - such as innocuous, troublesome, dangerous;

- Symbols with qualifiers - such as not threatening, threatening, very threatening;

- Numerical equivalents - where a systematic scale is used to denote the threatening power of different objects. For example, 0.2 for innocuous, 0.5 for troublesome, 0.9 for dangerous;

- Ranges - where a continuum of values specified by upper and lower limits of values (numerical or symbolic) is used to designate each header such that a particular cache column will be chosen when the related problem value falls within the upper and lower limits specified.

Each of these types has some merits/demerits in their use as column headers with respect to the cache-based architecture. A system designer should bear in mind that the headers should be so chosen that they help in retrieval and also in conducting interpolations. In chapter 6 we discuss the implications of different types of headers in a cache.

In the situations of extreme pressure of time, one cannot expect a better-quality solution than those obtained from the default level using these general abstractions as cues for retrieval. But when activities involve a more detailed reasoning with deeper-level cells of the cache, one may prefer a better-quality solution. For example, an optimum solution involving a broken live wire ought to differ from when the hazard comes in the form of a moving dog (even when everything else is the same); even if the degree of potential threat of the two objects may be regarded as comparable. One reason for this difference is, as indicated in figure 4.2, due to the mobility of the object. When the object is static, one may consider cordoning off the area thus indulging in minimum possible evacuation of the affected location.

Exploitation of similar past case(s); or use of rules such as,

IF the object is Threatening but Static,
THEN consider cordoning off the location and postponing evacuation.

can be used to improve upon the default solution. Thus movability of the object (which in related cases will occur as explanatory features) can be regarded as a suitable secondary feature. Relevant rule-bases and case-bases should be suitably partitioned using such features, and typical values of these features should be used as pointers from pertinent cache-cells to minimise the searching time on these cases/rules.

Evidently, the actual choice of indices is extremely domain-dependent; it is not possible to provide detail of tactics for all possible indices. However, the information given in this chapter can be considered as a case study which should provide a basic insight into the question and how to go about answering it. Since indexing is one of the major issues of building a cache-based system, building such a system for an arbitrary domain needs detailed inspection of the *viability* of finding suitable abstractions and their typical values that may play the roles of column dimensions and the column headers for the relevant cache. Building a cache-based system may not be feasible, otherwise. In fact, one can use this as a test of appropriateness of the cache approach in dealing with a new application: if it seems unusually difficult or impossible to find suitable indices and/or abstractions to serve as indices, this should be taken as evidence that our approach cannot be used for that application.

4.6. CONCLUDING REMARKS

One important remark regarding index selection for a cache is that in a domain there may be problems of varying types which do not fit into the same model solutions nicely. For example, in the AGC domain, two principal types of problems exist which are different from a solver's point of view. There are situations when a some hazardous object appears, causing an all-round panic. An essential step in any GOC's solution in these situation is considering the evacuation of the affected location. The other type of problem comprises situations when a particular flight cannot take off according to its schedule (as in the situation described in section 4.4). In an airport this often happens due to various reasons, such as mechanical problem in the aircraft; or some management problem (e.g. unfinished loading). In order to handle these problems, a controller does not need to evacuate any place with the apprehension of any danger. Hence such a step should be missing from a model solution. On the other hand, tackling these problems demands an extra step of mitigating the trouble (e.g. repairing/replacing the defective part), and consequent extra time allocation to finish the task.

Evidently, utilising the same cache for these two types of problems cannot be construed as ideal. Our suggestion in this regard is to develop two caches, each devoted to a particular type of problem. The caches ought to have different indices. But the overall method of choosing indices (and other design schemes) should remain the same. Whether or not to add such a complication will depend on analysis of the domain knowledge in any new area of application.

In building our cache for the shortwave radio domain, we have followed the same line as presented in this chapter for building the cache. Moreover, it has only been necessary to use a single cache. The general domain theory and our decisions regarding the abstractions (i.e. column dimensions and headers) will be discussed in chapters 8 and 9.

Chapter 5.

BASICS OF THE CACHE SCHEME: INTERPOLATION OF SYMBOLIC KNOWLEDGE

5.1. INTRODUCTION

Interpolation, as mentioned earlier, is a much-practised tactic for quick solution of numerical problems where one is trying to solve some problem of the form $Y = f(X)$ and where Y-values for known X-values are used for fast estimation of the unknown Y-value associated with a given X. Traditional paradigms in knowledge-based computations generally do not recommend interpolation-like tactics for solving problems. This is because the nature of such computations has normally not demanded the use of past values to help in solving a current problem. The main recent exception is case-based reasoning (CBR) which has been developed with the inherent intention to solve a current problem by analogy with old ones. A CBR system draws its power from a large case library (of past cases) rather than a set of first principles [Rich 91]. One of the key aspects of case-based reasoning is *adaptation,* i.e. making appropriate modifications to past case(s) in order to use their results in solving a current problem. A CBR system is designed to provide solutions to new problems by retrieving similar case(s) from its case-base and then modifying them appropriately, applying suitable adaptation techniques [Kolodner 92a].

Another AI area where one expects to reuse past solutions in present situations is *planning.* Although the task of machine planning is primarily deliberative, so that it generally involves calling for a broad consideration of the available actions and their consequences [Dean 91] (with the help of causal networks, heuristics, plan libraries etc.), the task often proves to be too expensive to start from scratch each time. Instead, planners are often inclined to use ready-made plans for a current problem either by invoking a used past plan [Hammond 87] or by generating an hypothesis heuristically and then debugging it (GTD - generate, test and debug paradigm) [Simmons 88]. However, a close look at these examples reveals that systems built accordingly have been considered either as early CBR systems (e.g. CHEF [Hammond 86], PERSUADER [Sycara 88a, Sycara 88b]) or their precursors (e.g. PLEXUS [Alterman 86]); or the techniques developed therein e.g. plan transformations using indices [Owens 88], and adaptation of explanations [Kass 88, Kass 89], seem to have been capitalised on by CBR specialists, as historical near ancestors of modern CBR. These observations make it hard to consider modern planning systems as being conceptually self-sufficient; we are prompted instead (by common usage) to consider them within the spectrum of CBR.

Employing an interpolation-like tactic is a natural and plausible way to deal with adaptation, provided that we can design some means of overcoming the obvious difficulties of handling symbolic quantities, which (unlike numerical values) may not have a self-evident straightforward ordering. In one of our papers [Chatterjee 94], produced during the work on this thesis, we have shown how designing interpolation schemes for case adaptation can be accomplished.

Given the way the cache-based architecture is designed, a system built around it needs to retrieve from the cache a solution or solving technique for tackling a current problem. This is equally applicable at different levels of the cache, irrespective of the underlying technique chosen for any one level. Evidently, in order to achieve a desired time-bounded performance, on occasions when a situation is not exactly matched with the cache indices, the system needs to adjust the retrieved answer to suit the current demand. The obvious tactic to make the system most effective is then to put much emphasis on these adjustment techniques with a view to generating solutions which can be computed quickly enough, on one hand, to meet the temporal demands, and also efficiently enough, on the other, to adapt even an imperfect match to a current problem. To any method that the system may apply to meet all these requirements, we give the name **knowledge interpolation.**

The major advantage of using interpolation-like tactics is that the nature of computations here is fairly algorithmic. Consequently, the temporal requirements for these computations are predictable, which is certainly not often true for traditional knowledge-based computations. Therefore, their use in time-bounded applications seems worth exploring. In this chapter we discuss how interpolation can be accomplished for symbolic computations and how its use is possible within a cache-based system.

5.2. INTERPOLATION IN SYMBOLIC COMPUTATIONS

In a numerical domain, in order to find the value of f(X) at a point x_0 of the independent variable X, one uses the distances of x_0 from x_1 and x_2 to interpolate between the values $f(x_1)$ and $f(x_2)$ where $x_1 \leq x_0 \leq x_2$.

The same notion can be translated intuitively into a problem-solving domain if one regards the key characteristics of problems, in the relevant domain, to be the independent variables while the actions that one takes towards solving a problem are the dependent ones. Thus, retrieving past cases where the problem features are similar to the nature of a current problem, followed by interpolation on the actions taken therein, is a possible tactic for generating a candidate collection of proposed actions. Generation of an action comprises generation of the *action name* and values for corresponding *action parameters.* The action name describes the action itself and the action parameters are used in describing how the action is executed.

For example, if the proposed action is to "send" something, then *send* is the action name, and some relevant parameters may be identified as: *what* (i.e. what is to be sent), *where* (i.e. the destination) etc.

If P_0 is the current problem parameter and if the system can retrieve cases with featural values P_1 and P_2 and corresponding action values (which may be action names or parameter values) A_1 and A_2, respectively, then a value for the action parameter A_0 for the current problem can be proposed by interpolation[1] on A_1 and A_2, using the distances of P_0 from P_1 and P_2 respectively. The quality of the result will of course be dependent on the precise relevance of A_1 and A_2 to the current situation, and also on the quality of the knowledge in the region $(P_1\ A_1)$ - $(P_2\ A_2)$ supplied by the domain experts. If there is any peculiar discontinuity in the region, so that interpolation leads to unacceptable outputs, it is up to the experts to refine the knowledge by describing a P, where P is between P_1 and P_2.

5.2.1. Past Work Involving Methods That Resemble Interpolation

In some case-based systems ordering/partial ordering of concepts has been exploited for decision-making, although the term "interpolation" has not been used explicitly. For example:

- CHEF, a model case-based meal-planner [Hammond 87], uses artificial numeric values for ordering objects according to their properties: *taste of broccoli is savoury with intensity 5* and *taste of beef is savoury with intensity 9* are prime examples of this kind.

- In CLAVIER, a fault-recovery system for automated machinery [Barletta 88a, Barletta 88b], various features have been categorised using fuzzy descriptors such as *relevant, partially relevant, irrelevant* (for utility), *too high, too cold* (for temperature) etc.

- The decision tactics practised in TRUCKER and RUNNER [Converse 89, Hammond 88], two time-bounded systems for route-planning, in adapting a new plan on the basis of heuristic judgement on the practicability of different possible routes, illustrate ordering (or at least partial ordering) of characteristics that may appear non-metrisable at first sight.

Any adaptive steps taken in these systems, on the basis of these ordered features, fall within the purview of "knowledge interpolation", under several of the headings that we suggest below.

---

[1] Whether the technique will be an interpolation or extrapolation depends on the relative order of P_0, P_1 and P_2. However, we use the word **interpolation** to cover both interpolation and extrapolation with respect to knowledge-based computations.

In a similar vein, the iterative methods applied in PERSUADER, a labour-problem mediator system [Sycara 88a] that suggests settlements for disputes between company officials and a labour negotiator,[2] and the heuristics such as *prepare a dish with egg as secondary component,* when meeting with two conflicting goals like *a dish with egg as primary ingredient is to be prepared* and *one guest is on a low cholesterol diet,* used in JULIA [Hinrichs 89, Hinrichs 91], also resemble interpolations.

This observation not only suggests that "interpolation" is a reasonable perspective from which to look at quick generation of solutions, but also paves the way for a more in-depth treatment.

5.2.2. Considerations For Implementation

We classify issues relating knowledge interpolation into two major categories:

- *theoretical considerations,* i.e. developing knowledge interpolation as a novel technique for manipulating knowledge of various forms, to facilitate quick computation of solutions.
- *cache-related considerations,* i.e. how to use different interpolation techniques while reasoning with the cache-based architecture.

5.2.2.1. Theoretical Considerations

Although the idea of knowledge interpolation has been initiated as a means to provide quick solution for time-bounded computations, it requires further considerations of several theoretical aspects before we can regard interpolation as an adequate knowledge-based technique for manipulating symbolic quantities.

5.2.2.1.1. *Imposition Of Orderings*

The foremost of the problems to resolve is: how to impose an ordering on symbolic quantities, i.e. identification of different ways for ordering (at least partially) symbolic quantities. The purpose of this ordering is at least to take a first step towards establishing some kind of metric on the relevant quantities. A metrication will enable one to compute distances between participating candidates, which is essential for carrying out any interpolation.

[2] The example provided therein goes as follows: the labour union wants (among other demands) 15% increase in piece-rates (remuneration for processing each product unit), 7% increase in pensions and no subcontracting. The company wants no increase in pension or piece-rate and unlimited subcontracting. However, each side gradually compromises to an agreed solution which says 12% increase in piece-rates, 5% increase in pension and subcontracting for a limited time period.

For numerical quantities, a natural order exists through their magnitudes. Consequently, computation of distances (and hence the practice of interpolation) is quite straightforward in a numeric domain. However, an order between two symbols is meaningless unless an attribute that is relevant in the current context is considered. For example, there is no obvious order on the animals deer, cow and elephant. But we can hope to find some order with respect to any specific property: cow falls between deer and elephant when the attribute is weight, when speed is important elephant comes between cow and deer, while deer falls between elephant and cow under "potential for domestication". A metric can, therefore, be set up in any one dimension to assign relative distances between pairs of entities, and distance in multiple dimensions (multiple properties) can then be calculated via standard metrics (Euclidean, Manhattan etc.). In our work we have identified several different ways to impose metric interpretations on symbolic quantities. We illustrate them through simple but realistic examples, in the following section.

5.2.2.1.2. *Designing Interpolation Schemes*

The second most important task, after determination of an order, is to design different interpolation schemes. While the use of interpolation is to pick up a value between the two reference values, there may not be one unique choice of the intermediate value. Just as there are different types of interpolation in numeric domains (e.g. linear, quadratic), having different degrees of accuracy, for ordered symbolic quantities too one can think of various ways of computing an interpolated value. However, there are difficulties if we try to imitate typical numerical interpolation methods too rigidly in time-bounded domains. In numerical interpolation a higher degree of accuracy is achieved by considering a greater number of interpolating points and/or fitting a higher-degree polynomial. But achieving this in time-bounded symbolic domains suffers from some obvious obstacles:

- searching for a large number of reference points calls for a massive knowledge representation and corresponding retrieval arrangements;
- in a time-pressed situation such retrievals may prove to be too time-consuming;
- the law of diminishing returns may apply, so that a disproportionately large computational cost may have to be incurred for a minor improvement in quality.

Instead, we suggest an alternative way of ensuring better quality in solutions, when more computing time is available. The key quantity, here, is **distance.** Computation of distance between the interpolating points is a primary step in carrying out an interpolation. For numerical quantities, the difference in their magnitude provides the requisite distance; but such a straightforward way for measuring distance between symbols is not obvious. We have developed a number of tactics towards its accomplishment (described later in this chapter), each of which aims at measuring the distance between two symbols in some meaningful way. Evidently, the accuracy of the distance measurement varies with the technique employed. And we suggest that variation in the quality of interpolated results (as forced by the imposed

time limits) can be achieved by resorting to an appropriate "distance measurement" scheme and then applying linear interpolation on the interpolating quantities. Thus, if x_1 and x_2 are the two interpolating values of the independent variable X and values of the corresponding dependent variable Y are y_1 and y_2 respectively (with respect to some symbolic features), then for the given value x_0 of the independent variable, the interpolated value y_0 should be such that:

$$\frac{d(y_0,y_1)}{d(y_0,y_2)} = k * \frac{d(x_0,x_1)}{d(x_0,x_2)}$$

where any *d(m, n)* is the distance between *m* and *n* (measured with an appropriate metric) and *k* is a constant of proportionality.

Hence the more accurately measured is the distance between the interpolating points, the better will be the interpolated result.

5.2.2.1.3. *Designing Schemes for Knowledge Representation*

The task of ordering symbolic quantities and computation of distances between them leads to the obvious problem of designing appropriate knowledge structures for their implementation. "Appropriate" knowledge structures are those that promote efficient computation of distances in order for interpolation to be carried out smoothly. A real-life domain is normally associated with a huge number of symbolic values. Where the number of quantities is large, listing pairwise distances for each possible pair of symbols is time-consuming and impracticable, because not every symbol is always comparable with every other.[3] Moreover, even if two of them can be compared, the need to do so may not always be evident a priori. For example, while dealing with the AGC domain one may not apprehend any reason for comparing the object *dog* with *fire,* say. Naturally, a priori computation of distance between them may then seem meaningless. However, to a GOC a past situation when

a small fire has broken out in the terminal building

is similar to a current problem when

an escaped dog is moving freely in the terminal building,

as both of them can be abstracted as "unwanted appearance, causing delay", and computation of distance between "dog" and "fire" may be deemed necessary in order that the solution of the former can be interpolated for use in the current problem.

Hence, the most practicable decision for the system designer is:

- to identify the appropriate properties upon which distances are to be measured; and
- to design suitable knowledge representation schemes to facilitate quick access to the stored information to make fast computation of distances possible.

[3] Also, with the introduction of an (n+1)-st new symbol, the number of possible pairs is increased by n; thus inducing a rate of increase of order n.

Also important from the representational point of view is the existence of some form of "inverse mapping". As we see below, because of the obvious way in which the interpolation schemes operate, a symbolic value is to be transformed into a suitable numeric equivalent and interpolation is performed on these numeric equivalents only. The numerical result obtained thereby should then be translated back into an appropriate symbolic value. We call this process of retranslation "**inverse mapping**".

5.2.2.1.4. *Interpolation in Problem Solving*

We now examine the feasibility of interpolation for our purposes - which implies identification of where and how interpolation can be applied. Solving any problem involves two essential steps: planning and execution. In order to deal with time-bounded problems, a computing system needs to pay special attention to speed in both the steps. Our work indicates that interpolation, as a means to expedite calculation, is helpful for both the processes, provided the right amount and type of ground-work is put in place. Application of interpolation at any stage requires the following actions:

- identification of the dependent and independent variables for the desired task.
- development of a meta-reasoning framework to decide which interpolation scheme is to be applied.
- producing the interpolated result.

We show later in this chapter the mechanisms that we have developed in order to apply interpolation at various steps of problem solving. In all our subsequent discussions we use the terms (x_1, y_1) and (x_2, y_2) for the interpolating tuples and x_0 for the given value of the independent variable for which the interpolated value is sought. The stage of the reasoning process dictates what the values of x's and y's will be.

We feel (and show below) that although *knowledge interpolation* had primarily been conceived as a tool to be used in conjunction with the cache-based architecture (see chapter 3), the technique has a substantial potential to stand on its own merit as an efficient knowledge-manipulation technique that can, at least intuitively, be applicable to different knowledge-based problem-solving paradigms in general. The rest of the present chapter discusses various steps of development of "knowledge interpolation" as a viable knowledge-based tool, for generating solutions under several different knowledge representation schemes. For the purpose of the thesis we concentrate on *rules* and *cases* as the primary sources of knowledge.

5.3. ORDERING ON SYMBOLIC FEATURES

The heterogeneity in the nature, utility and values of different quantities that are used in a real-life domain does not allow any all-encompassing ordering scheme to take care of all related features (symbols or their values). Consequently, different schemes must be designed for different types of features, taking into consideration their inherent natures and roles in different contexts. We have identified several ways of ordering features. We describe them below, with examples from illustrative situations.

5.3.1. Different Ways Of Imposing Order

The nature and type of the particular feature determine how symbols can be ordered. We consider three broad categories of data types: *scalar, interval* and *set.* For each category, the participating quantities can be ordered directly in several ways by comparing their values along some preferred scale. Also, an indirect way of imposing order involves comparison with some reference. This ordering is indirect in the sense that here the quantities under consideration are not directly compared with each other. Instead their relative positions with respect to some reference point are considered, and the ultimate ordering is based on their relative distances from the reference point.

Since the rationale behind ordering the quantities is to use them in interpolation, we illustrate with examples how interpolation-like techniques can be applied while making decisions involving these ordered quantities. However, it is not necessary that each domain should comprise all the different types of quantities listed below. The symbols used in a domain and their types are decided by the domain experts depending upon their relevance in the context, which in turn dictates the ordering tactic that will be most appropriate.

5.3.1.1. Scalar Values

With respect to scalar quantities we identify the following three types of ordering depending upon the nature of the values.

5.3.1.1.1. *Numerical Values*

There are certain features that can be characterised adequately by numerical values only, e.g. distance, time, weight. Decisions involving these features are normally made by considering their numeric values only.

In these situations people often use straightforward interpolation (usually, unconsciously and subjectively). For example: when A asks B how much it should cost to go to Victoria from King's Cross station by taxi, B immediately replies that it should be around 7 to 8 pounds, from his tentative idea that the trip should take around 20 minutes by road and the information that his last trip by taxi cost him 2.50 pounds to travel between Oxford Street and Euston station (which took nearly 6 minutes of driving).

5.3.1.1.2. *Symbolic Quantities Masking Numerical Values*

There are features that are measurable according to some standard scale (which may be known only to the domain experts and not the people who record the raw case data), yet in non-expert practice are expressed in symbolic terms. The common choice of an ordering of colours according to the wavelengths of the corresponding light (recall the primary school mnemonic VIBGYOR or its reverse ROYGBIV as used in some parts of the world); or ordering of musical notes by their frequencies; or field positions in a cricket field (e.g clockwise from the bowler's left-hand side - *mid-off, extra-cover, cover, point, gully, slip* etc.) are simple relevant examples here.[4] For each angular position given above there can be more than one field position depending upon their distances from the centre of the field: for the same angular value one may have 4 positions *slip, short third man, third man* and *deep third man* reflecting the distance of the fielder from the batsman. This, however, still fits in our original proposition of using ordered symbols in lieu of their inherent numerical value. (Consider, for example, the mapping of elements in a 2-dimensional array into an ordered 1-dimensional data structures by compilers in languages like FORTRAN).

Here too interpolation-like calculation (perhaps unconsciously) is practised. Thus the captain of the fielding side, in a cricket match, may place a fielder at the position "extra cover" (which is in the middle of "mid off" and "cover") instead of placing two fielders at these two positions, respectively, to prevent boundaries through either of them as well as the arc between them.

For features that are not directly metrisable, we suggest two different types of imposed metric: artificial enumeration through ordinals, and fuzzy.

5.3.1.1.3. *Fuzzy Quantifiers*

In practice people often use fuzzy terms e.g. very-big/big/small/fairly-small (for size), heavy/medium/light (for weight), very high/high/low (for temperature) etc. instead of actual

[4] Since a cricket field is a two-dimensional space, a full way of ordering is also possible involving two variables (just as both R and Θ are used in polar coordinate systems) in this context: distance and angular position from a specific point of the field (centre, say).

quantitative values. Here we can borrow from the standard techniques of fuzzy reasoning, e.g. translating fuzzy terms into distributions, performing convolutions to derive distributions expressing combinations of terms, and making inverse translations to find the right fuzzy quantifier for a result. Zadeh [Zadeh 79] discusses this in detail.

The formality of the treatment distinguishes this scheme from the one immediately above, even though their terms may overlap. The idea, here, is to use degrees of membership of different objects to different symbolic property-classes (e.g big, heavy, red). These membership values may provide a requisite for interpolation. Normally, one uses a real number in the range of [0.0, 1.0] to specify the *degree of membership of an object X to a class Y* {denoted as $\mu_Y(X)$}, [5] where 0.0 denotes unconditional falsity and 1.0 represents unconditional truth, respectively, regarding X's degree of membership in the class Y.

However, the fuzzy membership values often take more complicated forms with the introduction of hedges. People in daily practice use hedges like "not" (as in *not good,* to express some sort of disliking),[6] "very", "rather", "fairly", "somewhat", "more-or-less" etc. to convey their degree of appreciation. In order to take care of these hedges, different arithmetics on membership values have been suggested. For example, Zadeh [Zadeh 92] has suggested the following:

$$m_{\text{not } Y}(X) = 1 - m_Y(X) \text{ (for negation);}$$

$$m_{\text{very } Y}(X) = m_Y(X)^2 \text{ (for concentration);}$$

$$m_{\text{more-or-less } Y}(X) = m_Y(X)^{0.5} \text{ (for diffusion);} \quad \text{etc.}$$

Thus if $m_{old}(\text{Peter}) = 0.8$, signifying that the membership value of Peter in the set of old people, is 0.8, then $m_{\text{very old}}(\text{Peter})$ is 0.64. Similarly, one can (subjectively) define consistent operations for other hedges, mentioned above, in accordance with their *notional equivalence* in their daily use.

Studies of organisational behaviour by Wenstop [Wenstop 76] represent a more systematic effort of this kind. There arrays of values for various terms, either as vectors or as matrices, have been constructed. Each relevant term and hedge has been represented as a 7-element vector or 7×7 matrix. Each array element has then been assigned (based on intuition) a value between 0.0 and 1.0.

Here the term "high" has been assigned the value (0.0 0.0 0.1 0.3 0.7 1.0 1.0) and the term "low" has been set to its reverse, i.e. (1.0 1.0 0.7 0.3 0.1 0.0 0.0). Other fuzzy terms have been defined similarly, on the basis of subjective judgement. Wenstop was then able to

[5] In different literatures, other notations such as M-Y-X, m_Y(X), $m_Y(X)$ have been used, to denote the same.

[6] In classical logic, anything "not true" implies "false". But in fuzzy logic, due to the continuity in membership values, "not true" does not imply complete falsity. Therefore, use of the hedge "not" does not correspond to a null membership value. Rather, it reflects a low membership value in the corresponding class.

combine groupings of fuzzy statements to create new fuzzy statements using Max-Min multiplication, which can be translated back to natural language. For example, when asked to generate a label "lower than sort of low", it generates "very low". In a similar vein it produces "rather low" when asked to generate "slightly higher than low". This enabled him to use fuzzy terms as both inputs and outputs in his simulation.

These studies show that convincing interpolation-like activities are possible even when underlying terms are fuzzy.

5.3.1.1.4. *Artificial Enumeration Through Ordinals*

Often artificial orderings are applied to symbols to express their relative order. It is a common practice to arrange candidates according to their educational qualifications by using numbers such as: postgraduate 5/ graduate 4/ diploma 3/ secondary-school graduation 2 etc.[7]

These artificial numbers are normally created by experts in a given domain to impose orders among the participating items. But one should be careful in assigning such artificial numbers - as they not only provide the relative order, but may also be used for computing distances between the objects. Hence, during assignment, these artificial numbers should be so chosen that they reflect the expert's interpretations of the values they possess, expressed in a suitable scale.

It should also be noted that the ordinals are normally set with respect to some specific features (temporarily ignoring less relevant features that may also exist). Thus, for example, a postgraduate may get ordinal number 2 while a holder of a technician's diploma may get 5 when the relevant feature is "ability to do electrical repair work" (say).

5.3.1.1.5. *Referential Modifiers*

Often scalar quantities are expressed not by their own values but indirectly by referring to some other known ones. Terms such as *better-than, bigger-than, at-par, more-or-less* are quite common in our daily expressions where each of the terms carries a specific sense. Thus it is also possible to order quantities with the help of these modifiers. However, it should to be kept in mind that some of the terms are exclusive to some particular use only. For example, *bigger-than* can be used when the underlying property is *size,* say. On the other hand *better-than* can be used for *quality* (although the implicit meaning is same in either case) while *at-par* can be used with respect to both the properties. In our subsequent discussions on referential relations, we call these relational terms (at-par, more-or-less etc.) *modifiers* and the objects to be compared *comparands.*

[7] The ordering used in CHEF, as described above, falls into this category.

5.3.1.2. Intervals

Often features are represented in the form of intervals (e.g. time, age group, salary). Depending upon the context one can use average (or any other moment, if a distribution function is known), lower bound, upper bound, range etc. as the index for ordering.

Average

Normally, one uses average as a means of ordering when the objective is to concentrate on the most typical value of the interval. Thus a football team having the ages of players in the interval 18 to 25 is considered younger than a team for which the range of players' ages is 20 to 27, say (assuming that there is a similar distribution in both cases). For a more complicated distribution function one needs to take into account the weights for computing the average.

Upper Bound

When the objective is to pick up the best possible value, one may tend to order objects on the basis of the upper bound of the interval. Thus in a sports event, javelin throwing, say, one notes the distances covered in different attempts by different participants. Participants can then be ordered on the basis of the greatest distances that they could cover. Thus a person whose three attempts measure 29, 30 and 33 metres respectively wins over another whose three attempts are 31, 32 and 32.5 metres (say), because of the higher upper bound (although the latter has a better average).

Lower Bound

On the other hand, use of lower bounds for ordering can also be practised, most often when the objective is risk minimisation. For example, in order to choose from various companies whose shares one may buy, one may consider the past offers of the companies concerned and prefers the one whose history promises the highest minimum dividend. Thus comparisons between different companies are made here with the help of the lower bounds of the intervals representing the offered dividend rates by respective companies over several years.

Range

In situations where the span or duration of the interval is the key issue, the ranges of the intervals provide a suitable yardstick for ordering the candidates. In athletics, for a sprint a person is declared the winner whose "range" (stopping time minus starting time) to cover the designated distance is least.

Relations

For time-intervals there are 13 possible basic relations [Allen 84]. Some of these (e.g. *during, contained-in, initiates, initiated-by)* lead to an obvious ordering. For example, if an event A "initiates" another event B, we can say that event A has happened prior to event B. The relation "initiated-by" is just the opposite. In similar vein, the occurrence of event A "during" the

event B suggests that event B has taken place for a longer span of time than the event A. Thus with the help of these relations one can order different events cropping up arbitrarily during a process, on the basis of their period of occurrence and control them as per requirement.

5.3.1.3. Sets

Where features are in the form of sets, order can be imposed by considering set-related criteria that are most relevant to the problem in hand. Some of them are:

Cardinality

Often the number of elements present in different sets is a good yardstick for ordering them. However, this is most useful when one does not look for the presence of some special properties among the set members. Hence the most practicable situation for the use of "cardinality" (for ordering) occurs when the set is homogeneous in nature. For example, to carry a heavy object a set of 4 may be considered less effective than a set whose cardinality is 5 (if there is no evidence to suggest that typical members of the two sets are not of comparable strengths).

Composition of the Set

Often sets are compared on the basis of their composition. For certain purposes the number of elements in a participating set is either irrelevant or fixed a priori. Naturally, making comparisons on their cardinality is then meaningless. Under such situations the composition of the set often plays an important role in ordering the sets. For example, in a cricket team, a composition of 5 batsmen, 5 bowlers and 1 wicketkeeper is considered better (because of its adaptability to all strategic situations in a game) than one with 8 batsmen, 2 bowlers and 1 wicketkeeper, even though the cardinality is the same for both.

Statistical Measures

Depending upon the purpose, one may use different statistical measures (average, range etc.) of some property in ordering sets. This happens particularly when one is not interested in the behaviour of individual elements of the set, but rather the overall behaviour considering the set elements as a whole is the primary concern. In cricket, the West Indies team has been considered superior to India because of its better average performance over recent years and not because of the outcome of a particular Test match. In the airport domain, a GOC often makes a mental ordering of different airlines according to their average delay time, general efficiency of their own ground operations, average age of aircraft etc.

Specific Properties of Individual Members of a Set

Often sets can be ordered by reference to a property possessed by just some of its members. This becomes important when the performance of the entire set hinges on the ability of some

key element of the set. The experience of the commander-in-chief of an army in battlefields, or ability of the striker to convert "penalty corners" in a hockey team, fall into this category. In a similar vein, in choosing the crew for a flight to an airport with unusual terrain, a crew with a first pilot quite experienced in flying to this particular airport will be considered more suitable than a crew where the first pilot lacks this experience (even though the composition of the second group may be of higher quality otherwise).

5.3.1.4. Ordering Relative To Some Reference

Often objects can be ordered not by their properties but by their relative positions with respect to some standard datum. Prepositions such as *in-front-of, beside, behind, near-to, far-from etc.* (for specifying location); *at-par, between etc.* (for quantity); or adjectives such as *more, equal, less* etc. are examples of this kind.

Thus a statement such as "A is *more* than B and B is *more* than C" gives an intuitive feeling to the user regarding the orders of quantities A, B and C. In expert-systems use, an unusual one, coming from [Barletta 88b], is an embryo ordering based on ideas of normality, e.g. via rules such as: "since the tool has been changed *within* the last 2 hours, its functioning can be taken as normal".

Thus it is possible to order symbolic quantities in several meaningful ways, and these orderings can be utilised to carry out interpolation-like activities in real domains. In order that "interpolation" can be applied consistently in knowledge-based computations, development of a systematic approach in measuring distances among the participating symbolic quantities is essential, as "distance" has been found to be the key concept in interpolation (see section 5.2.2.1.2). Our response to this need is to identify different distance computation methods to encompass the various ordering tactics, as stated above.

5.3.1.5. Are These Orderings Meaningful?

The obvious question that arises now is: whether all the ordering tactics given impose true orders on the participating elements. By definition, a **linear ordering** (we use the notation «) on a set (S, say) of relevant elements is a binary relation on S, such that $\forall$ x, y, z $\in$ S, the following conditions hold:

1. *Transitivity:* If x « y and y « z, then x « z;
2. *Anti-symmetricity:* If x $\neq$ y, then either x « y or y « x, but not both;
3. *Non-reflexivity:* If x « y, then x $\neq$ y.

Evidently, these properties hold when the expressions concerned are naturally numeric or symbols masking numeric values. For other types of expressions application of all the three properties is not so guaranteed; in particular, "anti-symmetricity" does not always apply (although "transitivity" and "non-reflexivity" hold trivially). For example,

Example1: two different sets, one (A, say) comprising an engineer and a mechanic and the other (B, say) comprising two engineers, have the same value when they are compared by their *cardinality;*

Example2: two intervals [1, 10] and [5, 10], although different, have the same value when *upper limit* is the relevant feature and therefore are equivalent in this ordering.

But, we can justify the ordering tactics by referring to the concept *notional equivalence,* mentioned in section 5.3.1.1.3, concerning the similarities between certain fuzzy qualifiers in their significance of usage. We appeal to a similar notion of equivalence between different participating entities by considering their practical use within the domain of application. For practical use, as stated before, two different entities are compared with respect to certain properties (recall our example of *cow, deer* and *elephant*, section 5.2.2.1.1). Any of the ordering mechanisms described above has the implicit assumption that it will be used if the purpose demands so. Thus *cardinality* of sets will be used only when the number of members of the relevant sets is the feature of concern. Under such a condition it is immaterial for the user to determine what the actual members are. From such a perspective the sets A and B (see Example1, above) are *notionally equivalent* to the user (otherwise, the user will like them to be ordered on the basis of some other property such as *composition of the set*). Similarly, the two intervals of Example2 above, are *equivalent (notionally)* when the user considers the upper limit of the intervals as the ordering criterion.

Thus all the three properties (including "anti-symmetricity") of *linear ordering* hold good if one considers applying them for members of different equivalent classes. When two elements cannot be ordered strictly, they belong to the same equivalence class and one can make a selection arbitrarily by considering any member of the equivalence class concerned.

Hence for application of these tactics a designer needs to be sure of the underlying property for comparison and needs to compose the equivalence classes, a priori. For example in our application, we have defined some equivalence classes for *relational modifiers* - where the terms convey similar impressions. Thus expressions such as "at-par", "equivalent to", "around" fall into the same equivalence class. Similarly, "more-than", "greater-than", "superior-to" form another equivalence class. Under this assumption related terms can be ordered linearly according to the tactic described in section 5.3.1.4.

5.3.2. Use Of Orders For Distance Computation

Computation of distance between two quantities depends upon their type and the characteristics of the underlying ordering. These considerations have prompted us to design different distance measurement schemes. Evidently, the distance between members belonging to the same "equivalence class" is 0. Our schemes therefore apply when members of two different classes are under consideration. We describe these methods below.

5.3.2.1. Scalars

When the items under consideration are scalar, the distance between them is essentially the difference in their magnitudes. If the values are numerical (either inherent or artificially enumerated), then the distance is simply their difference in values. One may use the absolute value of the difference to represent the measure of the distance, while the signed difference can be used to indicate their relative positions (i.e. which one is higher or lower, in the ordering).

For terms expressed symbolically, one needs to refer to an appropriate ordered or partially ordered list containing the relevant items. The difference in their positions in the list then provides a measure of the distance between them. Thus in the ordered list VIBGYOR of rainbow colours the distance of "blue" from "orange" (positions 2 and 5, respectively) is -3.

When symbols involve fuzzy terms, the membership values of the items with respect to relevant classes provide a meaningful cue towards distance computation. For two objects the difference in their membership values with respect to a certain property class can be used as their distance with respect to the said property. However, we deviate from the conventional treatments, such as by Zadeh (described in section 5.3.1.1.3), here. In Zadeh's treatment hedges are used to distinguish one property class from another. Thus "very old" and "old" are two different property classes and one's membership values (μ) to these two classes differ with μ_{old} being greater than $\mu_{very\ old}$. Instead, we exploit the hedges to alter the membership value to a particular class, although we follow similar approaches as suggested by Zadeh in assigning the membership values.

Thus in the absence of any hedges, we use the value 0.75 (following Zadeh) as one's membership value to a given class. Thus if something is denoted as *big*, its membership to the class big (denoted as μ_{big}(big), following our notation) is 0.75. However, this value changes with the addition of some fuzzy hedges. In order to keep calculations simple, we use the following fuzzy hedges for our purpose: *not, fairly, more-or-less* corresponding to membership values 0.25, 0.44 and 0.56 respectively.[8] Thus, μ_{big}(not big) is 0.25,

[8] The choice of 0.56 follows Zadeh's convention: $0.56 \approx 0.75^2$ (mentioned earlier). The choice of 0.44 has been made to deal with complementary classes, described later.

μ_{big}(more-or-less big) is 0.56 etc. Similarly, we use two hedges *very,* and *very-very* to designate higher membership values 0.87 ($\approx 0.75^{0.5}$, following Zadeh) and 0.93 ($\approx 0.87^{0.5}$, to denote even higher membership) respectively. Thus according to our convention the distance between "very old" and "old" is: 0.87 - 0.75 = 0.12.

However, this collection of fuzzy hedges implies a non-uniform distribution of numerics over the interval [0, 1]. In order to obviate this drawback we introduce two more fuzzy hedges in our interpolation scheme: *rather* and *hardly* whose numeric equivalents are 0.65 and 0.35 respectively (keeping in view that one is complementary to the other).

Often, in common usage, humans alternate between two different expressions, conveying exactly opposite senses, with respect to the same property (we regard them as representing "complementary classes"). For example, with respect to the property "size" one uses "small" as well as "big" while their significance is in some sense opposite. Some other complementary classes are: high and low (for altitude); strong and weak (for strength); tall and short (for persons' height); heavy and light (for weight); dark and bright (for colour) etc. which we have come across during our applications. In order to handle these terms consistently we propose using only one of them. For our applications we have decided to use the term "big" for size, and values using the term "small" (with or without hedges) have been transformed into terms that use the value "big" along with some suitable hedges. We assume that the class "small" is complementary to the class "big". Thus membership of an object X to the class "small", $\mu_{small}(X)$, can be computed as : 1.0 - $\mu_{big}(X)$ (and vice versa).

Therefore, we have:

μ_{big}(fairly small)

= 1.0 - μ_{small}(fairly small)
= 1.0 - 0.44
= 0.56
= μ_{big}(more-or-less big).

Therefore, anything that is "fairly small" can also be referred to as "more-or-less big". Similarly, anything "not big" can be referred to as "small"; or "hardly big" is equivalent to "rather small". For the sake of completeness we suggest using two other hedges *not-at-all* and *in-no-way* for which the prescribed numeric values are 0 and 0.13 respectively in order that they can be used to complement the hedges "very-very" and "very". If one uses the hedge "very-very" with some property, we consider its membership to the complementary property class is 0.07. Thus, "very-very big" will be considered as having a near-null (i.e. 0.07) membership in the class "small". Similar treatment can be extended to other complementary property classes mentioned above.

There is a subtle difference between terms involving fuzzy hedges and those using relational terms. While fuzzy hedges are associated with *properties,* the relational modifiers refer to

objects as comparands (i.e. some sort of reference values) albeit there is an underlying property with respect to which the comparison is made). Naturally, no obvious numerical value can be assigned to these terms, and computation of distance involving relational terms requires consideration of both the modifier and the reference value. In order to deal with quantities involving relational terms the obvious difficulty that one faces is that of handling an abundance of modifiers that people tend to use in day-to-day practice. For example, expressions like *less-than, smaller-than, lower-than, inferior-to* convey a very similar notion but people tend to apply them in different contexts (e.g. "smaller-than" for size, "less-than" for quantity). A system designer needs tactics to convert the modifiers into numerical equivalents, to facilitate distance measurement. However, these modifiers do not suggest any values on their own. Rather, they hint at the discrepancy in value by drawing a comparison with some known values. Thus the ultimate numerical quantity that one aims at deriving depends upon the *comparand* i.e. the value of the term that is being referred to (X, say) by the modifier. It can be observed that the same modifier used with two different terms may reflect different degrees of discrepancy. For example, consider the term "at par with X". This expression may refer to any value between 8 and 12 when the comparand X is 10 (according to some chosen scale). On the other hand, the same expression may seem to be valid for the range 800 to 1200 (along the same scale) if the value of X is 1000. Therefore, the absolute difference in magnitude is unsuitable while dealing with these relational modifiers. Use of percentage values is more appropriate for this purpose.

Hence we have the following suggestions for handling quantities involving relational terms:

- Instead of considering each of the different terms separately, one should consider the equivalence classes (see section 5.3.1.5) to which they belong.

- To decide the percentage values that should stand for each class. Because of the imprecise nature of expressions one may use a range of values instead of a single one. We have chosen the following set of values to accomplish this: 90% - 110% for "at-par", 70% - 90% for "less-than", 50% - 70% for "much-less-than", 110% - 130% for "higher-than" and 130% - 150% for "much-higher-than". We assume that a different reference value is to be used when the discrepancy is more than 50% of the value currently being considered. We note that the same value applies to all members of the same equivalence class.

Depending upon the context, different tactics have been designed to compute the distance between two quantities:

Case 1. when the comparands of the two participating expressions are the same, the distance can be computed on the basis of the modifier alone. In order to achieve this, one needs to consider the percentile ranges that represent the relevant modifiers (as described earlier), and the difference of their midpoint values provides the distance information. For example, the distance between expressions *less than* A and *at par* A (for some term A) is 20% or 0.2. When

the reference values are the same, the ultimate interpolated result will also be a relational expression with the same comparand but (possibly) with a different modifier. Hence the abovementioned distance measurement is meaningful.

Case 2. in order to compute distances between relational expressions where the comparands are different or between a relational expression and another term of a different type, one needs to consider both the modifiers and the comparand to derive a numerical equivalent of the expression. The task is carried out in two steps:

1. forming a numerical equivalent of the reference value X. This is straightforward if the term itself is numeric. Otherwise, if X is a member of some ordered list, then as mentioned earlier, the position of X in the list may serve the purpose. Otherwise, if X stands for an object, or a set or an interval-type variable, the method needs first to find the underlying property, based on which the ordering is to be performed. The corresponding value (for the object X) provides the necessary numerical information. (Of course one may have to compute it iteratively, if the property value is symbolic or set etc.).

2. subsequent modification of the numerical value with the help of the modifier. The mid-value of the specified range can be used for this purpose.

5.3.2.2. Sets

Evidently, computation of the distance between two sets depends upon the tactic used in imposing an order on them. Each of the different possible ways of ordering (mentioned in section 5.3.1.3) warrants a specific approach for distance computation. We have identified the following as possible means of achieving this:

1. When *cardinality* is the key issue, computation of distance is straightforward. Here, the difference in the cardinality (i.e. the number of elements present) of the participating sets provides a meaningful measure of distance. In similar fashion, for sets ordered on the basis of various statistical measures (average etc.) the difference in the relevant numeric measure gives the distance.

2. In the event that a property of a specific member of the set is the means for ordering, one must first consider the value of the specific property for each of the participating sets. If the values are scalars, one uses one of the techniques discussed above. Otherwise (i.e. the values are in terms of intervals or sets) one needs to reduce them recursively to scalars or to apply an appropriate statistic (e.g. simple magnitude, average) for measuring distance.

The most complicated task is one where the actual compositions of the sets need to be taken into account for computing distance. Here, we suggest one of the following alternatives:

1. *Use of the difference vector of the two.* The idea, here, is that the composition of the set is listed in the form of vectors, and the sum of squares of the component-wise differences provide a meaningful measurement of distance between the two. For example, suppose a cricket team's composition is described by a 4-tuple (a_1, a_2, a_3, a_4), where a_1 stands for the number of batsmen, a_2 stands for the number of all-rounders, a_3 represents the number of bowlers and a_4 represents the number of wicketkeepers, taken in the 11-member team. Thus the distance (in strength) of two cricket teams having the compositions (5, 2, 3, 1) and (4, 3, 3, 1) is 2.

One can use weights, here, if there is a variation in the components' relevance in the overall composition. For example, for a cricket team there needs to be only one wicketkeeper. Having more than one of them in the team is not only superfluous, but may even degrade its strength in batting or bowling. Similarly, having at least one all-rounder adds to the strength of the team in both batting and bowling. In order to compute distance considering all the above factors, one can use weights decided subjectively on the basis of the importance of each component. For instance, a weight of 1 for difference in number of batsmen and number of bowlers and a weight 3 for difference in number of all-rounders and a weight -10 on the square of the difference in number of wicketkeepers, [9] to compute the difference of the two teams. A negative difference will imply less strength.

However, the problem with the abovementioned example is that it cannot effectively compare the strengths of two teams A (having no wicketkeeper) and B (having 2 wicketkeepers), as the difference of A from B and also of B from A may turn out to be negative. In order to circumvent these difficulties we suggest the following two alternative tactics.

2. *Indirect method.* Here, one stores I, one ideal composition for the sets. In order to compute distance between two sets having compositions A and B respectively, one computes D_A and D_B, the distances of A and B from the ideal I, respectively (possibly by using the abovementioned method). The desired distance then can be computed with the help of the values D_A and D_B.

3. *Use of rules.* For this approach, we store rules regarding evaluation of compositions. For example, with respect to formation of a cricket team, one can have rules such as:

   Rule 1: If the number of all-rounders in team A is greater than that of team B, then for each extra all-rounder add 3 to the score of team A;

   or

   Rule 2: If the number of wicketkeepers in a team is not 1, then subtract 10 from the score of that team.

---

[9] For any team, having one wicketkeeper is ideal. Having no wicketkeeper or more than one wicketkeeper may undermine the strength. Hence the square of the difference is used, here, to wipe out the effect of sign.

Use of these rules may be helpful in evaluating the distance between the participating sets (here, teams) by taking the difference of the final ratings of each team.

5.3.2.3. Intervals

Of the different yardsticks for measuring distance between two intervals, use of average or upper bound or lower bound or range essentially reduces to comparing two scalars - and therefore can be accomplished by a suitable technique (described earlier) depending upon the nature of the relevant scalar value, i.e. whether it is numeric or symbolic or a fuzzy expression etc.

The most difficult distance computation occurs when two intervals are to be compared by their relative positions, i.e. when relations such as *initiates, during* etc. hold. For example, consider computation of distance between [4, 6] and [7, 10]. Here, one possible solution is to use the distance between the mid-points of the said intervals. According to this measure, distance between [4, 6] and [7, 10] is the distance between points 5 and 8.5 (the respective mid-points) i.e. 3.5, which says that interval [7, 10] is at a distance of 3.5 units along the underlying scale and to the right of the interval [4, 6] (because the distance is positive). However, this simplistic measure does not hold good in all situations, particularly with overlapping intervals. For example, according to this notion the distance between intervals [2, 10] and [5, 7] is 0, (as both have a mid-point at 6), which evidently is counter-intuitive.

Common intuition suggests that interpolation between intervals [2, 10] and [5, 7] should generate an interval [x, y], say, where, ideally $2 < x < 5$ and also $7 < y < 10$. Thus computation of distance between two intervals should reflect not only their respective positions along the relevant dimension but also should take care of their respective ranges. In the absence of any suitable measure to capture both these aspects we propose to use a 2-tuple, namely the vector comprising the upper and lower bounds for measuring the distance. In this approach the distance between two intervals is the 2-tuple comprising the distances of the lower bounds and the upper bounds of the two participating intervals, respectively. Thus the distance of interval [2, 10] from interval [5, 7] under the new scheme is (-3, 3).[10] One should note that this vector representation of distance will be useful when the the method is required to give the interpolated result as another interval. In situations where the intervals play the role of dependent variables, use of range or other scalars will normally be sufficient in carrying out the interpolation and hence a suitable technique for scalars can be applied for computing distances.

[10] This does not mean the interval from -3 to +3. It says that the lower bound of the first interval is at a distance -3 from the lower bound of the second interval and similarly the distance in upper bounds is +3. Similarly, distance of the interval [2, 4] from [5, 7] is (-3, -3). Thus for checking the legitimacy of this definition (say, triangle inequality) one has to consider each entry of the distance vector separately and not the pair as a whole.

5.3.3. Algorithm For Computing Distance

In figure 5.1 we describe the algorithm for computing distances between two symbolic quantities. We call the function *distance* which computes the distance between two quantities X_1 and X_2. Our convention for this computation is that the distance will be measured in the direction from X_1 to X_2, so that a negative distance implies $X_2 < X_1$ and a positive distance implies $X_2 > X_1$. This function first checks if the terms are directly comparable for the purpose of computing distance and computes it when possible. Otherwise it reduces the term(s) to a convenient form, and the distance is computed by calling the same function recursively with values in the reduced form. We make use of the following functions for computing distance:

1. *Pos (X, L):* finds the position of the item X in the list L.

2. *Fuzzy-value (F):* returns the numerical equivalent for the fuzzy qualifier F.

3. *Complementary-fuzzy (FT_1, FT_2):* the input is two fuzzy terms FT_1 and FT_2, where each FT_i involves a fuzzy qualifier F_i and a class-name C_i. The function first checks whether the classes C_1 and C_2 are complementary. If they are not, the function returns a failure. Otherwise, it first converts the fuzzy qualifier F_2 such that FT_2 can be expressed as a fuzzy term GT_2 where the class-name is C_1 and the fuzzy qualifier has been modified appropriately (see section 5.3.2.1). The output of the function is the fuzzy qualifier involved in the fuzzy term GT_2.

4. *Mid-value (R):* returns the mid-point of the percentage interval specified by relation R (see section 5.3.2.1)

5. *Get-property-value (X, P):* determines the value for the symbol X with respect to the property P. If X is an object, it may involve simple retrieval from the knowledge-base. But if X is of *set-type* or *interval-type;* this function computes the relevant value.

6. *Modify-value (R, V):* modifies the value of V with the numerical equivalent of the relational term R.

Function distance (X_1, X_2)

```
IF both X1 and X2 are  numeric
THEN
    RETURN  X2 - X1
ELSE IF
  both X1 and X2 are members of same ordered list L
THEN
    RETURN Pos(X2, L) - Pos(X1, L)
ELSE IF
  both X1 and X2 are fuzzy terms
THEN
     Let  X1 = (F1 T1) and X2 = (F2 T2);
     If T1 = T2
     Then
        RETURN Fuzzy-value (F2) - Fuzzy-value (F1),
     Else
        Let F' = Complementary-fuzzy (X1, X2)
        RETURN  Fuzzy-value (F') - Fuzzy-value (F1)
ELSE IF
  both X1 and X2 are relational expressions
THEN
     Let  X1 = (R1 T1) and X2 = (R2 T2),
     If T1 = T2
     Then
         RETURN Mid-value (R2) - Mid-value (R1),
     Else
         Identify P, the relevant property for comparing T1 and T2;
         Let V1 = Get-property-value(T1, P); V2 = Get-property-value(T2, P);
         Let W1 = Modify-value(R1 V1); W1 = Modify-value(R2 V2);
         RETURN W2 - W1
ELSE IF
  both X1 and X2 are intervals
THEN
  Let  X1 = [l1 u1] and X2 = [l2 u2]
  Then
      RETURN  (l2 - l1, u2 - u1)

ELSE   {When the terms are not directly comparable, one needs to convert them into suitable form.}

  Identify P,  the underlying relevant property
               that is the basis for distance computation;
  If  X1 is a Set or Interval or an Object
  Then
     Let V1 = Get-property-value (X1 P)
  Else
     Let V1 = X1;

  If  X2 is a Set or Interval or an Object
  Then
     Let V2 = Get-property-value (X2 P)
  Else
     Let V2 = X2;

  CALL distance (V1, V2).
```

Figure 5.1: Algorithm for Computation of Distance between Two Symbols

5.3.4. Use Of Ordered Symbols In Interpolation

Computation of mutual distances between different symbols helps one to order them linearly. The symbols thus ordered can then be used to conduct interpolation for quick decision-making. For example, in an airport a GOC may have several different options for initiating a repair work: *call-ordinary-worker, call-junior-mechanic, call-senior-mechanic, call-foreman and call-engineer.* Evidently, the underlying property, in this context, for imposing an order among them is their relative effectiveness in responding to certain particular repair work. We call this property *maximum degree of effectiveness (***mdoe**). To enable comparison we attach to each of the options an artificial numerical value. In our system these values are: 0.1, 0.2, 0.5, 0.8 and 1.0, for these five options respectively, demonstrating the domain expert's intuitive feeling about their respective expertise. This information can be used in conducting interpolations in the following way. Consider, for example, two instances, in one of which *junior mechanics* have been called to carry out a repair job whose difficulty is described as *"not high".* An *engineer* has been used, in the other, for doing a repair whose difficulty is *"very high".* These two past instances can be used in suggesting the most appropriate type of workman to be called for tackling a *more or less* difficult job, by using interpolation. The independent variable, here, is the *difficulty* of the job whose values for the two interpolating situations are *not high* and *very high* and for the current problem is *more or less high.* We use for interpolation their fuzzy membership values which are 0.25, 0.87 and 0.56, respectively (see section 5.3.2.1). Interpolating values of the dependent variable (evidently, the mdoe) are 0.2 and 1.0. Our method hence suggests, using a linear interpolation:

$$D = 1.0 + \frac{(1.0 - 0.2) * (0.56 - 0.87)}{0.87 - 0.25}$$

i.e. that the desired degree of efficiency (D) that will be required for the current problem is 0.6. From this value the method can deduce (using the "mdoe" values of different options) and applying what we have named *optimisation through inverse mapping* (described in section 5.4.3) that the most appropriate option will be *calling foreman.* Thus the system avoids calling persons who are not capable enough to address the current problem and therefore are likely to be less effective in this situation and also does not resort to calling more effective technicians who can be employed better for dealing with more serious problems.

Use of some additional knowledge leads to further improvement through interpolation. In the abovementioned situation, the method now needs to determine the number of *foremen* that has to be used to tackle the current problem. We store in our domain knowledge-base along with each option the necessary quantity that would achieve the maximum degree of effectiveness. For example, with respect to the action *call-foreman,* we store the information *":number 4",* stating that the important factor to decide is *number,* (i.e. the number of persons to be called) and 4 of them are required for achieving the declared degree of effectiveness of 0.8 (following our expert's suggestion). Thus one interpolating pair of values (corresponding to the independent and dependent variables) is (0.8, 4). The other

interpolating point is (0, 0) stating quite obviously that no efficiency is achieved when no workers are called. Thus for the current problem, the desired number of "foreman" to be used is computed to be 3 by another application of interpolation.

5.4. DESIGNING DIFFERENT INTERPOLATION SCHEMES

For the purpose of knowledge interpolation, imposition of a suitable metric on the candidate items is logically followed by design of tactics for how to carry out the actual task of interpolation. The following observations, which have been made in this context, are significant for designing interpolation schemes for symbolic quantities:

1. The act of interpolation aims at selecting a point whose distances from the interpolating ones are in a pre-determined (set by the ratio of distances of the independent variables) ratio. The inherent continuity of numeric quantities makes it possible to fix the interpolated result at any point, maintaining any chosen degree of accuracy. But the discrete nature of symbolic knowledge, in general, often stands in the way of achieving such desired accuracies. Furthermore, knowledge interpolation usually needs to produce a symbolic answer, selected from its stipulated vocabulary. Therefore, it primarily involves mapping of the numeric result obtained through interpolation on relevant non-numeric equivalents (as illustrated in previous section) to the relevant list. The discrete nature of symbolic knowledge suggests that each symbol corresponds to a range of numeric values and therefore that the ability to express accuracy to an arbitrary high degree of accuracy is lost.

2. For numerical interpolation, higher degrees of accuracy can be attained by resorting to quadratic or other higher-order interpolations [Conte 72] , which involve three or more interpolating points as opposed to just the two points needed for linear interpolation. As knowledge interpolation is potentially more computation-intensive due to the overheads of the tasks of mapping and inverse mapping (which in turn may involve searching, computations etc.), using three or more computational points may have several disadvantages:
   - it implies proportionately higher demands on computing resources (and hence time);
   - the extra computational effort may not be sustainable in extremely time-pressed situations;
   - the process of mapping and inverse mapping may annul any extra gain in accuracy that one may hope for by resorting to higher order interpolation (while dealing with the numeric equivalents) in the first place.

Since (in most of the real time-bounded situations) maintaining temporal stipulations is more necessary than going for optimality, in the context of the abovementioned observations we shall restrict ourselves here to linear interpolation for symbolic domains. In the different interpolation schemes, stated below, we consider two interpolating points only. Yet depending upon the tactic of how the interpolated point is chosen, one can have different interpolation schemes, of varied degrees of accuracy, for knowledge interpolation. We next describe the different schemes that we have assembled with a view to carrying out knowledge interpolation.

5.4.1. Binary Selection

The simplest interpolation-like action can be termed *binary interpolation.* The approach here is to choose one of the two interpolating values as the answer. Although it is not an interpolation in the truest sense, in situations when time is too pressing it is acceptable human behaviour to resort to selecting one of the two immediately-accessible relevant items as the solution to the current problem. Consider for example, the following situation in an airport:

> *The pilot of an aircraft that is scheduled to take off in 2 minutes has noticed a technical problem in the aircraft and informs the GOC.*

While ideally the solution is to contact engineers to take care of the technical snag, this appears to be too time-consuming given the temporal constraint. Naturally, the GOC is left with two options:

Option 1: Either suspend the flight altogether and reschedule it once the fault is repaired;

or

Option 2: If the fault is not too serious, ignore it and allow the plane to fly.

While the former option implies delay on the part of the flight and subsequent extra work towards rescheduling the flight, the latter involves some risk. The GOC needs to consider the seriousness of the current problem and choose one of the alternatives as the solution. In the framework of interpolation, here, the two interpolating values of the independent variable are the respective estimates of seriousness of the two situations under which the options have been suggested. The available options form the respective values of the dependent variable. The seriousness of the current situation is a known (after a quick consultation with the pilot) value of the independent variable, for which the interpolation is to be performed.

In similar vein, in an airport the directions for aircraft landings and take-offs are governed by actual wind direction, which is arbitrary. However, the GOC needs to choose one of

the two possible alternatives (parallel or anti-parallel) with respect to the direction of a runway. Here, the wind direction is the independent variable while the direction of landing/ take-off is the dependent one.

We envisage the use of two flags, *distance-flag* and *optimist-flag,* to accomplish binary selection. These flags tell which of the two options will be taken. If the "distance-flag" is on, the method selects the option that is associated with a situation closer to the current one. However, if computation of distance is not straightforward one may select one of the alternatives by considering the role of the two alternative options in the current situation. Of the two possible options in many cases it will be possible that one of them suggests an *optimistic* solution while the other suggests a *risk-minimised* solution. For example, the "option 1", above, is less optimistic than the "option 2" solution. The method can be arranged so that if the "optimist-flag" is on it selects the more optimistic of the two alternatives. In figure 5.2 we present the algorithm for binary interpolation, where x_1 and x_2 are the values of the independent variable X, with corresponding values of the dependent variable Y being y_1 and y_2 respectively. The task is to find the value of Y for a given value x_0 of X.

Binary-int (x_0 x_1 y_1 x_2 y_2 &optional Distance-flag Optimist-flag)

```
Compute distances d1 and d2, where di is distance between x0 and xi.

IF  (d1 ≠ d2) and Distance-flag is ON

THEN      {interpolation based on distance}

     If (mod(d1) > mod (d2))
     Then
           RETURN y2
     Else
           RETURN y1;

ELSE      {interpolation based on optimistic solution}

    Determine between y1 and y2 the one that is more optimistic;
    Call it yo;
    Call the other alternative yp;

    If the Optimist-flag is ON
    THEN
          RETURN yo
    ELSE
          RETURN yp.
```

Figure 5.2: General-level Procedure for Binary Interpolation

5.4.2. Discrete Selection - Choosing One Of A Finite Set Of Alternatives

Discrete selection is similar to the previous binary selection, but the major difference is that here the choice is not restricted to the two possibilities only. Instead, there is some means of generating the set of values over which the choice is to be made. For example, in order to communicate with someone at a distance of 10 metres (say), the GOC can choose one of the alternative actions: *talk, run,* or *walkie-talkie.* In the absence of any constraints regarding use of some of the actions, two types of situations may exist:

- all the possible alternatives are equally acceptable to the solver.
- the alternatives are arranged according to decreasing order of preference.

In either case, it is safe for a computation to opt for the first element of the list. In the absence of any constraint, one may accept this selection as the right solution. Otherwise, one needs to select the next item in the list iteratively, until an acceptable (unconstrained) solution is achieved. The relevant algorithm is presented in figure 5.3.

Discrete-Select (x_0 x_1 y_1 x_2 y_2 Alternative-list Constraint-list)

```
Set Flag = Nil
REPEAT
       IF  Alternative-list is   Null
       THEN
            REPORT   Failure.
       ELSE
            Set E = first element of the list
            Check Constraint-list
            If no constraint applies
            Then  Set Flag = E
            Else  Set Alternative-list =  Alternative-list - {E};
UNTIL   Flag;
RETURN   Flag.
```

Figure 5.3: General-level Procedure for Discrete Selection

5.4.3. Optimisation Through Inverse Mapping

When the selection is to be made on the basis of a specific criterion, the selection cannot be random. In these cases the selection has to be made from the relevant list (S-list, say) in such a way that the selected solution satisfies the distance ratio for linear interpolation best. The example given in section 5.3.4 is an example of this technique where the

interpolated result is mapped into the list of possible options represented by their *maximum degree of effectiveness.*

In situations when the interpolated value does not match with any of the item on the relevant list, one selects the item that is closest to the interpolated result. We suggest the use of an "optimistic flag" when selection is to be made out of two prospective candidates (those residing on either side of the interpolated result). When the flag is on one may select the more optimistic of the two possible candidates and turning the flag off should lead to the selection of the less optimistic one. In figure 5.4 we present the underlying algorithm. Another flag (called the "distance flag") can be used to decide whether the selection will be made on the basis of distance alone or whether optimism about the solution will also be considered. In figure 5.4 we present the algorithm.

Inverse-mapping (x_0 x_1 y_1 x_2 y_2 S-list &optional Distance-flag Optimist-flag)

```
Identify the required property, P, for interpolation.
For each item i ∈ S-list let P_i be its value for the property P.
Compute R, the interpolated value based on x0  x1  y1  x2 y2,
        with respect to property P
If distance-flag is ON
Then
       Select from  S-list item i such that ∀ j ∈  S-list, j ≠ i,
          abs (Pj - R) > abs (Pi - R);
       RETURN i
 Else
       Consider  Pi and Pi+1  such that Pi > R > Pi+1 or Pi < R < Pi+1
       Determine between Pi and Pi+1 the one that is more optimistic
       Call it Po, and call the other one Pp
       If optimistic-flag is ON
       Then
          RETURN  the item associated with Po
       Else
          RETURN  the item associated with Pp
```

Figure 5.4: General Procedure for Optimisation through Inverse Mapping

5.4.4. Optimisation Of Certain Functions

In real life, situations often arise when a solution to a problem requires a compromise between two conflicting demands. In these situations the compromise is usually arrived at by providing an alternative that maximises some measure of satisfaction (or minimises some measure of dissatisfaction) of the two demands involved. While one may not consciously work out the value of the function for different alternatives, it is often possible to discover the underlying function. For example,

1. While purchasing a gift, if the budget is quite flexible, [11] one tends to buy the article

[11] When the budget is tightly fixed, one goes by "constrained" decision-making, described later in section 5.4.5.1.

that maximises the *average satisfaction for each unit of money spent.* For example, one often prefers a pair of trousers costing 30 pounds to one that costs 15 pounds, say, provided the buyer thinks that the satisfaction (in terms of quality, effective lifetime etc.) gained from the former is more than double the satisfaction of the second one. On the other hand, if one is convinced that two trousers together, of the second kind, will last longer than one of the first type, one may rather buy two of the 15-pound variety, as this selection provides more satisfaction for each pound spent.

2. In planning a trip, a couple faces the problem of selecting the place. The husband wants to go to Alpine Switzerland but the wife is against going to mountains and insists on going to the sea resort of Nice, France, which the husband does not approve. In the end they compromise by going to historical Rome, which neither of them strongly dislikes. Consequently, one can identify the choice to be that which minimises the total dissatisfaction. (This is an actual observed example!).

3. With respect to the AGC domain, we have met a situation where, in an airport, in the wake of some emergency (e.g. fire, bomb) a GOC may have to evacuate some lounge or terminal building. The process should minimise both time and panic among passengers. The quickest approach is to make a loudspeaker announcement - which, however, is likely to create wholesale panic. The least-panic solution is to approach each person individually, with quiet persuasion to evacuate, which can be time-consuming. Hence both are sub-optimal. However, the optimal solution can be worked out through interpolation between them. This is:

> *to instruct a team of staff members to go to different parts of the affected space. Each individual should address a small group of people and lead them to a safe exit.*

Obviously, to work out this interpolation a computing system requires the knowledge of the two parameters, representing the conflicting requirements and the values of these parameters for all possible candidates. However, to compute the result we deviate from *bivariate interpolation,* the technique available in numerical analysis, to perform interpolation involving two variables as time-pressed situations do not really require that degree of precision. Also, there may be situations when the number of conflicting variables is more than two (say, the preference of a child as well, with respect to the example 2 above). We therefore suggest storing the knowledge of the pay-off function that is relevant for a situation and to work it out for all the possible candidate solutions. The ultimate solution will be the one that has the optimum (maximum or minimum, depending upon the context) value for the desired function.

While the above suggestion seems clear with respect to examples 1 and 2 above (where finding the candidate solutions is straightforward), for situations like example 3 it is not so obvious, because in this case the method also needs to find the appropriate action. However, as we discuss later in this chapter (section 5.5.2.1), a deep analysis of the situations suggests that performing interpolation on actions often boils down to choosing an

appropriate parameter for performing a generic action. For instance, with respect to the example 3 the generic action is "to send an agent" for quick evacuation, and the purpose of the interpolation is to identify the parameter that minimises the panic yet finishes evacuation quickly. Naturally, the best solution should depend upon the given situation. A domain expert suggests that creation of panic is often made easier as the number of people to be evacuated increases. If the number of affected people is small enough, then even announcing through loudspeakers should not generate unmanageable panic. Hence, the properties to be looked for are: how many agents are to be sent, and the radius of effect for each agent. The method can then select, on the basis of these two properties, the one that will be more suited to a given situation, from among the candidate solutions which comprise *general loudspeaker, humans with hand-carried loudhailers, visual display, security personnel* etc. With the knowledge that panic increases as the number of persons to be taken care of by each agent (N, say) increases and completion time increases as N decreases, the approach should optimise the ideal value for N (which in turn gives A, the ideal number of agents to be deployed). Once A is known, a computation can find the radius of effect that is desired for each agent and then can inverse-map on the list of candidate agents to select the most suitable one.

This technique can evidently be applied to simpler situations (such as example 2), where a computation can first identify the candidate destinations by shortlisting them on the basis of one parameter (say, the wife's choice) and then can choose the optimal one from the shortlist with the help of the second parameter (here, the husband's preferences).

5.4.5. More Advanced Interpolation Techniques

While we are on the topic of designing interpolation techniques, we can discuss two more schemes: *constrained interpolation* and *iterative interpolation.* The major difference of these two from the ones stated above is that they are not intended to produce a quick solution; instead, they are aimed at increasing the quality of the solution through consideration of deeper aspects of the situations and using more detailed planning. Selection of a suitable solution worked out by the abovementioned interpolation schemes will seldom produce the best solution, primarily due to two reasons:

- constraints of various forms on employment of the interpolated solution often exist, in which case a more powerful interpolation tactic is required that can take care of these constraints;

- when the interpolation is aimed at satisfying two conflicting demands, it is unlikely that the solution, without any further qualification, will be accepted by the two parties involved. The method should have capabilities to improve upon the interpolated solution, incrementally, by using further inputs from the parties.

Constrained interpolation and iterative interpolation schemes have been conceived to cater for these needs.

5.4.5.1. Constrained Interpolation

Constraints on the solutions may originate from three primary sources:

- *Global,* which are the constraints that are inherent within the definition of the domain and therefore should be stored in a system's knowledge-base. For example, if there is a huge fire in an airport complex, use of multiple fire-brigade engines may be the most judicious action. But their use may be constrained if there is a lack of space for their manoeuvring in the affected area. The solver may then resort to using a smaller number of engines and some hand-carried extinguishers in parallel.

- *Situational.* Often the current state of the world imposes certain constraints which are not part of the global knowledge. For example, unavailability of a certain service on a particular day (or for a particular period).

- *User-specified.* Often the user may impose certain constraints along with the problem. These may be of the form that a particular type/class of solutions should not be considered with respect to the given problem.

Evidently, a sound interpolation should check the above sources, before delivering the interpolated result, if there exist any constraints on the answer that the interpolation scheme has produced. In extreme time-pressed situations, such an approach to a solution may not even be feasible. Hence use of this tactic within the *default level* of a cache architecture is not recommended. Higher-order levels of the cache, on the other hand, may pay attention to constraints, whose representation depends on the nature of the relevant levels. For example, as we see later, the *rule level* of the cache may store some do's and don'ts (which are nothing but constraints) corresponding to situations that they represent. Similarly, failures of past *cases,* while reasoning in the *case level* of a cache, may provide hints (in the form of explanations such as why some particular action failed, why a certain parameter has not been used and thereby imposing constraints on the reuse of some action/parameters in solving a similar problem in future) regarding certain constraints as well. However, these are not likely to present *situational* and *user-specified* constraints; and therefore the system should check with an appropriate knowledge-base the presence of constraints that may affect the course of an action or the result that will be generated through an action it intends to execute. For example, with respect to the AGC domain a system may check (possibly by asking a human user) before calling a particular agent whether that agent is free at the particular moment. Similarly, before suggesting a piece of equipment for use in a given situation a method may check if the user has any reservation against that particular item (e.g. the available workers may not be adept in using it).

The actual use of the *constraints* depends upon the way the plans are stored (i.e. in the form of rules or cases) and will be discussed while dealing with rule and case interpolations.

5.4.5.2. Iterative Interpolation

Often an improvement to the initial solution when there are two conflicting demands can be achieved by applying some bargaining criterion iteratively. For example, in a busy airport a GOC may ask the operations manager of a certain airline to ensure that the next flight leaves within 5 minutes of the scheduled time. The manager, apprehending that loading of passengers will not be completed by that time, asks for a 20-minute concession, with reasons. Through negotiations (exchanges of offers and justifications) they can arrive at a solution, which will interpolate between their two extremes. Thus, the reasoning resembles iterative numerical methods (e.g. bisection method), where an optimal value is reached progressively.

The intuition behind designing this scheme is that there are many situations where an acceptable (to the users) outcome cannot readily be achieved. For example, in the AGC domain it often happens for a departing flight that preparations are not complete so that it can leave at the scheduled time. The authorities of the airline concerned therefore ask the GOC to defer the flight time by a certain number of minutes (m_1, say). On the other hand, the GOC sensing that this will cause considerable traffic problems allows only an extra m_2 minutes, where $m_2 < m_1$. The final settlement is made after each party has compromised to a certain extent. Evidently, any optimum value achieved here has been computed iteratively. PERSUADER, a case-based system for negotiating settlements, deals with such iterative refinements of solutions.

The same technique is intuitively appropriate for a knowledge-based system. Upon computing a solution, the system will provide it to the user for comments, based on which the modifications will be made. This process can continue until no more improvement is possible. Evidently, the inherent nature of this technique prevents us from incorporating it within the time-bounded framework. Its use is mainly advisable if there are no restrictions on the computing time. Nevertheless, we introduce the ideas of the two said methods here for the sake of completeness of development of "knowledge interpolation" as a viable technique for reasoning in knowledge-based computations. However, their utility is limited within the cache-based scheme.

5.5. HOW TO APPLY INTERPOLATION

While ordering mechanisms, tactics for distance computations and interpolation methodologies constitute the groundwork for practising interpolation on symbolic domains, employment of these methods in practice requires making some tactical decisions. The first step towards this objective is to *identify the dependent and independent variable quantities* for the interpolation. We define a few terms towards this objective, and certain system requirements in handling them:

Problem-type: By a problem-type we mean the nature of the problem that hampers the current trouble-free activities of the underlying domain. A cache-based system should have the knowledge of problem-types that it can handle. For example, with respect to the AGC system some problems are "unwanted-appearance", "sickness", "fault" etc. The problem-type helps in identifying the key features that are to be used in accessing a cache and generating a solution.

Problem Parameters: Each problem-type is associated with a set of relevant parameters that differentiate one occurrence of a particular problem-type from another. For example, with respect to problems of type "unwanted-appearance" some relevant parameters are: "importance-of-the-location", "potential-degree-of-threat" (of the undesirable object that has appeared) etc..

Plan: A plan suggests the strategic approach towards tackling the occurrence of problem. Normally, a plan is broken into a series of actions.

Action: Each individual piece of a task that the system may suggest can be called an action. For example, for the AGC domain some valid actions are: "contact", "send", "repair", "evacuate", "move" etc. A system should have a list of actions within its repertoire that it can suggest for solving a problem.

Action Parameters: Each individual action is associated with certain parameters that characterise the performance of the action. Appropriate values for these parameters are to be suggested each time a particular action is invoked (or prescribed) by the system. For example, with respect to the action "send" some valid parameters are: "what" (i.e. which item is to be sent), "where" (i.e. the location), "number" (how many of the items) etc. In order to support a deeper reasoning, two more parameters are often associated with each executed action, *purpose* and *result,* stating the reason why the action has been invoked and what the outcome of the action is, respectively.

These definitions help us in identifying the appropriate set of variables required for the accomplishment of interpolation.

5.5.1. Selection Of Independent Variables

Since the purpose of a cache-based system is to prescribe suitable actions in the event of a new problem situation, the obvious choice for the independent variables (those that govern the decisions) should be the problem-related features. By maintaining appropriate knowledge (for example, the gravity of various problem situations, the key variables of each type of problem etc. that one may encounter in a domain) it is possible to estimate the distances between two problem situations. One can divide different types of heterogeneous problems that one may expect in a given domain into several classes, such that the situations belonging to a particular class are fairly homogeneous in nature and hence recommend similar actions towards their resolution. Within each class one can order the situations by their parameters. Thus for the problems of type "unexpected-appearance" one may have different problem situations depending upon the key object, and order them on the basis of their gravities. For example, the problem of a "bomb threat" is more alarming than a sudden outbreak of "fire" in an airport terminal. In the event that the key objects of two situations are matching, the gravity is determined by the intensity of the affecting object. Thus while comparing two situations in the "fire" category, one considers the intensities in the respective situations i.e. whether they are "high" or "very high" etc. Another key consideration is the "location" of the outbreak. If the affected location implies a greater chance of loss (in terms of life and property), then the situation has a higher degree of gravity.

In order to compute distances between two situations one considers their respective descriptions. If the two problem situations are the same, then distance is measured on the basis of their parameters; otherwise the gravity of the problem types should also be considered along with their parameters for that purpose.

5.5.2. Selection Of Dependent Variables

In a problem-solving environment the aim of a system can be characterised as generation of a series of actions that can mitigate a current problem. Evidently, the role of interpolation, here, is to derive a new sequence of actions, by interpolating on the remedial steps taken in the two interpolating instances. In complicated situations where simple interpolations on the recommended actions are not likely to generate suitable solutions, one must probe more deeply into the problem-solving mechanism to understand the planning that is needed for these situations so that new plans can be engineered for generating the desired solution. In these situations the underlying plan for the past instances serve as a source for the dependent variables for the interpolation process.

Based on this observation, we divide the choice of dependent variables into two levels:

Action level, where interpolation is carried out between two actions. Normally, the two

actions will be such that they have similar purposes with respect to the two interpolating situations.

Plan level, where interpolation is accomplished at a higher level of planning in order to replace a set of actions, meant to achieve a particular goal, by another set of actions, designed to achieve a new goal.

However, within each level there are several sub-levels (explained below) based on the nature of interpolation.

5.5.2.1. Interpolation at Action Level

When the current problem situation is quite close (according to our notion of distance) to the interpolating ones, it is expected that the steps to be taken for tackling the new problem should not be qualitatively different from those representing the interpolating points. Moreover, gross changes in the overall planning are normally not necessary for its accomplishment. Naturally, one tends to use interpolation at "action level" in these situations. There are two variants of this category:

Parametric type, where an old action is retained for a new situation but the action's parameters are altered to suit the new demand.

Tactical type, where the basic action is replaced by a more relevant action commensurate with the new situation.

5.5.2.1.1. *Parametric Type*

When the two interpolating actions match in their names and exhibit their difference only in the parameters, a parametric type of interpolation applies. This can be achieved in the following way:

Altering the size of parameter: suppose the current problem is that there is a *small* fire at some corner of the medical unit of the airport, and the two interpolating points represent situations where,

a) 1 bucket of water has been used to put out a *very small* fire;

   and

b) 3 buckets of water have been used to tackle a *fairly small* fire.

In this situation, the interpolation should work on deciding the number of buckets of water that should be necessary for solving the problem. Here, interpolation needs to respect only the size of the parameter.

Replacement of parameter: in a situation similar to the one just mentioned, but with the exception of having the second interpolating point representing a situation where 2 fire extinguishers have been used to put out a *rather large* fire, a good interpolating method first needs to decide upon the ideal equipment (water/extinguisher etc.) for that purpose. Hence the interpolation, here, seeks to find the required new parameter (by replacing those used in the interpolating situations). This can be achieved by applying inverse-mapping interpolation on the utility of different items of equipment. Once the equipment to be applied is decided upon, its size parameter needs to be calculated.

5.5.2.1.2. *Tactical Type*

Interpolation of this type comprises substitution of an action by another action suitable for accomplishing the same task. This can be assisted by ordering similar actions in accordance with their efficiency to comply with certain demands. Here too, there are two possible ways of doing this:

Simple Replacement: when the actions recommended in the two interpolating points are replaced by a more suitable one to solve a current problem. Thus for example, for the solver there are several possible alternative actions for sending messages (e.g. shout, run, walkie-talkie, telephone, email etc.) and their utilities vary along with the underlying distance to be covered. In a given situation, when the task is to send a message to a distance of 20 metres, and there are the following interpolating points: *shout* when *distance is 5 metres* and *telephone* when *distance is 100 metres,* interpolation on the distance may find that *run* is the best possible alternative.

Replacement by Compounds: when the action suggested by a simple replacement is untenable (possibly due to the effect of constraints), one needs to replace it by compounds. Thus in the earlier case, when the suggested simple replacement is *run for 20 metres* and the person concerned is constrained not to leave a fixed place for any significant time, a possible compound replacement can be *run for a short distance and then shout.*

5.5.2.2. Interpolation at Plan Level

Interpolation at action level may sometimes fail to provide a solution, as it relies on the simplistic assumption that all problems will require essentially similar sets of actions. Such an assumption, however, may not always be valid since there may be more complicated situations which are often not amenable to action-level interpolation only, and where deeper planning is required. One needs to apply plan-level interpolation to alter the underlying plans in order to accomplish the desired objective.

Some of the possible reasons for indulging in plan adaptation have been identified as:

Non-commensurate Parameters: Due to the difference in nature of the problem parameters between the current problem and the interpolating ones, a straightforward application of the actions suggested in the interpolating plans may not be possible. A method may have to drop some of the actions; or replace an action with another; or introduce a new action to the plan to accomplish the task. CHEF [Converse 89] provides an example of this kind, where, to replace beef by duck as the key ingredient in the preparation of pasta, the planner needs to carry out an extra step of removing the fat from the duck.

Non-implementable Action: If some action suggested by the action-level interpolation technique is not implementable (due to some constraint such as missing equipment, non-availability of an agent etc.), a method may have to resort to plan-level interpolation.

For example, if the suggested action for the GOC to communicate with the chief security officer is to *run some distance first and then talk,* but the situation is such that the GOC cannot leave the current place, the solution becomes untenable. He then may retrieve a past case when in order to leave the office for half an hour he had asked his junior to occupy his chair temporarily to discharge his duties. The GOC can now use interpolation between these two solutions - and the result is: *call a junior and send him to the security officer with the message.*

Plan-level interpolations of the above two kinds involves local modifications of the plan and therefore do not require a scrutiny of entire contents of the interpolating plans. But in situations when the available time allows one to do so, the method may indulge in some more ambitious interpolations involving plans. But as application of these interpolations requires knowledge of the entire plan and results obtained after each action, we suggest that their application should be restricted only to an appropriate reasoning paradigm (e.g. using a rich case environment). These two varieties of plan-level adaptation are given below.

Precautionary: Often a problem-solver resorts to some precautionary steps in order to avoid unnecessary later work. For example:

In an airport a fire alarm leads to a significant workload for the management, as hectic evacuation etc. is likely to follow. However, fire alarms are often proved to be misleading as they start sounding due to some other factors (dust etc.) which are far less dangerous than fire itself. Therefore, in the event of a sounding alarm, the GOC may think of a precautionary step of first checking if there is a real fire.

This action step can be generated by extrapolation of a case in which the news of a localised disturbance (bomb) was actually false or a hoax.

Enhanced Performance: In certain situations some guidance may be advisable for advance planning from past cases, if this guidance can increase the quality of the solution. For example:

Suppose the current problem for the GOC is that a plane that is to leave in 10 minutes has a problem with a window which has developed a crack. The two cases that may be retrieved are:

a) *the reardoor of a departing plane was not closing properly, and the GOC had to call a mechanic for repair. The mechanic finished the task in 5 minutes.*

b) *the radio in the pilot's cabin was not functioning properly for a departing plane, and an electrical engineer had to be called. The engineer repaired it but not before deciding that the radio was to be replaced by a new one. The time he took in fetching a new one from the store and installing it caused a delay of 20 minutes.*

From these two plans a system may interpolate on plans to decide an extra action of checking, through some reliable person, if any replacement is necessary. The resulting plan may therefore, thanks to interpolation, include an a priori action of checking if the crack is serious enough so that the window ought to be replaced, before calling for a mechanic, and this action may speed up the performance considerably.

Evidently, application of these techniques relies heavily upon the underlying reasoning tactic, as we have seen before that not all reasoning paradigms support every interpolation scheme mentioned above. Consequently, policies must be developed for carrying out interpolations under different knowledge representation schemes.

5.5.3. Interpolation For Different Paradigms

For our discussion on interpolation we consider two representational paradigms: rules and cases - not only because we have considered them to be suitable for two levels of the cache-based systems, but because of the wide range of knowledge-based systems built around them as the primary paradigms for knowledge representation and reasoning.

5.5.3.1. Interpolation for Rules

To keep close to our picture of interpolation in a problem-solving domain, the rules must be of the form:

IF <*precondition*> THEN <*action*>

where "precondition" refers to some relevant characteristics of the problem under consideration or some conditions that may influence the rectifying action. The purpose of interpolation, here, is to obtain the action that is most suited in a given situation, on the basis of two interpolating rules. Two possible variations of the scheme are:

1. when the preconditions of both of the interpolating rules are applicable to the current problem situation and the actions recommended in these two rules are to be interpolated;
2. when the value of the relevant feature for a current problem is between those expressed in the preconditions of the rules, and the interpolation is to be performed on the actions recommended therein.

In our discussions we have used rules having a single precondition. For rules having more than one precondition to satisfy, a system designer needs to write the rules in such a way that the two interpolating rules disagree only on a single precondition i.e. all but one precondition clauses should be the same for the two rules. The distance between the two rules can then be measured with the help of the two disparate clauses alone. If the precondition clauses in a rule are disjunctive, one ought to split the rule into more than rule each involving a single precondition clause only.

5.5.3.1.1. *Preconditions of Both the Rules Match the Situation*

We illustrate the possible courses of interpolation with examples from the AGC domain. For illustration, let the current problem be that fire has broken out in some part of the airport, and the system has to offer suitable actions. Suppose the system is primarily expected to suggest how to carry out the following basic actions:

- nomination of an appropriate agent whose *efficiency* depends upon the gravity of the situation to be called;
- how to evacuate the location (if necessary);
- how to control any possible spread of the trouble;

The system should have an overall knowledge about the minimum requirements of various factors (e.g. efficiency of the agent that has to to be called, the priorities of the tasks) to handle the situation. Rules can then (e.g. as in real AGC experience) express knowledge about how to modify these kinds of information so that they suit a current situation better. The two interpolating rules (let us call them A and B) guide towards better achievement of these actions. We identify 4 different possibilities in this situation. We explain them by examples arising from different possible variations of the above "unwanted-appearance" problem.

The rules are cooperative: the interpolating rules are such that they suggest the same or similar actions. For example, suppose the current situation is that: *a big fire has broken out near a terminal gate, where passengers are waiting to board a plane,* so that both of the following rules may apply:

Rule A: IF the *location is crowded* THEN *increase* the efficiency of the agent by *up to 10%.*

Rule B: IF the *location is very important* THEN *increase* the efficiency of the agent by *up to 15%.*

The interpolation mechanism should be able to observe that these two rules are not opposing each other and proceed accordingly. The system's action in this situation will be to compute a value of the efficiency of the agent. It may apply one of the following options:

1. use binary interpolation to choose either of the two values (10% and 15%) and find an agent that suits the updated value. One can set the "optimist-flag" suitably to generate desired output.
2. impose an ordering on all the possible agents that satisfy both the conditions, and choose (by discrete selection) one of them;
3. as in 2 (above), create the list of possible solutions and choose one that optimises some function (say, availability, ease of application etc.).

The rules are independent: the interpolating rules are such that neither affects the application of the other. For example, suppose the two rules that the system is considering are:

Rule A: IF the *location is crowded* THEN *increase* the efficiency of the agent by *up to 10%.*

Rule B: IF the *location is crowded* THEN *increase* the efficiency of the evacuation process by *up to 15%.*

Obviously, the task of interpolation in this example is simple, as these two rules do not affect each other and therefore both can be applied without any qualification. The only possible role of the interpolation mechanism here can be in the plan-level, to apply rule B ahead of rule A so that the task of evacuation because of "crowdedness" gets priority over controlling the fire (as saving life is more important than saving property, when choice is to be made between these two).

The rules are restrictive: the interpolating rules are such that the action suggested by one restricts the application of the other rule. For example, suppose the current situation is that:

a big fire has broken out in a hangar where planes are stored.

The system considers the following two rules applicable in this context:

Rule A: IF the *fire is big* THEN *take fire engines as close to the location as possible.*

Rule B: IF the *location is crowded* THEN *leave enough space for evacuation.*

A system, in order to apply these two rules, finds that Rule B restricts the application of Rule A, because with respect to a "hangar" the evacuation implies manoeuvring aeroplanes and finds from the associated knowledge-base that the space that is left for evacuation does not allow fire engines to be operated from close range. We call the postcondition that is flexible (the one corresponding to Rule A, here) the "flexible action" and the other one the "restricting action".

Evidently, the type of interpolation that the system should use is "constrained interpolation". The system should first decide the minimum requirement (M) for the "restricting action", then from the set of all possible solutions of the "flexible action" it should choose the one that satisfies M. With respect to the above example, it translates into first determining the space required for movement of the aircraft (on the basis of number of planes, the space requirement etc.) and subsequently, instructing fire brigades to position themselves as near as possible to the affected site while leaving the space M for evacuation of aircraft.

The rules are opposed in their effect: situations may occur when the interpolating rules that the system may want to use are such that one of them opposes the application of the other. For example, suppose in a situation where *a big fire has broken out near a crowded terminal gate* (as mentioned earlier in this section) the system considers the following two rules to apply:

Rule A: IF the *location is crowded* THEN *open all doors* (to arrange for fast evacuation).

Rule B: IF the *contiguous locations are important* THEN *close exit doors* (to make arrangements so that the trouble does not spread).

Consequently, one finds that the rules are opposing, because to arrange for "fast" evacuation the GOC would need to open as many doors as possible, while that would actually encourage the spreading of the fire. There are three options here:

1. Application of "constrained interpolation" where the system first estimates the minimum number of doors (and which ones) to keep open for safe evacuation and indicates closure of other doors to restrict the spread of the fire as much as possible;

   or

2. A plan-level modification: to keep all the doors open and to post people, with fire-extinguishers in hand, to prevent the fire from propagating until fire engines arrive.

or

3. The system selects one of the two alternatives on the basis of some domain-dependent criteria. For example, in the AGC domain, highest priority should be attached to safety of passengers. Therefore, a GOC, in the absence sufficient computing time for options 1 and 2 above, should apply rule A, ignoring the other rule.

Thus we find different tactical schemes for dealing with each situation. On the basis of available time and other optimisation criteria, the system should act judiciously as to which interpolation tactic should be taken. In our approach we use different combinations of flags to select specific tactics. At present the user should set the flags suitably before the interpolation routines are invoked from within the control program. (It would be an interesting research exercise, but one that we have not attempted, to decide how this process could be automated). We consider these issues further in chapter 6.

5.5.3.1.2. *The Situation Condition Lies Between Preconditions of the Rules*

Another possibility that may occur while reasoning with rules is that the precondition of the current situation falls between those designated by the two interpolating rules. This can happen when an expert has specific policies about extreme situations but is unsure about what should be done in an intermediate situation. For example, consider the following situation in an airport:

> *a GOC receives a message from an airline stating that one of its flights that is scheduled to take off after 15 minutes will not be ready as loading of baggage is taking more time than expected.*

Normally, in an airport the GOC prepares a medium-term schedule of flights and planes are instructed to take off accordance with it. Any alteration in that plan upsets the whole process (as it may mean substantial readjustments), and therefore a GOC is not inclined to make changes in the current schedule. Any request of the above form is therefore normally met with a long deferment (particularly when the activity level of the airport is very high) so that the relevant flight is incorporated in the next schedule. Only when the activity level in the airport is very low is the GOC likely to allow the airline concerned the length of time that it requires and rearrange its departure accordingly. Hence the two rules that describe the GOC's actions are:

Rule A: IF activity level of the airport is *high* THEN *defer the flight by 2 hours.*

Rule B: IF activity level of the airport is *low* THEN *do not impose anticipatory deferment.*

But the question arises: what should be done when the activity level is neither high nor low, but somewhat in between (medium or fairly high, using fuzzy descriptions, say)?

One can apply interpolation to this situation in several possible ways:

1. Binary interpolation: which in this case should take the safest decision, i.e. consider the activity level of any situations that cannot be described as "low" to be "high" and use the solution of rule A.

   Under this scheme, all intermediate descriptions (e.g. fairly high, rather high) are considered as high and the system suggests actions (or action parameters) stricter than actually necessary. The delay and poor performance that this solution inevitably suggests can be circumvented by carrying out one of the following interpolations:

2. Action-level interpolation: here, one considers the level of intensity as the independent variable, and applies interpolation, either based on order of the descriptors (e.g. low, medium, high) or converting the fuzzy description of activity level into numerics (as described in section 5.3.1.1.3), and decides the extent of the anticipatory deferment, which serves as the dependent variable. Thus, if the current situation is described as "fairly high" then the GOC decides the deferment time (D) by applying interpolation considering values 0.25, 0.44 and 0.75 for low, fairly high and high (for the class high) and 0 hours and 2 hours for the two given values of the extent of deferment. Thus the ideal value of D (in minutes) is given by the following equation:

$$D = 120 * \frac{(0.44 - 0.25)}{(0.25 - 0.75)}$$

   The system therefore suggests deferring the flight by 45 minutes and plans accordingly.

3. Plan-level interpolation: the solver may decide not to defer the flight, but in order to guarantee completion of baggage loading within the scheduled time it introduces an extra step in the plan, by referring to a rule of the following kind:

   IF *the deadline for loading is likely to be narrowly missed* THEN *ask the airline to hire help from another airline.*

In order to apply interpolation on rules, the designer of an appropriate method has to spell out policies regarding which particular interpolation technique should be used in a particular situation. In time-bounded applications this is often dictated by the available time. We use several flags to control the application of different interpolation techniques, as required by the situation. The overall reasoning algorithm takes care of setting these flags appropriately. In chapter 6 we discuss these issues with respect to the cache-based architecture.[12]

[12] Since the interpolation techniques are not meant exclusively for the cache-based architecture, their use in other knowledge-based systems is not precluded. But their use elsewhere requires suitable reasoning algorithms for setting various control flags that govern the course of interpolation.

5.5.3.1.3. *Algorithm for Rule Interpolation*

As we have observed in the discussion of the examples above, two participating rules may have two unrelated preconditions yet may have influence on the same action. In this situation we suggest concentrating first on the postconditions of the participating rules, for the purpose of interpolation.

We denote the two participating rules as R_1 and R_2. For each R_i, we denote the preconditions and postconditions as C_i and A_i respectively. We consider two postconditions to be interpolatable if they are operating on the same parameter. As a negative example, "increase efficiency of agent" and "increase efficiency in evacuation process" are not interpolatable, since the parameters of concern are different; although the action names happen to be similar.

The algorithm given below (figure 5.5) assumes that all the relevant flags are set by the control algorithm that invokes the interpolation steps. The preconditions C_1 and C_2 supply the relevant independent variables for conducting the interpolation. These (depending upon the situation) may be the relevant parameter or parametric values used in the C_is.

```
Step 1: IF A1 and A2 are interpolatable {rules are not independent}
        THEN
             Go to Step 2
        ELSE    {rules are not interpolatable}
             If C1 matches with the current situation
             Then RETURN  A1
             Else {i.e. C2 matches with the current situation}
                  RETURN  A2.

Step 2: IF C1 and C2 both hold good in the current situation
        THEN {apply interpolation}
            If A1 and A2 are cooperative
            Then
                Go to Step 3
            Else if A1 and A2 are restrictive
               Then
                  Find from A1 and A2 the flexible action Af
                      and the restricting action Ar;
                   Go to Step 4
                Else  {A1 and A2 are opposite}
                   Find from A1 and A2 the one that is more important;
                   Call this the restricting action Ar;
                   Call the other action  flexible action Af;
                   If plan-level-flag is on
                   Then
                       Go to Step 5
                   Else
                       Go to Step 4;
        ELSE {current situation lies between the preconditions}
            Go to Step 6.
```

```
Step 3: IF the binary-flag is on
        THEN
          Set A  the result of binary selection on A1 and A2
          RETURN  A;
        ELSE
          Generate S, set of possible alternatives.
          If optimise-flag is on
          Then
             Select A from S such that A optimises the relevant function;
             RETURN A
          Else
             Apply  discrete selection  on S to select B
             RETURN B.

Step 4: Find M the minimum requirement for the restricting action Ar;
        Find S the set of possible solutions for the flexible action Af;
        Arrange the members of S in decreasing order of quality;
        REPEAT
          If S is EMPTY
          Then   REPORT   Failure.
          Else   Consider next member, A ∈  S
                 If A satisfies M
                 Then    Set FLAG True
                 Else    Set Flag False;
        UNTIL Flag;
        RETURN M and A.

Step 5: Find S the set of possible solutions for the flexible action Af;
        REPEAT
          If S is EMPTY
          Then   REPORT   Failure.
          Else   Consider next member, A ∈ S
                 If   A is independent of Ar
                 Then    Set FLAG True
                 Else    Set Flag False;
        UNTIL Flag;
        RETURN Ar and A.

Step 6: {the current situation C is between preconditions C1 and C2}
        IF binary-flag is on
        THEN
            If distance-flag is on
            Then
                Find A intermediate solution between A1 and A2;
                RETURN A.
            Else If optimist-flag is on
                 Then
                    RETURN the more optimistic solution from A1 and A2;
                 Else
                    RETURN the risk-minimising solution from A1 and A2.
```

Figure 5.5: Algorithm for Rule Interpolation

5.5.3.2. Interpolation for Cases

Interpolation between cases is aimed at adapting similar past cases to a current situation. In a problem-solving domain this process translates into deciding an appropriate set of actions. But the advantage of using past cases, in comparison with the earlier methods, is that cases do not represent the sequence of actions alone; instead a case describes a solver's entire reasoning and actions in dealing with a particular problem. Consequently, when guidance comes in the form of cases the solver is expected not only to identify a set of actions and their parameters. In addition, a system indulging in reasoning with past cases is expected to learn from the past mistakes how to avoid any errors; to reorder task priorities for better performance; to anticipate possible pitfalls in a plan and take possible safeguards. Hence interpolation with cases should not merely engage in action-level interpolations but should also have provisions for interpolating in plan-level, as described in section 5.5.2.2. To be able to perform the aforementioned interpolations evidently requires:

1. storage of appropriate information in each case of the case-base;
2. designing a suitable representation scheme to assist interpolation;
   and
3. designing policies for how to carry out interpolation in different situations.

5.5.3.2.1. *What Should Be Contained in a Case?*

The three necessary components that should be represented in a case have been identified to be [Kolodner 93 (ch. 5)]:

- *the problem description;*
- *a solution* (which includes the actions that have been taken; the justification behind taking the action; the parameters for each of those actions and their values; the results);
- *an outcome* comprising explanations of why some suggested actions (if any) failed and also possible explanations of the expert as to what should be done in order not to repeat any such mistakes in the future.

However, a survey of CBR systems suggests that although there is no substantial debate about what should be represented in a case, there is a very wide selection of decisions about how to represent cases. Considering the wide range of applications that existing CBR systems are dealing with, one can possibly infer from the above observation that formalisms in representing cases for a particular application are decided primarily on the basis of the requirements of the application and the characteristics of the underlying reasoning scheme.

In a problem-solving environment, where the purpose of using past cases is to guide in deciding appropriate actions, interpolation is aimed at generating the required set of actions and estimating their parameters that suit the current problem best. Evidently, to achieve this desired performance the cases should be so represented in the case base that the interpolating variables can be isolated immediately. Hence, we suggest that the main emphasis in storing information in a case should be laid on the actions taken therein (apart from storing a description of the relevant problem, of course). Each action that has been carried out constitutes a component of the *solution* part of a case. Each of these actions ought to contain the obvious information of the action name and the values for its parameters which serve as the dependent variables for the interpolation. Additionally, we have identified several features which are necessary for any future decision-making using the case:

Relationships between actions: the successive actions that are carried out in solving a problem can have one of the following relationships:

- two actions may be *independent* so that one's execution is not dictated by the other. For example, in order that repair work for a defective aircraft be carried out, a GOC needs to call the engineers and also needs to relocate the on-board passengers. These two actions are independent of each other.

- two actions may have a *consequent* relationship, i.e. an action may need to be carried out as a consequence of another action. For example, relocation of a defective aircraft may take place as a consequence of engineers' preliminary examination that the repair work will take a significant time.

- one action may be the prerequisite of another action. For example, removal of obstructing objects from the path leading to a place that has caught fire may be seen as a prerequisite to arrange access for the fire brigade.[13]

We use the key ***initiated-by*** against each action to describe what prompted initiation of the action concerned. This field of some action A refers to some other action B if A is a consequence or prerequisite of B. If an action is such that the its initiation is necessitated by the nature of the problem, we store the value **problem** for this field (of the action concerned). When two actions have the value "problem" in their "initiated-by" field, they are considered to be independent.

Purpose: Each action that a solver executes is associated with a certain purpose. For example, a fire engine is called with a view to putting out some fire. Storing the purpose

[13] The information about what are the prerequisites for carrying out an action can be retrieved from a knowledge-base of valid domain actions, residing outside the case-base. We discuss this issue in chapter 6.

helps us in judging if an action suggested by the past cases is required for the current problem as well. This also helps in deciding if two actions are interpolatable. For example, "contact electrical engineering unit to send an engineer" is not interpolatable with "contact transport unit for sending a vehicle", as their purposes are different. The method therefore needs to distinguish between these two actions, albeit their names and parameters are same. We use the key ***purpose*** to denote the purpose of an action in the case solution.

Reason: what has prompted a solver to take up this action. A solver takes up an action either because it is dictated by the problem situation (e.g. an outbreak of fire leads to calling fire engines); or because an action with the same purpose has failed (e.g. one decides to call fire engine to put out a fire because an earlier decision to use hand-carried extinguishers has failed); or is warranted by an earlier action (e.g. a action to carry out some repair work ends in asking for a replacement of some component of the faulty item or a call for a more experienced mechanic). We use the key ***because*** to describe the underlying justification behind resorting to certain action or deciding some parameter values. In subsequent uses of the cases, the method can determine from this information both what feature in the problem in hand needs immediate consideration, and the value of the said feature as used in the past case (which will be used in conducting the interpolation).

Result: The end result of an action initiated by the problem solver. A solver suggests an action with a view to getting a certain result. However, on many occasions an action may fail to deliver the expected result. Moreover, on some other occasions an action may not fail outright, but may be considered as sub-optimal as the result is not as satisfactory as wanted. This information should be stored to help in designing better solutions (e.g. by using plan adaptation).

We store with each action the result of its application - which can be either *ok,* or *sub-optimal,* or *fail*, the terms having their obvious significance. An *explanation*-value attached to the *result*-field of an action that produced a "failure" or "sub-optimal" result are used in deciding whether to use the same action in future use of the case or whether some modifications are necessary. A key ***result*** is used to store this information in our representation.

Elapsed-time: The time taken by an action to get the result. This is important when the situation is time-bounded.

As we see below, all this information helps us in carrying out interpolation in various ways. Our scheme for classifying features in a case (as described in chapter 4) helps in extracting this additional information from a case and/or an expert's narratives.

5.5.3.2.2. *How to Represent a Case?*

The representation of cases within the memory should take account of our needs in carrying out interpolations. We have proposed a representation scheme which we call **Action-based Representation.** Here, one stores all the actions that the solver has taken in their order of execution. Assuming that a solver can execute only one action at a time it is straightforward to have a linear order (in terms of time of commencement) of the actions. Typically a human expert will execute the actions serially - thus this representation clearly reflects such behaviour and is quite meaningful. This representation also makes it quite easy for a case interpolation scheme to consider the actions sequentially and judge their relevance with respect to the current situation.

The three primary constituents of a case are represented as three components of the case, namely **problem, solution** and **outcome.** The "problem" part comprises description of the problem that helps in measuring similarities between a past case and a current problem and hence identification of the cases to be used in interpolation. The "solution" part provides the actions carried out sequentially in the past case, as mentioned above, along with the result of each action. The "outcome" part contains the ultimate result and necessary remarks of an expert which provide guidance in future use of the case. We illustrate this representation with respect to a case (Case A) from AGC domain:

Case A: where radio equipment in an aircraft appeared to be non-functioning. Here too, a two-task plan has been executed: *defer the flight by 15 minutes* and *call 1 engineer* (since associated degree of difficulty is "very high"). However, this plan has not been successful because the engineer after preliminary inspection found that a defective part should be replaced and he sent that request. Consequently, the repair could be carried out only after the new item had arrived. To accomplish this the GOC had to arrange for a vehicle in order that the replacement item could be supplied to the engineer quickly. The GOC, considering that it would take significant time, deferred the flight by a further 30 minutes to enable the supply and repair. However, "30 minutes" was found to be an overestimate as the entire work was finished 10 minutes early and therefore some unnecessary delay occurred (which does not reflect well on the GOC's performance). The GOC realised that all this extra work could have been avoided if it were ensured at the beginning that the engineer would bring along with him a replacement item as a form of insurance.

In figure 5.6 we show the action-based representation of the case. In the following section (5.5.3.2.3) we explain how to use this case in conducting a case interpolation in a given situation. In our system we are using this action-based representation of cases.

```
(caseA
  (problem fault-in-aircraft
     :item radio :type non-functioning :activity-level busy :location gate-14)
  (solution
    ((defer-flight :by 15 :because (degree-of-difficulty (very high)))
          (purpose (enable repair))
          (initiated-by problem)
          (result fail :explanation (insufficient-time)))

    ((contact :whom elec-engg :because (object radio)
          :by telephone :because (distance (less-than 500))
          :message (send :what engineer :number 1
                :because (degree-of-difficulty (very high))) )
          (purpose (repair :item radio))
          (initiated-by problem)
          (result fail :time-taken 10 :explanation needs-replacement))

    ((defer-flight :by 30 :because (busy-status high))
          (purpose (enable repair))
          (initiated-by (failed action 1))
          (result sub-optimal :explanation (extra-delay 10)))

    ((contact :whom transport-unit :because (object vehicle)
          :by telephone :because (distance less-than 500)
          :message (send :what vehicle :number 1))
          (purpose (transport :item radio :from elec-engg :to gate-14))
          (initiated-by (prerequisite action 5))
          (result ok))

    ((contact :whom elec-engg :because new-radio
          :message (send :what new-radio :number 1))
          (purpose (supply :what replacement))
          (initiated-by (failed action 2))
          (result ok :time-taken 10)))      ;; <end solution>

  (outcome
       (extra-delay :remedy (ask extra-time :in action 1))
       (extra-work :remedy (ask replacement :in action 2)))) ;; <end caseA>
```

Figure 5.6: Example of Action-based Representation of a Case

5.5.3.2.3. *How to Carry Out Interpolation?*

The method of "case interpolation" is aimed at deriving the right set of actions along with their parameters. We suggest a sequential approach for doing this.

At each step the relevant method should consider the next actions that had been carried out in both the cases, as a base for support of the interpolation. But this approach has the obvious problem that the two selected actions may not be interpolatable. For example, in order to deal with a sudden outbreak of fire, a method will select two past cases both dealing with problems involving "fire breaks out". But the actions taken therein may not be in the same order. In the first case where the fire is described as "big", the solver had first sent for fire engines to contain the fire and then arranged for the evacuation of the trapped passengers. On the other hand, in the second case where the degree of the fire had been medium but the number of trapped persons had been significantly large, the solver first sent for the security personnel to arrange safe evacuation, and subsequently called for hand-carried fire extinguishers for putting the fire out. Evidently, a straightforward sequential interpolation of actions is meaningless in this context, as the corresponding actions in the two cases may not be compatible. The following scheme is suggested to deal with this difficulty:

Step 1: Of the two cases, select one as the *guide case* to provide the planning i.e. the basic guideline regarding which action to take when confusion (as in the above example) arises. Thus with respect to the above example, if the first case is considered as the guide in the effort to solve the current problem, consider how to put the fire out in the first instance, and then consider arrangements for evacuation.

In selecting the "guide case" one may choose from a variety of heuristics:

1. the case that has greater similarity (according some similarity measurement scheme) to the current case;
2. the case that has eventually provided a better quality (according to some domain-dependent criterion) result;
3. the case that suggests the smaller number of separate actions;
4. the case that needs less human interaction (in the form of answers to prompts, to determine the value of parameters for the current situation);
5. the case that has required less time to complete;
6. With respect to the cache-based architecture the following approach in selecting the guide case may be helpful: if the two interpolating cases are picked from two different cells, one can assign weights to each of the two cells.[14] These weights may then

[14] Weights are attached to each of the selected cache cells on the basis of their relevance in the current problem situation. Details of the weight-assignment procedure will be explained in chapter 6.

provide the necessary cue, so that the case prescribed by a cell having more weight can be used as the guide case.

Our proposed method considers the sequence in which actions have been carried out in the guide case, to be its preferred plan for solving the current problem. For each action (call it A) of the guide case, the method first considers if the action is relevant for solving the problem. Once the relevance is established, the method turns to interpolation in order to decide if any changes in the plan (i.e. a plan-level interpolation) is required; what should be the next action; and then to determine the appropriate parameters for the selected action.

Step 2: The relevance of the task is established in the following way. The method checks what has prompted the initiation of the action in the past case. If the action has been initiated as a prerequisite of some later action, then it is considered irrelevant for decision-making. When it has been initiated by the nature of the problem (i.e. the value of the initiated-by field is "problem") or as a consequence of some earlier action, the method considers its purpose to decide if the action is relevant in the new situation. On the other hand, if the action is initiated due to the failure of some action, the method checks if any action has so far been carried out in solving the current problem which has the same purpose. If any such action has indeed been carried out and its result is considered "ok", then the action under consideration is no more relevant; otherwise it is still considered relevant for the current solving exercise.

The next task therefore is to identify the corresponding action from the second case. Since the corresponding action from the second case may not be in the same place as in the sequence in the guide case, the selection is not straightforward. We introduce the concept of *situation relevance* to aid the selection. In step 3 we explain the notion, formation and utility of a "situation relevant" set of actions.

Step 3: While solving a current problem we consider a particular action to be *situation relevant* if the current state of solving is similar to that when the action has been applied in the past case. We suggest that in order to select the action (call it B) from the second case that is interpolatable with the action A of the guide case, one should consider only those actions of the second case that are "situation relevant" to A. We propose the following approach to determine the situation-relevant set of actions from the second case:

First consider the set (S, say) of actions that have been initiated because of the nature of the problem. These actions are characterised by the value "*problem*" for the parameters "initiated-by" (see figure 5.6). To start with, the actions in the set S represent the "situation-relevant" set of actions. As one of these actions is carried out, it is deleted from the set S and any action consequent to the carried-out action is added to the set S. Thus the set is updated continuously.

Once the current situation-relevant set of actions is computed, selection of the most appropriate interpolatable action from the set can be achieved by considering the similarity of each member of the set S with the chosen action A of the guide case and then identifying the one that is most suitable for interpolation. Analogical reasoners [Wolstencroft 93] consider all the pertinent types of similarity, namely structural, semantic and pragmatic (see chapter 4) to develop a uniform way of measuring similarity. In his approach Wolstencroft considered these three types of similarity in that order for deriving a measure of similarity. However, we deviate from this approach because of the following observations:

- structural similarity, which primarily considers the syntactic form in which the actions are expressed (such as number of parameters etc.), is not intuitively appealing when we are trying to determine interpolatability.

- semantic similarity, which for two actions may reflect equality in action names, equivalence of purpose etc. seems to be useful for interpolation, but it has its own drawbacks as well. For example, two actions with the same action-name "contact" but one representing communicating with a fire brigade for putting out a fire and the other representing communication with security personnel for evacuation of people, are not suitable for interpolation even though they both bear the same action-name.

- pragmatic aspects, which for actions may refer to the *purpose* behind resorting to some action and what is expected out of it, appear to be most relevant for determining if two actions are interpolatable.

We therefore propose consideration of pragmatic similarity between the actions in set S and the action A, in the first instance. If more than one action comes into play, we should resort to considering their semantics. We suggest using a hierarchy of action-names to measure semantic similarity. For example, if two actions have the same name (contact, say), the nature of their semantic similarity is obvious. But it is not so straightforward to measure the similarity between actions "contact" and "send-a-person", although both are used for exchanging messages. Hence maintaining an hierarchy where both these actions are descendants of a more generic action ("communicate", say) helps in identifying the relevance. (We shall discuss these aspects in chapter 7). Generation of the interpolated action (from the two selected actions) is possible in the following way:

Step 4: The two actions under consideration are then subjected to interpolation. The nature of the interpolation, however, depends upon the outcome of each action. We describe the possibilities below:

(a) When both actions have been successful, the method resorts to some action-level interpolation between them, which (depending upon the nature of the actions) may be a parametric type or tactical type (see section 5.5.2.1): if the two actions are

semantically similar, the interpolation is done in the parameter level; otherwise a tactical-type interpolation is used to select the appropriate action and then its parameters are estimated (see section 5.3.4).

In the event of extremely time-pressed situations the method may use binary interpolation and select the values used in the action that comes from the case having higher weight. Otherwise, detailed interpolation (e.g. optimisation through inverse mapping) is used in selecting an appropriate action and/or the parameters.

(b) When one of the actions fails or produces a sub-optimal result, the method in a time-pressed situation may recommend the action that has succeeded (hence a binary interpolation). Otherwise, it may consider the reasons of failure of the non-ok action and modify it by analysing the cause of the failure (as given in the associated *because*-field, and taking any remedial step that may be prescribed in the *outcome* of the said case. from the result field of the failed action or from the outcome of the case (see figure 5.6). The modified action is then subjected to interpolation with the corresponding action of the other selected case (i.e. the one for which the action has succeeded).

If no remedial action is prescribed the method then ignores the action and searches the relevant case to find another action with the same purpose and which has been successful. This new action is then subjected to interpolation.

(c) When both actions have a failed or sub-optimal outcome, both are first modified using the remedial suggestions given in the *outcome* of respective cases, or a subsequent successful action is searched from the relevant case(s) (as for the failed case in step 2). The modified/new actions from both the cases are then subjected to interpolation.

(d) Occasionally there may be situations when no corresponding actions exist in the second case that match the current action from the guide case. If this happens, the method checks the *purpose*-field of the said action in order to estimate its relevance in the current context first and to decide if the action needs to be carried out. The choice is limited to deciding whether or not to carry out the action at all. Thus the decision becomes a binary interpolation. If the action needs to be carried out, the method checks its result. If it happens to be a success, the action is recommended. Otherwise, appropriate modifications are needed before suggesting its application.

(e) When all the actions of the "guide case" are taken care of, attention is given to any unmarked action of the second case. All these actions are then treated in a way similar to Situation 4.

5.5.3.2.4. *An Illustration for Case Interpolation*

We now illustrate how interpolation works, by solving the following current problem:

> *In a busy airport the GOC receives a message from the pilot of an aircraft, scheduled to take off soon, that a crack has been noticed in one of the windows of the plane and it cannot fly until this is repaired.*

Such problems disrupt the entire flight schedule in a busy airport and therefore need to be addressed quickly. The two cases that we want to use for solving the current problem are Case A (given in section 5.5.3.2.2 and figure 5.6) and Case B, given below.

Case B: where the rear door of an aircraft was jammed just prior to its departure. The plan that has been executed had two tasks in the following order: *call 1 junior mechanic* and *defer the flight by 5 minutes.* (The parameters have been so decided because the GOC reckoned the degree of difficulty of the problem to be "not high"). The plan has been successful as it could be carried out smoothly.

In our application the case has been represented as given in figure 5.7.

```
(caseB
  (problem fault-in-aircraft
     :item rear-door :type jammed :activity-level busy :location gate-20)
  (solution

    ((contact :whom mechanical-engg :because rear-door
          :message (send :what junior-mechanic :number 1
                   :because (degree-of-difficulty (not high))))
          (purpose (repair :item door))
          (initiated-by problem)
          (result ok))

    ((defer-flight :by 5 :because (degree-of-difficulty (not high)))
          (purpose (enable repair))
          (initiated-by problem)
          (result ok )))                ;; <end solution>
  (outcome done))                       ;; <end caseB>
```

Figure 5.7: Action-based Representation of Case B

Using the case B as the guide case we have the following:

Step 1: the selected action (B) from the guide case is "contact mechanical engineering" The situation-relevant set from Case A has two actions: "defer-flight" and "contact electrical engineering". Evidently, the latter is the similar (to B) action here. But this has resulted in a failure. The method therefore checks the "outcome" of the case A to find out that it is advised to ask for a replacement at this stage. It now considers the relevant

consequent actions in Case A which are the "contact for replacement" and its prerequisite "contact for vehicles". The method reckons the prerequisite action is unnecessary in this situation. The first action that it carries out is: *contact mechanical engineering* to arrange for a spare window pane. And the second action: *contact mechanical engineering* to send 3 foremen, by reckoning the difficulty in window repairing to be "more or less high" (appealing to the knowledge-base) and applying interpolations (as in section 5.3.4).

Step 2: the selected action here is "defer-flight" and the situation-relevant set of case B comprises the single action of deferring by 15 minutes. This being a failed action, the method refers to the "Outcome" of the case and the remedial deferment action to decide that a 35-minute deferment at the beginning should be adequate if the current problem's difficulty is "very high". However, difficulty in window repairing being "more or less high", the system interpolates on the relevant values to find that the ideal deferment time is 22.5 minutes and acts accordingly.

We now present an algorithm to carry out the interpolation between two cases.

5.5.3.2.5. *Algorithm for Case Interpolation*

The following algorithm has been designed to perform interpolation between two cases. Let the cases be C_1 and C_2, with C_1 representing the "guide case". It considers the actions in the guide case sequentially for interpolation and marks an action if it has been taken care of. Once two actions are selected, they are subjected to interpolation by invoking appropriate interpolation routines (e.g. binary selection, inverse-mapping). Similarly, this algorithm invokes interpolation routines to adjust the parameters for a selected action. In the absence of any temporal constraints the system using them should apply a sound interpolation tactic (e.g. optimisation through inverse mapping) that is appropriate. But in time-bounded computations, invoking interpolation routines necessitates decisions regarding which particular interpolation tactic will be employed, and how to set the appropriate flags associated with each routine (see section 5.4). These decisions are made on the basis of the available time at different stages of computations and the temporal requirements of the different tactics. For our present discussion we refer to any such calculations as "apply interpolation".

As discussed in section 5.5.3.2.3, this algorithm marks any action that has been considered for interpolation and concentrates on the unmarked actions only. If the selected action is not relevant for the current problem, then the next unmarked action is considered. If no suitable unmarked action is found, the system has to make decision if interpolation should consider both cases, or it will proceed with only one case. If the method decides to use a single case it sets the appropriate case (i.e. either C_1 or C_2) to C_1 and sets C_2 as null. It modifies the information on marked actions appropriately, and proceeds. When the decision is to continue further with both the cases, the method proceeds with an extrapolation on the selected action from C_1.

Step 1: Consider the next unmarked action from C_1; call it A_1.

```
IF A1 is Null   {i.e. there is no such action}
THEN   Go to Step 6
ELSE IF A1  relevant in the current situation
     THEN    Go to Step 2
     ELSE    Go to Step 1;
```

Step 2: Set S = the "situation relevant" set of actions from case C_2.
Consider action $A_2 \in S$ such that A_2 is interpolatable with A_1.

```
IF A2 is Null        {i.e. there is no interpolatable action}
THEN
      Consider if both the cases are still usable
      If "no"
      Then     Identify the single case to be followed, call it C;
               Set C1 = C; Set C2 = Null;
               Modify the "Marked" information accordingly
               Go to Step 1;
      Else
               Go to Step 3;
ELSE  Go to Step 4
```

Step 3: **{extrapolating on the single action A_1}**

```
IF result of A1 is OK
THEN
   SUGGEST A1 as the next action;
   RETRIEVE L, the list of parameters of A1;
   For all i ∈ L
      APPLY interpolation to compute the value.
ELSE      {i.e. if A1 has failed}
   Consider S1, the "outcome" of case C1;
   If ∃ B ∈ S1 such that B ⇒ avoiding the failure
   Then      {Plan modification}
       SUGGEST B as the next action;
       RETRIEVE L, the list of parameters of action B;
       For all i ∈ L, APPLY interpolation to compute the value.
       MARK B in C1 {as already taken care of}.
   Else
       Consider F1, the set of actions initiated
           by the failure of the action A1 in C1;
       Consider A2 ∈ F1 such that A2 is OK
           and A2 is pragmatically similar to A1;
       MARK A2 in C1;
   Go To Step 5.
```

Step 4: **{interpolating on $A_1 \in C_1$ and $A_2 \in C_2$}**

```
IF both A1 and A2 are OK
THEN Go to Step 5. {action interpolation}
ELSE      {Plan interpolation}
  IF A2 has FAILED
  THEN
      Consider S2, the "outcome" of case C2;
      If ∃ B ∈ S2 such that B ⇒ avoiding the failure
      Then    Set A2 = B;
              MARK B in C2;
```

```
    Else
            Consider F2, the set of actions initiated
               by the failure of the action A2 in C2;
            Consider A2 ∈ F2 such that A2 is OK
               and A2 is pragmatically similar to A1;
            MARK A2 in C2;
ELSE
    Consider S1, the "outcome" of case C1;
    If ∃ B ∈ S1 such that B ⇒ avoiding the failure
    Then   Set A1 = B; MARK B in C1;
    Else
           Consider F1, the set of actions initiated
             by the failure of the action A1 in C1;
           Consider B1 ∈ F1 such that B1 is OK
              and B1 is pragmatically similar to A1;
           Set A1 = B1; MARK B1 in C1;
Go To Step 5.
```

Step 5: {begin interpolation}

```
IF the action names of A1 and A2 match
THEN
  SUGGEST  A1 as the interpolated action. {interpolating on parameters only}
  RETRIEVE L, the list of parameters for the action A1;
  FOR each item i ∈  L DO
       APPLY interpolation to compute the value.

ELSE {Tactical interpolation}
   APPLY interpolation on the action names, to get interpolated action A;
    {Using Discrete Selection, Inverse mapping etc.}
   SUGGEST A as the next action;
   RETRIEVE L, the list of parameters for A;
   For all i ∈ L
      APPLY interpolation to estimate the value of i;
      SUGGEST Result;   Go to Step 1
```

Step 6: {when all actions of case C_1 have been taken care of}

```
REPEAT
    Set Flag 0
    Check if there is any unmarked action in C2;
    IF  "yes"
    THEN
      Check if the action is relevant {from the purpose}
      If "yes"
      Then
          APPLY the action after modification (if necessary)
          MARK the action in C2;
   ELSE   Set Flag 1;
UNTIL  Flag is 1;
STOP.
```

Figure 5.8: Algorithm for Case Interpolation

5.5.3.2.6. *Is the Interpolation Robust?*

In numerical analysis an interpolated result does not depend on the order of the interpolating quantities. While the same is evidently true for different knowledge interpolations discussed above, it does not obviously hold for cases. This is because for case interpolation one needs to choose one of the two cases to be the "guide case", and the sequence of actions that are suggested through interpolation is governed essentially by the action sequence of the chosen guide case. Therefore, while interpolating between two cases C_1 and C_2 (say) the result obtained by treating C_1 as the guide case is likely to be different from that obtained by using C_2 as the guide case. However, a deeper examination would reveal that the apparent discrepancy lies only in the plan level, i.e. the sequence of the actions may alter but the actions and their parameters will remain the same in either choice. Thus robustness of this algorithm can be justified as far as the action levels are concerned. Robustness in the plan is not guaranteed through this algorithm. Hence selection of the guide case is important in these activities. Evidently, the more similar (to the current problem) of the two interpolating cases should be tried as the guide case to obtain a better and more appropriate result. In cache-based systems, the selection of cache-cells (along with their weights) helps one to select automatically the more similar one as the guide case.

5.6. CONCLUSION

This chapter has investigated different possible ways of conducting interpolation for generating quick answers in a problem-solving domain. In this context we observe that the technique of "knowledge interpolation" for reasoning within the cache-based system is clearly both appropriate and helpful. In fact, the idea of "interpolation" was first conceived as a means to generate quick solutions when one is reasoning with a cache-based architecture. In situations when the selected cell of the cache has null contents, one possible tactic for getting a quick solution (if it does not alter the level of computation) will be to interpolate on the contents of the cells that are adjacent to the selected cell (see chapter 3). But as we probe into details of such a system, we find that utility of knowledge interpolation is not limited to the above situations alone; it is applicable in other aspects of cache-dependent reasoning as well. For example, what happens when the input features of the current problem do not match the column headers along some dimension, while reference to the cache is under way? Obviously, in this situation a straightforward retrieval fails to provide an acceptable solution. Consequently, attention has to be given to those columns (along the relevant column dimension) that represent values closest to the problem values. Solutions may be retrieved from the relevant cache-cells and interpolation can then be carried out on them. Even in situations when a single cell has been selected,

one may find that the contents of the cell, due to the abstractions used in accessing the cache, do not match with the current situation and therefore some adaptation is required. In chapter 7 we discuss how to reason with cached information for solving time-bounded problems.

Our discussion in chapters 7-9 refers to caches in which the order of knowledge types as we go from the shallowest to the deepest level is default-rules-cases. However, this is not a fixed feature of the architecture. For a given application, one will start with a default level as the shallowest row of a cache, but what types of knowledge are found at any deeper level will depend only on the nature of experts' problem-solving knowledge for that application, plus the requirement that the expected time to use it in a computation will be not less than the corresponding times for shallower levels, and not more than the times for any deeper level(s).

Although application of "knowledge interpolation" is not confined to any particular system architecture, one major advantage of its use from within a cache-based system is that the cache provides an easy access to the most appropriate solutions or solution schemes, so that there is no need to devise any additional rigorous procedure to determine the necessary interpolating values. Thus the cache-based architecture assisted with the knowledge interpolation technique provides a suitable launching pad for time-bounded problem solving.

Chapter 6.

BASICS OF THE CACHE SCHEME: DESIGNING A CACHE-BASED SYSTEM

6.1. INTRODUCTION

This chapter deals with a systematic approach towards building a cache-based architecture. In chapter 4 we have seen the feasibility of selecting appropriate abstractions (generally by analysing past cases) to design a cache and its accessing mechanism; and in chapter 5 we have introduced the notion of "knowledge interpolation" that facilitates an algorithmic (and hence having temporal predictability) method of deriving solutions from symbolic knowledge, in time-bounded situations. We now focus our attention on identifying the essential formalisms for building such a system in any particular domain.

A complete design of a cache-based system involves two primary requirements:

- identification and representation of appropriate knowledge
  and
- efficient reasoning schemes to exploit the stored knowledge.

Evidently, these two aspects are important for any knowledge-based systems. But with respect to a cache-based system they are more significant because in order to derive time-bounded performance special tactics are needed in representing the knowledge so that it facilitates application of "knowledge interpolation" at various stages of reasoning. Hence building a cache-based system for a particular domain warrants careful analysis of the domain in order that relevant knowledge can be identified.

Based on the utility and purpose we classify the knowledge into three categories:

1. **Cache-related Knowledge:** the knowledge needed to design the cache and the required indices to facilitate accessing the appropriate cells. We assume that the system receives as its input a problem description from which values for relevant indices (those describing the column dimensions) can be computed. These index-values determine the actual cell(s) to be accessed. From the accessed cell(s) the method retrieves essential information for solving the current problem.

2. **Action-related Knowledge:** a cache-based system should be able to produce suggestions of appropriate actions and to estimate their parameters to settle a current problem. Evidently, the system needs to have knowledge about different permissible actions and their parameters that are to be estimated by the solving mechanism. Knowledge is therefore necessary to identify the relevant problem features that may serve as "independent variables" in the interpolations.

3. **Reasoning-related Knowledge:** which comprises the information stored in different cells of the cache and other forms of knowledge (such as which particular interpolation tactic should be used at a particular juncture, how to select relevant rules/cases from the rule-base/case-base, residing outside the cache, for deeper-level reasoning) needed to carry out the reasoning. Using the information about the time that is available, a system determines the particular level of the cache to be accessed and the information stored in a relevant cell is then subjected to subsequent reasoning for generating the solutions. Hence the "reasoning-related" knowledge depends significantly on the reasoning tactic involved, and varies with the level of the cache.

In this chapter we discuss issues regarding acquisition, representation and use of the cache-related knowledge. Issues regarding reasoning-related knowledge will be discussed in chapter 7 where we consider the general functioning of the cache-based system and the particular tactics involved in different levels of the cache depending upon their respective underlying reasoning paradigms.

6.2. CACHE-RELATED KNOWLEDGE

A designer first needs to identify the classes of problem that the cache is aimed at; and the indices that characterise the problems. As mentioned earlier (in section 4.5), proper abstractions should then be applied in order to determine the cache indices. The tasks involve acquisition of the following forms of knowledge:

1. Identification of the column dimensions;
2. With respect to each column dimension, the typical column headers;
   and
3. How to access the (relevant) cache for a given problem.

In chapter 4 we have discussed the abstractions for identifying the appropriate column dimensions. Upon receiving a new problem, the system needs to identify the right cache columns that match the key characteristics of the given problem. Because of the way the cache has been designed (see chapter 3), each of the k column dimensions represents some key feature

of the class of problems for which that cache should hold knowledge about solutions and/or methods of solutions; and each column dimension, in turn, is divided into several columns representing some typical values along the dimension concerned. Having received a new problem the method should refer to the right cache cell in order to retrieve the most appropriate information. The task therefore is to identify the ideal column, from each column dimension, which represents the current problem best. Obviously, the task of identifying the appropriate columns for a current problem becomes easier if some metric can be imposed on key features of the problems so that the column headers of the cache can be arranged according to some specific order. Under this arrangement, along each column dimension one can measure the distances between the problem value (i.e. the appropriate value of a relevant key for the current problem) and the typical values representing the column headers. The column having the least distance from the problem value can then be considered as the closest to the current problem. All the different ordering tactics and corresponding techniques for measuring distance, described in chapter 5, can be applied, when one wants to compute the distance between the current problem and the cache columns. To achieve this objective a system designer needs to identify the relevant features to represent the cache columns and the types of values that will be assigned for each different feature. Consequently, some specific policies are necessary regarding selecting column headers along each of the selected dimensions. We discuss this issue below.

6.2.1. Selection Of Column Dimensions

In this context we have observed that not only does computing distances using variables that are not scalars (such as set and interval-type variables) warrant the extra computation of transforming the values into appropriate scalar quantities, but also their use as column dimensions do not fit naturally into our basic cache design. Hence we recommend ruling out any "set" and "interval" type variables from use as column dimensions and instead suggest limiting the selection to scalar-valued variables for describing the structure of the cache. At this point it is worth distinguishing between the two terms "interval-type" variable and "range". By a range of scalars we mean a set of values defined by a lower and an upper limit on a continuous scale. We regard a variable being of "interval-type" if the values that it takes are of the form of intervals or sets of continuous points. We regard these variables as being unsuitable for column headers. [1] We therefore suggest the use of features whose values are scalars, as column dimensions. This may of course mean transformation of non-scalar features into suitable scalar features. Thus, if the appropriate feature is deemed to have interval-type values, a designer needs to transform it into a suitable scalar, depending upon the most relevant underlying concept in the target domain. For example, in the airport domain the delays that are incurred by different airlines in preparation for departure constitute a key aspect for a GOC in deciding policies involving the airlines concerned. Evidently, the best knowledge about the current delays of any airline can be obtained by checking the delays that

[1] But, as we shall see later (in section 6.2.2), one may use ranges as column headers.

have occurred in the last few departures of its flights. Thus one may consider *delay-record* (of the last 25 departures, say) as a suitable dimension for the cache. Evidently, the values that this feature will take, for each airline, will have interval-type assignments. Computing distance between two such values is complicated. Hence use of a suitable scalar-encoding of the feature *delay-records*, such as *average-delay, maximum-delay* is more appropriate.

6.2.2. Selection Of Column Headers

The next task is to choose column-headers along each dimension of the cache. These should be the values that typify the most important aspects of the problems and/or their solutions, so that they can lead to appropriate solutions for a current problem. Evidently, *unambiguousness* and *distinctiveness* are the two key properties that a system designer should look for, while selecting the column headers for a cache. Hence use of inexact terms (i.e. those involving fuzzy qualifiers or relational modifiers) for column headers is unsuitable for the cache design, as they may introduce uncertainty into selection of the right cache-cells. We therefore suggest avoiding (as far as possible; if it is not possible, the suggestion in section 6.2.2.2 can be followed) the use of any term involving fuzzy qualifiers or referential relations as column headers in the cache. However, this does not preclude use of such terms in specifying a current problem, and the system can resolve these imprecise values by using the numerical equivalents (described in section 5.3.2.1) in order to measure of distances from the column headers.

To summarise the discussion: column headers along each of the cache dimensions should be either numeric quantities or atomic symbols.

6.2.2.1. Numeric Column Headers

Column headers can have numeric values in two ways:

1) when the feature, representing the column dimension concerned, itself has numeric values;
2) when values for the said feature have been enumerated artificially (see section 5.3.1.1.4).

In either situation, however, the problem values can be one of the following:

a) *discrete numeric value:* where the value depends on the nature of the headers, i.e. belonging to 1) or 2) above);
b) *specific range of values:* where all values falling within the range will be referred to the column header concerned;

   and

c) *imprecise:* using "modifiers" such as *around, at-par, greater-than* .

6.2.2.2. Symbolic Column Headers

Symbolic column headers can also be of two types:

a) when the feature concerned represents some property (say, weight) and the domain expert uses symbolic values such as *light, heavy* to denote the two principal values that may designate the underlying property value.

These columns can be referred to by problem values of any of the following types:

- discrete value matching some header exactly;
- fuzzy value involving a fuzzy qualifier along with some term representing a column header (e.g. *fairly* light, *rather* heavy);
- using modifiers (e.g. *less-than* heavy);
- a range of values (e.g. *"in the range heavy to very heavy"*).

b) when the symbolic values that the underlying feature may assume refer to some objects. For example, different locations in an airport. Possible problem values for such features can be either

- exact values matching some of the column headers (e.g. terminal-building);

or

- using modifiers (e.g. *near-to* hangar, *between* terminal-gate *and* taxiway).

We illustrate these points with the difficulties which we had to deal with in working on the AGC domain.

6.2.2.3. An Example

While designing a cache for tackling the class of problems that we call **unwanted appearance** for an airport, we have found that the "extent of threat" that the offending object (whose appearance is causing a problem) may induce is a key aspect that needs to be reflected along some column dimension. Our initial intention to divide all possible objects into meaningful sets (based on their classification as entities, such as "animal", "vehicle", "object") did not appear to be helpful - because the GOC's treatment of these problems does not depend on the most obvious class of the object. For example, a "dog" on the tarmac and an "escaped elephant" on the tarmac are not to be dealt with in the same way despite both being members of the same class "animal". Moreover, as mentioned earlier, such sets can be ambiguous if for the current problem the object whose appearance is unwanted does not fall into some specific class. For example, an "out-of-order jeep" on the taxiway should logically belong to the "object" class and not to the "vehicle" class, although one is likely to choose the latter (particularly when time is very limited). Similar exercises with other classifications such as on the basis of mobility (using symbolic column headers such as *static, slow-moving*) or weight (using column headers such as *light, heavy, very-heavy*) have also met with similar design

problems. Our discussions with domain experts reveal that in actual practice an expert tends to provide a quick solution to a problem by estimating (perhaps intuitively) the intensity of the threat and not merely the class to which the present problem belongs. For example, in dealing with a *fire outbreak* a solver is more concerned with the *intensity of the fire,* on which depends the relevant agent (e.g. fire engine, extinguishers) that is to be called for tackling the situation.

We therefore propose use of the variable "potential-degree-of-threat" (PDT) as the relevant column dimension after evaluating different problem situations that may crop up in the domain, in order that a value can be assigned, by subjective judgement, for the said feature. For each offending problem, one may describe the PDT-value of the any feature by consistently using a real number in the closed numeric interval [0, 1].[2] A domain expert should be able to assign values of this variable to different objects which may pose potential threats during the course of the application. (Use of past cases is also helpful in identification of the various objects that are related to the domain). Another possible alternative, for the system designer, is to use symbolic terms like *mild, very-mild, high* etc. which are more appealing to common human understanding and easier for an arbitrary domain expert to state. However, if symbolic terms are to be used it is mandatory on the part of the system designer to decide upon their numerical equivalents, preferably in the range 0 to 1, in order to adhere to the standard. These numerical values are required for interpolation (as mentioned in chapter 5). Additionally, the system designer needs to decide upon the possible column headers. The task here is:

1. to group the objects, on the basis of their values, so that one column (along the dimension of "potential-degree-of-threat") can be assigned to each group.

   and

2. to decide whether range values or discrete values should be used as column headers.

Thus, with respect to our example, there can be 4 possible types of column headers. Assuming that it has been decided to use 5 columns of equal width, the possible alternative forms for the headers are:

1. *0.1, 0.3, 0.5, 0.7, 0.9* (using artificial numerical quantities);
2. *(0.0 - 0.2), (0.2 - 0.4), (0.4 - 0.6), (0.6 - 0.8), (0.8 - 1.0)* (using ranges involving artificial numerical values);
3. *very-mild, mild, medium, high, very-high* (using symbols masking numerals);

   and

4. *(nil - very-mild), (very-mild - mild), (mild - medium), (medium - high), (high - very-high)* (using ranges involving symbols)

[2] One may choose some other interval if necessary. Our choice of unit length is because expressions involving fractions and percentages fit immediately into this scale.

The system designer should identify (depending upon the domain concerned) the most convenient form of expressing the headers.

6.2.3. Knowledge For Cache Reference

Having received a current problem as its input, the system refers to the cache. In this respect we put the related features into three classes:[3]

Raw features: These are the features that are used in describing a problem. Thus if the current problem (within our domain of illustration) is that *a big fire has broken out in the control building*, then the "raw" features are *big fire* and *control building.*

Primary features: These features are used in accessing the cache. Depending upon how a cache has been designed, one may use a raw feature as the value for a primary feature (e.g. for the example given just above, the raw feature "control building" can be used as the value for the primary feature "location-of-the-event"); or the value of a primary feature can be derived from the raw feature (by referring to the knowledge-base and/or computing some function). The value for the primary feature "potential-degree-of-threat" can be derived from the raw feature *big fire* by first retrieving the value for fire and then modifying it with the numerical equivalent of the qualifier *big.*

Secondary features: these features are not used in accessing the cache but present more subtle aspects of a given problem and are, therefore, used in subsequent reasoning (for deriving a solution) with the problem. For example, *"crowdedness-of-the-location"* is a secondary feature, for our domain of illustration.

The aspects of concern for a designer here are:

1. The primary features are not always the ones that occur naturally in describing a problem. For example, when an object of hindrance appears somewhere within an airport the most natural way of describing it is by using the name of the hindrance (e.g. a bomb, a big fire), which is a raw feature.

2. Even when a raw feature is used as a primary (or secondary) feature for accessing the cache (e.g. location-of-the-event should appear in the problem description as a raw feature, yet we may suggest using it as a primary feature to describe one of the dimensions), the input value may not correspond directly to the values used as column headers along that dimension. For example, one may use inexact values (such as *near-to terminal building*) in describing the location of the event.

[3] Some of them have been used earlier in our discussion in chapter 4 (see section 4.3), but we mention them here for the sake of completeness and recapitulation.

Our suggestions to deal with these aspects are as follows:

1. With respect each type of problem, a system designer should maintain a list of raw features whose values are needed to describe an upcoming problem completely. It should be noted that the raw features may be related not only to the problem description but also to the background. For example, the action steps that a GOC prescribes in the event of a hindrance depend significantly on the current activity level of the airport. However, inclusion of this feature in the problem description does not occur naturally.

2. For each cache the designer should suggest a list of primary features and the necessary secondary features. For example, with respect to the cache designed for handling "hindrance" problems, one should maintain 3 primary features: *potential-degree-of-threat, location-of-the-event* and *activity-level-of-the-airport.* Some of the related secondary features are: *movability-of-the-object* (stating the nature of the movement of the object of hindrance - whether it is static, predictable or unpredictable); *influence-of-the-object* (stating the set of objects that may be affected by the hindering object).

3. For each of the primary features there should be a list of column headers in the form suggested in section 6.2.2.

4. Acquisition of knowledge to enable calculation/determination of values of the primary and secondary features with respect to the current problem so that the values are compatible for measuring distances. There are many ways of achieving this objective:

   Using a function: a primary feature may be computed by applying some function to one or more raw featural values.

   Maintaining appropriate tables: suitable tables can be maintained for storing relevant information and the method may inspect it for retrieving the relevant value. For example, to determine the *influence-on-the-object* we consider four different types of vulnerable items: *humans, aircraft, vehicles* and *gadgets.* Corresponding to each hindrance object we maintain the list of objects which are affected by it: "bomb", "fire", affect all the four types of objects; "water" affects humans and gadgets; "poisonous-gas" affects humans alone; while an escaped animal may affect humans, vehicles and aircraft.

   Maintaining property lists: along with the relevant objects one stores values for their different properties and retrieves the values as and when required.

   Storing numerical equivalents: one maintains numerical equivalents for different symbols used. The symbols can be of two types: values and qualifiers. For example, the potential degree of threat for a bomb may be described as *high.* But if the column headers along this dimension involve numeric values (as described in section 6.2.2) one needs to transform it into an equivalent numeric value. Thus in a scale of (0 - 1.0) one may consider any value above 0.7 to be high. Similarly, numerical equivalents for fuzzy qualifiers and relational modifiers are stored to facilitate any transformation of values between symbolic and numeric domains.

6.2.4. Identification Of The Right Column Header

We now illustrate how this knowledge can be utilised for referring to the cache. With the help of the above knowledge, the values for the primary features are derived first. For each column dimension the computed value is then compared with the prescribed column headers (in the said dimension) to determine its distance from each of the headers. The one(s) having the minimum distance should then be considered for retrieval. However, if the problem value falls between two such header values one may consider both the columns for retrieval, i.e. so that the solution can be derived by applying interpolation on the retrieved values. In chapter 7 we describe details of this kind of reasoning. Here, we illustrate identification of appropriate column headers for a given problem.

Let us assume for the sake of illustration that we decide to use numerical values, so that either discrete numerics or intervals of numerics is the choice for the column headers. In the event of symbolic terms, for both discrete values and intervals, the treatments will be similar to the respective situations involving numerals, provided that an appropriate distance-measurement tactic is used. For a given problem, in order to identify the most useful column along this dimension, it is first necessary to compute the PDT-value for the object under consideration. Let us call this value a *problem value.* Evidently, both the column-headers and the problem value are to be considered for when one is selecting the right column. The problem value can be of one of the possible types

- specific scalar (e.g. *0.42*);
- a range of values (e.g. *in the range 0.35 to 0.45*);

  or
- inexact i.e. using appropriate modifiers (e.g. *around 0.4*).

The type depends upon the knowledge about the current problem situation. (With respect to the AGC domain, if the current problem is that of a *fire outbreak* in some part of the airport, one may expect values according to one of the above types to describe the "potential-degree-of-threat".)

6.2.4.1. Illustration of Column Selection Policies

We now illustrate how to select the right column when the problem is of one of the above three types, by first considering discrete numeric quantities and then intervals involving those quantities, as the cache column-headers.

6.2.4.1.1. *Discrete Column Headers*

With respect to our example, here, the column headers are: 0.1, 0.3, 0.5, 0.7 and 0.9. Depending upon the problem value, we can have any of the three following situations:

Situation 1: problem value is discrete.

In this situation, computation of distance is straightforward, as one computes the distance of the problem value from all the column values and chooses the one that has minimum absolute distance. So, in the above example, if the problem value is 0.42, one chooses the third column (i.e. header value 0.5), as this one is the nearest. However, when distances of two headers are more or less equal, one may select both the columns with appropriate weights (i.e. relatively higher weight for the nearer one). The weights can then be used later, if this is appropriate, for deriving a result from multiple retrieved items, e.g. by interpolation.

Situation 2: problem value is a range.

Here we propose to consider, for each column-header, the average of its distances from each point of the range, and take the one for which the absolute distance is a minimum. Hence, in the problem under consideration, if the range is [0.38, 0.48], its average distances from the five column-headers are: 0.23, 0.03, -0.17, -0.37, -0.57, respectively. Hence column 2 is the right choice. Here too, one may choose two columns with appropriate weights, if they are close to being equidistant from the problem value.

Situation 3: problem value is inexact.

Here, we propose to convert the inexact value into a range, and apply the method suggested for situation 2. In chapter 5, we have prescribed some suitable numerical equivalents for different inexact qualifiers. They can be used effectively for this purpose. These values help in creating appropriate intervals for the inexact values. For example, *around 4.0* will be transformed into the interval [0.35 0.45], and the choice for the right column will then be (imitating the technique in situation 2) column number 2.

6.2.4.1.2. *Range-type Column-headers*

For the purpose of our illustration let the column headers in this case be: [0.0, 0.2], [0.2, 0.4], [0.4, 0.6], [0.6, 0.8] and [0.8, 1.0].

Situation 1. problem value is discrete.

Here, the choice is simple because the interval that contains the problem value is the obvious choice. Thus in the above example, for a problem value of 0.42, one chooses the third column to be the right one. However, ambiguity arises if the given value falls on the border of two intervals. In these situations one can choose both the columns with equal weight.

Situation 2: problem value is an interval.

Here one considers the extent of overlap of the problem value with the column headers and chooses one that has maximum overlap. Thus when the problem value is the range [0.38 0.48], the overlaps with the five headers are: 0.0, 0.2, 0.8, 0.0, 0.0 respectively. Thus the third column is the right choice in this case. Alternatively, both of the second and third columns can be chosen with appropriate weights.

Situation 3: problem value is inexact.

As in section 6.2.4.1.1, once again we can convert the inexact value into an interval and deal with it as we would do for situation 2.

Similar analysis can be carried out when the feature values are ordered symbols.

6.2.4.1.3. *Suggestions for Column Selection*

The main decision problem for the system designer while dealing with numerals and ordered symbols, therefore, is whether to use discrete column headers or range-type column headers. The options have certain inherent properties:

Range-type headers span the entire range of possible values for the underlying feature. Consequently, if the problem value is discrete, it is likely that a single column, which includes the problem value, will be selected for retrieval. (The only situation when two columns are picked up even with range-type column headers is where the problem value is the upper limit of one header and the lower limit of the next one). On the other hand, in the case of discrete-type column headers, unless the problem value does not match exactly with some of the column headers or it is at one end of overall possible range of values, the problem value lies between two headers. Consequently, the two headers on either side of the problem value should be considered for selection. Thus discrete-type headers, by their nature, provide more options for consideration and cost somewhat more in time for computation.

We therefore suggest that unless a small variation in the problem value leads to a significant change in the solution, one should use "range-type" column headers (when the headers can be ordered) to keep subsequent reasoning less complicated.

However, if designing meaningful ranges does not seem to be natural, it is necessary to use discrete-type headers. In this case the reasoning mechanism should be tuned to select only a single column, in order that fast reasoning can be achieved by selecting a single column. This can be achieved in two ways:

(a) select the column that is nearest to the problem value. For example, if the problem value for the underlying feature is x, and the header values are $x_1, x_2, .., x_n$, such that $x_i \leq x \leq$

x_{i+1}, then select column i if $distance(x, x_i) \leq distance(x, x_{i+1})$ and select column $i+1$ otherwise.

(b) either adopt a *pessimistic selection* tactic, where one selects the column that corresponds to a graver situation; or adopt an *optimistic selection* tactic, where one selects the column that deals with a less grave situation. For example, if the underlying feature is "potential-degree-of-threat" and the column headers are: *0.1, 0.3, 0.5. 0.7* and *0.9,* then for a current problem with the feature value 0.45, one may choose the column with header value 0.5 if the selection mode is *pessimistic;* or may choose the value 0.3 if the user prefers to operate in *optimistic* mode. Evidently, the decisions regarding how to decide which one is *pessimistic (*or *optimistic)* depends upon the domain concerned and appropriate knowledge ought to be stored in the system.

6.2.4.1.4. *Unordered Column Headers*

As we have mentioned earlier in this section, features may exist which have no meaningful ordering of the values commensurate with the domain's needs. For example, another important feature for the class of problems *unwanted appearance* is "location-of-the-event" i.e. the location where the unwanted object has appeared. At first we wanted to use some artificial enumerators for establishing an order on the possible values (such as *terminal-building, tarmac*). The most meaningful consideration for the said purpose is: the importance of the location. However our attempt to assign a number in the interval [0, 1] to each location of the airport, reflecting their importance to the GOC, did not succeed for the following reason. We found that two locations *passenger-lounge* and *runway* were both extremely important to the GOC,[4] yet the actions that are carried out for the respective locations, during the emergence of some potentially dangerous object are significantly different. For example, a fire near a runway is unlikely to lead to evacuation of passengers although that is the most important aspect for a passenger lounge. Consequently, we resolved the problem by using discrete column headers: "passenger-lounge", "terminal-gate", "taxiway-&-runway", "others". The justification of using these four headers is that the "passenger-lounge" is primarily passenger infested;[5] a "terminal gate" needs attention to both passengers and planes; appearance of dangerous objects in taxiways and/or runways affects aircraft only; whereas other locations (viz. control building, hangar) may involve static costly items along with some human occupants.

Evidently, no range-type header exists involving these values. The relevant featural value for a problem can be (as stated before) one of the following two alternatives:

---

[4] We assigned the value 1.0 to each of them, as suggested by our experts.

[5] I am told that this is a strange usage to native English speakers, but on reflection it nevertheless seems to say something about many airports that cannot be said so neatly by any other single word or short phrase.

- exact, when the problem value matches with one of the column headers;
- relational, where one may use relational terms involving some column headers such as "*between* passenger-lounge *and* terminal-gate", "*near-to* hangar".

Selecting the right cache column for exact matches is straightforward as the method picks up the column whose label is the same as the problem value. For problem values involving relational terms we use the following heuristics for matching: we restrict our choice of relational terms to either *between* or *near-to.* When the relational term is "between", we pick up the two relevant values involved in the expression, and equal weight of 0.5 is assigned to both the columns. However, when the relational term is "near-to" we select only the column that is involved in the relational expression, for our subsequent reasoning.

Evidently, more complicated relational expressions such as "between A and B but nearer-to B", "near-to A but on the opposite side of B" are used in ordinary discourse. One can define one's own heuristics to handle these expressions. We, however, do not pursue this issue any further as this is not a significant target for this thesis.

6.3. ACTION-RELATED KNOWLEDGE

The reasoning method applies the technique of "knowledge interpolation" (developed in chapter 5), for quick reasoning with the knowledge retrieved from the cache. Application of interpolation inevitably requires some preprocessing (e.g. identifying the property on which the interpolation is based, imposition of ordering) of the interpolatable variables. For the domain whose problems the system is set up to solve, interpolatable quantities, as retrieved from the cache, are the actions. And the interpolation is governed by the problem values. Consequently, some processing of the various actions (those that are permissible within the domain) is necessary in order to make them amenable to the interpolation tactics. This section deals with the necessary knowledge and their representations in this regard. However, we first define some relevant terms that will occur in our subsequent discussions.

6.3.1. Definition Of Some Action-Related Terms

Plan: A plan consists of a sequence of tasks. A system, in its attempt to deal with a problem, first designs a plan and then carries out the tasks sequentially. Some of the pertinent tasks with respect to the AGC domain are:

- *Control the Spread* - suggesting limiting the effect of some troublesome entity.
- *Evacuate* - suggesting evacuation of a certain location.

It is implied that the tasks suggested in a plan should be accomplished in the order in which they occur in the description of the plan.

Generic action: A generic action is a high-level description of something that needs to be done to accomplish some particular task.

For example, generic actions related to the task "evacuate" are:

- *Evacuate-humans* - arranging evacuation of humans who are likely to be affected by an impending object.
- *Evacuate-planes* - arranging for removal of aircraft from an affected area.
- *Evacuate-objects* - arranging for removal of inanimate objects (gadgets, vehicles etc.) from an affected area.

Similarly, some of the generic actions associated with the task "control the spread" are:

- *tackle-fire* - suggesting s step taken to control some outbreak of fire.
- *tackle-gas* - controlling some gas emanating through (e.g.) some leakage.

The generic actions attached to a certain task may be either disjunctive (i.e. only one of them is to be used) or conjunctive (i.e. some or all of the generic actions need to be carried out). For example, in order to carry out a task "evacuate" it is first necessary to find from the context whether to apply all the generic actions listed above, or only a subset of them. On the other hand, for the task "control the spread" one is required to make a taxonomic decision (from the nature of the object) on which particular generic action is most suitable at this juncture.

Action: An action suggests the name of a particular step in a plan. For each generic action, we store a list of particular actions that are capable of doing the task. For example, some actions for the generic action "control the spread" are: *call-sand-bucket, call-water-bucket, call-fire-extinguisher, call-fire-brigade.* Similarly, relevant actions for *communicate* are: *send-email, telephone, walkie-talkie, talk, fax, run.*

Degree-of-effectiveness: This is an artificial number that we attach to each action to denote its effectiveness in achieving its purpose. Not all the actions associated with each generic name are equally efficient in achieving the objective. We therefore propose to attach to each action a numerical value that suggests the maximum degree-of-effectiveness that can be achieved with the action concerned.[6] We express the degree-of-effectiveness of the actions with the help of real numbers between 0 and 1. For example, corresponding to the action *call-fire-extinguisher* we have stored in our knowledge-base the information (0.4) suggesting that the maximum desired effectiveness that can be achieved with this action is 0.4 (obviously for the purpose of controlling the "spread of fire").

[6] Evidently, it is a translation of domain expert's subjective views.

The information on degree-of-effectiveness of the different actions is utilised in the following way. In each plan that is stored in the cache, we assign along with each task the effectiveness (E) that is desired in accomplishing the task. This value depends upon the gravity of the particular problem. For example, the desired effectiveness for the task of "evacuation" in the outbreak of a fire is much more when the fire is described as *big* than when the fire is described as *rather big*. The selected action should be such that its degree-of-effectiveness is greater than or equal to the E-value of the task concerned.

Action parameters: Each action is associated with a list of parameters called action parameters. Some of the parameters refer to individual actions while others are specified with the generic action and the related actions inherit them from their generic action. For example, for the generic action "communicate" the parameters are: *whom* (the person/department to contact with), *message* (the message to be sent). Similarly, a relevant parameter for the action *run* is *distance* (suggesting how far one has to run).

Action prerequisite: Often in order to apply an action we need to determine if any prerequisite is necessary for initiation of the chosen action. For example, in order to display a message one first needs to frame the message; before calling fire-brigades one must ensure that the path is clear for their manoeuvring; in order to announce through a microphone one should first check if any microphone exists in the location concerned and should arrange for it if it is not there already; in order to remove humans/items from a plane, buses/trolleys should be sent to the location depending upon whether the task is "evacuate-human" or "evacuate-object".

Initiation of an action: Initiation of an action is normally a 2-step procedure. First, we check if any prerequisite action exists which is to be carried out before initiating the action concerned. After initiating the prerequisite actions (if any) and obtaining the outcome, we must determine the relevant parameters (of the action concerned) and calculate their values. Once we have a control structure for carrying out one step of the "prerequisite" procedure, it is simple to permit this to work recursively.

Completion time of an action: For time-bounded computations the temporal requirements of each action are a matter of concern for the designer. An action that has a temporal requirement greater than the time dictated by a current problem is unworthy of initiation even if it has the optimum degree of effectiveness for the said purpose. Hence a designer should arrange to store information regarding the time that the action requires for completion. A designer should have a policy, for a particular domain, regarding the mode of expression of this information, e.g. whether to store the minimum or average or maximum completion time, with respect to different permissible actions and/or information about observed past variations in these times.

Default outcome of an action: In a realistic domain, initiation of an action (or some of its prerequisites) often needs human interactions. For example, the different actions under the generic action "communicate" with respect to the AGC domain requires active participation of the user. Due to the unpredictable nature of these actions, the time taken to complete them may exceed the allowed temporal bound. In these situations, in order to continue with the same plan, a system needs to assume some default outcome of the action and proceed with the reasoning accordingly. For example, if the current action of the GOC, in the event of a fire alarm, is to send a person for checking if there is a real fire and the person does not return within the time allowed, the GOC should assume that the fire is genuine and continue on that basis.

The designer of a cache-based system needs to acquire all this action-related information for the domain concerned in order that "knowledge interpolation" can be applied effectively for dealing with time-dependent problems.

6.3.2. Representation Of Action-Related Knowledge

All the action-related knowledge should be represented within the system, so that:

1. It is easily accessible to the reasoning process irrespective of the level of the cache currently under consideration (although it is possible that depending upon the reasoning technique used at each level, the exact requirement of the knowledge will vary);

   and

2. It can easily be handled by the knowledge interpolation procedures in a way that ensures that time-bounded performance can be achieved.

Information about the actions is accessed by the reasoning mechanism for conducting interpolation on the cached answers. We therefore suggest that the general information regarding actions should be stored in the system physically outside the cache so that its access from all the levels of the cache is uniform. This suggestion has the further advantage that any updating of this knowledge will be easier to carry out and less prone to error since it will be done at a single place (unlike the situation where relevant information is stored at each level of the cache). For each of the generic actions we propose to have a frame-like structure where parametric knowledge about the generic action is stored. For example, corresponding to the generic action "tackle-fire" we maintain a frame given as in figure 6.1.

generic action	tackle-fire	
	parameters	*values*
	d-o-e	*<potential-degree-of-threat>*
	where	*<location-of-the-event>*

Figure 6.1: Frame for the Generic Action "tackle-fire

In general, parametric knowledge comprises the following:

(a) The parameters whose values are required to be computed for initiation of this action. For example, according to figure 6.1 the relevant parameters for the generic action "tackle-fire" are: *degree-of-effectiveness* (d-o-e) and *the location* (where);

(b) Against each of the items mentioned in (a) the problem feature (raw or primary or secondary) that influences the value of the parameter and therefore may serve as the independent variable for interpolation. In figure 6.1, for computation of "d-o-e" the relevant independent feature is "potential-degree-of-threat"; whereas computation of "where" depends upon the primary feature "location-of-the-event".

Such a frame may be considered as a node of a tree where the descendants will be the actions that are permissible within the domain to achieve the action's purpose. On certain occasions the descendant of a generic action may be another generic action.

Each of the generic action nodes is in turn a descendant of a parent node describing the purpose for which this generic action is meant. There may be more than one generic-action for the same purpose. For example, with respect to the AGC domain, for the purpose of *passing-message* we have two generic actions: *communicate* and *send-messenger;* each having its own set of actions (as descendants). For example, some of the actions for "communicate" are *talk, run, fax.* Similarly, for *send-messenger* there are two descendant actions: *send-person* (when the person to be sent is near) and *ask-person* (when a specific agent is to be contacted to pass the message). Figure 6.2 considers such a tree where the purpose "passing-message" is the root node; the two generic actions are intermediate nodes, each having their descendant actions, along with their maximum degree-of-effectiveness values, (for the purpose of passing-message) as the leaf nodes.

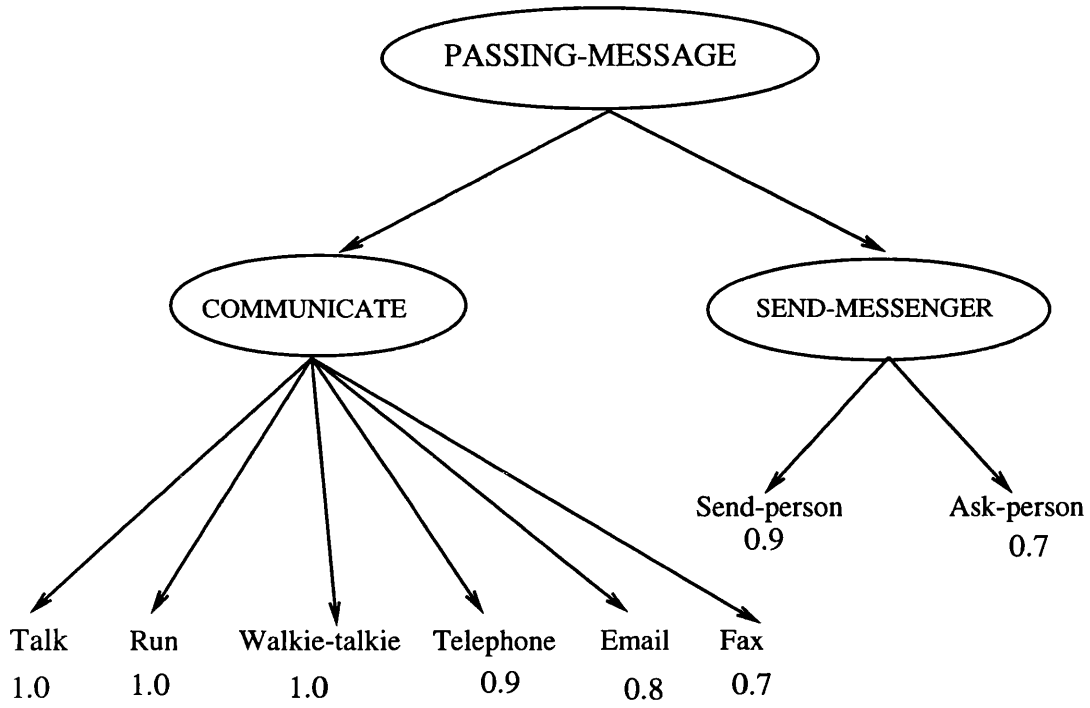


Figure 6.2: Hierarchy Tree for the Purpose "passing-message"

For certain actions this maximum degree-of-effectiveness can be achieved only by adjusting certain parameters. The hierarchy tree contains this information as well, whenever it is appropriate. Consider for example, the various possible actions under the domain for controlling the spread of fire. Various actions suitable for this task are: *call-sand-bucket, call-water-bucket, call-fire-extinguisher,* and *call-fire-brigade,* with their respective maximum degree-of-effectiveness being 0.1, 0.2, 0.5 and 1.0. But to achieve the maximum performance from these actions, an adequate number of items should be employed. This information about the desired numbers should also be incorporated in the knowledge-base. Figure 6.3 illustrates the relevant portion of the hierarchy tree, where the root node represents the purpose "control-spread". Note that "tackle-bomb", "tackle-animal", "tackle-gas" are some of the other generic actions for the same purpose.

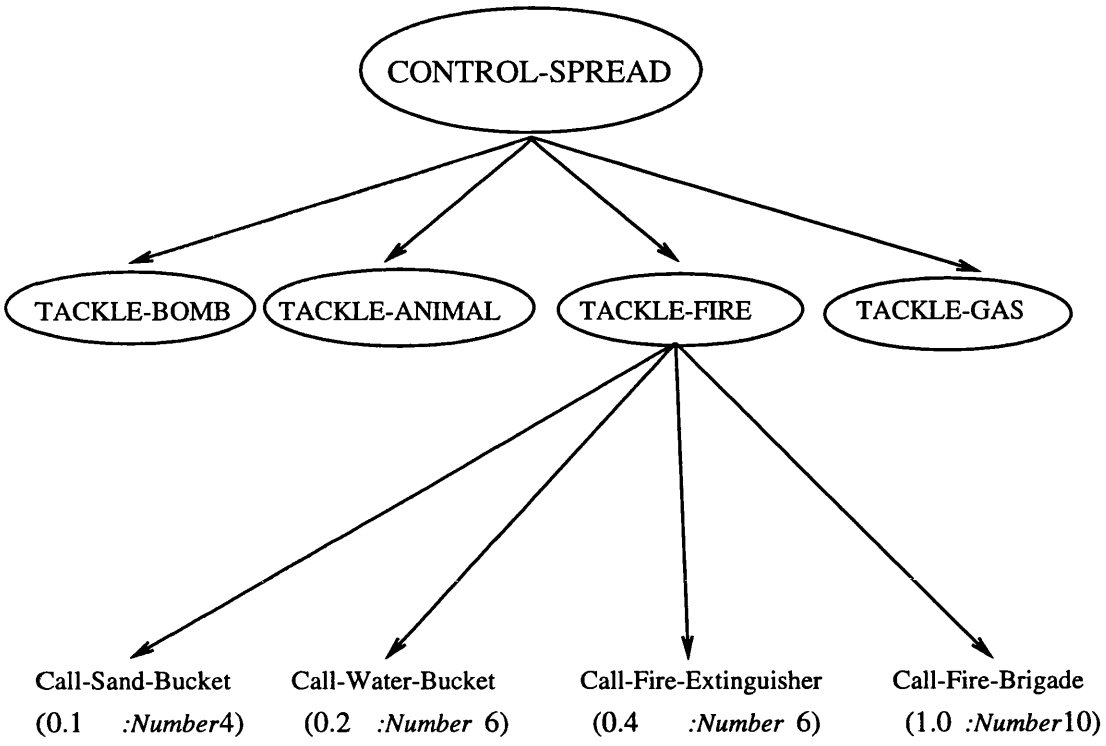


Figure 6.3: Hierarchy Tree for the Purpose "control-spread"

Corresponding to each action there should be a frame-like node consisting of the following information:

- the degree of effectiveness of the said action in accomplishing the generic action;
- the list of prerequisite actions necessary for initiation of that action;
- the parameters that are not inherited from the parent generic action;
- for each of these parameters, the problem feature that acts as the independent variable; and
- default outcome (if any).

Figure 6.4 shows such frames for the action "call-fire-brigade". It suggests that the *number* of fire-brigade engines (necessary for tackling the current outbreak of fire) should be computed from the "potential-degree-of-threat" of the current problem; the *where*-value should be inherited from the generic-action that activates this action; the *prerequisite* for using fire-brigade is to clear the path for its manoeuvring; the *default outcome* of this action is that it fails; and the *average-time* for its completion is 5 minutes.

action	call-fire-brigade	
	parameters	*values*
	number	*<potential-degree-of-threat>*
	where	*[inherit]*
	prerequisite	*<clear-path>*
	default-outcome	*"failed"*
	average-time	*5 min*

Figure 6.4: Frame for the Action "call-fire-brigade"

The knowledge that one stores should cover all the kinds of tasks that may occur in a plan. For example, this involves:

- determining the most suitable generic action for a given task;
- identifying the most suitable action(s) representing the selected generic action;
- fulfilling the prerequisites of the action(s);

  and
- estimating the parameters of the action(s).

Also, some additional knowledge is required for carrying out these steps efficiently. In the following section (6.3.3) we describe how they are carried out. The notion of the additional knowledge will be described during the explanation of these steps.

6.3.3. Additional Knowledge For Reasoning

Evidently, to reason with this action-related knowledge any system that uses it ought to have an underlying plan of tasks that should be carried out. In the cache-based system the cached knowledge is used for this purpose. In chapter 7 this issue will be discussed in detail. For our present discussion we assume that the system is working on an underlying plan.

First the method finds the relevant generic action(s) that should be performed for any task suggested in a plan currently being executed. Simple domain-dependent knowledge can be utilised to achieve this. We have used an additional parameter, *Index,* along with the task to identify the right generic action. Thus in our use, we invoke the task *evacuate* by specifying the necessary parameter, such as *(evacuate* :Index *'human)*; *(evacuate* :parameter *'plane)*,

depending upon what the plan wants to accomplish. Similarly, the task *control-spread* will be invoked as (*control-spread* :Index *'fire*) in order to shift the attention to the generic-action "tackle-fire". Simple conditional statements on the "Index" value can be used to make this taxonomic decision. In the absence of any associated "Index", the system needs to consider all the generic actions associated with the said task. The related "Index" values are set by the designer while specifying the plan: in our case, in the cells of the cache. We discuss these issues in section 7.4.2 while discussing activities involving the cached knowledge.

The next step in accomplishing each task is to determine (with the help of the selected generic action) the most suitable action that needs to be performed. This can be decided with the help of the desired degree of effectiveness suggested in the underlying plan. Thus when the suggested desired degree of effectiveness is x (say), where $0 < x \leq 1.0$, all the actions whose associated value of degree-of-effectiveness is greater than (or equal to) x may represent a meaningful choice for the relevant task. In the event that more than one action satisfies the criterion, one refers to some additional knowledge to accomplish the right selection. We store such knowledge in tables called *action-selection* tables. In figure 6.5 we show the Lisp implementation of such a table for the purpose of "passing-message", where all the actions associated with a particular criterion have equal degree-of-effectiveness for the task. For example, if the purpose is to pass a message to a person at a distance of 1000 metres, reference to this information indicates that the possible actions are: *telephone, email fax.* The method should look at this table in addition to searching the hierarchy tree given in figure 6.2 to determine the right action in a given situation. For each purpose we maintain such tables (if necessary).

```
(communicate  (distance)                ;; in metres
   ((<= distance 5) (talk))
   ((and (> distance 5) (<= distance 15))   (walkie-talkie talk run))
   ((and (> distance 15) (<= distance 50))    (walkie-talkie run))
   ((and (> distance 50) (<= distance 500))   (telephone ))
   ((and (> distance 500) (<= distance 5000)) (telephone email fax)))
```

Figure 6.5: Example of an Action-selection Table

In the absence of any other constraint the user may apply the "discrete selection" method to choose any one of them. The stored knowledge can further be used in conducting *tactical-type* (see section 5.5.2.1.2) interpolation for determining a new action (if necessary) from two actions that are not fully appropriate for the given situation.

However, on certain occasions a more judicious selection of the right action can be made by using constraints that may have been imposed on some of the actions. For example, in the situation above the following are valid constraints:

- do not talk, if the message should be treated as confidential.
- talk, only if the addressee is visible.
- do not run, if currently dealing with some other urgent task.
- prefer telephone to fax or email if the message is extremely urgent.

If any such constraints exist for an action, the information will be stored in corresponding action-frame. In figure 6.6 we show the frame for the action "use-microphone" where two fields "constraints" and "useful" have been used to represent this knowledge. This knowledge helps in achieving efficient decision-making with actions, as illustrated in section 5.5.2.2.

action	**use-microphone**	
	parameters	*values*
	number	*<location-of-the-event :size>*
	prerequisite	*<frame-message> <arrange-microphone>*
	default-outcome	*"failed"*
	average-time	*2 min*
	constraints	*"cause confusion"*
	useful	*(crowd-density "low")*

Figure 6.6: Frame for the Action "use-microphone"

6.3.4. Example: Use Of Action-Related Knowledge

For illustration let us assume that the current problem is:

there is a rather big fire outbreak at the front gate of the engineering building of the airport.

Also, suppose the system wants the following plan to be carried out (after retrieval from the cache):

- control the spread with degree-of-effectiveness 0.3

  and

- evacuate (human) with degree-of-effectiveness 0.5.

Our system uses the stored knowledge in the following way.

6.3.4.1. Determination of Action

Since the object of hindrance is *fire,* then with the help of the raw feature "fire", the system can determine the generic action *tackle-fire* and subsequently can retrieve from its stored knowledge the following information on actions (see figure 6.3): *call-sand-bucket (0.1 :number 4), call-water-bucket (0.2 :number 6), call-fire-extinguisher (0.4 :number 6), call-fire-brigade (1.0 :number 10).* The triple attached to each action refers to the maximum degree-of-effectiveness that can be achieved with that action; the parameter that determines the effectiveness; and the required value of the parameter for attaining the maximum degree-of-effectiveness. For example, since for the current problem the outbreak of fire has been described as *rather big,* the system calculates using fuzzy descriptors (see section 5.3.2.1) that the "degree-of-potential-threat" for the problem is 0.65. The system then determines from the cached knowledge the required degree-of-effectiveness for controlling the spread. Suppose in this situation it comes out to be 0.7 (in chapter 7 we explain how this is achieved at different levels of the cache). Since the desired degree-of-effectiveness is 0.7, the system determines that "fire-brigade" is to be called, as other alternatives do not match the required demand of degree-of-effectiveness.

Similarly, to decide upon actions for "evacuation of humans from the affected location" the system maintains (similar to the task "controlling the spread") the knowledge about a set of relevant actions: *use-microphone (1.0), send-rescue-personnel (0.9), send-airport-personnel (0.7), display-on-electronic-board (0.3), display-on-board (0.1).* Here too, the selected action is determined depending upon the value of the primary feature "potential-degree-of-threat" (PDT) for the current problem. Only when the PDT-value is sufficiently small will a good GOC want to display the message so that humans can distance themselves from the location. A GOC wants also to avoid the confusion caused among the affected persons by a sudden microphone announcement of a danger (this knowledge is retrieved from related *action-constraint table* discussed section 6.3.3) and therefore will restrict the use of a microphone only to cases when the situation is extremely grave. Otherwise, a GOC should decide to send security personnel to apply quiet persuasion to the persons in danger to evacuate.

6.3.4.2. Fulfilling the Prerequisite

Suppose the GOC decides to use a microphone to ask the people inside to evacuate the building. The system that models GOC behaviour refers to its knowledge-base to determine that the action *use-microphone* (see figure 6.6) has the following two prerequisites:

Framing the message: The task comprises suggesting the safest exit through which people can leave. The system may refer to some specific knowledge-base (or may ask the human user) in this regard. Here, the system will find that the rear door of the said building should be the safest, because of the knowledge that if the trouble is at the front then the rear-side is the safest.

Arranging Microphones: Arrange-microphone itself is a task having two related actions: *check-microphone* and *supply-microphone* with degree-of-efficiencies 1.0 and 0.8 respectively. The system therefore first tries the action *check-microphone* which translates into checking for data indicating the presence of an internal microphone in the building. If the answer is negative, then this action fails. It therefore tries an alternative action *supply-microphone* which translates into sending someone to use a loud-hailer from outside the building.

Consequently, the task of "evacuating the affected location" can have as many as three steps:

- Framing the evacuation message (E), i.e. "evacuate through the rear door";
- Asking the security department to send a loud-hailer to the engineering building;
  and
- Passing the message E to the security department.

6.3.4.3. Estimation of Parameters

The list of variables associated with each action parameter helps in identifying the right values of the parameter. For example, for deciding the value of the parameter *where* for the action "call-fire-brigade" (see figure 6.4) the system retrieves the information that the necessary problem feature is "*location of the event*". Hence estimating the value for the "*where*" parameter becomes straightforward. Further, to estimate the parameter *number* the system refers to the problem feature "potential-degree-of-threat" and computes the desired number of items. For example, for estimating the number of fire-extinguishers required to achieve a certain effectiveness (*e,* say) the system retrieves the pair information (1.0 :number 10) suggesting that 10 fire-engines are needed to accomplish a degree-of-effectiveness 1.0. This pair of values can then be subjected to interpolation where, evidently, there is also an implicit interpolating point (0 :number 0) because of the general fact that 0 effectiveness is achieved if one uses no relevant tools. This value suggests that the users of the system's output need to employ "*fire-extinguisher*" to control the spread of this fire. In order to determine the desired number the system again applies interpolation on the pair of values (0, 0) and (1.0, 10) to find that 7 fire extinguishers are appropriate for the purpose. This is the message that is then passed to the user.

In certain situations estimation of parameters may need further computations. For example, to decide how many microphones are necessary to conduct some announcement the system retrieves from the knowledge-base (see figure 6.6) that not only does this depend upon the feature "location-of-the-event", but also that its size is the important underlying factor. The method retrieves the interpolation-related information from the related hierarchy tree and computes the interpolated result for the size of the affected location.

6.4. CONCLUDING REMARKS

In this chapter, we have described the nature of knowledge and its efficient representation for rapid decision making in determining suitable actions and their parameters in a given context. The knowledge can be exploited in the cache-based reasoning environment where the cache-cells contain (or point to) the required plans. Reasoning tactics associated with different levels of the cache can then make use of these concepts for deriving solutions in a given problem situation. We discuss reasoning with the cached information and the overall control of the reasoning process in chapter 7.

Chapter 7.

BASICS OF THE CACHE SCHEME: REASONING AND PROBLEM SOLVING

7.1. INTRODUCTION

Although the technique of interpolation provides some support for computations that are intended to be fast enough to respond to stipulated temporal constraints, and the cache acts as a means to access the required information for relevant problem-solving methods quickly, the effectiveness of the interpolation within the cache-based framework depends largely on the underlying reasoning scheme which controls:

- monitoring the remaining time at any stage of the calculation;
- decisions regarding the most suitable action that is to be taken at any juncture;
- retrieval of appropriate information from the cache;
- identifying appropriate interpolating variables;
- setting suitable (commensurate with the available time) interpolation flags;
- retrieval of required domain knowledge to carry out the interpolation

  and finally
- providing the solutions to the user.

This chapter first describes the outline of a reasoning scheme that controls the operation of the cache-based system while taking account of any imposed time bounds. We then elaborate each major step in detail, and address various subtleties that need to be considered while dealing with the cache and applying interpolation to its contents. We also identify different complications that may crop up while reasoning with the cache-based system and propose possible ways of averting them. In our initial discussions we adhere to the simple descriptions of the cache and its contents given in chapter 3. However, with the emergence of more complicated problems within our domains of discourse and development of tactics for reasoning and time management for dealing with these situations, we recognise certain additional requirements for a cache-based system. Although these additional requirements do not alter the basic cache structure, responding to them can increase the effectiveness of the system in handling time-dependent problem solving. We discuss these issues below.

In our discussions on reasoning with the cache we use the following structural specifications and notation:

- The cache is of dimension **1+k**.

- While the first dimension refers to a given time-limit, the remaining **k** dimensions are domain-dependent indices for problem specification, and are described as *column dimensions*.

- Along the first dimension of the cache, there are **n** levels, corresponding to the different characteristic times for the different methods that may be employable for solving a problem. We call these times T_1, T_2 .. T_n, with increasing size as the subscript increases.

- Each column of the cache refers to some significant property, characteristic of the problems for which the cache holds information relevant to their solutions. We refer to the property represented by the i-th column dimension of the cache as P_i.

- Each cell of the cache is referred to by a **1+k** tuple (j, v1, v2, .. vk), where **j** denotes the level and vi, $1 \leq i \leq k$, denotes the column number along the i-th column dimension of the cache. We refer to the contents of the cells as $C_{j,v1,v2,\ldots vk}$.

7.2. BASIC REASONING TECHNIQUE

The system is provided with two inputs: the problem description and the temporal bound. Upon receiving the input, the system begins the execution phase, where it starts reasoning about the problem in order to derive a solution. In an outline of the essentials, reasoning with the cache-based system consists of four major steps:

1. *Identification of the right cache-cell:* with the help of the imposed time bound, the system decides the level of the cache to concentrate on. Level x is so chosen that $T_{x-1} < T < T_x$, $1 \leq x \leq n$, where T_0 is the minimum time that has to be allowed to derive any solution at all. If the allowed time, T, is smaller than T_0, the system will refuse to provide any answer and report that insufficient time is available. This is because the cache needs some overhead time to analyse a situation and access the cache. T_0 stands for this minimum temporal requirement.[1] The system also determines, by isolating the key features of the current problem, the column (along each column dimension) that describes the problem best. These indices show where to access the cache, as each cache-cell can be identified uniquely by the level and a single column from each column dimension.

[1] For practical applications one may assume that the time bound T will be greater than T_0.

2. *Retrieval from the cache:* accessing the prescribed cell to retrieve information required for a solution (explicit, or an indication of how a solution can be found) that is stored there. The nature of the contents, in particular the primary knowledge representation used, depends upon the level of the cache. This retrieved contents are then to be used in determining a solution of the current problem.

3. *Adaptation to the new situation:* if the conditions of the retrieved solution exactly match the current problem situation, the retrieved solution can be applied without change. But in a complicated domain, such an exact match is unlikely to happen on typical occasions. Consequently, the system needs to modify the cached answers suitably in order to generate a solution that can resolve the current problem. We call any such modification technique "adaptation" because of its similarity in purpose with "case adaptation"; one of the fundamental steps in case-based reasoning [Kolodner 92a], by which a past case is modified so that its solution can be used in a current context. Any such modification process, in general, requires some domain knowledge and therefore should have the necessary capabilities to identify and retrieve relevant information from the domain knowledge-base.

We advocate the use of "knowledge interpolation" for the purpose of adaptation. (In fact, this is our original motivation behind developing knowledge interpolation). However, the approach and purpose of the adaptation both vary with the level of the cache. When the activity is at the shallowest level (the level that corresponds to extreme time-pressed situations), a system is still expected to suggest a default solution (or some simple extension of it, e.g. relying on interpolation, that still respects the imposed time limit). Consequently, the purpose of the adaptation at this level is to estimate the appropriate values for the parameters of the suggested actions. On the other hand, when available time is not so tightly limited, a system should use more efficient reasoning schemes (e.g. case-based reasoning, evaluation of sets of rules) and therefore apply more elaborate interpolation techniques which modify the plans (see section 5.5.2.2) in order for the quality of the solution to improve.

4. *Carrying out actions:* the solution thus generated is transformed into a set of actions, to be carried out sequentially in order to solve the current problem. For a default solution, the actions are to be carried out unconditionally in order to resolve a problem. But in a more reasoned approach the actions can be of two types - either some computations that the system can execute or (in many real domains) some instruction to the human user. In the latter case the system receives the result of the action subsequently, as further input. The system may then change the course of the next actions depending upon the outcome of previous ones, as in plan-level interpolation (see section 5.5.2.2).

Once the actions are carried out and their results are obtained, the system needs to combine them to form the ultimate solution. There should be a control strategy to hold these basic steps together, to achieve the desired objective. We now discuss the requirements that we consider essential for any such control program while reasoning with a cache-based system.

7.2.1. Properties Of The Control Program

The primary role of the control program is to ensure that a solution is produced for the current problem within the allotted time. Although the cached solutions are so designed that when translated into actions they ought to be able to find an answer to the current problem within the available time, for practical use in a real domain this is not guaranteed to happen in all situations. The expected performance from an action may not be achieved in some situations, as hindrances may appear through different extraneous factors. For example, with respect to the AGC domain, the GOC has the knowledge that the average response time of the baggage loaders' team of some well-organised airline (XYZ, say) is 2 minutes. Cached solutions will incorporate or have pointers to this knowledge - so that in an urgent situation when some other airline is unlikely to complete the loading of baggage into a departing aircraft in time and approaches the GOC for an immediate solution, the GOC's answer may well be: *ask for help from XYZ Airlines* (in order to maintain timely departure). But on some particular day this solution may not work, as XYZ cannot achieve its expected performance due to an unusual inconvenience (e.g. sickness, agitation, national patriotic holiday of the country XYZ). Consequently, any attempt on the part of the GOC to use this solution proves futile. Another approach is then needed.

Therefore, the task of the cache-based architecture does not end with the prescription of some solution and carrying out the basic steps in the right order. The control program should also monitor the elapsed time periodically - so that if during the execution of a task, any threat of exceeding the time limit occurs, the system can abandon the action concerned and can switch to appropriate alternative steps for the same task or assume some default result for the task concerned and proceed. For example, in an airport in the event of a fire alarm a GOC may decide to have a quick check of the intensity of the fire (or of whether there is a real fire at all!). Any subsequent tasks (e.g. calling an appropriate agency to control the fire) can be deferred until a report comes. However, if no report is received within a specific time (2 minutes, say) the GOC should assume that the fire is serious and proceed accordingly. The same notion should be implemented within the control program. During the course of the periodic inspection of the available computing time, the system is required to estimate the time needed to complete the task still remaining. Should the system, at some instant, find the estimated time to exceed the imposed time bound, the control should abandon the current execution, and initiate an alternative computation which is expected to end within the remaining time. We suggest the following policy in this regard:

- if the current action is just a precautionary action (e.g. in the event of fire alarm, a GOC sending a person to check if there is a real fire), the system may assume a default outcome of the action. Thus in the above case if the person does not return within a specific short time period, the GOC can assume that the fire is real and proceed accordingly.

- if the action concerned is an important step of the solving procedure, then if there is a similar alternative action and there is sufficient time for the execution of the alternative

action, the system may initiate it so that the original plan can still be followed. But if there is no alternative action available or if the remaining time is insufficient to carry out the alternative action, the system should then abandon the execution of the current plan and switch to a new plan which can be retrieved by accessing an appropriate level (shallower than the current level of activity) of the cache where the reasoning is indicated to be likely to occupy no more than the time still remaining. Thus (refer to the example given at the beginning of this section), if the GOC prescribes "*asking help from XYZ Airlines*" as the solution to a loading problem and it appears that XYZ is currently too busy to respond to the request, the GOC may then check how much time is left. If the remaining time is sufficient to call help from another airline (an alternative to XYZ), then that is the next action that the GOC suggests. But if the GOC finds the available time to be too small for loading the baggage (even with the help from a second airline), the GOC should then abandon the plan of getting the baggage loaded and therefore refer to a shallower-level (i.e. associated with smaller temporal requirement) cell. A possible solution for the current solution, found at such a level (e.g. a default level), may then be: *defer the flight.*

We have evolved the following to be the basic algorithm that controls reasoning with a cache-based system.

7.2.2. An Elementary Control Program

We assume that the cache has **n** levels and **k** column dimensions. To help in restricting the imposed temporal bound, we propose using two counters for time: *solving time* (T_s) and *action time* (T_a). The first one is for the total elapsed time after the start of the plan and the other is for any plan-action that is being carried out at any particular time. The solving time is directly related to the time bound provided by the user; this is the time that the system adopts as a target for finishing the entire computation. In order to maintain the imposed temporal bound, the system fixes a prescribed maximum time limit for every action that is to be carried out as a part of the entire problem-solving exercise. The time counter T_a is reset at the initiation of each action, and subsequently is compared with these limits by reference to the temporal knowledge (attached to the relevant row of the cache) that stores information relevant to the expected completion time for each action. The time counter T_a is associated with a flag F_a which indicates (e.g. see the algorithm in Figure 7.1) whether the time allotted for the completion of the said action has expired. If no solution is obtained within the allotted time, the method resorts to one of the two following options:

- it may look (in the action-related knowledge-base; see section 6.3.1) for some default outcome ("dummy solution" in the presentation of the algorithm) that the action may recommend, and the method then proceeds with the reasoning scheme by considering the next action on the list; or
- the method may abandon the currently executed scheme and start some fresh line of reasoning by accessing an appropriate cache level determined by the time still left.

The method first resolves the retrieved information into a sequence of generic actions. These are the high-level actions describing the basic steps that are to be carried out (plan) for solving a current problem. Subsequently, the method considers sequentially each generic action and its application. By application we mean deciding the best way of fulfilling the task, estimating the parameters (using interpolation), considering any requirements that need to be satisfied beforehand to start the action, [2] and then initiating the action. If the action is to be carried out by some external agent and the outcome is needed to decide the course of the subsequent actions, the system waits for news of the outcome of the said action. Evidently, the system needs to have a knowledge-base of the allowed generic actions for the domain concerned and the possible ways of accomplishing the objective. In figure 7.1, we present the overall structure of the control algorithm for the cache-based architecture.

7.2.3. Further Requirements For The Control Program

Having laid out the overall reasoning scheme for the cache-based system and the principal steps towards achieving the goal, we now change focus to developing methods for carrying out each of the individual steps of the entire reasoning exercise. A close examination of the reasoning steps that are stated above reveals certain difficulties that need to be addressed adequately if such a control algorithm is to perform to match its expectations. For example,

- what happens when the problem specification is not clear enough (due to use of fuzzy descriptors or relational modifiers, say) to identify a single column unambiguously from all the column dimensions of the cache?
- which cell(s) of the cache are to be considered for retrieval, if more than one cache-cell appears to be relevant in a given situation? [3]
- is it necessary to fill in all the cache-cells at the beginning? What happens if the cache-cell that has been identified as a place from which information is to be retrieved does not have any contents at all? (An obvious consequence of any answers to these questions is identification of what minimum contents a cache-cell should have at the outset).
- what should be the contents at each level of the cache? For each level of the cache, how do we identify the relevant quantities that are required to carry out interpolation?
- which interpolation tactic is to be applied at a particular juncture to derive a solution, for the problem being dealt with, from the cached information?
- how should the total available time be distributed among different actions?

In the following sections we study the above aspects in detail and identify their different ramifications in a domain. Also, we suggest some ways to handle these issues.

---

[2] For example, in order to call fire brigades to tackle a fire, it is necessary to ensure sufficient clear space for their expected operations to occur freely.

[3] We see later that this pertains to situations when more than one column is selected along some of the column dimensions.

```
Step1: {Input}.
      Input the problem description and the time-limit (T) for solving the problem
Step2: {Start clock and set flags}.
      Set solving time (T_s) = T; Start Clock

Step3: {Preparation}
      Identify/compute for the current problem the index-set {P_v1, P_v2, .. P_vk}.
            {the values of the "k" column dimensions}
      Choose the level x such that T_x-1 < T_s < T_x, 1 ≤ x ≤ n.

Step4: {Access the cache}.
      Retrieve information from cache-cell C_x,v1,v2,..vk.
            {the cell determined by the above indices}
      Resolve the information into a sequence (G) of generic actions {A_1, .. A_m}

Step5: {Solving procedure}.
      REPEAT
        Consider next A_i from G
        When no such action exists Go to Step 6
        Set action time (T_a) = t_a; {based on the action and remaining time}
        Set action flag (F_a) = 0
        Apply action A_i;
   {Start monitoring}.
        WHILE F_a = 0 and   elapsed time < T_a
              {action A_i did not exceed its temporal bound}
           Continue with action A_i
           When  a solution is found  Set F_a = 1;
   {End monitoring}.
        IF  elapsed time > T_a
        THEN
             If some dummy solution exists for action A_i
             Then
                 Retrieve the dummy solution
                 STORE the solution;
             Else          {Change Solving Tactic}.
                 Calculate residual time (T_r) = T_s - elapsed time
                 Set T_s = T_r
                 Go to step 3;
         ELSE
              STORE the solution;
        {REPEAT-loop now carries out the next action}
      UNTIL no more actions remain to be carried out;

Step6 : Output the solution.
```

Figure 7.1: Algorithm for Reasoning with the Cache Architecture

7.3. SELECTION OF CACHE-CELL FOR RETRIEVAL

In order to retrieve a solution from the cache, we need to identify the right cell - which implies fixing an appropriate level of the cache and also finding column headers along each column dimension of the cache. The time bounds associated with each level of the cache help in fixing the right cache level (as in the algorithm of figure 7.1) commensurate with the imposed temporal bound that is supplied along with the problem description. The column headers identified along different cache dimensions provide the remaining indices for retrieval of solutions. But effective retrieval from the cache requires us to invent tactics to overcome two notable obstacles:

1. If only one column is selected along each dimension, then the appropriate cache-cell can easily be identified as the combination of the indices for the different dimensions that refers to a single cache-cell (assuming that the appropriate level for retrieval has already been decided on the basis of the temporal limitation). But we have observed earlier (in section 6.2.1) that under many circumstances the system may recover two columns from the same column dimension, because of their proximity to the current problem value. This results in identifying a number of cells as possible candidates wherefrom retrieval should be made. One therefore needs to have policies to select only the most appropriate cells from among the competing items.

2. What happens when the cache-cell chosen for retrieval is found to have null contents? A cache having several dimensions is normally very large. Evidently, to fill in all the cache-cells may require quite large amounts of information about solving problems under different reasoning paradigms, which even a domain expert may not the patience to provide when the cache is being initialised. For example, to use the cache for solving problems with case-based reasoning requires past cases covering all sorts of problems. It may not be possible for a domain expert to recollect prototype cases covering the full variety of the problem field. In similar vein, to spell out rules for all sorts of problems is not straightforward, as the rule-set grows empirically with experience. Consequently, it may not be possible for a system designer to fill in all the cells with appropriate solutions, when the cache is set up. Hence it is possible that the retrieval process may identify a cell to be the most appropriate for a current problem only to discover later that it has a null content. The system designer needs to have an answer for this difficulty as well.

7.3.1. Choice Of Cells When More Than One Are Contending

The column-selection method described in section 6.2.1 can choose at most two columns from each of the k column dimensions. Thus, in theory, as many as 2^k cells may be selected for solving a current problem - resulting in an exponential growth in the number of contending cells as the number of column dimensions increases. But to deal with too many cells may

prove to be cumbersome as it does not fit into the linear interpolation schemes developed in chapter 5, where only two pairs of values for dependent and independent variables are needed. Hence we recommend selection of only two cells out of the contenders (when selection is possible from more than two cells). Certainly, the key concept that ought to be looked for, here, is significance of the cells for the current problem. We therefore propose the use of some weighting scheme to ascertain weights for all the competing cells. In our cache scheme, the distances of the chosen column headers from the problem values may act as the natural yardstick to measure the relevance of any particular cell in a given context. Once the weight for each cell in question is determined, the system can apply a suitable criterion for the final selection of two most appropriate cells, contents of which will be forwarded for interpolation. We have developed the following weighting scheme to assign weights to each of the contending cells, by suitably combining the distances of the columns (whose intersection designates the particular cell) from relevant problem values. An approach for selecting two cells is provided in the next subsection.

7.3.1.1. Weighting Scheme

For each of the k dimensions we assign a weight 1, which will either be given to a single column (when only one column is chosen) or divided between two columns when the header values for both of them are reasonably close to the problem value. If the weight is to be computed on the basis of distance, then the column header having a higher distance from the problem value should receive a lower weight. We assign weights according to the simplest (linear) scheme, as in general there is no justification for a more complicated approach: if c_1 and c_2 are the column headers with distances d_1 and d_2, respectively, from the problem value, then the weight for c_1 is $d_2/ (d_1 + d_2)$ and the weight for c_2 is $d_1/ (d_1 + d_2)$.

On the other hand, if weights are calculated on the basis of overlaps, the column having greater overlap with the problem value should receive a higher weight. Hence, for two columns c_1 and c_2, having overlaps o_1 and o_2 respectively, with the given problem value, the weights will be $o_1/ (o_1 + o_2)$ and $o_2/ (o_1 + o_2)$.

The weight of a particular cell is calculated on the basis of the weights of the columns that serve as indices for the cell concerned. We use a simple linear function of weights of each column-index for this purpose. Thus, for a cell having column indices c_1, ... c_k, where c_j is the chosen column from dimension j, the weight is $(w_1 + .. + w_k)/k$, where w_j is the weight of the column c_j. For example, suppose we have a simple design with two dimensions (A and B, say) and two columns have been chosen from each of them (a_1 and a_2 from dimension A and b_1, b_2 from dimension B, say). If the weights of a_1 and a_2 are 0.7 and 0.3 respectively and the weights for b_1 and b_2 are 0.65 and 0.35, then the weights of the 4 cells can be computed using the above scheme. Figure 7.2 shows the weight assignment among different cells.

A/ B	0.65	0.35
0.7	0.675	0.525
0.3	0.475	0.325

Figure 7.2: Distribution of Weights for a 2-dimensional Cache

Thus the maximum weight that a cell may have is 1, and this happens when a single column is chosen from all the dimensions. The weight, however, decreases as the number of dimensions from which the above methods lead to selection of pairs of columns are picked increases.

7.3.1.2. Criterion for Selection

In the event that a single column is chosen from each dimension of the cache, the choice of the right cell is obvious. The selected columns provide the indices for the best cell for the purposes of retrieval. But the system needs to have policies for identifying the right cells from which to retrieve, when more than one cell is found among the contenders. We have evolved the following scheme for this kind of situation:

1. When the available time is too pressing for any extended selection process to be advisable, the most sensible choice for the system is to take the cell that has the maximum weight. This can be achieved easily by selecting the column with the highest weight, along each dimension. Conflict occurs when two columns of the same dimension have the same weight. We suggest choosing one of them arbitrarily to resolve this conflict.

2. When the system can afford time for a higher quality of reasoning, we can demand that it should look to retrieve the contents of more than one cell, and construct a solution by applying appropriate interpolation techniques, on all the retrieved material.

Because of the way the cache is designed, two consecutive cells, along any dimension, will have all their other indices in common. Hence they represent rather similar problems and are the most likely to have easily-interpolatable solutions. On the other hand, cells that differ in a large number of indices represent two quite different situations and therefore are not likely to contain materials or solutions directly compatible with each other.

Under this observation, considering too many cells implies:

- reasoning with heterogeneous solutions, which may not be feasible to tackle by an interpolation-based approach.
- cells with negligible weights draw undue attention for little reward;
- a lot of time is spent in reasoning, with diminishing marginal utility.

By contrast, interpolation with just two cells is fast, easy to comprehend and when these two cells are contiguous the interpolation is between two quite comparable situations and therefore is meaningful. Hence we suggest considering only two cells for the purpose of interpolation. Logically, one should select the two cells with highest weights for interpolation as they are the most relevant ones for the current problem. However, the question arises of whether the two highest-weighted cells contain comparable solutions. It can be proved that under the above selection and weighting schemes for columns, the topmost two cells are adjacent in the cache, and hence are interpolatable.

Proof: Let us consider a cache having k column dimensions as a k-dimensional vector. When k is 1, the cache is a 1-dimensional vector, and the selected columns are consecutive by the selection scheme.

When k is 2, there is a cell at the intersection of the columns from each dimension. Let us assume that a_1, b_1 are the weights of the two consecutive columns from dimension 1, and a_2, b_2 are the weights of two consecutive columns in dimension 2, where, a_j and b_j are the two highest weights in dimension j and $a_j > b_j$. The 4 cells at the intersection of the columns will have weights $w_{11} = (a_1+a_2)/2$, $w_{12} = (a_1+b_2)/2$, $w_{21} = (a_2+b_1)/2$ and $w_{22} = (a_2+b_2)/2$. Call the 4 cells c_{11}, c_{12}, c_{21}, c_{22} respectively. Certainly, the cell c_{11} has the highest weight, and c_{22} has the lowest weight among these cells. Hence the second highest cell is either c_{12} (if $w_{12} > w_{21}$) or c_{21} (if $w_{21} > w_{12}$). Evidently, in either case it is contiguous to the highest weight cell c_{11}.

We now use the method of **induction** to prove this assertion for any arbitrary number of column dimensions. We have already shown that our assertion holds good for 1 and 2. So what remains to be proved that: if we assume that the assertion is true for caches when the number of column dimensions (k) takes a value in the integer set 1 to n ($n \geq 2$), then it is true when $k = n+1$.

Consider a cache of n+1 column dimensions to be an (n+1)-dimensional vector space and call it **S.** Consider the subspace ($\mathbf{S_n}$) of the **S** formed with the first n column dimensions (of **S**) and focus on the two cells corresponding to the two highest weights in $\mathbf{S_n}$. As per our assumption, these two cells will be contiguous. Call them C_1 and C_2 with weights W_1 and W_2 respectively. Without loss of generality we may assume that $W_1 > W_2$.

In order to determine the two highest-weight cells in **S** , one needs to consider the projection of the two cells (C_1 and C_2) on the dimension (n+1) (which represents two lines) and intersect them with the two highest-weight columns (call them e_1 and e_2) of the (n+1)-th dimension (which are contiguous by our process of selection). Without loss of generality, let us assume that weight of e_1 (w_1) is greater than the weight of e_2 (w_2). The two highest-weight cells in the dimension (n+1) will be selected on the basis of these weights. Certainly, the cell at the intersection of the C_1 line and e_1 will be the one having highest weight. The cell having second-highest weight will be either:

- the one at the intersection of C_1 and e_2 (when $n^* W_1 + w_2 \geq n^* W_2 + w_1$)
  or
- the one at the intersection of C_2 and e_1 (when $n^* W_2 + w_1 \geq n^* W_1 + w_2$)

The contiguity between C_1 and C_2 and also between e_1 and e_2 implies that in either case the two highest-weight cells will be contiguous, and this completes the proof.

However, it is not always necessary for the system to select two columns from a dimension. When the demand of time is too pressing, the system may restrict its activities within a single column by choosing the column having a higher weight (in the event of equal weights of the columns one can be selected arbitrarily). Use of a single column evidently makes subsequent computations less time-consuming and therefore is more likely to meet the temporal stipulation. Evidently, the price is paid in terms of quality of the resulting solution.

Once the desired number (one or two) of cells is selected, the control process shifts attention to retrieving the information stored in the cells concerned in order to design solutions for the current problem. In this context, the obvious difficulty that we see a system facing is: what happens if the selected cell is void? As mentioned earlier, this can well happen in a cache-based system, particularly in the developing stage, when the designer does not have sufficient information to fill in all the cache-cells. For example,

- an expert may not be able to frame appropriate rules for some cells of the cache where the relevant knowledge is stored in the form of rules;
- it is possible that no cases appropriate for storage in some of the cells can be recollected where reasoning is based on past cases.

In the following section we describe some suitable heuristics to take care of these eventualities.

7.3.2. Retrieval When Selected Cells Have Null Contents

We describe two different sets of heuristics depending upon the number of cells that have been selected for retrieval.

7.3.2.1. Retrieval From a Single Cell

In situations when the system selects a single cell for retrieval, columns from all the dimensions that describe the cell concerned match exactly with the problem. Consequently, the cell receives a weight 1.0, and no information is available regarding the relevance of other cells as far as the current problem is concerned. Naturally, it is not reasonable in time-bounded situations for a system then to check other cells in that level of the cache. Instead, it is worthwhile to go one level up in the cache (i.e. in the direction of a lower-quality result but one that is computationally less expensive in time) and try to retrieve information from the cell at that level with the same column indices. Thus if the cell concerned (which has a null content) is $C_{j, i1, i2, .. ik}$, then the next cell from which retrieval should be tried is $C_{j-1, i1, i2, .. ik}$. The same policy is applied recursively, if the content of the new cell is null as well, until the search is in the "default" level of the cache, where ready-to-use solutions are stored for problems of all different characteristics.

Application of this heuristic imposes one essential requirement for the cache to start functioning: the contents of the default level of the cache need to be complete if all the cells in that level are able to provide solutions readily, when accessed. But since the default level corresponds to situations demanding almost immediate solutions, such a requirement seems to be justified - because if any cell in this level is empty and the system selects this cell for retrieval of information, normally there will hardly be any time to check the relevance of other cells for the problem concerned. Hence, from the point of view of efficiency in time-boundedness too this requirement seems both meaningful and essential.

7.3.2.2. Retrieval From Two Cells

When the system selects two cells for retrieval, heuristics of many forms can be developed depending upon the particular situation. For illustration, call the two cells A and B; and let us assume without loss of generality that the weight of the cell A (W_A) is greater than the weight of B (W_B). Here, we may have the following possibilities:

Case 1: Only cell B is null.

In this case, we suggest one of the following alternatives:

a) the process may ignore the cell B completely and rely for interpolation on the contents of cell A alone; and proceed as it would do if the cell A had been the unique selection. However, since the weight of cell A is less than unity, the solution thus generated will not be as well suited to the situation as it would have been for a situation when cell A had been the unique choice.

b) As described in the situation for a single cell, the process may decide to check at a shallower level of the cache for the cells corresponding to cell A and B in the level concerned. This search can be continued recursively until a level is reached where both the cells are able to provide some information. The system then may retrieve the information from the two cells and subject them to an interpolation method that fits the reasoning scheme associated with the level from where retrieval has been made.

   Here too, the quality of the solution will be inferior to what is available when both A and B are able to provide some information. While devising the control scheme a system designer should experiment with both the heuristics to arrive at a specific policy. For example, one may consider introducing some threshold (whose value is to be determined through experiments) such that option (a) will be applied only if W_A is greater than the threshold value and option (b) will be followed otherwise. In chapter 8 we mention experimental results with different options.

We have considered a third alternative, where one may discard the cell B and choose a cell (C, say) that corresponds to the third-highest weight (after A and B), and apply interpolation on the contents of cells A and C. However, there is no guarantee that the cell C will be contiguous to the cell A in the cache. Consequently, there may be discrepancies between cell A and C with respect to more than one feature, and therefore an interpolation scheme that involves more than one independent variable will be required, to formulate a solution. In our current research we are not considering such complicated interpolation methods and therefore do not examine this alternative further in the present thesis.

Case 2: Only cell A is null.

As the weight of cell B is not expected to be high, we do not recommend formulating any solution on the basis of this cell alone. Instead, we suggest that one should shift attention to a shallower level of the cache, as in the option (b) above, until a level is reached where both the cells corresponding to A and B are able to provide some information. The same heuristics should apply equally when both A and B are empty in the current level of the cache.

An alternative to the above suggestion may be finding a non-empty cell (A', say) with the same column headers as cell A, but at a shallower level. One may then consider interpolation between the contents of cells A' and B. However, conducting such an interpolation is not

straightforward, because of the different representational schemes adopted in different cache levels, or because of possible differences of contexts in which some terms are understood at different levels. Only in some special situations (such as when both levels are associated with rules, but with different search horizons) may a system designer conceive carrying out such an interpolation. We do not have any general suggestions (except caution) in this regard.

7.3.3. Algorithm For Cell Selection

In this section, we present an algorithm that generates iteratively the indices of the cells that are most relevant for a current problem. It considers columns from each dimension and builds a partial list of indices. At each stage of iteration the algorithm maintains at most two partial lists of indices and thereby saves time on comparing weights of all the possible cells. Figure 7.3 provides the algorithm for cell selection, which assumes that the problem description and a time-bound (T), within which a solution is required, have already been stated and the values of the k parameters (representing the column dimensions) have already been computed. The algorithm tries to generate a list of k numbers (v_1, v_2, .. v_k), where v_i, $1 \leq i \leq k$ denotes the most relevant column for the i-th column dimension of the cache, representing the current problem (with respect to the relevant parameter) most closely. The list is generated by considering each column dimension 1 to k successively, through appending the value v_i at the i-th stage to the partial-list that has been generated up to and including the (i-1)-th stage. We use a variable *Index* to hold this list. *Index,* however, may contain two such lists in order to facilitate considering two cells. At each stage *Index* records the two highest-weighted partial-lists generated so far. This is then modified at each stage by considering the weights of the updated partial-lists. Two variables *Weight1* and *Weight2* are used to record the weights of the generated partial-lists. The output of the algorithm is the list of triples consisting of *Index, Weight1* and *Weight2,* where *Weight1* > *Weight2* and *Weight2* is 0 if a single list of column indices can be identified unambiguously. It also outputs information identifying L, the level of the cache that is to be concentrated upon. We use the term *wt-function* to denote the function that modifies the cumulative weight of the cell using successively the weight of each column. In our practice we have set the wt-function to be the *arithmetic mean* (as described in section 7.3.1.1). One may choose one's own *wt-function,* to respond best to the requirements of the domain.

Upon recognising the right cells, the scheme should retrieve the information stored therein, as a preliminary to carrying out adaptation of the stored information to suit a current situation. In the following section we discuss related issues in some detail.

```
Step 1: Determine the preferred level (L) of the cache from imposed temporal bound T
Step 2: Set Index = Null list; Set Weight1, Weight2 = 0 {initialisation}
Step 3: FOR i = 1, k DO  {k is no. of column dimensions}
        Find the nearest column(s), along dimension i, for the current problem
        IF only one column is chosen, {Call the column i1 and its weight w1}
        THEN If Index contains a single list I1
             Then
               Append the list I1 with i1
               Set Weight1 = wt-function(Weight1, w1);
             Else  {i.e. when Index contains two lists I1 and I2}
               Set Index = list of I1 appended with i1 and I2 appended with i1
               Set Weight1 =  wt-function(Weight1, w1)
               Set Weight2 =  wt-function(Weight2, w1);
         ELSE {i.e. two columns selected from dimension i}
             Call the two columns i1 and i2, and their respective weights w1 and w2;
             If Index contains a  a single list I1
             Then
                Set Index = list of I1 appended with i1 and I1 appended with i2
                Set Weight2 =  wt-function(Weight1, w2)
                Set Weight1 =  wt-function(Weight1, w1);
             Else {i.e. Index contains two lists I1 and I2, say}
                Set Temp1 =  wt-function(Weight1, w2)
                Set Temp2 =  wt-function(Weight2, w2);
                If Temp1 > Temp2
                Then
                   Set Index = list of I1 appended with i1 and I1 appended with i2
                   Set Weight1 = wt-function(Weight1, w1); Set Weight2 = Temp1;
                Else
                   Set Index = list of I1 appended with i1 and I2 appended with i1
                   Set Weight1 = wt-function(Weight1, w1); Set Weight2 = Temp2;
      {End of FOR-loop}

Step 4: Check the cell(s) at the level L of the cache where
             column indices are prescribed by Index;
        IF Index contains a single list
        THEN If the cell is non-null
             Then
                RETURN L, Index and Weight1;
             Else
                REPEAT
                   Set L = L -1
                UNTIL the cell formed by L and Index is non-null
                RETURN L, Index and Weight1;
        ELSE {Index contains two lists I1 and I2}
        REPEAT
          If both the cells have non-null contents
          Then
              RETURN L, I1 and I2 with Weight1 and Weight2;
          Else Consider the non-null cell (I) and its weight (Weight)
               If Weight > TH      {TH is a pre-assigned threshold}
               Then
                    RETURN L, I and Weight;
               Else
                    Set L = L-1;   {go one level up}
        UNTIL L = 1 or a non-null cell is returned.
```

Figure 7.3: Algorithm for Cell Selection

7.4. ACTIVITIES CONCERNING CACHED ENTRIES

Two important considerations on the part of the system designer in this regard are:

- what should be stored in different cells of the cache. (This has been called "reasoning-related knowledge" in section 6.1).

  and

- designing policies to adapt the stored information in a current situation that calls for some adaptation.

Evidently, these two tasks are not quite independent for a system designer, because any information that the designer intends to store in a cell of the cache should have a clear purpose - i.e. a piece of information will be considered for storing in the cache only if it is required in subsequent reasoning for generating solutions to a current problem. Also, effective application of interpolation techniques is possible only if relevant information is stored in such a way that the identification of independent and dependent variables that is required for the interpolation is straightforward. We suggest the following steps to deal with the issues above:

Policies for Basic Reasoning: First, a designer should decide upon the reasoning techniques that are to be used in different levels of the cache, i.e. whether to use default actions or rules or past cases at a given level, for solving a problem. In chapter 5 we have seen that the nature of interpolation depends significantly on the underlying reasoning paradigm.

Representation of the Information: Depending upon the underlying reasoning paradigm, for each level of the cache, the system designer needs to specify the dependent and independent variables for conducting interpolation. Specific policies are further required to deal with the fact that the number of cells that will be selected for the purpose of retrieval may be one or two. Evidently, the policies for selecting interpolating values will vary along with the number of cells chosen.

Application of Interpolation: We have seen in chapter 5 that many of the interpolation schemes have associated with them several flags (e.g. *optimistic flag* used in *binary interpolation*) which dictate the course of the interpolation. The reasoning process needs to set the flags on the basis of the time still remaining so that the interpolated result is in principle obtainable within the available time. In subsequent sections (7.4.1 to 7.4.3) we show our policies to meet this demand. For illustration, we take an example from the AGC domain.

Consider a cache for the class of problems "unexpected appearance", i.e. appearance of surprising or unwanted entities in an airport. In designing the cache we take into account only two primary features for this class of problem: *potential-degree-of-threat* (PDT) and *location-of-the-event,* each providing a dimension of the cache.[4] Along each of the

[4] In actual AGC use the suggestion of the expertise is that another problem feature, *activity-level-of-the-airport,* should be used if one is to build a successful cache (section 3.2.3).

dimensions we can choose several columns for representing some typical values of the feature. For example, for each object of hindrance we have assigned a numeric value in the interval [0, 1.0]. Since this range of values represents a wide spectrum of the objects from material as innocuous as a body of water to as devastating as a powerful bomb or very big fire, a small difference in the PDT-value may lead to a considerable difference in the tactics of handling the problem. Consequently, we have chosen 5 columns along this dimension with header values 0.1, 0.3, 0.5, 0.7 and 0.9. Each of these columns stands for an interval of 0.2 along the PDT-scale. In extreme time-bounded situations the method will consider the column that is nearest to the current problem value. Otherwise,

(i) either two columns i and (i+1), $1 \leq i \leq 4$, will be chosen such that the problem value lies between the header values of column i and (i+1);

or

(ii) pessimistic selection (see section 6.2.4.1.3) will choose the column that stands for a PDT-value immediately higher than the problem value.

Along the dimension of "location-of-the-event" there are 4 column-headers: "passenger-lounge", "terminal-gate", "taxiway-&-runway" and "others", representing the most typical positions in the airport (see section 6.2.4.1.4).

We now describe the related aspects of storing information in a cache and their use in computing a solution for a given problem. The following points need to be considered in deciding what should be stored in the cache-cells:

- the method should be able to generate from the stored information a sequence of generic actions whose execution looks appropriate for solving a current problem (see figure 7.1);

- for each generic action there should be a method to determine, through tactical-type interpolation (see section 5.5.2.1.2), the most suitable action for accomplishing the task;

- how to make quick estimation of the parameters for each action through parametric interpolation (see section 5.5.2.1.1);

- finding a dummy solution for each task needed if the action fails (see figure 7.1)

and

- quick determination of interpolating variables (both independent and dependent) taking into consideration the number of cells chosen for that purpose.

Evidently, any decisions on how to deal with these operations in a cache will depend on the reasoning process associated with each level of the cache. We focus our attention successively on three different types of levels:

- default level, where solutions are prescribed in the form of a series of actions;
- rule level, where sets of rules are used to modify some pre-suggested actions according to the requirements of the current problems;

  and
- case level, where past cases are used as guides in formulating solutions.

For each level, we first investigate how one can reason with the knowledge stored in cells of this level of the cache, and examine how it influences the choice of what should be stored in the cells of this level, with respect to the independent and dependent variables. While (evidently) the ultimate policies with respect to all of the above will depend on the particular domain, these discussions should serve as guidance for designing cache-based systems for other domains.

7.4.1. Considerations For The Default Level

The *default level* of the cache is accessed when the available time is extremely small. Consequently, the system cannot use any rigorous method of finding a solution. Our suggestion for the contents of each cell at this level is a default plan i.e. a set of unconditional tasks which, if carried out sequentially, has a reasonable chance of resolving the current problem.[5] For example, with respect to our example, the content of the cache cell with column headers *terminal-gate* (for index "location-of-the-event") and *"0.5"* (for index "potential-degree-of-threat") is the following plan of tasks that a GOC will have to consider to tackle any such problem:

1. Control the spread of the problem;
2. Evacuate humans from the location concerned;
3. Evacuate objects from the location concerned;
4. Decide about the affected flights.

In actual representation each task is associated with parameters which help in identifying the suitable actions. In figure 7.4 we show the representation of the contents of this cell. Use of different parameters will be explained in section 7.4.1.1.

[5] Domain experts' views are necessary in identifying the exact steps and the order in which they will be applied.

Tasks	Index	Independent	Dependent	Values
control-spread	*<object>*	*PDT*	*d-o-e*	*(0.4 0.65)*
evacuate	*"human"*	*PDT*	*d-o-e*	*(0.45 0.7)*
evacuate	*"aircraft"*	*PDT*	*d-o-e*	*(0.45 0.7)*
decide-flight	*nil*	*PDT*	*(defer-time defer-area)*	*((30 45) "terminal-wing")*

Figure 7.4: Contents of a Default-level Cell for AGC Cache

Other cells of the cache also have plans of similar tasks, with three major differences:

1. The sequence in which the tasks will be considered may differ from cell to cell. For example, for cells referring to high density of humans (e.g. passenger lounge), the task of "human evacuation" should be considered ahead of "controlling the spread".

2. For certain cells some of the tasks may be unnecessary. For example, when the affected location is "runway", normally any trouble there will not directly affect any humans; therefore any task concerning human evacuation is absent from the relevant cells.

3. In some cells some additional tasks may also appear for consideration. Thus when the affected location is "other" (and it refers to some building) a task that is important to consider is *evacuation of objects.* Thus each cache cell contains knowledge pertaining to specific requirements of the situation represented in the cell concerned.

The control program accesses the relevant cell and retrieves the plan. The reasoning mechanism is responsible for executing the steps suggested in the retrieved plan. Tactics explained in section 6.2.4 are employed for selecting the appropriate cache columns. However, to achieve efficient time-bounded performance one needs to store certain extra knowledge in the cache-cells. We now describe the additional knowledge requirement and our policies for time-bounded decision making for default-level cells.

7.4.1.1. Additional Knowledge and Its Use

For each task the system needs to determine the action (and its parameters). This is a 2-step procedure: first the generic-action is selected, which is followed by selection of the appropriate action. The cached knowledge helps in their accomplishment in the following way. Consider, for illustration, the first task (i.e. "control-spread") in the cell given in figure 7.4. The associated "Index" is <object>, suggesting that the object (which poses the danger) should be

considered in choosing the right action. The method refers to the related hierarchy tree (figure 6.3) and select the appropriate generic-action (e.g. when the "object" is *fire,* the generic-action should be *tackle-fire*), as described in section 6.3.3.

Now, the method aims at determining the action. The field "independent" (figure 7.4) suggests that the determining feature of the problem ("raw" of "primary") is *PDT,* i.e. potential-degree-of-threat. The method should already have information about the PDT-value (*p*) for the current problem. And the system should compute the value of the "dependent" variable which in this case is *d-o-e,* i.e. degree-of-efficiency. Application of interpolation makes the task straightforward provided the interpolating values are known. We store in the cells additional information (in the field "Values", see figure 7.4). For the current task the field contains a pair *(0.4 0.65).* The principle behind storing this information is the following:

A cache cell may be accessed not just for a single value of the problem feature ("raw" or "primary") but for a range of values. Thus corresponding to each column two situations may exist, which refer to minimum gravity and maximum gravity of a current problem for which this particular column will be selected. The selection of the minimum-gravity and maximum-gravity situations can be made in the following way:

- *Range-type headers:* here, the selection is obvious. The upper limit and the lower limit of the range concerned provide the maximum-gravity situation and the minimum-gravity situation respectively. (It could be just the opposite if the underlying feature were such that the greater the value, the less grave is the situation).
- *Scalar-type headers:* In this situation one considers the header values of two successive columns. The mid-point of their values provides the maximum-gravity and the minimum-gravity situations for these two columns.

In each cell, we store (along with the recommended tasks) the desired values of the dependent variables corresponding to the minimum-gravity and maximum-gravity situations for which the cell will be accessed. These values help in the reasoning in the following way: for the current cell, the value of the header along the dimension "potential-degree-of-threat" is *0.5.* The two adjacent column headers being *0.3* and *0.7,* the method calculates that the minimum-gravity situation for the current cell is when the PDT-value is 0.4; and the maximum-gravity situation is 0.6.

1. If this is the only cell selected for decision-making, then these two values serve as the two independent-variable values (x_1, and x_2) for the interpolation. The pair information *((0.4 0.65),* here) stored in the field "Values" offer the interpolating values y_1 and y_2, the values dependent variable (*d-o-e* here), for the minimum-gravity and the maximum-gravity situations respectively. The system now interpolates on the pair of values *(0.4, 0.4)* and *(0.6, 0.65)* to determine the desired degree-of-efficiency for the current problem where the PDT-value is *p.* (see above).

2. When two cells are selected for retrieving information and therefore for subsequent interpolation, the system needs to consider the plans stored in both the cells for deciding the solution. For example, if the current location of a fire outbreak is "*between* passenger-lounge *and* terminal gate" while the PDT-value is 0.5, the system resorts (by interpreting the modifier "between")[6] to selecting two columns i.e. those having headers "passenger-lounge" and "terminal-gate" along the dimension "location-of-the-event" and therefore identifying two cells for decision-making. Consequently, for the current problem, for the purpose of "controlling-spread" (and other tasks as well) a method should consider the tasks and the parametric values suggested in both the cells. (In section 7.4.1.2 we provide our heuristics to control the reasoning process). Application of interpolation then determines the desired "degree-of-efficiency" value for the action that is to be initiated for carrying out the said task. The system now refers to the related hierarchy tree (see figure 6.3) to decide the actual action. Furthermore, in order to initiate an action the method needs to determine the values of the necessary parameters, the prerequisites etc. We have discussed the relevant reasoning tactics in section 6.3.4.

The same steps should be carried out for the subsequent tasks specified in the current cell. For example, the second task suggested in this cell is: *evacuate* (see figure 7.4). The associated parameter, here, is the symbol "human", implying that "evacuation of humans" is to be carried out next. The associated independent and dependent variables and the stored values help in identifying the required action, in a way just described.

For certain tasks the reasoning needs to have different tactics. Consider, for example, the fourth task (see figure 7.4) suggested in the cell concerned. Here, the method takes up the task of deciding what should happen to the affected flights. Ideally, it should consider only the exact areas of the airport that are affected by the incident and decide the course of those departing flights whose preparations for departure (e.g. loading of passengers, food, baggage) are affected by the trouble and those arriving flights which are to pass through the troubled areas of the airport. Therefore, the response of the system, here, is to defer a set of flights; it becomes necessary to compute the flights to be deferred and the extent of the deferment. Consequently, the need here is not to compute the ideal action; instead the focus of attention is to compute the time up to which flights will be deferred and the area within which the flights are to be considered. Evidently, these are calculated again on the basis of the "potential-degree-of-threat" of the current problem. However, no theory exists regarding how much space and time ought to be considered for this purpose. A GOC normally works on an intuitive appreciation of the problem under consideration. We note their heuristics in this regard as the following:

[6] The modifier "between" puts equal weight, namely 0.5, on each of the columns.

1. If the location is "passenger-lounge", all departing flights are deferred but no change in schedule is made to the arriving flights. For an incident in a "terminal-gate" all activities surrounding the terminal concerned are stopped temporarily by deferring the departing flights and diverting the arriving flights to a safe location. On the other hand, if the location of the event is "taxiway-&-hangar", all flight-related activities in the airport are suspended. In these situations all the departing flights are deferred and the arriving flights are asked not to land but to enter a holding pattern until further instruction. A disturbance in other sites of the airport does not affect any flight.

2. The extent of deferment can be decided by means of the following ad hoc policies. The deferment time may be non-uniform but evidently never decreases as the degree of potential threat increases. For a typical AGC scheme, if the PDT-value is 0.1, the recommendation is to defer activities by 10 minutes, and for a PDT-value 0.9 the deferment is 2 hours. However, if the PDT-value is even higher (i.e. between 0.9 and 1.0) a GOC will defer activities indefinitely.

3. Identification of surroundings for the affected terminal gate (see 1, above) is also based upon the PDT-value and is computed using the following default policy. When the PDT-value is less than 0.2 the "surroundings" implies the affected gate alone. For a PDT-value between 0.4 to 0.6 it includes all the gates in the same terminal-wing as the affected gate. For higher PDT-values the entire terminal building is considered as the area for deferment.

Consequently, our cell contains the relevant values (see figure 7.4):

a) a *nil* in the "Index" field indicates that no action needs to be determined:
b) the "Independent" feature, i.e. the decision-making factor, is *PDT*.
c) the "Dependent" feature. i.e. on which decisions are to be made is a pair *defer-time* and *defer-area*, indicating the time period and the space (of the airport) respectively through which deferment is to be made.
d) The pair value *(30 45)* in the "Values" field suggests the deferment times (in minutes) corresponding to the minimum-gravity and maximum-gravity situations represented by the column; and the value *terminal-wing* implies consideration of the terminal-wing concerned.

Thus the information contained in the cells within the "default level" of a cache is rather approximate but still prescribes what must be done to tackle a typical problem. However, it should be observed that we have considered only the *raw features* and the *primary features* of the current problem for interpolation within the "default level", because identification of secondary indices relevant to the current situation and their computation may use up too much computing time.

Evidently, such a solution is far from being optimal, because in order to ensure safety the action steps and the parameters are so chosen that the estimates always err on the side of safety and therefore are likely to incur extra costs.[7] But in an extreme time-pressed situation when calculation of a better-quality solution is computationally infeasible, one's choice has to be limited to these default solutions. However, if the available time is greater, one can expect a solution of higher quality by accessing deeper levels of the cache where, as explained below in subsequent sections (7.4.2 and 7.4.3), deeper reasoning tactics are utilised.

7.4.1.2. Policies for Default-level Decision-making

A system needs definitive policies for checking the available time periodically and deciding accordingly upon which computations can be undertaken. For a default-level cell, due to the paucity of available time, our policy is not to consider initiating a prerequisite action. For any such prerequisite action (needed for any action identified to be carried out) we suggest assuming the default outcome (see figure 6.4), and proceeding accordingly. Policies should also be developed for

(*i*) determination of the right actions (for the suggested tasks);

(*ii*) estimating the parameters for the suggested actions;

Evidently, these policies depend on the number of cells chosen. We first describe our policies when the number of cells is 1, which will be followed by our policy for dealing with two cells.

7.4.1.2.1. *Policy for a Single Cell*

When a single cell is under consideration, the tasks to be carried out are predetermined by the contents of the cell. In figure 7.5 we present the relevant algorithm for decision-making. In our algorithm we use the function *m-doe(action)* to denote the maximum degree of efficiency that can be achieved using action *action.*

[7] For the airport domain the extra cost is reflected in terms of extra delays in handling the flights.

```
Step1: [Preliminary]
       Retrieve from cache the set S, of the suggested tasks.

Step2: [Preparation]
       Set N =  cardinality of S.
       Determine T, the total time for computation.
       Let T_1 be T/ N.
       Start time counter C.

Step3: [Reasoning]
       For each task TK_i ∈ S, 1 ≤ i ≤ N, do

       - Set  T_1 (time left) = T - elapsed time on C;
       - Set T_a (allowed time) = T_1/(N - i + 1);
       - Compute the desired degree of efficiency D, by applying interpolation
         on minimum-gravity and maximum-gravity values suggested with the task.

       - Consider the generic action (G) on the basis of relevant "raw" feature.
       - Determine A, the set of actions a_k associated with the generic action G
       - Delete from A all actions a such that m-doe(a) < D.
       - If A is empty, then REPORT failure
         Else
            - Select desired action a_d by applying discrete interpolation on A.
            - For all the parameters associated with action a_d

              . Retrieve the 2-tuple (x y) such that:
                  x = maximum efficiency that can be achieved through this action;
                  y = desired value of the parameter to achieve this efficiency

              . Determine the parameter values in the following way:
                If still there is sufficient time left (from the allowed time T_a)
                Then
                  apply interpolation on the pair of values (0, 0) and (x, y)
                Else
                  select y to be the desired value of the parameter.
              . map the interpolated result on the appropriate symbolic set of values.
```

Figure 7.5: Default-level Reasoning Scheme on Contents of a Single Cell

7.4.1.2.2. *Policy for Two Cells*

For reasoning involving two cells, we alter the above algorithm in the following way:

When two cells are involved, selection of the task is not straightforward as the tasks recommended in the two cells may not be the same. Our heuristic here is:

1. Identify from the two cells the one having the higher weight. Call this cell C_1. We call the other cell C_2. In case of equal weights choose C_1 and C_2 arbitrarily. Call the set of

tasks suggested in cell C_i, S_i, i = 1 and 2.

2. For each task in S_2 we check if the same task is prescribed in S_2 as well. If there is no such task, add the said task at the end of S_1. If such a task exists in C_1, then modify the associated values (needed for interpolation) in the following way:

   2.a The modified minimum-gravity situation corresponds to the minimum of the two minimum-gravity situations described in C_1 and C_2.

   2.b The modified maximum-gravity situation corresponds to the maximum of the two maximum-gravity situations described in C_1 and C_2.

3. Apply Step 2 and Step 3 of the algorithm in figure 7.4 on the modified set of tasks S_1.

7.4.2. Considerations For The Rule Level

"Rule level" refers to any level for which the information contained in the cache-cells is in the form of rules. Outside the cache, the system holds a general rule-base pertaining to the domain of application, i.e. as for an expert system. However, pointers are also maintained from the cache-cells to relevant segments of the rule-base such that the reasoning mechanism has easy access from each cache-cell to the rules whose preconditions match the given situation (as characterised by the column headers of the cell concerned). Postconditions of these rules offer guidance as to how to generate suitable actions to deal with a current problem. Thus the cache has in effect an organisation of the rule-base that reflects the common expert-system idea of partitioning or indexing the rules to improve the relevance (and hence efficiency) of access. To achieve this, the designer may assign an identification number (*rule-id*) to each rule and store in each cache-cell the relevant rule-ids.

But building such rules for an arbitrary domain suffers from difficulties such as:

> "*acquiring large bodies of rules can be extremely difficult due to inexpressive representation languages, incomplete or uncertain domain understanding, or simply the sheer volume of knowledge*" [Kolodner 92b].[8]

Even when designing such rules is possible, in non-trivial domains the solving process may prove to be too time-consuming to meet time-bounded demands. In these situations we find that it is easier to work on a skeleton plan consisting of basic steps that are necessary for dealing with the situation (specified by a relevant cache cell) and to use rules for deciding upon appropriate actions and action parameters that transform the suggested plan into a sequence of situation-specific actions. It should, however, be noted there is no basic difference

[8] In fact, this is why AI researchers have been looking for alternative reasoning paradigms such as case-based reasoning.

between these two approaches as far as reasoning and interpolation is considered, as the same method of reasoning with the rules (given below) can be carried through in either case. The difference only is that the former does not presuppose any plan of actions for a given problem, while the latter stores a tentative plan of actions for solving a problem. For using rules within the cache-based system we have adopted the latter approach, as will be illustrated in the following section. The outcome of the reasoning with the rules is:

1. to determine, from the tentative plan, the exact sequence of actions that are to be carried out in dealing with the current problem;

   and

2. to determine the parameters of the suggested steps.

We discuss below the additional information that needs to be stored in the cache cells of this level.

7.4.2.1. What Should be Stored?

Contents of the cells in the rule level of a cache, therefore, are a plan to be carried out along with pointers to appropriate rules. The tasks that we consider here for each cell are similar to those in the corresponding "default level" cell (i.e. the one having the same column headers) of the cache, with two exceptions:

1. Unlike reasoning with the default level, here the tasks do not have to be carried out in the same order as provided in the cell. There may be rules (described below) that may modify the plan by rearranging the actions, depending upon some deeper aspects of the situation. Various secondary features can be used to describe these deeper aspects.

2. The suggested parametric values attached to the tasks are not purposely biased to take care of the worst possible (maximum-gravity) situation. Instead, the rule-level values indicate some optimum values for the parameters concerned, which are in general required for tackling a situation represented by the cell that contains them. However, there will be rules to modify these values subsequently, if there is sufficient time for applying appropriate rules.

Hence, additionally, each rule-level cell maintains:

a) a list of secondary features which reflect deeper aspects of a situation

   and

b) pointers to a set of rules which have possible applicability in a situation that refers to the relevant cell. The preconditions of the rules may involve listed secondary features along with the *primary* and *raw* features.

For example, suppose that the current activities of the AGC system are within the cache-cell whose column headers are: *terminal-gate* and *0.5,* corresponding to primary features "location-of-the-event" and "potential-degree-of-threat", respectively. The stored plan, imitating the corresponding default-level cell, is the following sequence of tasks:

- *Control-spread :Index <object>;*
- *Evacuate :Index 'humans;*
- *Evacuate :Index 'objects;*
- *Decide-flights;*

along with the safest values for different parameters (as discussed in earlier section).

The relevant rule-level cell contains the same sequence of tasks as its tentative plan. Additionally, it contains a list of secondary features relevant for reasoning from within this cell. In our example these features are:

1. *population-density:* which refers to estimation of the number of humans in the location that is involved.

2. *mobility-of-the-object:* which refers the nature of movement of the object. For example, a bomb is static; fire moves but the movement is predictable; an unleashed mad dog is unpredictable.

3. *approaching-aircraft:* which indicates whether there is an aircraft currently approaching the affected location;

4. *influence-of-the-object:* which suggests the items that are affected by the occurrence of a particular object of threat. We consider four different types of vulnerable items: *humans, aircraft, vehicles* and *equipment.* Objects, such as "bomb", "fire", affect all the four types of objects; "water" affects humans and gadgets; "poisonous-gas" affects humans alone; while an escaped animal may affect humans, vehicles and aircraft.

These features are used for judging the significance of the current situation and helping to achieve necessary modifications of the stored plan and subsequent actions appropriate for the current situation. This is achieved with the help of rules where these features are used in the preconditions (see sections 7.4.2.1.1 and 7.4.2.1.2).

Evidently, the rules associated with a particular cell should take into account in their preconditions the values of the relevant features and suggest in their postconditions how to improve the quality of the solution. The rules are of two types: *plan-modification rules* and *action-improvement rules.*

In figure 7.6 we show the knowledge representation within a rule-level cell for the AGC cache.

Secondary features	*mobility-of-the-object* *influence-of-the-object* *approaching-aircraft* *population-density*				
Rules	*((20 28) (37 43) (45 55) (65 73))*				
Tasks	*Task-name*	*Index*	*Independent*	*Dependent*	*Values*
	control-the-spread	*<object>*	*PDT*	*d-o-e*	*0.55*
	evacuate	*"human"*	*PDT*	*d-o-e*	*0.6*
	evacuate	*"aircraft"*	*PDT*	*d-o-e*	*0.6*
	decide-flight	*nil*	*PDT*	*(defer-time defer-area)*	*(40 "terminal-wing")*

Figure 7.6: Contents of a Rule-level Cell for AGC Cache

7.4.2.1.1. *Plan-modification Rules*

Plan-modification rules are used for altering the order of tasks given in the stored plan as and when necessary. These rules consider the values of different relevant features (including the secondary features) for the current problem to generate a plan that prescribes the right sequence in which the tasks ought to be carried out in order to achieve the best result. For example, with respect to the AGC domain, some of the relevant rules for the cell under discussion are:

(1) IF *population-density* is *large* THEN start *evacuation of humans* first.

(2) IF *population-density* is *not large* THEN start *controlling the spread* first.

(3) IF *approaching-aircraft* is *imminent* THEN start *decision about the aircraft* first.

(4) IF *approaching-aircraft* is *somewhat imminent* THEN start *decision about the aircraft* after considering *evacuation of humans.*

(5) IF *mobility-of-the-object* is *predictable* THEN stick to the *tentative plan.*

(6) IF *mobility-of-the-object* is *unpredictable* THEN start *evacuation processes* first.

(7) IF *influence-of-the-object* does not include *aircraft* THEN ignore *evacuation of aircraft.*

The cell concerned maintains pointers to all these rules (which are scattered over the rule-base according to the organisational design). The reasoning process selects from among them the ones that are relevant in a given situation and applies them.

7.4.2.1.2. *Action-improvement Rules*

These rules aim at improving the quality of the solution. Reasoning on the default level relies on default values for different parameters. As stated in section 7.4.2.1, these parametric values are calculated without considering the secondary features of the problems and also are over-estimated or estimated pessimistically. The action-level rules are designed to improve on these shortcomings. For example, some relevant rules corresponding to the abovementioned cache-cell are:

a) IF the object is *bomb* THEN increase the efficiency in human evacuation by 20%.
b) IF the object is *water* THEN decrease the efficiency in human evacuation by 20%.
c) IF the object is *unleashed animal* THEN consider evacuating children first.
d) IF the object is *bomb* THEN increase the minimum distance of evacuation by 20 metres.
e) IF the object is *gas* THEN do not remove gadgets.
f) IF the object is *poisonous gas* THEN use maximum efficiency in controlling it.
g) IF the object is *unleashed animal* THEN consider those gates that are at most 50 metres away, for flight-deferment.
h) IF the object is *water* THEN consider the affected gate alone for flight-deferment.

The raw problem features are used in selecting the appropriate rule for the current problem.

7.4.2.2. Reasoning with the Knowledge

The system should first apply the plan-modification rules to generate the exact sequence in which the tasks will be carried out. Subsequently, action-improvement rules are used for estimating the appropriate parameters from the typical values suggested along with the tasks.

Application of the rules at both stages follow certain policies:

First, the method checks all the relevant rules in the appropriate set (plan-modification or action-improvement) and checks also the preconditions of the rules successively. It discards the rules that are not applicable and considers only those rules that match with a current situation and are therefore directly applicable, or a pair of rules that partially match the situation such that the current situation falls between them (see section 5.5.3.1.2).

The selected rules are then considered for application, which is governed by the following policies. The particular policy to be adopted depends upon the number of rules selected for the purpose and also the weights of the rules where the weight of each rule is determined by the weight of the cell concerned (as determined by the cell-selection technique in section 7.3.1.2) and the distance of the preconditions from the relevant feature values of the current problem. The discussions here refer to various flags mentioned in connection with the different interpolation schemes developed in section 5.4.

Case 1: No rules have been selected

In this instance the method checks if two rules exist where the current situation does not match exactly with either of them but where the current situation lies between them. If this is so, the method then applies binary interpolation between the postconditions to find the answer. The optimistic flag is turned off for this purpose, in order that a risk-minimising solution is obtained. For example, if the current situation suggests that the *population-density* is *rather large* and therefore falls between the situations covered by rules (1) and (2) (see plan-modification rules), the computation uses the postconditions of both the rules. The answer therefore is "evacuate humans first" as the other solution may cause harm to the affected humans.[9]

However, if no such pair of rules is found, the tentative plan stored in the cell is executed.

Case 2: A single rule has been selected

This is the most straightforward of all the situations. The single rule is applied directly to obtain the desired result and no interpolation is considered for application.

Case 3: Two rules have been selected

Evidently, given our emphasis on interpolation, the system now applies interpolation on the related postconditions by invoking the rule-interpolation method described in section 5.5.3.1.1. Our policies in this regard are as follows:

1. If the selected rules refer to plan-modification, we suggest applying *binary interpolation,* where the selection is based upon the weights of the rules concerned and the one having the higher weight will be selected for application. The weight of a rule is determined by the weight of the cell that points to it. When two rules have the same weight, the selection can be arbitrary.

[9] To a GOC, safety of humans is of primary importance.

2. If the rules are meant for improving actions, application of interpolation is not limited to "binary interpolation". Some improved tactic can be employed, depending upon the relations of the preconditions, i.e. if they are cooperative, restrictive or opposing (see section 5.5.3.1.1) and the available time.

   a) if the rules are cooperative and the available time is sufficient to conduct non-trivial interpolation (e.g. inverse-mapping), we invoke the rule interpolation routine with the binary-flag off. But when the available time is not enough for such a method, we suggest using "binary interpolation" with the optimistic-flag off with a view to selecting the pessimistic solution so that the worst possible situation can be tackled safely.

   b) if the rules are opposed in their effects, we suggest using an interpolation technique such as *inverse-mapping* or *optimisation of functions* (depending upon the requirements), provided that the time is sufficient for its application. Otherwise we suggest binary interpolation with the optimistic-flag off for the same reason mentioned just above.

   c) if the rules are restrictive, then constrained interpolation (section 5.4.5.1) is the right way to devise the solution.

Case 4: More than two rules have been selected

Here, interpolation cannot be applied in a straightforward fashion. Therefore the following tactic has been adopted. For each rule the method demarcates the postcondition and notes the weight of the relevant rule. All the possible postconditions are next arranged in decreasing order of their total weights considering all the rules together. The selected postconditions are then subjected to "discrete selection" (see section 5.4.2) so that the alternative having maximum weight is chosen. For plan-modification rules it means that the plan of tasks suggested by the majority of the rules (measured in terms of total weight) will be selected. For example, consider (with respect to the AGC domain) the situation when

> *there is a leakage of gas in a terminal building and the population density is described to be sort-of large.*

Consider also the set of rules given in the earlier section 7.4.2.1.1 on plan-modification rules. Rule (6) applies immediately as the gas spreads unpredictably and therefore its postcondition (*start evacuation first*) receives a weight 1.0. On the other hand, rules (1) and (2) are on either side of the current problem. The rule (1) refers to a precondition of *large* population density; while the precondition of rule (2) stands for a situation where population density is *not large.* Consequently, weights assigned to these two rules are 0.38 and 0.62 respectively; by considering the fuzzy equivalents of *normal, not* and *sort-of,* which are 0.75, 0.25 and 0.44 and by using our distance computation method. Thus the postcondition *"start evacuation first"*

receives a total weight 1.38, while the other relevant postcondition *"start controlling the spread first"* has a total weight 0.62. We therefore suggest taking up "evacuation" first, followed by the task of "controlling the spread".

The same tactic applies for action-modification rules. However, if the number of options is just two, and the time for computation is sufficient, we advocate use of non-trivial interpolation such as *fuzzy interpolation, relational interpolation, inverse-mapping* depending upon the natures of the terms involved.

It should be noted that it is immaterial to the method described above whether the rules being considered are from one cell or from two different cells. We therefore suggest application of the same tactic irrespective of whether one cell is under consideration or two cells. In the latter situation, however, the weight of each rule should be modified additionally by the weights of the respective cells for computation of the total weight of the postconditions.

Evidently, the reasoning at the rule level of the cache provides qualitatively better solutions than any obtained through reasoning with the default-level entries. However, here too, the reasoning is still based on a fixed set of foreseeable tasks. But for many situations the solving process needs to be flexible in order that a plan can be generated depending upon the needs of the current situation. As mentioned in section 5.5.2.2, a user may like to have some "precautionary" steps as safeguard against a future difficulty; or some extra action in advance for "enhanced performance" in dealing with a current difficulty. We find that representing knowledge in the form of past cases, as in CBR, is helpful here, in deriving the solutions for a current situation. To achieve this objective a process typically needs to have more computing time than for the earlier methods. Consequently, a deeper level of the cache that we discuss is related to reasoning with cases.

7.4.3. Considerations For The Case Level

When available time is even greater, a cache-based system can refer to a case level of the cache. Since each past case contains its own sequence of executed actions and their purposes (see discussion on case representation section 5.5.3.2.2), we do not recommend storing any predesigned plan of tasks in cells of this level. Instead, we suggest that the reasoning mechanism should select the most appropriate cases for applying interpolation and generate the required sequence of tasks which are then translated into purposeful actions.

In order to apply cases in support of a reasoning process, the overall system needs to have a case-base of past cases. This will lie outside the cache. We give each case some specific identification number, called *case-id.* Each cell of the cache maintains a list of case-ids that are relevant for a situation for which the cell concerned is accessed. To accomplish this, a designer needs to analyse the available cases to extract their values for the "primary features". A particular cache cell contains case-ids of those cases only whose primary feature values

match with the column headers denoting the cell concerned. In certain situations, a particular case (i.e. its case-id) needs to be incorporated in two cells. For example, if a current problem has the "location-of-the-event" value (*between* terminal-gate and *taxiway*), cases may be needed from both the cells which have their values, along the said column dimension, *terminal-gate* and *taxiway-&-runway,* respectively (other header-values being the same). (We admit that a more detailed analysis of cases may lead to a better organisation of cases in a case-base. In this respect our belief is that by storing individual case-ids in each cell one can have quick access to the relevant cases, although storing in this way is somewhat painstaking. However, CBR researchers are actively engaged in devising quicker methods for case retrieval - the gamut ranges from designing appropriate similarity metrics for retrieving imperfectly explained cases [Bento 94] to massively parallel CBR [Myllymaki 94]. This, however, is not within the basic target of the present thesis. And we hope that any such method should, in principle, be usable from within the cache as well. In some sense, if the hope is well satisfied, this will be an element of validation of the cache scheme itself).

We observe that not all the features related to a particular situation are equally important from the reasoning point of view. For example, suppose the current problem is that

A mentally deranged and aggressive man is roaming freely in the terminal building.

Having found no single case with *mentally-deranged man* as the object, the method may refer to the secondary feature *mobility-of-the-object,* and consider a case where a problem of a *freely moving unleashed dog* in the terminal building has been dealt with. The justification behind this selection is that "unleashed dog" being unpredictable in the spreading of its effect, is more similar to the current problem than other objects such as *fire* or *bomb*. And the only notable difference in their treatments will be in the name of the "controlling agent".

Consequently, not all the "raw" features of the case are of primary importance here to measure similarity. Hence a designer needs to identify the most important features which influence the decisions significantly. In this domain we find that the two most important features are: *approaching-aircraft,* and *population-density* (in the affected location). Hence we consider these two features for identifying cases that are similar to a current problem situation. We first select from the set of cases pointed to by the cell those that match on the *key features.* If any of the suggested cases matches the current situation exactly, the case can be related directly to the current situation. When no cases match the current situation exactly, we use the secondary features to compare the similarity between cases at this stage. Consequently, in each cell of the case-level we suggest storing the lists of relevant *key features,* and the *secondary features* along with the list of *case-ids* pointed to by the cell. In figure 7.7 we present the contents of a case-level cell of the AGC cache. In subsequent sections (7.4.3.1 and 7.4.3.2) we present our policies for case selection and conducting interpolation between cases.

Key-feature	*(approaching-aircraft population-density)*
Secondary-feature	*(mobility-of-the-object influence-of-the-object)*
Case-ids	*<list of case-ids>*

Figure 7.7: Contents of a Case-level Cell for AGC Cache

7.4.3.1. Policies for Case Selection

Once a particular cache-cell is accessed, the case information is retrieved and the suggested cases are compared with the current situation to decide upon their applicability. We have observed that it is sufficient to use simple techniques such as counting the number of common features [Ashley 89a], where the set of *key features* is to be defined by the user or a modified version of the above where user-defined weights affect evaluation of the significance of each feature and measurement of the degree of similarity. (One can possibly maintain some statistic (say, frequency of use) to determine the relative importance of the cases for a given type of situation and adjust the pointers to the cases (from the cache-cells) accordingly, in order that least time is spent in searching for cases. Researchers in traditional CBR [Kolodner 89a] have made studies and developed heuristics for efficient arranging of cases in case-bases. We have not studied the relative usefulness of different heuristics in this thesis).

Policies are needed to select two cases from among the suggested ones for carrying out interpolation. When a single cell from the cache is chosen, both the cases should come from the same set of cases pointed to by the relevant cell; and the one having the higher degree of matching with the current problem serves as the "guide case" for case interpolation (see section 5.5.3.2.3). On the other hand, when the activities involve two cells, we consider the most similar cases from each cell for the purpose of interpolation and the case suggested by the higher-weight cell serves as the "guide case". The actual task of interpolation, however, is carried out in the same way (as discussed in section 5.5.3.2.4)[10] irrespective of whether the cases are prescribed by a single cell or two.

We first compare the cases with the current situation using the *key features* and select those that share similar values with respect to the current situation. The selected cases are then compared for similarity in "raw" features. If no case matches exactly, we compare them along

[10] Evidently, one prerequisite for the approach is that past cases should be represented in the case-base following the action-based representation scheme explained in section 5.5.3.2.2.

the secondary features. The distance-measurement techniques described in chapter 5 help here in measuring the distance with respect to each feature between the current problem and a past case. The case having the minimum composite distance is selected for interpolation. When activities involve a single cell, the closest two cases are considered for this purpose.

An improved decision-making policy can be devised where one considers use of some threshold (T, say) such that a case will be considered for interpolation if its composite distance is below the threshold T. If no cases satisfying this condition are found, the reasoning method should abandon using cases for resolving the situation and move to a shallower level (depending upon the time left) of the cache and apply the method prescribed in the new level.

7.4.3.2. Policies for Case Interpolation

Before invoking the routines for case interpolation, a system needs to judge the temporal aspect of the situation and set various control flags, required for controlling the interpolation, accordingly. If C1 and C2 are the two interpolating cases and the time limit to accomplish the interpolation is set at L, we suggest the following approach for accomplishing the task from within the cache-based environment. (We assume, in this discussion, that of the two interpolating cases the case C1 will be used as the guide case for performing the interpolation).

Initially the system should allocate equal amounts of time for each action to be carried out. If the number of actions taken in the "guide case" is N, then for each action the system allocates a time (T_a) of length L/N. This number will usually be modified subsequently in the following way: at each stage the system counts the number of unmarked actions (i.e. which have not yet been considered in the interpolation). Having computed the time that is still left, out of the initially-allocated time L, the system divides it equally for all the actions that are yet to be taken care of.

While dealing with a particular action the system first determines the number of parameters to be estimated. It then determines the type of each of the parameters, as the requirement of time depends upon the type of the data, and estimates the amount of time (E) that will be required if rigorous interpolation is carried out with all of these parameters. The order in which the parameters will be interpolated depends upon the value of E and T_a.

1. If $E < T_a$, the parameters are arranged in decreasing order of importance and interpolation is initiated for the most important parameter first, with the binary-flag off. The idea is that the system tries to find the value for the most important parameter using some non-trivial interpolation technique (*inverse mapping, optimisation through some special function*). The same consideration is then given to less important parameters in succession. But if the time required in doing so exceeds the expected limit, the system switches itself to

binary-interpolation mode by turning the binary-flag on. The advantage in this approach is that the poorer tactics of interpolation will be applied only on comparatively less important features.

2. If $E > T_a$, then the approach will be just reversed. The system then considers the parameters with the least important one first, and sets the binary-flag on. But, as the binary selection method is relatively fast (because of the absence of any substantial computation) it may happen that later while considering the most important parameters the system may have more time than initially allocated to opt for rigorous interpolation.

The interpolation scheme successively generates the necessary actions along with the required parameters, which constitutes a plan that should be appropriate for tackling the current problem. Having determined all the actions and their relevant parameters, if sufficient time still remains for further improvement of the quality of the solution, the system may go for improving the solution by applying either "constrained interpolation" or "iterative interpolation" where a feedback for the proposed solution is sought from the user and the system changes the necessary parameters accordingly. We omit these issues from our present discussions.

One may design a more elaborate approach, particularly in selecting cases from the case-base by considering more abstract features for comparing situations. Similarly, one may consider a full-fledged planning method where subtleties of each situation will be considered in designing a solution. Any methods involving such advanced techniques will inevitably have larger temporal demands and therefore can be incorporated within the framework of the cache-based system, e.g. by adding further cache rows and attaching different time limits and different schemes of computation to the different copies. We have not considered explicit plan-generating methods in this thesis.

7.5. CONCLUSION

The techniques of "knowledge interpolation" and "ordering and distance measurement between symbolic quantities" play a crucial role within our scheme of reasoning with the cache-based system. We identify three possible stages of reasoning where these techniques are of paramount importance:

- Meta-planning - when a system prepares itself for retrieval from the cache and therefore tries to identify the right columns. In designing the cache, one should keep the column dimensions and the headers along each of them unambiguous. If the raw features are simple too, there is no scope for any interpolation. But in actual use the raw features may take unspecified values involving *fuzzy qualifiers, relational modifiers* etc. In this

chapter, we have prescribed how this can be handled by choosing multiple relevant cells with appropriate weights, based on the distances of their column headers from relevant problem parameters.

- Planning - when a system is framing a plan for the solution. At this stage the system is retrieving relevant information (defaults, rules, cases) and applying interpolation to generate the desired solution.

- Execution - we envisage that in actual application a system may have to tackle more than one problem at a time (we do not suggest any parallel computations here, but if two problems have occurred within a very small interval of time, a system may be called on to start computations for the second problem while it is waiting for some result of an action needed for the first problem). The system may also recognise that some action that is needed for the first problem has relevance for the second problem too, and can (at least in principle) apply some interpolation there by considering relevant features of the second problem too. Any real-time system should have some capability to perform opportunistic reasoning. In our current work, however, we rule out this possibility for the sake of simplicity, and we assume that the cache is to be accessed sequentially for each problem.

Chapter 8.

DESIGN OF SYSTEM FOR A NEW DOMAIN: SHORTWAVE RADIO PROPAGATION

8.1. INTRODUCTION

The dissertation so far has discussed the cache-based architecture and its different aspects for dealing with time-bounded demands. We now concentrate on implementing the ideas developed therein for a particular domain. For illustration we shift our attention from the AGC domain to the domain of shortwave radio propagations.

Analysis of different facets of an airport's incidents, which we collected from our experts and different newspaper stories, helped us in identifying the varieties of problems that one would encounter in a realistic domain and the different types of symbols or descriptions that are used for practical problem-solving there. These observations, in turn, made it possible for us to design our different ordering techniques in the first place and to develop consequent distance-measurement methods and interpolation tactics (as described in chapter 5) subsequently. However, in trying to build a full-fledged cache-based system for handling any substantial range of AGC problems we faced several difficulties:

1. The problems that may occur in the AGC domain are too varied and heterogeneous to lead to an obvious set of indices capable of characterising all the different problems. Also, there is no corresponding set of generic actions that can be prescribed for all the different types of problems at large. For example, there is a class (we call it *hindrance*) of problems that may appear somewhere in the airport and therefore affect a number of flights. Problems such as an *outbreak of fire* or *appearance of an unwanted object* fall into this category. Any action that a GOC may take needs consideration of the set of flights influenced by the hindering object. On the other hand, there may be a class, which we have named *disturbance*, of problems that do not affect any flight directly. For example, if the location of the event is such that no aircraft is in the vicinity, e.g. *a fire in the engineering unit*. Consequently, any plan of actions that a GOC may initiate does not normally contain any decisions regarding flights. But if the situation is too grave, or if the affected locations have direct roles in arrival or departure of flights (e.g. the control building), the GOC needs to take decisions about the flights affected indirectly by the incident.

Additionally, there may be problems that affect a single flight. We call this class: *trouble*. Problems when a particular aircraft, due to leave soon, *develops a snag;* or when an airline *requests a deferment of departure time* (possibly because it has failed to load passengers/ baggage in a timely way); or in the event when an aircraft, just arrived, has *problems in opening the baggage-door* fall into this class.[1] Consequently, a GOC's plans of action vary considerably along with the nature of the problems. An important task for the GOC in tackling a "hindrance" or "disturbance" problem is to arrange for immediate evacuation of the affected location; but such a step is normally not required in handling a "trouble" in an aircraft. Furthermore, occurrence of a hindrance makes it mandatory to halt (at least) temporarily some of the imminent scheduled flights, but trouble in an individual aircraft rarely affects the schedule of other flights (unless the airport is so busy that the same gate is to be used for a succession of flights).

2. The most practical way of dealing with the above problem is to build separate caches for each class of problems that the system is expected to handle. This is because building a single cache for all the types of problems will mean having different column dimensions for different classes of problem leading to a cache with too many dimensions, most of which will remain unused in any particular problem context. It was clear that the size of the AGC domain would require collection and representation of knowledge that would be too extensive to manage single-handedly within the limited time period of this project.

3. Most of the actions pertaining to the AGC domain involve significant human interactions. Consequently, any gain in efficiency (measured in terms of milliseconds) that one should expect while using the cache-based system easily evaporates as the human involvement (which is measured in terms of minutes per human action) has a dominant effect on efficiency in comparison with the computational demands. Furthermore, the variation of human performance makes it difficult for us to have reasonable estimates for different action times which are crucial to the success of a cache-based system. Also, many of the problems in an airport take place so infrequently that accruing all the knowledge at the beginning is practically impossible and the system needs a regular updating of knowledge.

4. Also, it is often difficult in an airport to ascertain the quality of a solution that has been suggested by the cache. Experts are not always unanimous regarding the quality of a solution as they are often guided by their individual apprehensions of the problems. Since the purpose of the cache-based system is to produce higher-quality solutions as more computational time is available, it is sensible to look for a domain where the solutions can be ordered easily according to their qualities.

[1] One possible exception is the situation when some potentially dangerous object such as a bomb is suspected to have been planted in an aircraft. In such a case a preventive action of the controller affects not only the aircraft concerned but also other aircraft in its immediate vicinity.

For the purpose of demonstration, we therefore focus our attention on a different application that does not show these drawbacks: *radio-wave propagation,* where a system is expected to produce a recommendations for how to transmit a message as requested by the user (client) to a given destination with an expected reliable quality of reception. The task involves:

1. identifying the right frequency for the transmission;
2. deciding the appropriate power of the transmitter that is required for the propagation.

Consultants on shortwave radio propagation engineering are often requested by their clients to guide them in choosing the appropriate values for the above two features for carrying out the propagation. The users of the facility include (among others) aeronautical and maritime radio services; mobile and stationary services of all kinds; world-wide transmission of broadcast programmes by different radio stations, large or small, all over the globe; weather forecasting and exchange of information between different public institutions and private organisations. Depending upon the nature of urgency the expert may need to make a time-bounded decision for the current purpose. However, the task is straightforward relative to the AGC application: the ideal values of power and frequency may vary with the target location, source location and many other physical phenomena as we describe in the following section (section 8.2); but with manageable dimensions and with characteristics that are well adapted to cache labelling and to interpolation. The greater is the time available to the expert, the more accurate and well-considered will be the answer. Furthermore, the task involves manipulations with different symbolic quantities and quick approximations of various featural values. The domain is both interesting in itself and appropriate for treatment from the cache-based architecture and knowledge interpolation points of view. Moreover, it has enough structure to provide examples and tests of all the key features of the design that we have proposed and illustrated in previous chapters by reference to the AGC domain.

This shift of attention in no way means that a complicated domain like the AGC is not amenable to the cache-based reasoning technique. We feel that use of multiple caches, each assigned to a particular class of problem (such as hindrance, trouble mentioned above) is an ideal modification of the system design that should be able to cope with the versatilities of a domain like AGC. In such a modified system, meta-knowledge ought to be used additionally for identifying the appropriate cache that may provide solution to a current problem. However, collection of enough knowledge to fill all the relevant components of such an extended system is too large a job to add to our existing work in one thesis project. The homogeneous nature of the radio-wave propagation problems helped us in achieving our goal (i.e. design and implementation of a cache-based system) with a single cache. Actual use of a cache-based system for radio-domain will need a constant updating of cached-entries as the transmission conditions vary with solar condition annually. On the other hand, a cache built for the AGC domain should need no regular updating, as the background situations should remain constant, unless there is a significant change in the airport layout.. Furthermore, with advancement of knowledge about the physics of electro-magnetic propagation, new rules (and cases) may be incorporated into a radio-cache, for an AGC cache we do not foresee such periodic refinement of knowledge.

In this chapter we describe the radio domain and its properties in detail, along with the types of knowledge for building the cache and other relevant knowledge-base(s). In the succeeding chapter (chapter 9) we describe our experimental results.

8.2. DESCRIPTION OF THE DOMAIN

The use of short waves in the range of about 2.5 to 30 MHz is probably still the best (in a perspective that combines flexibility, price and efficiency) means of transmitting information over arbitrary terrestrial paths and distances. The path that is to be traversed by the radio wave can range from a few kilometres (km) to half the circumference of the Earth (about 20,000 km). Planning a shortwave link therefore requires detailed consideration of the radio path, involving the length of the path and other geographical and geophysical conditions [Braun 82, (chapter 2)] .

The propagation of the wave can be of two types: *ground wave* and *sky wave.* The "ground wave" is propagated along the surface of the Earth in proximity (with some interaction) to different media such as soil, water and rock. The sky wave propagates in a different way. The wave is emitted through an antenna towards the sky, and then rebounds from the ionosphere, which consists of electrically conducting layers of charged particles surrounding the Earth. There can be many such layers of charged particles, at different heights from the Earth's surface. Some of them are relevant for shortwave propagation: in particular, the D-layer (60 to 90 km above the Earth), E-layer (110 km), F1-layer (170 to 220 km) and F2-layer (225 to 450 km above the Earth). However, not all the layers reflect at any particular time of the day, neither do their densities and reflecting powers remain constant throughout. In fact, the ionisation and concentration (and therefore reflecting capacities) of these layers vary with time. The angle of incidence of the wave on the ionosphere and the current ionised state of the layers determine the distance that can be covered by the wave. But there are many obstacles to straightforward propagation, which can be natural as well as created by man.

8.2.1. Natural Obstacles

The natural phenomena that have influence on propagation are the following:

Daylight path: The ion densities and consequently the reflecting capacities of different layers of the ionosphere vary with the time of the day. The ultraviolet radiations of the sun split the molecules in the atmosphere into electrons and positive ions. Although the free electrons collide with the ions for mutual neutralisation (recombination), the extent of ionisation and recombination varies with the solar radiation. As the solar radiation becomes stronger,

ionisation increases in strength and as a result the capacity of the individual ionised layers in reflecting high-frequency waves increases. The maximum is achieved when the sun is at the highest position. As the sun sets the ionisation declines and recombination increases so that after the sunset a fairly neutral state is reached. Furthermore, the lower-altitude layers are normally formed during the daytime, and disintegrate quickly as the sun sets. These layers, having higher densities of electrically charged particles, are responsible for wave propagation during daytime. On the other hand, more stable higher-altitude layers of the ionosphere act as the reflecting layers at night. Consequently, not only does the ion density of the layers vary with the time of the day, but the effective layer at which the best reflection for a given range of frequencies takes place changes between daytime and nights. An expert therefore needs to consider the time of the day for deciding the best frequencies for transmission. The task is even more interesting for long-distance propagation as the effect of solar radiation is specially enhanced at a given location near its sunrise and sunset times. As those may occur along various segments of a propagation path, it causes favourable disturbances in propagation from or to other places near their sunrises and sunsets according to what in radio engineering jargon is called the *grey-line effect.*

Seasonal effects: As the solar radiation effects change with the season, the usable frequencies also vary with the time of the year. Particularly, one needs to distinguish between two seasons that for convenience we call *winter* and *summer.* For detailed calculations one should also consider *autumn* and *spring,* as solar radiations in these two seasons are not as extreme as in summer or winter. In addition, while considering seasonal effects one needs to consider the hemisphere (north or south) where the activities are going on.

Water Path: This is particularly important for long-distance transmission, as the sky wave that propagates from a distant source may not reach the target by one reflection in the ionosphere. Instead, there will be multiple reflections between the earth and the ionosphere before the wave reaches the target zone. Consequently, the condition of the Earth's surface from where the wave rebounds to the sky is important. Particular distinction needs to be made according to whether there are long stretches of water in the path as reflectivity of water differs from that of land. Waves of higher frequencies can be reflected more easily from water surfaces than from land. Thus typical propagation within the southern hemisphere has different properties from that within the northern. Even within the northern hemisphere propagation over land (say, Europe to Central Asia) has different characteristics from propagation between two points with a vast water path between them (e.g. between Europe and North America).

Sunspot Number: The nuclear activity occurring in the sun, producing light and heat, does not follow a constant pattern. Instead, it is in a state of continual change. However, it has been observed that a cyclical behaviour of this activity exists, whose best-known component is called the *sunspot cycle* [Bird 92]. It is an 11-year cycle during which the intensity of the solar activity periodically increases and decreases. For each year a sunspot number is assigned, based on the average solar activities of the last 6 months and the estimated activities in the next 6 months. Observations over many years have shown that the annual

minimum value of this average may lie in the range [0, 10] and the maximum values in the range [50, 190]. When the activity is low the ionosphere is weaker and propagation of frequencies higher than 15 MHz is virtually impossible by reflection in the ionosphere. On the other hand, when the sunspot number is high, there will be higher radiations in the sun causing higher degrees of intensity in the ionosphere. Consequently, radio waves of higher frequencies can be propagated.

Atmospheric Noise: Natural phenomena such as storms, heavy rain, snowfall, and sandstorms release copious electrostatic discharges in the atmosphere causing radio noise. This noise interferes with the electromagnetic radio wave and may hinder its propagation, besides being heard as an interfering distraction by the radio listener. But the level of these disturbances varies over the Earth's surface. In higher latitudes of northern and southern hemispheres the activities are comparatively low; while in the equatorial region (20^{o} North - 20^{o} South) the activities are very intense (called *tropical interference*). Furthermore, these activities have seasonal and diurnal variations as occurrence of these electrostatic discharges is greater in daytime and disturbances from further locations may affect the propagation. Further, these natural phenomena are more common during summer than in winter, causing less interference in the winter.

Auroral Activities: The polar regions of the Earth experience electromagnetic discharges, primarily caused by flows of ions from the sun. These discharges show their presence in the form of streaks of coloured light, called *aurora.* Their occurrences are greater in the winter than in summer. Normally, these activities are around the magnetic poles of the Earth and covers an annular-shaped region (60^{o} - 68^{o} latitude in the Northern hemisphere; possibly slightly different values in the South). The electromagnetic nature of the aurora causes interference in the propagation of radio waves. This is more acute for long-distance propagation which may have to cross an auroral zone twice on its path.

8.2.2. Human-Created Obstacles

Apart from the natural phenomena which influence the propagation of radio waves, some man-made problems may affect the propagation as well. The radio waves emitted by other transmitters often affect the audibility of the message that is sent. In order to avoid this confusion there is an *International Frequency Registration Board* (a dependency of the *International Telecommunications Union* or *ITU),* where all regular users register, on the basis of the expected geophysical conditions, the frequencies that they intend to use at different times of the year and different times of the day for different transmissions. Various broadcasting corporations (e.g. BBC, Voice of America, All India Radio) register the frequencies that they want to use for different transmissions, well in advance, in order that the listeners can receive early data and therefore have improved chances of hearing their target programmes uninterruptedly. Nevertheless, because of the large numbers of broadcasting organisations and the fact that not all are as efficient or as disciplined as those just mentioned, any attempt at

sending messages using any such frequencies, pre-assigned to some organisation, must expect to face the possibility of interference.

Further difficulties may arise from the following factors:

1. **Small-scale Radio Centres:** Many local radio centres often do not abide by the international rules, and broadcast programmes on certain frequencies (aiming at some local population) that block the regular transmissions.

2. **Jamming:** Deliberate interference from certain countries, primarily for political reasons. These countries often broadcast roaring sounds, interspersed with some coded messages (e.g. programme tapes played backwards) in order to block the propagation of certain programmes in some target areas. Countries such as Iran, Iraq, China and to some extent North Korea, Burma and Turkey are the principal jammers in the current political world.

3. **Unintended Interference:** Often very close frequencies (with difference of a few kHz) are allotted to more than one broadcasting station for transmission to different target areas. Simultaneous operation by two such broadcasters may interfere with one another. Although such allocations are so made that the level of interference will be negligible, in reality (often due to lack of selectivity in the receiver) such interference may become significant. The degree of interference is directly proportional to the power attached to the interfering transmitter (IT) and is roughly inversely proportional to some power (near 2)[2] of the distance of IT from the target area.

An expert in shortwave propagation needs to consider all these factors for determining the frequency and power for some particular transmission that is requested.

8.2.3. Who Are The Users?

An expert may receive a time-bounded request from users of any of the following types:

1. **Broadcasting Companies:** Although the frequencies for transmissions by various broadcasting companies are planned well in advance and their frequencies for this purpose are pre-determined, they often need assistance from experts for the following purposes:

   a) private communications, particularly with journalists, covering some events (such as fighting) at remote interior places having no telephone facilities;
   b) relaying some programmes (such as ceremonies, sporting events) which have not been planned far enough ahead to figure in ITU-registered frequency occupancies;

[2] We are using the value 2.3 as per our experts' advice.

c) sorting out difficulties in regular broadcasting services which can be created by jamming or by the appearance of some new small or undisciplined station that interferes with a regular frequency allocation or by sudden natural disturbances.

d) emergency announcements such as asking the nationals of a particular country X who are located in a different country Y to contact the embassy of X in Y for quick evacuation (normally, in the wake of some political hostility).

2. **Other Regular Users:** Multinational organisations (e.g. airways) maintain radio communications between offices located at different sites, and between aircraft and offices, for transaction of business-oriented information such as regular updating of situation data, or reports of any unusual happenings.

3. **Irregular Users:** Aeronautical and maritime users who often need to send important messages to an arbitrary location at arbitrary times;

4. **Military:** Military officials also need to transmit messages to some specific locations depending upon their requirements, e.g. for national contingents in UN peacekeeping forces (consider the 1995 situation in Bosnia, for example). These requests too may come in irregular fashion;

5. **Sporadic Users:** Many private or public organisations (e.g. oil companies) occasionally need to transmit messages through radio with a view to expediting their business.

We have divided all the different types of transmission requirements into two classes: *broadcast* and *pressing*. By "broadcast" we mean transmission for a considerable or long period, while "pressing" stands for problems where an urgent message is to be transmitted to an appropriate audience. The obvious difference between these two types is that in the former situation it is necessary to determine a transmission set-up free from any interruptions. But in the latter situation, a transmission with some amount of interruption may be acceptable. Therefore, although the same set of tasks is recommended for both the problem-types, the actions to accomplish them may differ.

8.2.4. Amenability To The Cache-Based Architecture

The task of the proposed system is to receive the request and suggest the most appropriate transmission requirements (frequency, power) that should be used for the intended transmission. The time limits allowed for solving current problems can, in practice, be no more than a few minutes. In some urgent cases an user needs the information within a few seconds only.

There are other factors (such as antenna design, antenna orientation) as well which influence the transmission process. In ideal planning a system designer should make decisions for these

aspects too. But we are not incorporating them in the present form of our system as these features are not considered to be so important from the point of view of heuristics. Moreover, even when any relevant heuristics are taken into account along with the basically algorithmic treatment that experts give them in practice, their processing would not require any additional developments in the *architecture* that we develop and evaluate in this thesis.

To accomplish this objective it is necessary to explore the amenability of the shortwave domain to the overall demands of the cache-based architecture. In this respect we first observe that an expert in the domain normally uses a variety of reasoning tactics for solving a current problem: in extreme time-pressed situations one may provide some ad hoc (based on some typical observations) solutions; in other situations one tends to use some rule-like knowledge (e.g. *if the season is winter then waves of high frequency are not likely to rebound from the ionosphere*) or some past experience, e.g. expressed in cases (e.g. *in 1993 an attempt to send a message to Baghdad using frequency 6560 kHz had failed, and the reason has later been discovered to be a jamming operation by the Republic of Iraq Radio*), to design a well-considered solution. We further observe that an inherent smoothness exists in the solving mechanisms so that marginally disparate problem situations do not result in vastly different solutions. Consequently, the technique of knowledge interpolation looks highly appropriate for the underlying reasoning schemes. These observations prompted us to design a cache-based system for the shortwave domain, as a good test of our methods.

8.3. ANALYSIS OF THE DOMAIN

Evidently, the first step towards building a cache-based system is to analyse the domain with a view to identifying the different types of knowledge: *cache-related, action-related* and *reasoning-related* (see section 6.1) required for the system.

In some of the earlier chapters (3, 6 and 7), we have described our policies of acquiring this knowledge with the help of examples from the AGC domain and explained the rationales behind our selections. In the absence of any well-developed theories that experts follow unconditionally, we take the same approach for the radio domain as well. We have found that the same general reasoning methods developed with reference to the AGC domain can be followed equally well in the new domain. This similarity of treatment of two entirely unrelated domains makes a good argument for robustness of these methods. At places in the design, we nevertheless make marginal deviations in the details of the cache architecture for the AGC domain to take account of local differences in properties of the two domains:

a) the problems in the radio domain are not as varied as in the AGC domain. Consequently, unlike the AGC domain (as indicated in section 8.1), here we have identified a single set of the tasks that may be suggested in different situations. These are:

comp-frequency, i.e. deciding about a suitable frequency that can be used for transmitting the message under the stated geophysical conditions (existing at the time when the transmission occurs);

comp-power, i.e. estimation of the power that should be used in the transmitter for the transmission;

and

comp-transmitter, arranging for transmitter(s) that can perform the transmission. We consider a single cache to be appropriate for the radio domain.

b) Unlike the AGC domain we do not attach any degree-of-effectiveness to the actions attached to a particular task. Instead, as per suggestions of our expert, for each type of problem we arrange the actions in decreasing order of preference. The method tries to obtain a solution using the most preferred action. It resorts to a less preferred action only if more preferable actions fail to produce any solution. However, the sequence in which the actions are considered often depends upon the problem itself. For example, when the problem is a "broadcast", the method considers the sequence *(search-band search-neighbour-band exit-band)* for determining the right frequency. But if the problem is "pressing" the method's preferred sequence of action is *(search-band exit-band search-neighbour-band),* particularly when the the time is strictly limited. A "band" is a range of shortwave frequencies received by the ITU primarily for public broadcast transmissions.

8.3.1. Cache-Related Knowledge

Relevant cache-related knowledge expresses decisions about:

1. identification of "raw" features (see section 6.2.3) for input and characterisation of new problems (and for subsequent use of the features in communication with the system in feeding the right knowledge into the cells and retrieving it);

2. identification of "primary" features, i.e. those representing the column dimensions of the relevant cache. The task also involves deciding upon the column headers with respect to each column dimension.

8.3.1.1. Input of a Problem Description

A problem should be fed into the system by inputting the problem type (i.e. whether the intended transmission is "broadcast" or "pressing") and then specifying the values of the "raw" features. For an user of this system the current problem is to transmit a radio message or broadcast at some certain time or period of the day. Consequently, the relevant features are:

- the location from where the transmission should take place (*source*);
- the destination of the message (*target*);
- the time of the year (*season*)
- the time when this transmission starts (*begin-time*).

  and
- the time when this transmission ends (*end-time*).

A user needs to input to the system the values of these "raw" features for a current problem. Two further relevant features that are required for any decision-making are: whether or not there are any preassigned frequencies from which to make the selection (regular broadcasters often register certain frequencies for their exclusive use) and whether there is any upper limit to the available power. These features are used in deeper-level reasoning with the cache, and therefore the system will ask for values for these two features only when the available time is sufficiently long.

8.3.1.2. Structure of the Cache

The next design decision is to identify the "primary" features of the domain to determine the column dimensions of the cache. Normally, a user specifies the location information in the form of name of the place (or the country's name). But using the country names or location names, as they are, for accessing the cache not only leads to a huge cache due to the multitude of terms, but is also unnecessary for the following reasons:

1. propagation of waves is guided by geophysical conditions and not by political boundaries;

2. the distance, D, between the *source* and the *target* is important in determining the propagation path, where the distance between two places can be measured using the latitude-values and the longitude-values of the two locations;

3. yet, the measure of D as such does not help directly in determining any policies for transmission. For example, transmission between two places which are at a distance of 3000 km (say) but having close longitude values will be influenced by entirely different physical conditions from when the two locations are close in their latitude values. In the former case the wave is unlikely to pass through a polar zone (hence it should not be subject to auroral disturbances) but may pass through the tropical zone and is therefore likely to be influenced by tropical interference (see section 8.2.1). On the other hand, the transmission path in the latter situation is not affected by the tropical disturbance unless the source and target locations are in the tropical zone; and it is also not affected by auroras unless the locations concerned have very high latitude values (in either hemisphere);

4. transmission for a comparatively smaller distance between two higher-latitude regions may be influenced by the aurora, while transmission for a much larger distance near the equator are not influenced by auroras;

5. the longitude-values of the *source* and the *target* do not as such have any particular impact on transmission frequencies, except that a greater longitudinal difference may imply considerably greater distance between the *source* and the *target* - and consequently, transmission between them may be influenced by other physical conditions such as *water-path, daylight-path* (see section 8.2.1).

Based on these observations we select five column dimensions for the cache: *source-latitude, target-latitude, longitudinal-difference, seasonal-effect* and *daylight-path.* However, considering that a cache with 5 column dimensions is somewhat complicated to deal with; and that the interesting values with respect to both of the features "seasonal-effect" and "daylight-path" are two in number, we have decided to use only the first three column dimensions to design the cache. In each cache cell, information regarding dealing with "daylight-path" and "seasonal effect" is then incorporated in the form of simple conditional statements. Thus in effect we have a 4-dimensional cache where the other dimension ("row") is for specifying different levels of computations.

8.3.1.3. Choice of the Cache Columns

Along each of the column dimensions we now select column headers that typify the expert's decision-making. The numeric nature of the selected primary features prompts us to use range-type values (see section 6.2.4.1.2) for the column headers. In this respect we observe that an expert while dealing with latitudes (of the "source" and the "target") relies more on the position of the locations concerned on the Earth (e.g. whether it is in the arctic zone or tropical zone) than the explicit numerical coordinates of the locations. Consequently, we select 5 columns along the two latitude-governed features (i.e. *source-latitude* and *target-latitude): N-arc* (Northern Arctic), *N-temp* (Northern temperate), *Tropical* (Equatorial region), *S-temp* (Southern Temperate) and *S-arc* (Southern Arctic). It is worth noticing that the five symbolic header values have an inherent ordering from north to south, in the given order. Consequently, these values are amenable to knowledge interpolation if necessary. For example, if a given location is within the "northern temperate" region but nearer to (i.e. the latitude difference is within some pre-determined limit, see figure 8.2) the "northern arctic" region, a user may consider selecting both the columns (with due weights) for retrieval and consequent interpolation on the retrieved information.

With respect to the feature *longitudinal-difference* we have noted that the difference in longitude values between two places can lie between 0^{o} and 180^{o}. We have divided this entire span into four groups: *0 - 20, 21 - 50, 51 - 100, 101 - 180.* But evidently, for longitudinal-difference values lying near the fringes one should be inclined to pick up the two adjacent

columns for cell selection. Hence for the purpose of the cache we use symbolic values (*A, B, C, D* respectively) instead of numeric ranges because use of range-type headers restricts the selection to a single column from the dimension (see section 6.2.4.1.3) and therefore restricts the use of interpolation. The other two primary indices *seasonal-effect* and *daylight-path* are considered with respect to the source. For each of these features we use two possible headers. With respect to "seasonal-effect" the two headers are: *winter* and *summer;* while for "daylight-path" the two headers that we use are: *daytime* and *night.*

8.3.1.4. Determination of Column Headers

In order to access the cache, the system needs to transform the "raw" featural values which are provided by the user as input. This can be achieved by calling certain functions and/or looking up relevant tables. One important source of information in this regard is: the *latitude-longitude table,* where we store the latitude and longitude values for different locations over the globe. In figure 8.1 we present a portion of the said table, where the latitude and longitude values have been stored as a list of triples containing the degree value, minute value and a direction, i.e North or South for latitude and East or West for longitude.[3] The same table can be used for identifying the latitude and longitude for both the "source" and the "target" locations.

Location	Latitude	Longitude	Region
Calcutta	(22 35 N)	(88 21 E)	S-Asia
Johannesburg	(26 10 S)	(28 2 E)	S-Africa
London	(51 30 N)	(0 10 W)	W-Europe
Melbourne	(37 50 S)	(145 0 E)	E-Australia
Newyork	(40 40 N)	(73 50 W)	N-America
Oslo	(59 55 N)	(10 45 E)	Scandinavia
Singapore	(1 20 N)	(103 50 E)	SE-Asia
Zurich	(47 23 N)	(8 33 E)	W-Europe

Figure 8.1: A Portion of the Latitude-Longitude Table

Additionally, the table contains which region on the Earth a particular place belongs to. We have in our database certain broad regions (e.g. West Europe, Scandinavia, East Europe, Middle East, South Asia) and depending upon the geographic position we assign each place to a particular region. This regional information is helpful in storing certain information pertinent

[3] In actual use for the experiments covered for this thesis, we have at least 100 entries for different cities all over the world.

for ascertaining certain transmission characteristics. For example, to ascertain whether a particular transmission path is primarily a path over water, we use the regional information such as that transmission paths from South Africa to West Australia or Europe to East America or Far East to South America are essentially covered by water. Storing such information using latitudes and longitudes is not practicable.

We now describe policies for computing the values for the five primary features, from the raw-feature values given as input, so that the best cache column(s) are selected.

Computation of Source-latitude

Evidently the relevant "raw" feature for this computation is the *source* (see section 8.3.1.1). Depending upon the *source*-information, the computation works in the following way:

The user establishes the *source*-information by giving the name of the location (e.g. London, Newyork, Calcutta) of the transmitter. The reasoning method first refers to the *latitude-longitude-table* to retrieve the latitude value (LatV) of the location concerned. The *source-latitude* value is then computed by using simple conditionals, e.g. "if the source latitude value is greater than 65° then the value is either *N-arc* or *S-arc"* (see section 8.3.1.3) depending on the hemisphere concerned; a source location with latitude value less than 23.5° is called *Tropical* irrespective of the hemisphere where it is found.

Computation of Target-latitude

Although the header values along this dimension of the cache are the same as the *source-latitude,* our system computes this value for a given problem with more careful attention to the details of the problem. This is due to the facts that:

- often a user needs to send messages to multiple locations. Consequently, the method needs to identify the relevant value for each of the target locations intended;
- transmission to a target situated near the fringe of a given target-latitude region (which can be computed by simple conditional statements as in the case of *source-latitude*) is normally influenced by transmission conditions prevailing in the neighbouring region. Consequently, the two related column headers are chosen with appropriate weights. (While the same is also true in principle for computation of the *source-latitude,* our expert did not consider it necessary).

Target is a single location: when the target is a single location, the method checks if the latitude-value is centrally located in one of the five ranges used as column headers, in which event a single column is chosen. Otherwise, two adjacent columns are selected with appropriate weights. Figure 8.2 explains the tactic for targets in the northern hemisphere. A similar

tactic should apply for targets in the southern hemisphere. The function works on the latitude value (LatV) of the target location. The two functions *degree(X)* and *direction(X)* compute the degree value and the direction (i.e. north or south) of a given location X.

```
IF direction(LatV) is "N" THEN
    If degree(LatV) > 70
    Then
 source-latitude  is N-arc with weight 1.0;
    Else
    If degree(LatV) > 60
    Then
 source-latitude  has two values:
          C1 =  N-arc with weight W1 = (degree(LatV) - 60)/ 10;
          C2 =  N-temp with weight  (70 - degree(LatV))/10;
    Else
    If  30 ≤ degree(LatV)  ≤ 60
    Then
 source-latitude  is N-temp with weight 1.0;
    Else
    If degree(LatV) > 20
    Then
 source-latitude  has two values:
          C1 =  N-temp with weight W1 = (degree(LatV) - 20)/ 10;
          C2 =  Tropical with weight  (30 - degree(LatV))/10;
    Else
 source-latitude  is Tropical with weight 1.0;
```

Figure 8.2: Simple Selection of Target-latitude Columns for Northern Hemisphere

If the problem value uses the relational modifier *near-to* (e.g. *near-to* Newyork), a single target location, given by the comparand (i.e. New York, here) is selected.

Target as a larger space given by a set of locations: Evidently, in this situation the method needs to consider the entire range of latitudes, as given by the set of target locations. The latitudes of the northernmost and the southernmost locations provide the range of target-latitudes that needs to be considered. Appropriate columns can then be chosen by considering the overlaps of the target range with the column headers. The two columns having the maximum overlaps are chosen for retrieval. Their respective weights can then be determined using the tactic explained in section 7.3.1.1. For example, for broadcasting from London to the target area of South Asia and South-East Asia, the computation considers the typical cities as Delhi (latitude 28^{o} 40' "N") and Kualalumpur (latitude 3^{o} 10' "N"). The target range of latitude, therefore, is 29 to 3,[4] in the northern hemisphere. The overlap of this range with the columns

[4] using *ceiling* and *floor*, respectively, to convert into integer values.

having header values "Tropical" and "N-temp" are 20 and 6 respectively (23^{o} "N", being the border). The weights of the two columns, therefore, are 0.77 and 0.23 respectively. Use of the relational modifier *between* is treated in the same way, considering the two comparands as the extremes of the range.

Computation of longitudinal-difference

The longitude values of the *source* and the *target* are considered, to compute their difference. If a single target is considered, then the difference is a single value. If the value is centrally located in any of the four column headers, that column is chosen with weight 1.0. When the *longitude-difference* for the current value is near the fringe of any of the header values, the method considers the two adjacent columns with appropriate weights. For multiple target locations, the method considers the entire range of values for the said feature, and two columns are considered (if necessary) with their weights adjusted using the overlaps.

Computation of seasonal-effect

Computation of seasonal effect, for retrieval from the cache, reduces to whether it is winter or not in the *source* location. From the latitude of the location concerned we first identify whether the source is in the northern or in the southern hemisphere. From the date of despatch we retrieve the month (M) when the transmission is to be made. Simple conditional statements on these two values then compute the value of the primary feature *seasonal-effect* for the current problem. The feature can have five possible values: winter, almost winter, between winter and summer, almost summer and summer. These values help in assigning appropriate weights for the two column headers along this dimension.

Computation of daylight-path

Value for the feature "daylight-path" can be calculated if the sunrise time (SRT) and sunset time (SST) are known at the source site. We assign the value *"daytime"* if the values of the raw features *begin-time* and *end-time* fall in between SRT and SST; we assign the value *"night"* otherwise. When the "begin-time" and "end-time" designate different daylight conditions (i.e. the transmission begins at daytime but continues beyond sunset or vice versa) we describe daylight-path by specifying the period during which it is a daylight transmission and the period during which it is a night transmission. The time periods one hour before the sunrise and one hour after the sunset are described as "dawn" and "dusk" respectively and during this time both the column-headers receive equal weight. In order to calculate the sunrise and sunset times we follow the algorithm by Hofmeister [Hofmeister 85] which computes the sunrise and sunset times for a given location (using the latitude and longitude information) for a given day.

Thus the column indices of each cell can be described uniquely by a set of five index values (I_1, I_2, I_3, I_4, I_5), where the j-th index value refers to the column number in the j-th column dimension. For example, by the index value set (2 3 4 2 1) one refers to the cell whose source-latitude is *N-temp*; target-latitude is *Tropical; value of longitudinal difference is D* i.e. in the range 100^0 to 180^0 (see section 8.3.1.3); value of seasonal-effect is *summer*; and the value of day-light path is *night.*

8.3.1.5. Determination Of Cache Levels

The temporal aspect of solving problems is taken into account in determining the levels. Unlike the AGC cache we have chosen four levels for the shortwave domain: *default1, default2, rule-level* and *case-level.* We have observed that in this domain often the requests are associated with such pressing time limits that no effective computation (including a default-level interpolation) is possible. Evidently, the solving method in such a critical juncture does not even consider two cells of the cache. Instead, the method selects the cache cell that receives the maximum weight under a given problem description. The output, therefore, is the stored values in the selected highest-weight cell. We call the cache level associated with the abovementioned scheme "default1 level".

In default2, the method considers two cells (selected by the tactic suggested in section 7.3). Here too, the solution consists of the right frequency and the power that will be required to perform the transmission. However, in suggesting the frequency and power, the method considers contents of both the cells and applies a suitable interpolation on them.

At the rule level, one applies rules to improve upon the default suggestions of the two selected cells. The rule level also takes up the issue of the choice of transmitter. It asks the user for any power limitation. If the suggested power is less than the available one, the method operating at the rule level suggests whether the same transmitter could still be used or it is advisable to try some alternative form of transmission (e.g. relay, use of a second transmitter).

Finally, there is the case level, where the reasoning method uses cases and therefore is able to make suggestions regarding not only whether a relay or a second transmitter will be needed, but also on the most effective way of utilising it. We have found that, in the context of shortwave radio propagation, an expert does not really rely on cases to suggest the standard transmission parameters (e.g. frequency, power). One normally tends to suggest these values from one's gross summarisation about these parameters considering various factors[5]; and/or their modification under certain conditions using rules. But one relies on specific past experiences only when an attempt to make a transmission fails, i.e. the transmission quality is not

[5] Evidently, the values in the default-level cells represent these summarisations.

acceptable due to presence of some disturbance (such as interference, fading); or the relay-path/telephone-path suggested by the expert is not attainable (due often to some non-technical problems, e.g. financial or political). As no clear-cut theory has been established to pinpoint the roots to these problems, an expert relies on past cases to solve them. Hence in our system, we expect to refer to cases only when there is some difficulty of the above nature.

Thus activities in the case-level are in the following line: the system first suggests the default-level solution - and then waits for a response from the user if there is any problem with the suggested solution.[6] Upon receiving the user's response the system selects appropriate cases to solve the reception problem.

Once the levels to be used are determined and the associated activities are designed, a designer of a cache-based system should concentrate on assigning the temporal bounds to different cache levels. These bounds will govern the access to an appropriate level when a time limit is imposed on the computation. Fixing of the temporal bounds for each level can be done only after repeating experiments a sufficient number of times (on a training sample), and estimating the average temporal requirements and other statistics (e.g. standard deviation, various percentiles) for each level. For each level of the radio cache, we have run experiments on 500 instances, to arrive at a decision about the temporal requirements. In chapter 9 we discuss the results obtained from our experiments on training samples, inferences drawn therefrom, and evaluation of the results.

8.3.2. Action-Related Knowledge

Under this category we place knowledge about the different types of permissible actions that are used for accomplishing the three tasks: *comp-frequency, comp-power* and *comp-transmitter.* Each of these tasks can be accomplished in several possible ways, and the system's choice of action depends upon the particular problem concerned. Here too, we organise knowledge in the form of a *task - generic action - action* hierarchy (as suggested in section 6.3.2) which facilitates action interpolation. We now describe the knowledge regarding the three tasks.

8.3.2.1. Computation of Frequency

Frequencies in shortwave public broadcast transmissions are classified into several bands depending upon the underlying frequencies, in megahertz (MHz). Each band comprises a set frequencies (most of which are divisible by 5, implying a standard 5 kHz spacing) within a

[6] In practice, ascertaining any such difficulty may take a significantly long time. However, for our system we assume a reasonably prompt reply from the user, which is not always realistic!.

specified range. In figure 8.3 we summarise relevant information in this regard.

Band (MHz)	Lower Limit (MHz)	Upper Limit (MHz)
2	2.3	2.495
3	3.2	3.4
4	3.9	4.0
5	4.6	5.1
6	5.875	6.25
7	7.1	7.525
9	9.4	9.99
11	11.6	12.05
13	13.6	13.8
15	15.05	15.7
17	17.5	17.9
21	21.45	21.85
25	25.74	26.04

Figure 8.3: The Frequency Limits for Each Shortwave Band

There are two major approaches to selecting the right frequency: *band-based* approach and *frequency-based* approach. In the band-based approach the system first determines the ideal band (B) that suits the prevalent condition best. Given the band information, a choice about frequency can be made from among the alternative frequencies (e.g. by avoiding frequencies where interference is likely). However, the task is not straightforward, as a chosen frequency may have already been allocated to some regular broadcasting organisation; or the frequency is known to be jammed at some specified location. The task for the system is therefore to search through a relevant database to find if a certain frequency is blocked for transmission (due to presence of interference) for the given target location at the given time (specified in the problem description). The search can be on the entire band or it can be restricted to the upper or lower region within the band limit. If all the frequencies in the band B are found to be unsuitable, the search is extended to the neighbouring bands (of B). Often an expert resorts to choosing a frequency that is not within the specified range of some band; rather, the frequency lies outside such a range. However, this action is recommended only when the purpose of the transmission is a short-term one and not a prolonged broadcast, and when the user thinks it is safe or desirable to ignore or bend ITU conventions.

The "frequency-based" approach applies when solving a regular broadcast problem. As mentioned earlier, the regular broadcasters often have some frequencies registered for them. Naturally, in such an event the choice of the frequency should be the one that is nearest to the ideal band (B). Thus the preassigned frequencies are considered sequentially in increasing order of their distance from the ideal band so that the frequency lies within the limits of the

maximum and minimum usable bands, until a frequency is found with an acceptably low level of interference. If all the preassigned frequencies are found to be unsuitable, the user's choice is limited to requesting some other broadcaster or relay centre (X) to carry out the transmission. The request can be of two types: request for a *relay* of the transmission when X receives the transmission from the source and retransmit it to the target area; and request for *announcement* when X temporarily halts its own current programming to broadcast the message requested by the transmitting location considered in the problem (evidently, this is applicable only when the purpose of the intended transmission is message-passing). From the specification of the problem the method can decide which of the two forms of request is relevant in a given context.

Figure 8.4 shows the organisation of the actions related to computation of frequency.

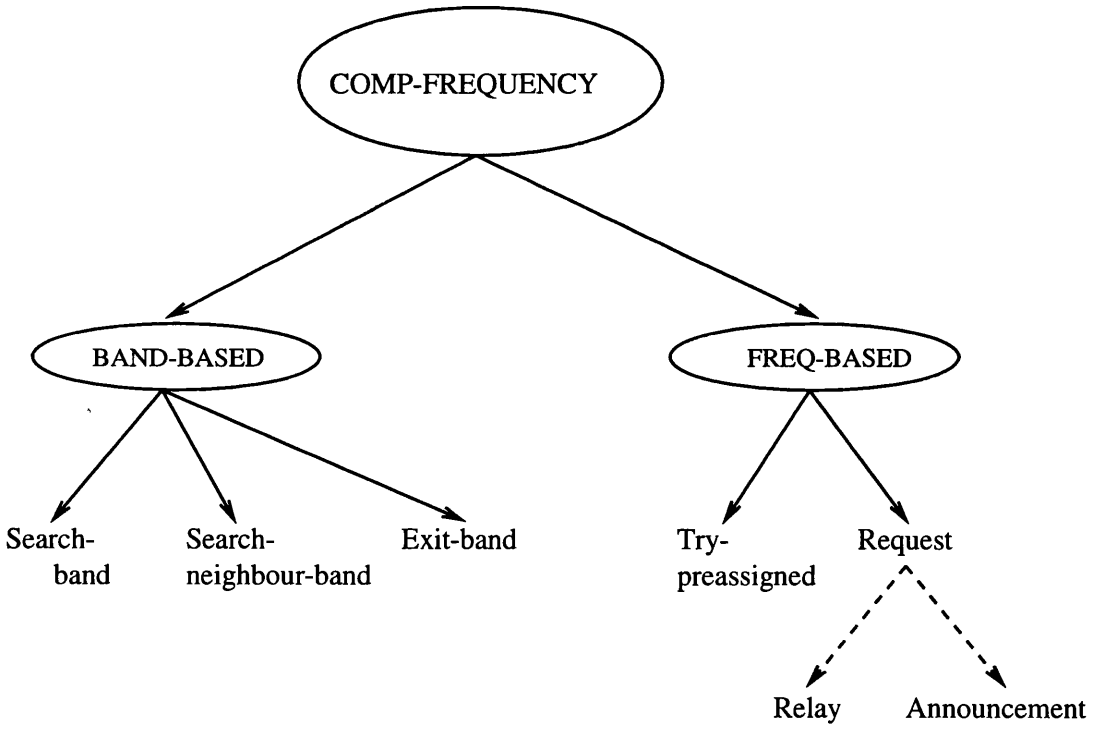


Figure 8.4: Organisation of Actions for Determining Frequency

In order to compute whether a particular frequency is blocked or not at a given time for a given location we maintain another table of information, called *blocked-frequency table.* A guide to radio frequencies such as [Bird 92] contains listing of all standard frequencies along with at which hours (in Universal Standard Time, UTC) of a day transmissions are made in those frequencies and from which location and for which target areas. However, storing such a huge table is not a very effective way of contributing to our research. In order to be more efficient (while remaining realistic) we store only segments of each band information in our

knowledge-base. In figure 8.5 we show the information that we have regarding band 11 in our "blocked-frequency table".

```
(11
   (when-blocked
          (11605 (1300 1345 300 (W-Europe Scandinavia) Tel-aviv)
                 (1800 1900 300 (N-America C-America) Tel-aviv)
                 (1600 2000 300 (W-Europe Scandinavia) Tel-aviv))
          (11620 (1730 2230 500 (W-Europe E-Europe) Bangalore))
          (11630 (1100 1230 250 (W-Europe) Sofia))
    ) ; <when-blocked>
  ) ;; <end 11>
```

Figure 8.5: A Portion of the Blocked-Frequency Table

The information stored in this table can be interpreted in the following way:

a) the frequencies that we are considering for this band are 11605, 11620 and 11630 kHz.

b) Transmissions are made on the frequency 11605 kHz as per the following schedule:

i) between hours 13:00 and 13:45 UTC, from Tel Aviv, for which the target areas are West Europe and Scandinavian countries. The power used for the transmission is 300 kW;

ii) between hours 18:00 and 19:00, from Tel aviv, for which the target areas are North and Central America using 300 kW transmitter;

iii) between hours 16:00 and 20:00, from Tel Aviv, for which the target areas are West Europe and Scandinavian countries. The power used for the transmission is 300 kW.

Blocked information for other frequencies in this band (and in general for frequencies belonging to other bands) can easily be ascertained from this table.[7] This information is required for searching a band. Similar information is stored for frequencies which lie outside the conventional band-limits.

8.3.2.2. Computation of Power

Determination of the necessary power to carry out the transmission job depends upon whether any upper limit exists on the usable power at the transmitting location. If the user does not specify any upper limit for the available power, the system is free to suggest any power that is considered best for the purpose. Hence the two relevant actions here are: *use-suggested-power* and *use-modified-power* where any suggestion and subsequent

[7] In actual use one may store additional transmission information such as the days in a week and the months in a year when a transmission is made. But we do not consider this inessential complication, in our implementation.

modifications may come from the cached entry and/or the cases and rules used for solving the problem. In situations when there is an upper limit on the usable power at the sender's site the action may be quite different, particularly when the suggested power is more than the available maximum power. The actions that a system may apply are:

- *use-suggested-power,* i.e. the suggested power is used (possibly because it is less than the upper limit);
- *use-maximum-power,* i.e. use whatever is available (the system resorts to this action only when the situation is "pressing");
- *use-boosted-power,* i.e. the system calls for the use of another transmitter with power such that the sum of powers of the two transmitters is not less than the suggested power; and
- *use-relay-power* when the system abandons the plan of transmitting from the site to the target area directly and instead recommends the use of a a relay centre (R, say) such that a transmission is made from the source site to the relay centre R which in turn relays its message to the target. Evidently, the system in this case needs to execute a prerequisite action - namely choosing an appropriate relay centre R. The action then also suggests the required power to transmit from the source site to R.

Figure 8.6 shows the organisation of the actions related to computation of power.

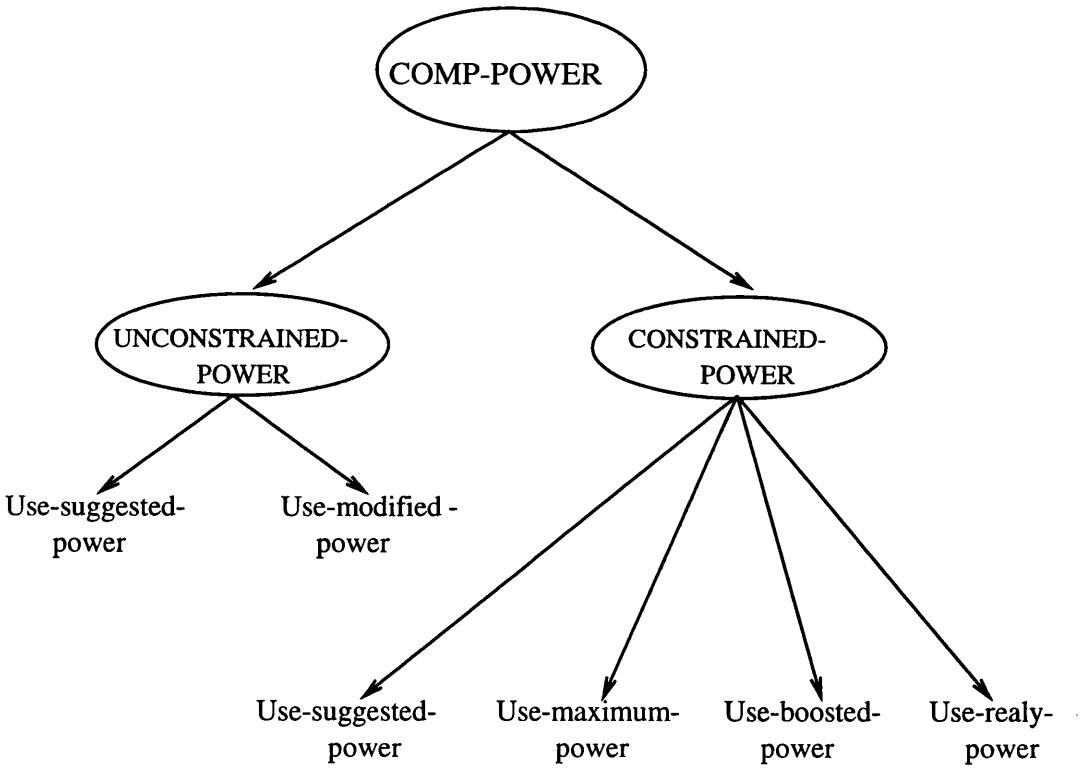


Figure 8.6: Organisation of Actions for Determining Power

8.3.2.3. Selection of Transmitter

Selection of a suitable transmitter that can perform the transmission with the suggested frequency and power can be achieved using one of the following actions.

- *use-local-transmitter:* when there is no restriction on power or when the suggested power is less than or equal to the available power, the system's suggestion amounts to using a transmitter that is available locally.

- *use-boost-transmitter,* when the system decides to use a boost transmitter or transmit remotely, it is necessary to find a transmitter with the required power. Corresponding to each transmission centre T, we maintain a list of other centres that can be connected to T using a reliable link such as a telephone line. For each of these centres, we maintain a record of the maximum power available. The given action selects from the said list one that satisfies the suggestion of power and frequency.

- *use-relay-transmitter* - the system resorts to choosing a relay transmitter if it has already recommended the use of a relay power. The action here essentially suggests the relay site R (selected while deciding the relay power) and recommends a frequency F which is at least 50 kHz away from the suggested frequency (to avoid undesirable coupling between receiving and transmitting equipment at R). This frequency F is used for transmission from the source site to the relay location.

Evidently, to carry out these actions we store information about the possible relay centres and telephone-links from each transmission location.

8.3.3. Reasoning-Related Knowledge

Reasoning-related knowledge covers how to reason with the cache's contents to derive the desired solution. Three major categories are:

1. isolation of the "secondary" features that represent more subtle or less important aspects of a problem or background details that are not described in the input. (We would expect these features to be used in the deeper levels of a cache where elaborate reasoning tactics are involved);

2. knowledge about contents of each cell and how to reason with them;

3. knowledge about interpolation, i.e. how various domain symbols can be used in carrying out interpolation to accomplish the goal.

8.3.3.1. Knowledge about Secondary Indices

Use of higher-order reasoning schemes such as rules and cases requires values of certain secondary parameters prevalent for the time of transmission as proposed in the current problem. The system therefore needs access to these values. In rule-level reasoning these values are used to check if any rule is directly usable in the current situation or if the current situation lies within the preconditions of two rules. The values of the relevant feature are used to ascertain the applicability of some rule(s). When reasoning is based on past cases, the system takes into consideration all the features (both primary and secondary) that characterise the current problem and the cases. For each relevant past case (pointed to by the cache) the system computes its composite distance from the current problem by suitably combining the distances with respect to each feature and choosing those case(s) whose composite distance(s) are at the minimum.

These secondary features can be of two types:

1. those for which values can be computed independently of any help from the user (provided sufficient knowledge is already stored in the system). For example, features such as grey-line effect, whether the path to be traversed by the wave is primarily water-path, or the reliability of the telephone-line at a given location, can easily be determined either by using some elementary computations or by table look-ups;

2. those that cannot be determined without the user's direct help. The level of solar activity, or the existence of any tropical disturbances on some particular day, fall under this category. In our approach, the system asks the user to provide the value. In order to make computations fast, the system provides the user with a selection of values and the user's task is to select the relevant value. For example, corresponding to the feature *solar activities* the choice of values is: *very-high, high, moderate, low;* and the user is expected to feed in the value that is deemed to be prevailing at the time of transmission.

In our approach, decision-making at the rule level does not require values of any features (except the maximum power available from the local transmitter) that needs a user's direct help, as this will inevitably be too expensive from the point of view of computational time. Use of such features is restricted to the case level alone. Thus "case level" is considered the deepest level for our cache.

8.3.3.2. Contents of the Cache

The default-level cells, as stated earlier, contains recommendations that can lead to ready-to-use solutions. With respect to the radio domain it implies suggestions for the best (i.e. the one considered to be most suitable) band for the conditions designated by the column indices of

the cell concerned. Since the method searches for suitable frequencies in adjoining bands too, in the event that no frequency providing an acceptably low level of interference is found in the best band, we also store the lower and upper limits of the band values within which the search is to be restricted. Thus the band information is a triple of numbers. In each cell we actually store four such triples in order that the retrieval procedure can consider the *seasonal-effect* and *daylight-path* prevailing at the time of transmission and make judicious choice. In a similar way we store the power information, in terms of kilowatts (kW) that is suitable for the given transmission condition. In figure 8.7 we show the contents of the default-level cell corresponding to the the following headers: *N-temp* (for "source-latitude"), *Tropical* (for "target-latitude") and *C* (for "longitudinal-difference", see section 8.3.1.3).

Season	Daylight-path	Best-freq	Min-freq	Max-freq	Power
Winter	Daytime	12	8	17	120
Summer	Daytime	15	9	19	110
Winter	Night	9	6	14	120
Summer	Night	12	8	17	110

Figure 8.7: Contents of a Default-level Cell for the Radio Cache

The rule-level cells contain default values with respect to band and power, as stored in the default-level cells. Additionally, they contain lists of any secondary features (e.g. solar activity, auroral activity) that are relevant for the cell concerned. For example, transmission from the tropical region to the northern temperate region is not affected by any ordinary auroral disturbances. Consequently, the secondary feature "aurora" is absent from the corresponding cache cell. Each cell also contains rule numbers which are relevant to the underlying state of transmission governed by the cell concerned. These rules (when found to be relevant) are applied to improve upon the default solution.

Each of the case-level cells contains (along with the list of relevant secondary features) a list of case-ids (see section 7.4.3) which are relevant for it. The method selects from them the relevant case(s) on which to apply interpolation. Also, each case-level cell has access to the default-level values of preferred band and powers.

8.3.3.3. Knowledge about Interpolation

While application of interpolation takes place in similar fashion to what is described in chapter 5, one notable difference from the tactics suggested while dealing with the AGC domain is absence of any interpolating feature such as "degree-of-effectiveness" which may provide a suitable dependent variable for interpolation. However, for each task the method has a list of actions arranged in accordance with the preference of the expert (see section 8.3.2).

Therefore interpolation of actions in reasoning with deeper levels of the cache (e.g. case level) may reduce to one of the following:

- a binary interpolation for choosing one action out of the two contending actions and selecting the appropriate parameters. For example, if the action for selecting the right frequency happens to be "*search-band*" in one case and "*exit-band :direction upper*" in the other, the interpolated action will be "*search-band :direction upper*" for taking care of the fact that higher frequencies have been found to be more effective in the given situation.

  or

- one selects an action from the preferred list (P) of actions by interpolating on the positions of the two given actions in the list P. The necessary parameters are determined subsequently.

8.4. CONCLUDING REMARK

Our summary of the new domain explains also the design considerations for a cache-based system for a new (i.e. non-AGC) domain. As we can see by comparing this chapter with our previous discussion of an architecture particularised for the AGC problem (chapter 6, 7), the basic characteristics of caching and interpolation are the same for the two.

Having designed the set-up, we now concentrate on experiments - i.e. on determining how the cache-based system performs. In chapter 9 we explain our experiments and the results we have obtained.

Chapter 9.

EXPERIMENTS AND RESULTS WITH SHORTWAVE RADIO CACHE

9.1. INTRODUCTION

In this chapter we discuss our experience with the radio cache that has been built according to the scheme presented in chapter 8. The experiments have been conducted on a representative set of problems that a full-fledged cache may be expected to handle. The examples chosen for this demonstration are sufficiently detailed to highlight the key aspects of the architecture, i.e.

- qualitative improvements in the solutions of a given problem, as one refers to deeper levels of the cache;
- the effectiveness of the interpolation technique in improving upon the default solutions;
- improvement in results by using parametric and action-level interpolations;
- use of interpolation on rules and cases;

  and
- the system's ability to respect temporal stipulations while solving a problem.

We illustrate the abovementioned characteristics by considering the following three transmission problems:

Problem 1: Transmission from Calcutta to London;
Problem 2: Transmission from Singapore to Los Angeles;
Problem 3: Transmission from Auckland to Johannesburg.

For each of these three problems we consider transmissions in the month of June from 11:15 hours to 11:45 hours UTC; and in the month of December between the hours 18:35 and 19:20 UTC. For each of the three problems we call these two "variation A" and "variation B".

9.2. OUTCOMES AT DIFFERENT CACHE-LEVELS

We first examine how the system reasons at each level and what solutions are generated thereby. In our discussion, we concentrate on four levels, namely reasoning at default1, default2, rule and case levels (see section 8.3.1.5) successively. We refer to a cache cell by specifying the set of five index values, as explained in section 8.3.1.4. For both the default levels the system sticks to the predetermined plan: choosing an appropriate frequency and choosing the right powers. Reasoning at the default level does not try to find whether a relay or boost transmitter needs to be used for carrying out the transmission task. Therefore when activities are within one of the two default levels of the cache, the system does not ask the user if any upper limit on the available frequency exists, and it does not investigate whether a direct transmission may be subject to interference. Thus solution at this level recommends using the transmitter available at the geographical location of the transmission. That is, it assumes that a transmitter with desirable power should already be in the possession of the user. These simplistic assumptions change when the activities are in a deeper level of the cache, as we shall see in section 9.2.3 and 9.2.4.

9.2.1. Reasoning At Default1 Level

At the default1 level the system considers only a single cell and reproduces the solution stored in the chosen cell. The input parameters are the source and destination along with the month and date and time of the day. In order to determine the frequency, the method considers the band suggested to be the best in the chosen cell, and then searches in its data-base the entire range of frequencies for the said band to find an unoccupied frequency for the given destination. The value of the recommended power is taken directly from the cached answer. The results that we get are given below.

In "Problem 1", we consider transmission from Calcutta to London. When the transmission request is of variation A (i.e. for the month of June between 11:00 and 11:30 (UTC)), the system chooses the cell indexed as (3 2 3 1 1) for retrieving information, and we get the following output:

```
Transmission from calcutta to london
Time of transmission (UTC): 11.15  to 11.45

Seasonal-effect:  summer
Day-light-path:   day

Use Frequency 15530 kHz
Use Power         110 kW
Use Transmitter local-transmitter
```

On the other hand, for a transmission request in the month December between UTC times 18:35 and 19:20 hours (i.e. variation B), the system chooses the cell (3 2 3 2 2) for retrieval and we get the following output:

```
Transmission from calcutta to london
Time of transmission (UTC):  18.35  to 19.20

Seasonal-effect:  winter
Day-light-path:   night

Use Frequency 9430 kHz
Use Power         120 kW
Use Transmitter local-transmitter
```

(It is interesting to note that All India Radio actually beams its General Overseas Service to Europe at this time of the day and year in the 9 MHz band, on a frequency that is actually not as free from interference as 9430 kHz - probably the force of tradition is responsible for the lack of change!)

The computations for "Problem 2" give the following two suggestions for the two variations respectively:

```
Transmission from singapore to los-angeles
Time of transmission (UTC): 11.15  to 11.45

Seasonal-effect:  summer
Day-light-path:   night

Use Frequency 11605 kHz
Use Power         120 kW
Use Transmitter local-transmitter
```

and

```
Transmission from singapore to los-angeles
Time of transmission (UTC):  18.35  to 19.20

Seasonal-effect:  winter
Day-light-path:   night

Use Frequency 11620 kHz
Use Power         135 kW
Use Transmitter local-transmitter
```

where, the selected cells have been (3 2 4 1 2) and (3 2 4 2 2), respectively.

Finally we consider "Problem 3". The selected cells for the two variations have been (4 4 4 2 2) and (4 4 4 1 1). It is worth noticing that for this problem the values of seasonal effect and

day-light path change due to the source being in southern hemisphere. The solutions suggested by the system for the two variations are:

```
Transmission from auckland to johannesburg
Time of transmission (UTC): 11.15  to 11.45

Seasonal-effect:  winter
Day-light-path:   night

Use Frequency 9425 kHz
Use Power         110 kW
Use Transmitter local-transmitter
```

and

```
Transmission from auckland to johannesburg
Time of transmission (UTC): 18.35  to 19.20

Seasonal-effect:  summer
Day-light-path:   day

Use Frequency 15615 kHz
Use Power     100 kW
Use Transmitter local-transmitter
```

While Radio New Zealand does not have any programmes specifically for South Africa, in fact the best receptions reported in 1995 to radio club magazines at the specified times are in fact on the same band as for the frequencies given in the results above.

9.2.2. Reasoning At Default2 Level

Reasoning at default2 level considers two cells selected by the cell-selection method described in section 7.3. First the solutions are made from each of the two cells and they are interpolated (using the weights of the selected cells) to produce the final suggestion. Thus the interpolation involved at this level is parametric in nature. For illustration, we consider the problem 1, with variation A. Reasoning at default2 level chooses the following two cache-cells: (3 2 3 1 1) and (3 1 3 1 1) with respective weights approximately 0.52 and 0.48. The interpolation now takes place in the following way:

To compute the frequency, the system considers the best bands suggested in the two selected cells. For the given problem these are band 15 and band 17 respectively, with the search to be conducted on frequencies near to the upper limit of each band. These two values are now subjected to weighted interpolation to determine the ideal frequency - or to be more precise, to determine the best band and which region of frequencies should be searched for a unoccupied frequency. The interpolated result is *(17 'lower)*, suggesting that the lower-order frequencies of band 17 are ideal for the job. The system then searches for an unoccupied frequency in the

specified zone, and the result thus obtained is 17510 kHz. Computation of power is performed using straightforward weighted numeric interpolation on the recommended powers i.e. 110 kW and 120 kW, and then rounding it off to nearest multiple of 5.[1] Thus use of the second cell in decision-making has resulted in a higher frequency and higher power. This is because although in the "Northern Temperate" zone, the latitude value of London is quite high, and therefore some influence of the considerations for transmitting to the "Northern Arctic" zone have involved. The final suggestion is computed as:

```
Transmission from calcutta to london
Time of transmission (UTC): 11.15  to 11.45

Seasonal-effect:  summer
Day-light-path:   day

Use Frequency 17510 kHz
Use Power         115 kW
Use Transmitter local-transmitter
```

Similar improvements over the results obtained through reasoning in default1 level have been obtained in our experiments with other problems. For illustration, the solution given to version B of "Problem 3" at this level reads as:

```
Transmission from auckland to johannesburg
Time of transmission (UTC): 18.35  to 19.20

Seasonal-effect:  summer
Day-light-path:   day

Use Frequency 15090 kHz
Use Power         110 kW
Use Transmitter local-transmitter
```

The result that have been obtained from the cache have been approved to be of reasonably good quality by our experts. For an arbitrary user to ascertain the quality of the cached answers, we suggest comparing the result obtained from the cache with commercially available tables for selecting suitable bands as given in [Jacobs 92]. In these tables best MHz values for transmission between two regions of globe (such as Europe, SE-asia) are tabulated for different time periods of a day (normally expressed as four-hour intervals) and for each month of the year. However, these tables are not foolproof, and not all the frequencies in the band behave in the same way in accomplishing a transmission task, as they may be subject to interference by some other regular broadcast at the same time. But experts' comments on the viability of the frequencies suggested are based on their experience and observations on various past transmission exercises and are therefore more reliable.

[1] Powers of transmitters are normally specified here in multiples of 5.

9.2.3. Reasoning At Rule Level

Reasoning at rule level first checks whether the suggested default plan should work for a current transmission problem. Then the method resorts to one of the following policies:

1. If the default plan seems workable, the system executes the same plan for the two selected cells, and subsequently interpolates on the two solutions. However, unlike the situation in default2 level reasoning, here the method improves upon the suggested solution by considering domain-dependent rules (associated with the cells under consideration), and also deals with the issue of making a good choice of transmitter. In order to ascertain the best way of carrying out the transmission, the system asks the user whether the available local transmitter has any associated upper limit of power. In the event of no such existing stipulation, or when the suggested power is less than the maximum usable power of the transmitter, the method suggests using the local transmitter (as in the default levels). Otherwise, depending upon the nature of the transmission, the system proceeds as follows:

a) when the nature of the transmission is *broadcast*, the method has to look for a boost transmitter available locally (see section 8.3.2.2), failing which it suggests "use-boosted-power", i.e. the method now tries to locate some other transmitter that can be connected using a telephone or some other better-quality line to the source. However, before resorting to a solution that suggests connecting through a telephone, the method applies domain-dependent rules such as:

Rule17: IF problem type is *broadcast*
AND
telephone line in the source region is *dependable* [2]
THEN use telephone.

If the rule applies in the context of the given problem, the system suggests use of telephone for connecting to the available centre. Otherwise, the only option open to the system is to use a relay, which is solved as given in 2. below.

b) The abovementioned tactic varies slightly if the situation is *pressing* (see section 8.2.3). In this situation the system does not aim at achieving a disturbance-free transmission. Hence, it first checks if the available maximum power is significantly lower than the recommended one. If that is not so, it suggests using maximum power. Otherwise, the

[2] Evidently, the dependability of the telephone line is a secondary feature which the system needs to evaluate for the given source region. In our implementation, we maintain a table to determine this value, for a given location.

system aims at establishing a telephone link. Selection of the telephone link is now governed by the following rule:

Rule18: IF problem type is *pressing*
AND
telephone line in the source region is *more-or-less dependable*
THEN use telephone.

When none of the options is workable, here too as above the system suggests transmitting by use of a relay.

2. When the conditions stated in a current problem and/or the non-availability of a suitable transmitter demand a relay-based transmission, the system switches to a new plan. Under this new plan the system needs first to suggest an appropriate relay centre (R). Subsequently, it determines the best frequency and power to transmit from the source location to R, and also for transmitting from R to the target location (where both source and target locations are given as part of problem specification).

9.2.3.1. Use of Rules at Plan Level

In order to determine whether to continue with the default plan or to switch to a relay-based transmission, our method continues a binary interpolation. The key secondary feature that dictates this decision is: *polar-influence,* i.e. whether the transmission is likely to pass through either of the two poles, or some considerable fraction of its path should cover areas in the vicinity of a pole (i.e. *auroral zones).* Whether a transmission path will be influenced by a pole depends upon the latitude values and the longitude values of the source and target locations. For example, if both the source (S) and the target (T) locations are near a pole (i.e. either both S and T are in the Northern Arctic zone or in the Southern Arctic zone), then even a small longitudinal difference may make an important change in the amount of influence of an auroral zone on transmission. The physics of wave propagation suggests that the closer the source and/or target is situated to the equator, the greater should be the threshold on longitudinal-difference, to consider a relay-based transmission. If both source and target are located near the equator, a longitudinal-difference of about 100^{o} should suggest consideration of a relay-based planning for transmission.[3]

Use of standard spherical trigonometric formulae can ascertain whether the transmission path between a given source and target passes through the vicinity of a pole. The result of the computation is summarised here using fuzzy description such as *possible, fairly possible.* Depending upon the nature of the problem (i.e. whether it is a broadcast or pressing type), the

[3] Therefore, plan-level rules that determine whether a relay-based transmission is necessary are included only in a subset of the cache cells.

system fires different rules that suggest the relevant thresholding schemes for deciding on the plan to be executed. For example, we have the following two rules for choosing a type of plan:

Rule01: IF problem type is *broadcast*
AND
polar-influence is *fairly possible*
THEN use relay-based transmission.

and

Rule02: IF problem type is *pressing*
and
polar-influence is *possible*
THEN use relay-based transmission.

Thus plan-level interpolation in this reasoning paradigm resorts to a binary interpolation that selects one of the two alternatives.

When the version B of "problem 3" is treated in the rule level, the system suggests a relay-based transmission. The ultimate solution that the method computes is:

```
Transmission from auckland to johannesburg
Time of transmission (UTC): 18.35  to 19.20

Seasonal-effect:  winter
Day-light-path:   night

use-relay-transmitter  -  where  the  relay-centre  is  colombo

Transmission from auckland to colombo
Use Frequency 11630 kHz
Use Power     100 kW
Use Transmitter local-transmitter

Transmission from colombo to johannesburg
Use Frequency 11605 kHz
Use Power     60 kW
Use Transmitter local-transmitter
```

It is worth noting that once the method indicates a relay-based transmission, then for computing the required parameters for the two phases of transmission (i.e. from source to relay-centre, and from relay-centre to target) it resorts to some shallower level of the cache (in order to avoid increasing the time it needs for computations). In the above example, we have used default2-level calculations for estimating the frequencies and powers that would be appropriate for transmission between the two segments. In full operation of the cache-based system, the method should decide on the basis of the remaining time which one of the two default levels is to be accessed.

9.2.3.2. Use of Rules at Action Level

Let us consider variation B of "problem 1". The latitude and longitude values of the source and destination (i.e. Calcutta and London respectively) suggest that the transmission path does not intersect with the polar region. Hence our method adheres to the default plan. However, in order that a decision can be taken more quickly regarding whether this plan needs to be abandoned, the method first computes the required power. Subsequently, upon receiving a feedback from the user regarding a possible upper limit, if any, of the available power, the system decides its subsequent actions.

Experiment 1.

Suppose we assume that the transmitter under consideration has an upper limit of 100 kW for the available power, but it is also known that the transmitter centre can be connected through a reliable telephone line to some transmitter so that the effective power can be boosted up. In this arrangement, the result that we obtain is as follows:

a) For estimating the required power, the method uses the following two rules:

Rule14: IF water cover of the path is *more or less high*
THEN increase the default power by 10%.

and

Rule15: IF auroral spread of the path is *between 4^o and 15^o*
THEN increase the default power by 20%.

The method applies the rules on the default values suggested in the cells that it has selected. In this problem the two cells are: (3 2 3 2 2) and (3 1 3 2 2) with respective weights 0.52 and 0.48 (see section 9.2.2).

When applied to the cached value of the first cell (which is 120 kW), the two rules produce suggestions 132 and 144 kW respectively. The same rules, when applied to the cached value obtained from the second cell, produce the suggestions 121 kW and 132 kW respectively. Weighted interpolation on these suggestions yields the required power to be 132.48 kW. However, in order to express the value as a multiple of 5 (see footnote 1 of this chapter), our method modifies this value to 130 kW.

b) From our assumption (i.e. maximum usable power is 100 kW), the method therefore decides that a boosted-transmission would be required.

c) The method next tries to estimate the required frequency. The relevant rules in this regard are:

Rule11: IF effect of Greyline is *nil*
THEN use the suggested *best band.*

and its complementary rule,

Rule12: IF effect of Greyline is *100%*
THEN use the suggested *lowest band.*

and a third rule which reads as:

Rule13: IF water cover of the path is *more or less high*
THEN use the band which is *one band higher than the best band.*

Since the transmission time is such that the effect of greyline is 100% (see section 8.2.1), we find the rules 12 and 13 apply completely in the given context. The result of application of the two rules for the selected first cell suggests the required band to be 7 and 10[4] respectively, while the corresponding results obtained from the second cell are 4 and 8. The weighted interpolation on these values suggest the use of a frequency from the lower end of band 7. Our method now searches the data-base to obtain an unblocked frequency for transmission to London. and the result obtained is 7110 kHz.

The overall suggestion that we receive from the system, therefore reads:

```
Transmission from calcutta to london
Time of transmission (UTC): 18.35  to 19.20

Seasonal-effect:  winter
Day-light-path:   grey-line-time

Transmission from calcutta to london
Use Frequency 7110 kHz
Use Power     130 kW
Use Transmitter boost-transmitter-via-telephone-line
```

By contrast, for the sake of experimentation, when we looked for a solution for a similar transmission by advancing the transmission time by 1 hour (so that the effect of grey-line is considered 60%), we find that Rule11 and Rule12 get weights of 0.4 and 0.6 respectively, causing the selection of the appropriate frequency to be 7510 kHz, i.e. a frequency from the upper end of band 7.

[4] Officially, band 10 has no meaning, e.g. see figure 8.3. But for the purpose of interpolation it is considered as an interpolating value.

Experiment 2.

Here we deal with the same problem but the assumption is that the telephone connection in the source location is not good enough to make a good-quality transmission (possibly more realistic for Calcutta!!). Now the method abandons the possibility of a direct transmission, and the recommendation thus obtained is as follows:

```
Transmission from calcutta to london
Time of transmission (UTC): 18.35  to 19.20

Seasonal-effect:  winter
Day-light-path:   grey-line-time

abandoning direct transmission because not sufficient power
use-relay-transmitter - where the relay-centre is dubai

Transmission from calcutta to dubai
Use Frequency 9965 kHz
Use Power     70 kW
Use Transmitter local-transmitter

Transmission from dubai to london
Use Frequency 9430 kHz
Use Power     105 kW
Use Transmitter local-transmitter
```

These solutions clearly indicate the qualitative improvements in the solutions with the use of domain-specific rules. As indicated earlier, the improvement is not confined within a better estimation of frequency and power, it also considers the feasibility of a direct transmission and readjusts the transmission plan by suggesting the best mode of transmission.

9.2.4. Reasoning At Case Level

In the case level, the reasoning tactic refers to relevant cases for resolving any transmission difficulties. The past cases to which an expert will refer do not necessarily describe transmissions between the same source and target as in a current problem. Past cases, therefore, cannot be used directly to estimate the key parameters (i.e. frequency and power). Hence, the method refers to the past cases for designing only the overall strategy to overcome difficulties. The actual values for frequency and power need to be determined from the context of a current problem alone. For initial estimation of the key parameters the method refers to the cached answers. Hence the overall solving procedure at this level follows three primary steps:

1. Estimation of initial values for the key parameters;
2. Asking from the user any difficulties with the proposed solution;

3. Retrieval of appropriate cases;
4. Adaptation of the cases for the current situation, using knowledge interpolation.

As mentioned earlier, getting the feedback for a proposed transmission (in step 2) may take a considerably longer time, but for the sake of experiments we assume a prompt reply.

9.2.4.1. Estimation of Key Parameters

The cache contains values for the key parameters corresponding to two seasons: summer and winter. For shallower levels of the cache, the method decides, using some conditional statements, from the month and the latitude of the source whether the time can be considered as summer or winter. But in reality, there are many intermediate situations such as *almost summer, between summer and winter.* Our method enquires of the user how one should describe the transmission season of a current problem. The method then interpolates (from the user's description of the season) on the cached answers for summer and winter and determines the estimates.

For example, to carry out day-time transmission from Calcutta to London, the cached values for summer are: 9, 21, 15 and 110 suggesting the lowest, highest and best bands (in MHz) and the ideal power (in kW) respectively. Values for the same parameters for winter are: 7, 17, 11 and 120. If the transmission season is specified as "between summer and winter", the method interpolates on the summer and winter values of the same parameters with equal weights. The result we obtain therefore is: 8, 19, 13 and 115 representing the four key-feature values respectively. On the other hand, if the transmission time is specified as "almost summer", the weights on the cache cells concerned alter, and the result thus obtained is: 8.8, 20.6, 14.6 and 111. [5]

Having estimated the key values, the method now looks to retrieve appropriate cases for guidance towards solving a given difficulty. As mentioned in section 8.3.1.5, the difficulty can be either unacceptable audio-quality or difficulty in transmitter allocation.

9.2.4.2. Retrieval of Appropriate Cases

Each cache cell contains a list of cases that are relevant for the transmission conditions (i.e. time of the day, season) etc. Ideally, some of the stored cases should match with the current problem in the source and target as well. However, practically it is not possible to collect cases matching every situation. Hence, with respect to the source, we determine for each case

[5] It should be noted that the decimal values in the bands do not have any realistic significance as such, but in our system we use this fraction to determine the frequency range within a band's limit where the search for a frequency that is free of likely interference should be made (see Section 8.3.2.1).

the geographical distance of its source location from the source location stated in the current problem. This distance is then suitably normalised by dividing it by the half of the globe's circumference (i.e. the maximum possible distance between two locations on the Earth). Regarding the target, we compare cases not by the exact location but by the region (see figure 8.1) containing these two locations. We select only those cases that correspond to transmissions to the same region as specified in the current problem. [6]

The ultimate selection can then be made by taking these factors into account, and some secondary features whose natures depend on the type of difficulty. Corresponding to the transmitter-related difficulties, the relevant features may be: *upper limit of usable power, the reliability of telephone line* (in the source location), *list of relay centres* (which may be connected to the source) etc. But for the difficulty "unacceptable-audio", the relevant features are: *degree of solar activity* (at that particular time) and *presence of tropical disturbances.* In the absence of any appropriate knowledge-base, the system depends on the user's feedback in this regard. Users in this field normally do not have any precise method of quantification. Hence the system accepts symbolic values such as *high, low* or a more fuzzy description such as *rather high.* These values are then used to select cases from the identified list of cases. The system now applies interpolation on them.

For illustration, let us consider the version A of "problem 1". The method first suggests default2-level solution: frequency 17510 kHz and power 115 kW (see section 9.2.2). But this results in poor audio quality, and the factor has been diagnosed to be a *rather high* nature of solar activity. The associated cases that the system retrieves (for "unacceptable-audio") are:

(a) from Karachi to London, where solar activity has been described as high;

(b) from Tehran to London, where the solar activity has been "low";

(c) from Colombo to Moscow, where the solar activity has been "very high";
and

(d) from Kazakhstan to Frankfurt under a "very high" solar-activity condition.

We first measure the distance of each case from the current problem. The overall distance of each case is: the sum of the square of distance along each relevant parameter i.e. source, target and solar-activity level. We then choose the two cases having least distances from the current problem. By using the normalised geographic distance (described above) for both source and target, and the fuzzy-distance for solar-activity level, we find that the case (a) and case (d) are selected for solving the current situation. The weights of each case are then normalised and the respective weights assigned to case (a) and case (d) are 0.74 and 0.26 respectively.

[6] However, if no two cases having the same target region are found, here too the distance ought to be measured geographically as in the case of the source location.

The cases are now subjected to "case interpolation" for deriving a solution for the current problem. Evidently, because of the weights attached, the method selects case (a) to be the "guide case" for performing interpolation, and the case (d) is selected to be the other interpolating case. Descriptions of the cases and how the interpolation works are illustrated in the following section.

9.2.4.3. Case Adaptation Through Interpolation

The two cases we are considering here have both dealt with the difficulty in audio quality by various trials with frequencies and/or power. The two cases read as follows:

Case (a): This is a problem of planning a broadcast from Karachi to England (London). The transmission on the usual frequency has been marred due to sudden high solar activity leading to a geomagnetic storm, and the authorities have to find a quick temporary solution.

Such occasional high solar radiations, in an otherwise normal situation (as described by the *sunspot number* (see section 8.2.1) unsettle the ionised layers causing disturbances in transmission. Since the higher layers are more likely to receive the bulk of the additional solar discharge, they are more likely to be unsettled. Hence, an expert hopes that lower-altitude ionised layers may have a better chance of behaving predictably and therefore decides to try whether a frequency can be allocated in a lower-frequency band.

An attempt therefore has been made on frequency 6245 kHz. However, it was found that the this frequency is blocked by deliberate jamming. Hence this frequency was ignored and a next attempt was made in the next band (in order of increasing frequency). In this case, it has been 7025 kHz. This frequency and the ones around it have been found to be heavily occupied by many amateur radio operators in West Europe. The third attempt, on frequency 9133 kHz, however, has been found "ok". But this frequency is normally booked for special non-broadcast use, such as aeronautical communications. Hence an unblocked legitimate frequency in the neighbourhood of this frequency has been looked for. The final solution has been 9440 kHz. The power has been calculated to be as high as 240 kW. However, it was found that the maximum capacity of the transmitter has been 200 kW. Considering that this is much less than the suggested requirement, the method suggests a boosted-power transmission, so that another Pakistani transmitter of about 40 kW should be used simultaneously to broadcast the same program on the same frequency.

Case (d): This has been related to a broadcast from Kazakhstan to Germany. Under a similar circumstance as in case (a), except that the degree of sudden solar activity is described as "very high", the authority first determined the appropriate power to be 300 kW. They first tried to use 5260 kHz, which is one of their standard frequencies for domestic broadcasting. There has been no reception at all. An attempt on a higher frequency, actually 7470 kHz, has been found to be unclear in reception. Transmissions at a still higher frequency, 9150 kHz (in

the 9 MHz band), have been very noisy, possibly due to unusual solar activities. The authorities therefore understood that good audio quality could not be ensured by direct transmission. Hence they tried for a relay. Ankara was selected as the relay centre, as it is between the source and target and the transmission on 9 MHz frequency has been clearly audible there. The consequent actions (see section 5.5.3.2.1) therefore are determining the frequencies and powers for each phase of the transmission. The final solution therefore reads as: transmission on 9 MHz from Kazakhstan to Ankara using power 250 kW; and relay transmission from Ankara to Germany on 11710 kHz using 120 kW. 11710 kHz has been selected because that is one reserved by the ITU for regular Turkish international broadcasting.

Having selected these two cases, the method now turns towards interpolating between them. At this point the method distinguishes between the two types of problems: *broadcast* and *pressing* (see section 8.2.3). The requirements for broadcasting companies is assurance of a strong signal with a high audio quality. Normally these organisations are equipped with powerful transmitters and/or access to relay centres - hence allocation of power and transmitter is not the main concern here. However, they are often constrained about the frequency (as only certain frequencies are earmarked for each organisation). Also, they are normally stipulated to restrict their transmission within a specified band. Using a frequency outside a band is not recommended from them. On the other hand, in pressing situations a good audio quality is not mandatory. But in these situations, one is normally forced to use a limited-capacity transmitter, and resorting to a relay is not always possible here. The case-interpolation routines are flexible enough to take appropriate decisions depending upon the nature of the given problem. The overall interpolation now follows the algorithm given in section 5.5.3.2.5.

Case Interpolation When Broadcast

With respect to the problem under consideration (i.e. version A of "problem 1"), we get the following answer when the problem is considered to be a *broadcast.*

The method first takes the purpose of "determining frequency". Having considered all the actions taken in case (a) in this regard, the method decides that a lower-order frequency in band 9 should be the best choice. The method now tries to select an interpolatable action from case (d). However, the attempt at selecting a frequency has ended in a failure in this case. Consequently, the method is forced to make a binary interpolation between the two options (i.e. whether or not to use band 9). Since the problem is a "broadcast", the method switches into a pessimistic mode, and the interpolation therefore suggests that we should not expect a direct broadcast on a single frequency.

The subsequent actions of case (a) therefore become irrelevant, and the method now makes decisions from case (d) alone. Hence the next action that the method carries out is *find-relay-centre.* We assume that the solution given in case (d), (i.e. Ankara) cannot be used directly,

possibly due to insufficiently speedy handling of the request. Hence the discrete selection method is applied on the list of possible relay centres associated with Calcutta.[7] The most appropriate relay centre is thus identified to be "Tel Aviv". The method now determines the other necessary parameters from the case (d). For transmitting from Calcutta to Tel Aviv, the power is recommended as 240 kW. The frequency for this phase of transmission is recommended as a low-end frequency on band 9. A search of our stored database suggests the frequency 9425 kHz for this purpose. The ideal relay-power is taken directly from case (d), and it is 120 kW. The frequency for this phase of transmission is derived to be in the lower part of band 11, and a database-search results in selecting the frequency 11605 kHz for this purpose. The final interpolated solution therefore reads as:

```
Transmission from calcutta to london
Time of transmission (UTC): 11.15  to 11.45

Seasonal-effect: summer

No direct transmission is possible due to solar activity: high
use-relay-transmitter  -  where  the  relay-centre  is  telaviv

Transmission from calcutta to telaviv
Use Frequency 9425 kHz
Use Power     240 kW
Use Transmitter local-transmitter

Transmission from telaviv to london
Use Frequency 11605 kHz
Use Power     120 kW
Use Transmitter local-transmitter
```

Evidently, the solution can be improved further by referring to appropriate rules/cases for each phase of transmission. But in our current approach, the level of complication above is enough to demonstrate the essential operations with cache and interpolation.

Case Interpolation When Pressing

The method proceeds in a similar way to that given for the "broadcast" situation. However, since the situation is pressing, the method switches itself to an optimistic mode. As a result, it does not recommend a relay; instead it recommends a direct transmission on a low-end frequency of band 9. Thus the recommended frequency is decided (having searched the database) as 9425 kHz. The method now tries to estimate the power. Case (a) suggested 240 kW when the solar activity was "high", while case (d) recommends 300 kW when the degree of solar activity is described as "very high", to make a direct transmission. Hence for the current situation (where the degree of solar activity is "rather high") the method suggests the ideal

[7] The method orders the possible candidates on the basis of their distances from Ankara.

power to be 190 kW, by interpolating on (*very high, 300*) and (*high, 240*) for the value *rather high.* [8] But here again it is found that the source does not have sufficient power to match this requirement, as the maximum available power was recorded as 150 kW. Considering that it is a "pressing"-type problem, the method suggests simply the use of the maximum possible power, with the warning that the reception quality is likely to be poor.

9.3. ESTIMATION OF TEMPORAL REQUIREMENTS

The above experiments clearly show that the quality of the answers varies with the underlying reasoning paradigm. Evidently, better-quality results take a longer time to compute. To make the cache-based system perform as per expectations, we therefore need to estimate the temporal requirements for computations under each reasoning paradigm. Our aim is to determine from these estimates the time limits that should be attached to different cache levels, and then test the viability of the cache-based architecture in maintaining the temporal bounds while producing solutions.

We envisage three possible ways of estimating temporal requirements:

- Measuring the actual clock time.
- Exact temporal requirements measured in terms of CPU cycles.
- Measuring internal run time, which includes not only the computing time of the given task, but also many behind-the-scene activities that include file management, overlaying, managing memory, I/O processes etc. The speed of the processors is also a significant factor here, unlike what happens when it is measured in terms of CPU cycles. We call these background activities "uncontrollable" factors.

Measuring in terms of actual clock time is not suitable in domains where computing needs significant user feedback, which can be viewed best in terms of seconds (if not minutes), while computing time is a matter of milliseconds. Hence even a negligible delay on the part of the human user may cause upset the temporal estimation significantly. On the other hand, measuring in terms of computing cycles is too one-sided, as it discards other activities going on in the computing environment completely. Although these activities have no direct relevance in the intended computation, they often prolong them. Therefore, to be realistic, one cannot ignore these effects, and any measurement of temporal requirements should take them into account. Hence we calculate estimations and temporal bounds on the basis of the internal run time. [9]

[8] The numeric equivalents (see section 5.3.2.1) have been considered for interpolation involving the fuzzy expressions.

[9] However, in order to minimise the effect of background processes we ran experiments during weekends when the computational load is expected to be rather less and also to show less short-term variation.

Hence the time requirement (R) for an experiment can be expressed as:

$$R_i = C_i + U_i + \varepsilon$$

where, R_i is the requirement for the i-th experiment, C_i is the computational requirement, U_i is the uncontrollable factor of time, due to the computational environment, and ε is the error term. We do not see any ways of getting separate estimates of C_i and U_i. Hence we want to estimate the total requirement (i.e. C + U), and like to choose the bounds for different cache levels based on them.

To estimate the temporal requirements for reasoning under each paradigm, we have tried experiments for different source and target locations. We have also excluded the time that is required to feed in the problem specifications - as this is not part of the reasoning process. (Another explanation is that normally the cache-based problem-solving tactic should work in conjunction with some real-time processing environment, where the cache will be accessed as and when a new problem occurs. Naturally, the problem will be fed to the reasoning process on-line and not through a human operator, unlike the situation in which our system has been developed).

Our experiments have been for the following types of computations:

1. Default1, reasoning with entries of one cache-cell alone.
2. Default2, reasoning with two cache cells.
3. Rule-1 - when the transmission conditions do not force the method to change the default plan at all;
4. Rule-2 - when the transmission conditions suggest continuation with the default plan but inadequacy of power later suggests finding a boost transmitter which can be connected with the source via a telephone line;
5. Rule-3 - when the transmission conditions suggest replanning right at the outset, and therefore the method needs to look for a relay centre, and then to compute powers and frequencies for the two phases of transmission - which is achieved by resorting to default2 level of reasoning.
6. Rule-4 - when the transmission conditions do not recommend a relay as such, but inadequacy of power in the transmitter and an unreliable telephone network at the source location forces abandonment of direct transmission and suggests relaying instead.

   and
7. Case - when transmission using a recommended frequency and power does not succeed, due to some natural disturbances, and the method therefore needs to refer to some past cases where a similar problem has been dealt with.

In figure 9.1 we present a summarisation of the results obtained from our training data generated by running each experiment (i.e. with each of the seven reasoning types) 500 times.

Here, time has been measured in terms of milliseconds (ms). Statistics of different types that have been calculated have their obvious definitions, excepting (possibly) *coefficient of variation,* which is the ratio of standard deviation and average, expressed in terms of percentage, and *skewness,* which is the third moment around the average.

Statistic	Reasoning Scheme						
-	*Default1*	*Default2*	*Rule-1*	*Rule-2*	*Rule-3*	*Rule-4*	*Case*
Average	312	600	612	645	1007	1139	1553
Standard Deviation	41.6	42.7	40.4	47.1	55.9	59.9	60.4
Coeff. of Variation	13.3	7.11	6.53	7.3	5.55	5.25	3.89
Minimum Reqd	267	533	567	567	917	1033	1417
50th %-tile	317	600	617	633	1033	1167	1567
75th %-tile	333	633	650	700	1067	1183	1617
95th %-tile	417	683	700	733	1083	1233	1650
99th %-tile	433	717	733	767	1117	1267	1717
Maximum Reqd	583	733	767	800	1133	1317	1967
Skewness	+2.91	+0.75	+0.8	+0.67	-0.08	-0.07	+0.5

Figure 9.1: Temporal Requirements for Different Reasoning Schemes

From this table, we make the following observations:

Observation 1: As expected, the temporal requirements measured in terms of average, minimum required time, or various percentiles etc. do increase along with the level of computations.

Observation 2: Standard deviations of the observed temporal needs increase as the required computational time increases. (This is likely to be attributable to gross effects of uncontrollable factors of computations). Since the average temporal requirements for different levels differ quite widely, absolute standard deviation is not the true indicator of the dispersion, and a relative measure of dispersion ought to be found [Waugh 52, chapter 6]. We therefore compute the "coefficient of variation" for each of the different types of computation. This suggests that the coefficients of variation decrease steadily along with a considerable increase in the computational time, and therefore also suggests that as the computational time increases the relative variability decreases. This observation clearly rules out any multiplicative model of computational requirements, i.e. U = C * F, or R = C * (1 +F) (which says the the uncontrollable factor is directly proportional to the average computational time. We therefore stick to our simple additive model, given earlier.

Observation 3: For all the levels we find that the average requirements are very close to their respective medians, i.e. 50-th percentile (which is not surprising assuming a standard normal distribution).

But the most interesting observations, however, originate from the upper limits of the respective ranges (i.e. the interval of minimum required time to maximum required time) and different higher-order percentiles. We observe different patterns in them in different levels (evidently, based on the computational requirements).

Observation 4: We observe that for default1-level computations the 75-th percentile value is near to one-fifth of the range. Our conclusion therefore is that on about 25% of the occasions the default-level computations are affected by the uncontrollable factors which extend the computational time significantly (by even as much as 100%). Thus distribution of the temporal requirements at this level is strongly positively skewed.

Observation 5: There is no appreciable computational difference in requirements between the default2 level and the rule1 and rule2 (i.e. where the method sticks to the predetermined plan) levels. For all these three levels the 75-th percentiles occur near their respective midpoints. Thus the distributions are found to be positively skewed, but to a much lesser extent than that for the default1 level.

Observation 6: For the rule3 and rule4 levels the 50-th percentiles have been near to the midpoint of the entire range. Thus distributions of computational time for these two levels do not show any significant skewness.

Observation 7: For case-level computations, where the requirement is still higher, we find that the 50-th percentile is at the midpoint of the interval of minimum required time and the 99-th percentile. On the remaining 1% of the occasions the computations have been prolonged by the uncontrollable factors, causing a positive skew in the overall distribution.

We thereby conclude that the uncontrollable factors have an intermittent effect on the computational time, i.e. they recur after certain intervals of time (X, say) within the computational environment. If the computational requirement is less than X, then there is a chance that it is not affected by the recurrence of the delaying factor. Since the computational requirements for the default1 level are small, it seems evident that computations at this level remain unaffected by the uncontrollable factor on most occasions. However, on certain occasions it influences the computations highly (of the order of 100%), causing strong positive skewness and high coefficient of variation. For default2, rule1 and rule2 levels, however, the computational time is such that they are uniformly affected by the factor. Thus although it prolongs the calculations, the overall variability reduces in these levels. But deeper-level computations being more demanding on the resource, each individual computation, here, has suffered by repeated recurrences of time. Thus a higher variability is observed here, since the number of recurrences of the uncontrollable factor (and its duration) is itself a variable. But since the computational requirements at these levels are high, the relative dispersion decreases.

Although the exact values of various measurements given above depend upon the environment where experiments are being conducted, we feel that the inferences that we can draw should not vary drastically.

In order to set up the cache-based system for our own computing environment, we need to accept this fact and to choose temporal bounds for each level of the cache so that the interference of the uncontrollable factors with the computations as the user sees them is taken care of as effectively as possible.

Based on the experimental results on our training samples, and the inferences drawn from the observations above, we choose the temporal upper bounds for different levels of the cache. We intend to assign such bounds on each level so that a particular level i of the cache will be accessed only if the available time is greater than the bound attached to the level i - 1, but is less than (or equal to) the bound associated with the level i. We repeat an experiment several times (where a time limit is provided to the system along with a given problem), and note the temporal requirements at each trial, in order to see whether it is possible to establish the viability of the cache-based reasoning in achieving the time-bounded performance and to examine what can be ascertained about the reliability of the time-bounded reasoning.

Realistically this should be done for just four of the cache levels, i.e. *default1, default2, rules* and *cases* only. However, for testing whether different reasoning schemes can indeed observe the time-boundedness, we consider the four different modes of rule-dependent reasoning (i.e. *rule1, rule2, rule3* and *rule4*) as four different levels of the cache. Evidently, this arrangement will provide much stricter bounds for those rule levels that are associated with smaller amounts of computations (i.e. rule1 to rule3) than what one will assign normally if all the varieties of rule-based calculations are combined into one level.

Obviously, selection of bounds for a level depends on the average and standard deviation obtained from the training samples for the level concerned. Here, one has to strike a balance between quality and time, i.e. the greater the bound assigned to a level, the higher will be the success rate of meeting deadlines (imposed on the computation) for activities involving the level concerned. We have examined three different ways of bound selection:

Soft1, where the bounds are somewhat demanding in the sense that here one wants to use the best possible reasoning method that is achievable within the imposed time bound, although it is likely to disobey the stipulation on certain occasions.

Soft2, where one wants to ensure a higher success rate (although one does not expect to have a 100% success rate).

Hard, where one does not want to allow any failure to meet the deadlines.

In the following sections (i.e. 9.3.1 to 9.3.3) we explain our heuristics for selecting the bounds for the different types of computations using the three schemes of bound selection (i.e. soft1, soft2 and hard). We also present the success rate achieved with each scheme.

9.3.1. Experiments Using The Soft1 Scheme

Computation in the soft1 scheme tries to ensure that the best possible reasoning technique that is achievable within the given temporal bound is used in solving a given problem. Hence our suggestions regarding selection of bounds is: for each level L_i we compute the lower bound (LB_i) as *Average - SD;* and an upper bound (UB_i) as *Average + SD.* We then consider *maximum* (UB_i, LB_{i+1}) as the upper bound for level i. Evidently, the upper bound for the (i-1)st level will act as the lower bound for the i-th level. The method refuses to provide any answer if the allotted time is less than the observed minimum time for the shallowest level. For the deepest (case) level we assume the observed maximum requirement plus corresponding standard deviation as the upper limit, although in real use there may not be any upper bound for the deepest level of the cache.

In figure 9.2 we summarise the bounds that we obtain from our observed data by following the above criterion.

Reasoning Type	Upper Bound
Default1	*maximum (312 + 41.6, 600 - 42.7)* ≈ *553*
Default2	*maximum (600 + 42.7, 612 - 40.4)* ≈ *643*
Rule-1	*maximum (612 + 40.4, 645 - 47.1)* ≈ *652*
Rule-2	*maximum (645 + 47.1, 1007 - 55.9)* ≈ *951*
Rule-3	*maximum (1007 + 55.9, 1139 - 59.9)* ≈ *1080*
Rule-4	*maximum (1139 + 59.9, 1533 - 60.4)* ≈ *1473*
Case	*1928*

Figure 9.2: Selection of Upper Bounds in the Soft1 Scheme

For each of these levels we now repeat experiments 100 times, by assigning a completion time, selected randomly, and determine the success ratio in meeting the deadline. We compute the following statistics in this regard:

a) Min Reqd - the maximum temporal requirement observed during this experiment.

b) Max Reqd - the maximum temporal requirement observed during this experiment.

c) No. of Successes, i.e number of occasions on which the deadline has been met;

d) Crossed Upper, stating on how many occasions the computational time exceeded the upper bound (assigned for the type of reasoning concerned);

e) Crossed Maximum - i.e. how many occasions the computation has taken more time than the Maximum required during training (see figure 9.1).

f) Max Deficit - the greatest margin by which the computation has failed to meet the deadline;

g) Max Gain - the greatest margin by which the computation has been concluded before the deadline.

The results obtained for the seven different types of computation are shown in figure 9.3.

Reasoning Scheme	Min Reqd (ms)	Max Reqd (ms)	No. of Successes (%-value)	Crossed Upper (%-value)	Crossed Maximum (%-value)	Max Deficit (ms)	Max Gain (ms)
Default1	*267*	*650*	*80*	*5*	*5*	*266*	*233*
Default2	*533*	*767*	*42*	*26*	*3*	*202*	*109*
Rule1	*517*	*766*	*71*	*28*	*0*	*120*	*134*
Rule2	*550*	*784*	*92*	*0*	*0*	*85*	*364*
Rule3	*884*	*1150*	*30*	*53*	*15*	*192*	*194*
Rule4	*1050*	*1367*	*67*	*0*	*0*	*199*	*398*
Case	*1466*	*1700*	*91*	*0*	*0*	*128*	*607*

Figure 9.3: Observed Results from Experiments in the Soft1 Scheme

Evidently the success rate is not very high, particularly for default2 and rule3 levels. The obvious reason is that for these two levels the upper bounds have been significantly less than their respective maximum observed requirements from the training data. In the soft2 scheme, we try to rectify this drawback (at least partially) by considering the maximum observed requirement too in deciding the upper bounds.

9.3.2. Experiments Using The Soft2 Scheme

Here, for each level i, we calculate a pessimistic upper bound (UB_i) which is *Average + SD * 3,* where, average and standard deviations have been observed from the training data. Similarly, we obtain a pessimistic lower bound (LB_i) *Average - SD.* The final upper bound for the i-th level can then be obtained by: *maximum(UB_i, LB_{i+1}, maximum observed).* We make this choice because we want to restrict the computations to a shallower level of the cache (and therefore to an inferior reasoning scheme), even when reasoning with more than one cache level seems viable. In figure 9.4 we summarise the selections.

Reasoning Type	Upper Bound
Default1	*maximum (312 + 41.6 *3, 600 - 42.7, 583) ≈ 583*
Default2	*maximum (600 + 42.7 *3, 612 - 40.4, 733) ≈ 733*
Rule-1	*maximum (612 + 40.4 *3, 645 - 47.1, 767) ≈ 767*
Rule-2	*maximum (645 + 47.1 *3, 1007 - 55.9, 817) ≈ 951*
Rule-3	*maximum (1007 + 55.9 *3, 1139 - 59.9, 1133) ≈ 1174*
Rule-4	*maximum (1139 + 59.9 *3, 1533 - 60.4, 1317) ≈ 1473*
Case	*2080*

Figure 9.4: Selection of Upper Bounds in the Soft2 Scheme

We have run similar experiments as for the soft1-type selection of bounds. Results thus obtained are provided in figure 9.5, where the columns have the same significance as in figure 9.3.

Reasoning Scheme	Min Reqd (ms)	Max Reqd (ms)	No. of Successes (%)	Crossed Upper (%)	Crossed Maximum (%)	Max Deficit (ms)	Max Gain (ms)
Default1	*267*	*516*	*80*	*0*	*0*	*97*	*279*
Default2	*533*	*733*	*82*	*0*	*0*	*126*	*179*
Rule1	*533*	*700*	*100*	*0*	*0*	*0*	*231*
Rule2	*550*	*783*	*100*	*0*	*0*	*0*	*371*
Rule3	*900*	*1117*	*74*	*0*	*0*	*161*	*232*
Rule4	*1017*	*1267*	*98*	*0*	*0*	*9*	*407*
Case	*1367*	*1617*	*93*	*0*	*0*	*86*	*676*

Figure 9.5: Observed Results from Experiments in the Soft2 Scheme

As expected, we find a much higher success rate for the computations in maintaining the stipulated bounds. This still does not guarantee success, but such a scheme can be acceptable for a "soft" real-time problem domain. In order to obtain a guaranteed success rate a much stricter scheme for choosing the bounds ought to be used.

9.3.3. Experiments Using A Hard Scheme

Here, in trying to obtain a guaranteed success we proceed in a way different from the above two schemes. Instead of choosing an upper bound for each level, we choose first a lower bound for it so that the level will not be considered for access unless the available time is higher than the designated lower bound. The lower bound for a level is chosen here as *Average + S.D. *3.* In figure 9.6 we show the bounds thus obtained, where evidently the lower bound of a level i will work as the upper bound of the previous level. For the deepest level the maximum observed value will be used as the upper bound. This scheme has the obvious disadvantage that the threshold of time bound below which the method refuses to provide any answer is much higher than in the previous examples.

Reasoning Type	Lower Bound
Case	*1553 + 60.4 *3 ≈ 1734*
Rule-4	*1139 + 59.9 *3 ≈ 1319*
Rule-3	*1007 + 55.9 *3 ≈ 1175*
Rule-2	*645 + 47.1 *3 ≈ 787*
Rule-1	*612 + 40.4 *3 ≈ 734*
Default2	*600 + 42.7 *3 ≈ 726*
Default1	*312 + 41.6 *3 ≈ 438*

Figure 9.6: Selection of Lower Bounds in the Hard Scheme

For this scheme too we repeat the experiment 100 times for each of the different types of reasoning we have considered. In figure 9.7 we summarise the result.

Reasoning Scheme	Min Reqd (ms)	Max Reqd (ms)	No. of Successes (%)	Crossed Upper (%)	Crossed Maximum (%)	Max Deficit (ms)	Max Gain (ms)
Default1	*233*	*650*	*97*	*0*	*3*	*-115*	*459*
Default2	*516*	*717*	*100*	*0*	*0*	*0*	*213*
Rule1	*516*	*733*	*100*	*0*	*0*	*0*	*255*
Rule2	*566*	*767*	*100*	*0*	*0*	*0*	*567*
Rule3	*900*	*1150*	*100*	*0*	*2*	*0*	*377*
Rule4	*1017*	*1450*	*99*	*0*	*2*	*78*	*642*
Case	*1400*	*1700*	*100*	*0*	*0*	*0*	*637*

Figure 9.7: Observed Results from Experiments in the Hard Scheme

As figure 9.7 shows, our expectations were not fulfilled entirely. We hoped to get 100% success in all the seven types of computations. But it failed on three occasions for the default1 level, and on one occasion in the rule4 level. Examination of the examples indicates that on each of these occasions computations have been prolonged to an extent much higher than what we obtained from our training data. Moreover, on repeating the failed examples, we have observed successes. We cannot attribute this failure's extra computational requirement to any factor other than any sudden change in the overall load in the system (possibly due to some other user).

9.4. CONCLUSIONS

We conclude from the above experimental results that we have been successful in providing an algorithmic method of solving problems for knowledge-based domains by efficient access to relevant knowledge sources through a cache, and subsequently subjecting the pertinent knowledge to an interpolation scheme (thus avoiding an unrestricted search through the knowledge-base). That the experiments did not succeed 100% is not due to any inherent drawback of the approach but because of some uncertainty in the computing environment. However, we can propose two possible ways of overcoming this difficulty. In the first approach, we suggest a dynamic updating of the bounds attached to each level, on the basis of past performances. Such a periodic upgrading should in the long run result in a still higher

level of success even for "hard" time-critical computations. Alternatively, for the deeper levels of the cache one may first calculate the solution using the default level of the cache and store it before resorting to a computation suggested by the associated reasoning scheme (in the light of progressive reasoning, mentioned in section 2.4.2). The advantage there will be that if the time bound is exceeded, the method will still have some solution to offer. Such a scheme will have the obvious drawback that after finishing the shallower-level calculations the method may not have enough time to start computations for the designated level and will have to turn to some shallower-level computations and thus have a poorer-quality solution. But for extreme hard-type situations, no other scheme seems to us to be infallible.

Chapter 10.

DISCUSSIONS AND CONCLUSION

10.1. GOALS AND MOTIVATION

The primary goal of this research has been to impose some control on the inherent uncertainty of knowledge-dependent computing in obeying any temporal stipulations. As the application of knowledge-based techniques in real-time domains is on the increase, demands for timeliness, i.e. ability to provide an answer within a stipulated time, must become greater. While increases in hardware speeds and/or use of multiple processors may seem to alleviate the demands in a given context, the general problem of whether it is possible to guarantee timeliness for knowledge-based computations is not yet in a satisfactory state. Our research has aimed at making some improvement in improving the situation.

We observe that the difficulty with knowledge-based computation is in its open-endedness, as the exact nature of computations that will be required to solve a given problem is not known a priori. We envisage that this drawback can be reduced significantly by efficient knowledge organisation in order that the extent of search can be stipulated within some predecided knowledge-space - so that the open-endedness of the searching can be controlled and the maximum temporal requirement for searching can be estimated. We have proposed a cache-based system to achieve this goal. The idea of caching for storing ready-to-use values in dealing with extreme demands for speed is quite well-known in computer science. But use of such a simplistic solution in a realistic domain is deterred by the obvious question: whether it is possible to store suitable solutions for all possible problems that may emerge in such domains.

Evidently, storing solutions for *all* possible problems is not feasible. More importantly, even if it had been possible to do so, finding the right solution for a given problem would then boil down to the ubiquitous search problem of knowledge-based computations. Hence one needs to look for a more generalised cache-structure to tackle this problem. In our research, we have tried a step by step improvement towards implementing a cache-based system, to identify the expected difficulties and then to address them. The final outcome of the research is the definition of an architecture, followed by its validation by the building of a cache-based system in the domain of solving transmission problems in shortwave radio propagation and ascertaining its viability in achieving time-bounded performance.

As a first step, we have designed a more general cache structure that does not merely contain the immediate solutions for all typical possible situations. Instead, the cache has different levels where each level is associated with some solutions or problem-solving paradigms. The immediate solutions are stored only at the topmost (shallowest) level of the cache, while deeper cache levels are associated with increasingly more complicated reasoning tactics. The idea behind this structure is that as the available time is increased, a deeper level of the cache is accessed to generate the solution. Since the extent of searching for each level is stipulated, an upper bound of temporal requirement for each levels of the cache can therefore be estimated and consequently, temporal bounds that will govern selection of the appropriate level in a given context can be determined. We have explained in section 1.2.1 how such a design conforms with human traits in solving time-bounded problems where better-quality but computationally or intellectually more expensive solutions are attempted as the amount of time available to do this job increases, and have also justified this for a realistic domain where normally a variety of methods of solution exists.

Implementation of such a cache demands answers to two problems:

1. How to access the relevant cached answer in time-bounded mode;
2. What happens if the answer obtained from the cache is not suitable for a given problem.

A significant part of the thesis has been devoted to dealing with these two difficulties.

Our answer to the first difficulty has been the use of a multi-dimensional (similar to a multi-dimensional array) cache, where each dimension corresponds to some key abstract feature. The set of these features is so chosen that its members represent a problem situation just adequate to frame an immediately-usable solution. Each dimension is divided into some typical column headers, and at the shallowest level of the cache we store in each cache cell the immediately-usable solution for the situation represented by the cell concerned. Other relevant features of the domain which may be needed to design an elaborate solution will be required only in deeper levels of the cache, and therefore their values in a given context are determined only when the need arises.

To respond to the second difficulty, we have proposed a novel reasoning technique which we have named "knowledge interpolation". Here, instead of spending time in the search for a best solution, one frames a solution from two not-so-appropriate ones, in a way similar to the linear interpolation approach that is common in numerical analysis. The advantage of this method is its predictability. Since the interpolation technique is very much algorithmic, it paves the way for estimating the temporal requirements in a meaningful way, unlike traditional knowledge-based computations. Moreover, even when time is not at all an issue, interpolation on cases can be viewed as a new way of taking case adaptation, which is still a central research topic in case-based reasoning.

The cache should work in unison with the knowledge-interpolation technique. Since for each of the two steps (i.e. retrieving solutions from the cache and subsequent improvement through knowledge interpolation) an estimation of temporal requirement is achievable, the total temporal demand of the overall solving procedure can be measured. Thus one should (at least intuitively) be able, in a suitable domain, to measure the temporal needs for typical computations under different reasoning schemes. Based on these observations, one should be able to make policies regarding which particular level of the cache should be accessed. The right cell(s) for attention at the level can then be identified through comparison of problem features with the cache's column-header(s) which should express abstractions suitable for describing a range of related problem-situations. Identification of such abstractions that work as indices in facilitating retrieval is a major requirement for building a cache-based system. We have discussed this point in section 4.5, where we have also indicated that inability to identify such indices in a domain may be a strong indication that the domain is not amenable to the cache-based approach.

10.2. WHAT WE HAVE DONE

We have studied the various ramifications of implementing a cache-based system for the domain of solving problems in controlling ground operations in an airport (AGC). We have investigated selection of features for accessing a cache cell, and the secondary features used in deeper levels of the cache that are required for designing good solutions at these levels. We have noted that in the absence of a strong domain theory identification of these features is not straightforward. Following this experience, we suggest that for an arbitrary domain one should use past cases (as narrated by experts) as a primary basis for acquiring domain-related knowledge, and expect that any cache will have at least one case level. We have designed a classification scheme (see section 4.4) of case features that will enable one to identify the roles of different entities in a case and will thereby facilitate selection of features necessary for retrieving items from cache.

Designing knowledge interpolation schemes necessitated finding solutions to the following requirements:

- how to order different symbolic entities.
- how to measure relative distance between two symbols.
- identification of the features, in a given context, on which interpolation should be carried out.
- how to accomplish the task of interpolation.

In our work we have provided some answers for each of these. We have shown ways to achieve ordering among different symbols and how to measure distance between two entities. We have also illustrated that this metrication is not confined to scalars alone, and we have devised different ways of achieving the same for non-scalar data types such as interval and set. In a given context, depending upon the needs, one should be able to choose at least one appropriate distance measurement tactic from the selection that we have provided. Evidently, the entities that will be subjected to interpolation depend upon the underlying reasoning scheme. We have shown that the knowledge interpolation technique is applicable not only to the shallowest cache-level, but to deeper levels of the cache too. In particular, we have demonstrated use of interpolation and a resulting improvement in the solutions for rule-based and case-based reasoning schemes. For each of the two major reasoning paradigms we have also underlined different ramifications and provided algorithms to accomplish the task of interpolation. Furthermore, for case-based reasoning we have designed a case representation scheme that facilitates identification of the interpolating variables and thereby makes interpolation less time-consuming. In addition, we have shown that there is more than one method of achieving an interpolated result, which increases the flexibility and the likely applicability of the overall approach.

Finally, we have applied the ideas generated by dealing with the AGC domain to another domain of solving problems of quick allocation of resources: shortwave radio transmission. There, we have shown how the solution improves with deeper levels of the cache and have also demonstrated how the method changes its courses of action dynamically depending upon the context of a current problem. We have subsequently estimated the temporal requirements for different types of reasoning by working on training data of 500 samples for each type of reasoning. Based on the training data we have estimated the temporal bounds for both *hard* and *soft* real-time problems, and experimented with the viability of the cache-based architecture, with each type of reasoning represented in its rows, in respecting the temporal stipulations.

We summarise our achievements as:

1 The cache-base architecture produces a viable form of knowledge representation (possibly at a meta level) that can impose an upper bound on the computational cost.

2 The cache and its associated reasoning scheme with knowledge interpolation do indeed provide means for an algorithmic treatment of knowledge-dependent reasoning, and therefore improve significantly the estimation of temporal requirements in knowledge-dependent computing.

3 The experimental results have shown a high success rate for both hard and soft real-time mode. However, the success rate is not 100% yet, apparently mainly because there is an uncontrollable part of some computations (due to the programming environment) that can prolong the computing time. However, this observation does not suggest infeasibility of

the cache-based scheme. We feel that this can be still be tackled and a better success rate can be achieved after by resorting to more lenient temporal bounds than those that we have attached to our cache, or (preferably) by repeating the experiments in a dedicated computing environment.

10.3. THE CACHE SCHEME IN THE PERSPECTIVE OF MIXED-PARADIGM SYSTEMS

The cache-based approach is not unique in capitalising on more than one reasoning scheme. Diversity in knowledge sources has often compelled researchers in expert systems to develop different reasoning paradigms, such as rule-based, model-based, case-based etc. Early expert systems have used only one form of knowledge (rules, primarily). But with the development of other forms of knowledge representation (e.g. case, model, causal network) which appeared to be more suitable for certain purposes and also with the horizon of knowledge-based applications spreading to encompass more substantial applications, KBS researchers have made efforts to integrate more than one form of knowledge and corresponding reasoning scheme in the same KBS, in order to utilise the knowledge more efficiently and/or to enhance the performance. CABARET, a mixed-paradigm system that integrates rule-based reasoning with case-based reasoning [Rissland 91, Rissland 89], and one system for technical diagnosis [Macchion 94] that integrates model-based, case-based and rule-based reasoning, are just two examples of this kind. Similarly, attempts are being made to integrate CBR with neural networks in various applications such as identification of non-genuine transactions through stolen credit cards [Reategui 94].

In modern AI terminologies such systems are called "hybrid systems" [Goonatilake 92]. However, we would like not to describe the cache-based system as a hybrid system in the sense used in that reference, as the term is yet to be defined unequivocally in AI. Depending upon various factors such as functionality, processing architecture and communication (between components) the term "hybrid" has many interpretations. Instead, we would prefer to describe the cache-based architecture as a "mixed paradigm system" (which is more straightforward) following Skalak [Skalak 89], who classified systems engaged in both rule-based reasoning and case-based reasoning into 4 categories depending upon the model of integration:

- Co-equal reasoners, where performance of the different reasoning mechanisms are controlled centrally by a separate control module which requests services from these mechanisms and uses their answers to determine the final result;
- Co-equal reasoners with two-way invocation, where both CBR and RBR are in operation and one invokes the services of the other as and when required;

- CBR dominant, where the CBR component has the key control which, if required, uses the services of the other reasoning mechanism (but not vice versa);
- CBR non-dominant, which is same as the one before with role of CBR and RBR interchanged.

According to this classification, the cache-based system may fall under the first heading, as a precise central control exists to decide which reasoning technique should be used at a particular juncture to solve the problem.

However, one major difference between existing mixed-paradigm systems and the cache-based architecture is that in all these systems the nature of the knowledge source has forced them to switch to a mixed-reasoning approach in order to get a higher-quality result. On the other hand, for the cache-based system the variety of temporal stipulations that may be imposed on the system, as dictated by a particular situation, naturally prescribes a mixed paradigm system in order that the best possible solution can be provided while observing the temporal bound. The cache-based architecture may not resort to multiple modes of reasoning, but it will do so automatically if on a given occasion it detects that there may be a possible failure on the part of the initially-chosen reasoning scheme in maintaining the imposed temporal bound (i.e. by stopping its calculations with the first scheme and switching to another reasoning scheme that allows completion of computation within the remaining time).

However, we find two potential drawbacks with the cache-based architecture. Firstly, not all possible AI techniques are amenable to the cache treatment. For example, we have mentioned earlier (see section 3.4.2) that in paradigms such as backtracking in Prolog an inherent uncertainty exists, for which no algorithmic interpretation can be designed. Hence building cache-based system in such a domain is unlikely to achieve any success. Also, building cache-based system implies identification of appropriate column indices and headers, possibly by analysing a large number of past cases. If a manageable set of such abstractions is not found, the architecture cannot be pursued any further. Additionally, filling the cache cells involves a non-trivial amount of work, particularly if the number of column dimensions and/or number of column headers along each dimension is rather large. We do not have any suggestions regarding how to circumvent this difficulty.

10.4. POSSIBLE FUTURE DEVELOPMENTS

We have used the cache-based system with exercises having only a single target location. But in a realistic domain, situations may occur where more than one problem needs to be solved within a given time limit. For example, with the shortwave domain, suppose the problem is to transmit messages to more than one location. The simple approach of solving both the problems separately may not be acceptable as the sender may have access to no more than one transmitter. Evidently, a more complex form of planning will be required for solving such problems. We anticipate use of cases to sort such a problem out, and we feel it is achievable

from within the cache-based reasoning approach. Apart from being an interesting extension, which can be tackled by using the kinds of reasoning that are helpful in resource-allocation problem where there are competing demands, (e.g. TRUCKER and RUNNER [Hammond 89]), we know that this will increase the attractiveness of such software to potential users in shortwave frequency-allocation planning and consultancies.

We also intend to investigate elaborately how a cache-based system will perform in a more dynamic world. So far the dynamism that we have incorporated is to abandon a direct-transmission plan and switch over to relay-based transmission, if it is found that conditions are not suitable. In a realistic domain more complicated situations may be expected - where a more urgent problem needs attention causing abandonment (at least temporarily) of the currently-executed plan. We intend to investigate these issues for a cache-based system too. Cases and other data are available from military shortwave applications, but collecting a good stock of them has been partly a "political" problem that needed more time and contacts than we had available during the thesis project.

In our experiments we have used static temporal bounds (obtained by a repeated measurements on a training set) for each level. We envisage a possible improvement on these bounds by updating the bounds whenever the result of a new computation is available. Evidently, it is worth investigating whether a better success rate can be achieved and/or a better-quality can be accomplished by this dynamic readjustment, and what information (e.g. simple averages will not be enough) will make any such improvements possible.

Additionally, we can think of two possible enhancements of the effectiveness of the cache:

1. Upward propagation of knowledge through the levels of the cache. As we have mentioned in chapter 3, it is evident that as the cache-based system solves more and more problems, it should be able to express more rules and better ready-to-use solutions. A system that exploits this material fully should propagate this experience gained in a deeper level of the cache to a shallower level. Evidently, this is not straightforward as the knowledge representation and related reasoning mechanism varies with cache level. Hence we consider this to be a challenging problem of how to exploit machine learning, and "knowledge interchange schemes with the same kinds of motivation as the KIF/KQML proposals (for reference, see [Genesereth 94]).

2. Another possible improvement is in selecting indices for a cache. In our design we have used manual analysis of the cases to determine the appropriate cache indices. It is worth considering how and to what extent this process can be automated, where the past cases or behaviours of a cache-based system should be analysed to ascertain (statistically, or by the progressive growth of an abstraction hierarchy of terms (e.g. the abstraction "potential-degree-of-threat" mentioned in chapter 6, and particular instances of it in new problems)) the importance of a feature in describing and/or solving problems in the domain.

3. We anticipate questions regarding possible parallelisation (and consequently an improved performance) of a cache-based system. We can think of three types of parallelisation:

   - Parallel computations of the cache indices in a multi-dimensional cache, which will evidently increase the speed of accessing the cache, and corresponding retrieval processes.

   - Parallelisation in reasoning, where more than one level of the cache is accessed and corresponding reasoning scheme is carried out. Evidently, it does not increase the computational speed. But it may improve the quality of the solution in the following way: instead of monitoring the time and resorting to a lower level of the cache when failure to maintain the temporal stipulation using the current reasoning scheme is anticipated, a processor may devote its entire time to finish the computation assigned to it. Because, even if a processor is indulged in a particular type of reasoning that is unlikely to complete within the allotted time, the processors carrying out shallower-level reasonings should be able to find an answer within the given time.

   - Parallesiation of cache, where more than one cache are simultaneously explored. Such a situation may occur in domains that are not precisely defined, so that solving a particular type of problem may involve getting suggestions from more than one cache. In general, we did not apprehend such a situation and assumed that a particular type of problem should require only a single cache to access. However, in an event where multiple caches are necessary, parallelisation should speed up the computation. But implementation of such a system requires further development of various interpolation schemes (e.g. rule interpolation, case interpolation) which should be able to carry out interpolations when there exist some disparity in the representations of corresponding items. This issue needs to be addressed first before contemplating implementation of parallel caches.

BIBLIOGRAPHY

[Aamodt 91] Aamodt A.: "A Knowledge-intensive, Integrated Approach to Problem Solving and Sustained Learning". Doctoral Dissertation, Department of Informatics, University of Trondheim, Norway, 1991.

[Allen 84] Allen J.F.: Towards General Theory of Actions and Time. Artificial Intelligence, Vol 23, No. 2, 1984, pp 123 - 154.

[Allen 85] Allen J. F. and Hayes P. J.: A Common-Sense Theory of Time. Proc. IJCAI-85, Morgan Kaufmann Publishers Inc. San Mateo, CA 94022, 1985, pp 528 - 531.

[Alterman 86] Alterman R.: An Adaptive Planner. Proc. AAAI-86, American Association for Artificial Intelligence, 1986, pp 65 - 69.

[Ashley 88] Ashley K.D. and Rissland E.L.: Waiting on weighting: A Symbolic Least Commitment Approach, Proc. AAAI-88, 1988, pp 239 - 244.

[Ashley 89a] Ashley K.D.: Assessing Similarities Among Cases, A Position Paper. Proc. DARPA Case-Based Reasoning Workshop, 1989, pp 72 - 76.

[Ashley 89b] Ashley K.D.: Defining Salience in Case-Based Arguments. Proc. IJCAI-89, Morgan Kaufmann, 1989, pp 537 - 542.

[Ashley 89c] Ashley K.D.: Indexing and Analytical Models. Proc. DARPA Case-Based Reasoning Workshop, Morgan Kaufmann, 1989, pp 197 - 202.

[Bareiss 89] Bareiss R. and King J.A.: Similarity Assessment in Case-Based Reasoning. Proc. DARPA Case-Based Reasoning Workshop, Morgan Kaufmann, 1989, pp 67 - 71.

[Barletta 88a] Barletta R. and Mark W.: Explanation-based Indexing of Cases. Proc. DARPA Case-Based Reasoning Workshop, Morgan Kaufmann, 1988, pp 50 - 60.

[Barletta 88b] Barletta R. and Mark W.: Explanation-based Indexing of Cases. Proc. AAAI-88, pp 541 - 546.

[Barletta 89] Barletta R. and Hennessy D.: Case Adaptation in Autoclave Layout Design. Proc. DARPA Case-Based Reasoning Workshop, 1989, pp 203 - 207.

[Behn 82] Behn R.D. and Vaupel J.W.: "Quick Analysis for Busy Decision Makers". Basic Books, New York, 1982.

[Bento 94] Bento C. and Costa E.: A Similarity Metric for Retrieval of Cases Imperfectly Explained. Lecture Notes in Artificial Intelligence, Volume 837, Springer Verlag, Berlin, 1994, pp 92 - 105.

[Bird 92] Bird M.: Solar Activity 1992. "World Radio TV Handbook", BillBoard Publications Inc. New York 10036, Vol 46, 1992, pp 34 - 40.

[Bradtke 88] Bradtke S. and Lehnert W.G.: Some Experiments with Case-Based Search. Proc. AAAI-88, 1988, pp 133 - 138.

[Brandau 91] Brandau R., Lemmon A. and Lafond C.: Experience with Extended Episodes: Cases with Complex Temporal Structure. Proc. DARPA Case-Based Reasoning Workshop, Morgan Kaufmann, 1991, pp 1 - 12.

[Branting 89] Branting L.K.: Integrating Generalisations with Exemplar-Based Reasoning. Proc. DARPA Case-based Reasoning Workshop, 1989, pp 224 - 229.

[Braun 82]: Braun G.: "Planning and Engineering of Shortwave Links". Simens Aktiengesellschaft; John Wiley and Sons, 1982.

[Buchanan 84] Buchanan B.G. and Shortliffe E.H.: "Rule-Based Expert Systems", Addison-Wesley, 1984.

[Campbell 90] Campbell J.A. and Wolstencroft J.: Structure and Significance of Analogical Reasoning, Artificial Intelligence in Medicine, Vol 2, 1990, pp 103 - 118.

[Chatterjee 92] Chatterjee N. and Campbell J.A.: A Caching Scheme for Time-Critical and Case-Based Reasoning. Proc. EXPERSYS-92, IITT-international, 94 Promenade A. Ballu, F-93460 Gournay sur Marne, France, 1992, pp 477 - 482.

[Chatterjee 93] Chatterjee N. and Campbell J.A.: A Caching Scheme for Time-Critical Knowledge-based Computations. Proceedings Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Gordon and Breach Science Publishers S.A., Y-Parc, Chemin de la Sallaz, 1400 Yverdon, Switzerland, 1993, pp 61- 70.

[Chatterjee 94] Chatterjee N. and Campbell J.A.: Adaptation Through Interpolation for Time-Critical Case-Based Reasoning. Lecture Notes in Artificial Intelligence, Volume 837, Springer Verlag, Berlin, 1994, pp 221 - 233.

[Conte 72] Conte S.D. and de Boor C.: "Elementary Numerical Analysis : An Algorithmic Approach". McGraw Hill, 1972.

[Conte 80] Conte S.D. and de Boor C.: "Elementary Numerical Analysis: An Algorithmic Approach". McGraw Hill, 1980.

[Converse 89] Converse T., Hammond K. and Marks M.: Learning Modifications Rules from Expectation Failure. Proc. DARPA Case-Based Reasoning Workshop, 1989, pp 110 - 114.

[Dean 88] Dean T. and Boddy M.: An Analysis of Time-Dependent Planning, Proc. AAAI-88, 1988, pp 49 - 54.

[Dean 91] Dean T. L. and Wellman M. P.: "Planning and Control". Morgan Kaufmann, San Mateo, CA, 1991.

[Dodhiawala 89] Dodhiawala R., Sridharan N.S., Raulefs P. and Pickering C.: Real-Time AI Systems : A Definition and An Architecture, Proc. IJCAI-89, Morgan Kaufmann, 1989, pp 256 - 261.

[Forgy 82] Forgy C. L.: RETE : A Fast Algorithm for the Many Pattern/Many Object Pattern Matching Problem. Artificial Intelligence, Vol 19, No. 1, 1982, pp 17 - 37.

[Genesereth 94] Genesereth M.R. and Ketchpel S.R.: Software Agents. Communications Of The ACM, Vol 37, No. 7, 1994, pp 48 - 53.

[Goel 89] Goel A. and Chandrasekaran B.: Use of Device Models in Adaptation of Design Cases. Proc. DARPA Case-based Reasoning Workshop, 1989, pp 100 -109.

[Goonatilake 92] Goonatilake S. and Khebbal S.: Intelligent Hybrid Systems. Proc. First Singapore International Conference on Intelligent Systems. Japan-Singapore Artificial Intelligence Centre, September 1992. pp 207 - 212.

[Gupta 85] Gupta A.: Parallelism in Production Systems: The Source and the Expected Speed Up. Fifth International Workshop on Expert Systems and Their Applications, Agence de l'Informatique, 26-57 Avignon, France. 1985.

[Gupta 86] Gupta A.: "Parallelism in production Systems". PhD Thesis, CMU-CS-86-122, Department of Computer Science, Carnegie Melon University, PA, 1986.

[Hammond 86] Hammond K.J.: CHEF : A Model of Case-Based Planning. Proc. AAAI-86, AAAI Press/MIT Press, 1986, pp 267 - 271.

[Hammond 87] Hammond K.J.: Explaining and repairing Plans that Fail. Proc. IJCAI-87, 1987, pp 109 - 114.

[Hammond 88] Hammond K., Converse T. and Marks M.: Learning from Opportunities, Storing and Re-using Execution Time Optimizations. Proc. AAAI-88, 1988, pp 536 - 540.

[Hammond 89] Hammomd K.J.: "Opportunistic Memory", Proc. IJCAI-89, Morgan Kaufmann, San Mateo, California, 1989, pp 504 - 510.

[Hayes-Roth 85] Hayes-Roth B.: A Blackboard Architecture for Control. Artificial Intelligence Journal, Vol 26, No. 3, 1985, pp 251 - 321.

[Hinrichs 89] Hinrichs T. R.: Strategies for Adaptation and Recovery in a Design Problem Solver. Proc. DARPA Case-Based Reasoning Workshop, 1989, pp 115 - 118.

[Hinrichs 91] Hinrichs T. R. and Kolodner J. L.: The Roles of Adaptation in Case-Based Design. Proc. DARPA Case-Based Reasoning Workshop, Morgan Kaufmann 1991, pp 121 - 132.

[Hinrichs 92] Hinrichs T. R.: "Problem Solving in Open World: A Case Study in Design". Erlbaum, Northvale, New Jersey, 1992.

[Hofmeister 85] Hofmeister T.: Sunrise-Sunset Calculation. In "Shortwave Software", Edition 2, Radio Nederland Wereldomroep, Hilversum, Nederlands, August 1985, pp 16 - 18.

[Holyoak 89] Holyoak K. J. and Thagard P.: Analogical Mapping by Constraint Satisfaction. Cognitive Science, Vol 13, 1989, pp 295-355.

[Hsu 89] Hsu F.: "Large Scale Parallelization of Alpha-Beta Search: An algorithmic and Architectural Study with Computer Chess". Ph.D Thesis Carnegie-Mellon Univerisity, Pittsburgh, PA, 1989.

[Ingrand 92] Ingrand F.F., Georgeff M.P. and Rao A.S.: An Architecture for Real-Time Reasoning and System Control. IEEE EXPERT, Vol 7, No. 6, December 1992, pp 34 - 44.

[Jacobs 92] Jacobs G.: Do-It-Yourself Propagation Forecasting Table for the Most Suitable Megahertz Broadcasting Bands. In "World Radio TV Handbook", BillBoard Publications Inc. New York 10036, Vol 46, 1992, pp 30 - 32.

[Kass 88] A. M. and Leake D. B.: Case-Based Reasoning Applied to Constructing Explanations. Proc. DARPA Case-Based Reasoning Workshop, pp 190 - 208.

[Kass 89] Kass A.: Strategies for Adapting Explanations. Proc. DARPA Case-Based Reasoning Workshop, 1989, pp 119 - 123.

[Kibler 87] Kibler D. and Aha D. W.: Learning Representative Exemplars of Concepts: An Initial Case study. Proc. Fourth International Workshop on Machine Learning, 1987, pp 24 - 30.

[King 88] King J.A., Klein G.A., Whitaker L. and Wiggins S.: SURVER III: An Application of Case-Based Reasoning. Proc. Fourth Annual Aerospace Applications of AI Conference, 1988.

[Kobayashi 91] Kobayashi S. and Nakamura K.: Knowledge Compilation and Refinement for Fault Diagnosis. IEEE EXPERT, October 1991, pp 39 - 46.

[Kolodner 88a] Kolodner J. L.: Extending Problem Solving Capabilities Through Case-Based Inference. Proc. DARPA Case-Based Reasoning Workshop, 1988, pp 21 - 30.

[Kolodner 88b] Kolodner J. L.: Retrieving Events form Case Memory, A Parallel Implementation. Proc. DARPA Case-Based Reasoning Workshop, 1988, pp 233 - 249.

[Kolodner 89a] Kolodner J. L.: Judging Which is the "Best" Case for a Case-Based Reasoner. Proc. DARPA Case-Based Reasoning Workshop, 1989, pp 77 - 81.

[Kolodner 89b] Kolodner J. L. and Simpson R. L.: The MEDIATOR: Analysis of an Early Case-Based Problem Solver. Cognitive Science, Vol 13, No. 4, 1989, pp 507 - 549.

[Kolodner 92a] Kolodner J.: An Introduction to Case-Based Reasoning. Artificial Intelligence Review, Vol. 6, No. 1, 1992, pp 3 - 34.

[Kolodner 92b] Kolodner J and Mark W.: Case-Based Reasoning: Guest Editor's Introduction. IEEE Expert, Vol 7, No. 5, October 1992, pp 5 - 6.

[Kolodner 93] Kolodner J.: "Case-Based Reasoning", Morgan Kauffman Publishers Inc., San Mateo, CA 94403, 1993.

[Kopeikina 88] Kopeikina L., Brandau R. and Lemmon A.: Case-Based Reasoning for Continuous Control. Proc. DARPA Case-Based Reasoning Workshop, 1988, pp 233 - 249.

[Korf 85] Korf R. E.: Depth-first Iterative Deepening : An Optimal Admissible Tree Search. Artificial Intelligence, Vol 27, No. 1, 1985. pp 97 - 109.

[Korf 88] Korf R. E.: Real-Time Heuristic Search: New Results. Proc. AAAI-88, pp 139-144.

[Koton 88a] Koton P.: Reasoning about Evidence in Clausal Explanations. Proc. DARPA Case-Based Reasoning Workshop, 1988, pp 260 - 270.

[Koton 88b] Koton P.: Reasoning about Evidence in Clausal Explanations. Proc. AAAI-88, 1988, pp 256 - 261.

[Kowalski 91] Kowalski A.: Case-Based Reasoning and the Deep Structure Approach to Knowledge Representation. Proc. 3rd International Conference on AI and Law, ACM press, 1991, pp 21 - 30.

[Kumar 88] Kumar V.K.R. and Rao V.: Parallel Best-first Search of State-space Graphs: A Summary of Results. Proc. AAAI-88, 1988, pp 122 - 127.

[Ladkin 86] Ladkin P.: Time Representations: A Taxonomy of Interval Relations. Proc. AAAI-86, Morgan Kauffman, 1986, pp 360 - 366.

[Ladkin 87] Ladkin P.: The Completeness of a Natural System for Reasoning with Time Intervals. Proc. IJCAI-87, Morgan Kaufmann, 1987, pp 462 - 467.

[Laffey 88a] Laffey T.J., Cox P.A., Schmidt J.L., Kao S.M. and Read J.Y.: Real-Time Knowledge-Based Systems. AI Magazine, Vol 9, No. 1, 1988, pp 27 - 45.

[Laffey 88b] Laffey T., Weitzenkamp, Read J., Kao S. and Schmidt J.: Intelligent Real-Time Monitoring. Proc. AAAI-88, 1988, pp 72 - 76.

[Leake 91] Leake D. B.: ACCEPTER : A Program for Dynamic Similarity Assessment in Case-Based Explanation, Proc. DARPA Case-Based Reasoning Workshop, pp 51 - 62.

[Lee 88] Lee K.F. and Hon H.W.: Large-vocabulary Speaker-Independent Continuous Speech Recognition Using HMM. IEEE International Conference on Acoustics, Speech and Signal Processing, 1988, pp 123 - 126.

[Lehnert 87] Lehnert W.: Case-Based Problem Solving with a Large Knowledge Base of Learned Cases. Proc. AAAI-87, 1987, pp 301 - 306.

[Long 83] Long W.F. and Russ T.A.: A Control Structure for Time-Dependent Planning. Proc. IJCAI-83, 1983, pp 230 - 232.

[Macchion 94] Macchion D.J. and Vo D.P.: A Hybrid Knowledge-Based System for Technical Diagnosis Learning and Assistance. Lecture Notes in Artificial Intelligence, Volume 837, Springer Verlag, Berlin, 1994, pp 301 - 312.

[Masui 83] Masui S. , McDermott J. and Sobel A.: Decision making in Time-Critical Situations. Proc. IJCAI-83, William Kaufmann Inc., CA 94022, 1983, pp 233 - 235.

[Matick 77] Matick R.E.: "Computer Storage Systems and Technology". John Wiley and Sons, 1977.

[McDermott 82] McDermott D.: A Temporal Logic for Reasoning about Processes and Plans. Cognitive Science, Vol 6, No. 2, 1982, pp 101 - 155.

[Miranker 87] Miranker D. P.: TREAT: A Better Match Algorithm for AI Production Systems. Proc. AAAI-87, 1987, pp 42 - 47.

[Myllymaki 94] Myllymaki P. and Tirri H.: Massively Parallel Case-Based Reasoning with Probabilistic Similarity Metric. Lecture Notes in Artificial Intelligence, Volume 837, Springer Verlag, Berlin, 1994, pp 144 - 154.

[Nilsson 82] Nilsson N.J.: "Principles of Artificial Intelligence", Springer-Verlag, Berlin, 1982.

[Owens 88] Owens C.: Domain Independent Prototype Cases for Planning. Proc. DARPA Case-Based Reasoning Workshop, 1988, pp 302 - 311.

[Owens 89] Owens C.: Plans Transformation as Abstract Indices. Proc. DARPA Case-Based Reasoning Workshop, 1989, pp 62 - 65.

[Owens 93] Owens C.: Integrating Feature Extraction and Memory Search. Machine Learning, Vol 10, 1993, pp 311-339.

[Porter 89] Porter B.W.: Similarity Assessment, Computation vs. Representation. Proc. DARPA Case-Based Reasoning Workshop, DARPA, 1989, pp 82 - 84.

[Porter 90] Porter B. W., Bareiss R. and Holte R. C.: Concept Learning and Heuristic Classification in Weak Theory Domains. Artificial Intelligence, Vol 45, 1990, pp 229 - 263.

[Ralston 93] Ralston A. and Reily E.D.: "Encyclopedia of Computer Science". IEEE Press, 1993.

[Ram 93] Ram A.: Indexing, Elaboration and Refinement: Incremental Learning of Explanatory Cases. Machine Learning 10, 1993, pp 201 - 248.

[Reategui 94] Reategui E. B. and Campbell J. A.: A Classification Systems for Credit Card Transactions. Proc. Second European Workshop in Case-Based Reasoning, Acknosoft Press, 1994, pp 167 - 174.

[Rich 91] Rich E. and Knight K.: "Artificial Intelligence", McGrawhill Inc., 1991.

[Rissland 89] Rissland E.L and Skalak D. B.: Combining Rule-Based and Rule-Based Reasoning: A Heuristic Approach. Proc. IJCAI-89, 1989, pp 524 - 530.

[Rissland 91] Rissland E.L and Skalak D.B.: CABARET: Rule Interpretation in a Hybrid Architecture. International Journal of Man-Machine Studies, Vol 34, 1991, pp 839 - 887.

[Schank 86] Schank R., Collins G. and Hunter L.: Transcending inductive category formation in learning. Behavioral and Brain Sciences, Vol 9, No. 4, 1986, pp 639 - 651.

[Schnelle 92] Schnelle K. D. and Mah R.S.H.: A Real-Time Expert System for Quality Control. IEEE Expert, Vol. 7, No. 5, 1992, pp 36 - 42.

[Shekhar 89] Shekhar S. and Dutta S.: Minimising Response Times In Real Time Planning and Search. Proc. IJCAI-89, 1989, pp 238 - 242.

[Shoham 89] Shoham Y.: Time for Action: On the Relation between Time, Knowledge and Action. Proc. IJCAI-89, 1989, pp 954 - 959.

[Simon 59] Simon H. A.: Theories in Decision Making in Economics and Behavioral Science. American Economic Review 49, 1959, pp 253 - 283.

[Simmons 88] Simmons R. G.: A theory of Debugging Plans and Interpretations. Proc. AAAI-88, 1988, pp 94 - 99.

[Skalak 89] Skalak D. B.: Options for Controlling Mixed Paradigm Systems. Proc. DARPA Case-Based Reasoning Workshop, 1989, pp 318 - 323.

[Smith 91] Smith D. B.: Principles for Case Representation in a Case-Based Aiding System for Lesson Planning, Proc. DARPA Case-Based Reasoning Workshop, 1991, pp 351 - 362.

[Solvic 90] Solvic P.: "Choice in Thinking - An Invitation to Cognitive Science" Vol 3. Ed. D.N. Osherson and E.E. Smith. MIT Press, Cambridge Massachusetts, 1990.

[Stankovic 88] Stankovic J.A. and Zhao W.: On Real-time Transactions. SIGMOD RECORD, 1988, Vol 17, No. 1, 1988, pp 4 - 18.

[Stottler 89] Stottler R.H., Henke A.L. and King J.A.: Rapid Retrieval Algorithm for Case-Based Reasoning. Proc. IJCAI-89, 1989, pp 233 - 237.

[Sycara 88a] Sycara K.: Using Case-Based Reasoning for Plan Adaptation and Repair. Proc. DARPA Case-Based Reasoning Workshop, 1988, pp 425 - 434.

[Sycara 88b] Sycara K.: Resolving Goal Conflicts via Negotiations. Proc. AAAI-88, 1988, pp 245 - 250.

[Thagard 89] Thagard P. and Holyoak K.J.: Why Indexing is the Wrong Way to Think about Analog Retrieval. Proc. DARPA Case-Based Reasoning Workshop, 1989, pp 36 - 40.

[Tsang 87] Tsang E.P.K.: Time-Structure for AI. Proc. IJCAI-87, 1987, pp 456 - 461.

[Veloso 91] Veloso M. M. and Carbonell J.G.: Variable-precision Case Retrieval in Analogical Problem Solving. Proc. DARPA Case-Based Reasoning Workshop, 1991, pp 93 - 106.

[Vila 94] Vila L.: A Survey on Temporal Reasoning in Artificial Intelligence. AI Communications, Vol 7, No. 1, March 1994, pp 4 - 28.

[vonNeumann 47] von Neumann J. and Morgenstern O.: "Theory of Games and Economic Behavior". Princeton University Press, NJ, 1947.

[Waugh 52] Waugh A.E.: "Elements of Statistical Methods". Mcgraw Hill, 1952.

[Webster 90] "Webster's New Dictionary and Thesaurus", concise edition. Russel, Geddes and Grosset, 1990.

[Wenstop 76] Wenstop F.: Deductive Verbal Models of Organizations. International Journal of Man-Machine Studies, Vol 8, 1976, pp 293- 311.

[Winston 84] Winston P.H.: "Artificial Intelligence", Addison-Wesley, 1984.

[Wolstencroft 93] Wolstencroft J.: "A Unifying Approach To Reasoning By Analogy". Ph.D thesis, University of London, 1993.

[Wright 86] Wright L. M., Green M.W., Fiegl G. and Cross P. F.: An Expert System for Real-time Control. IEEE Software, March 1986, pp 16 - 24.

[Zadeh 79] Zadeh L. A.: A Theory in Approximate Reasoning. Machine Intelligence 9, eds J. E. Hayes, D. Michie and L. I. Mikulich. New York, Halstead Press, 1979, pp 149 - 194.

[Zadeh 92] Zadeh L. A.: Knowledge Representation in fuzzy Logic. In " An Introduction to Fuzzy Logic Applications in Intelligent Systems", eds R. R. Yager and L. A. Zadeh, Kluwer Academic Publishers, 1992.