TASK AND CONTINGENCY PLANNING UNDER UNCERTAINTY

by

VOLKAN C. KUBALI

Nuclear Engineer and M.S. in Nuclear Engineering, MIT, (1992)
B.S. in Nuclear Engineering, Hacettepe University, Turkey, (1987)

Submitted to the Department of Nuclear Engineering
in partial fulfillment of the requirements for the degree of

DOCTOR OF SCIENCE

in the field of

Applied Artificial Intelligence

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1994

Author .... [illegible] ....
Department of Nuclear Engineering
August 23, 1994

Certified by ..... [illegible] ....
Lawrence M. Lidsky, Professor of Nuclear Engineering
Thesis Supervisor

Certified by ....
Andy B. Dobrzeniecki, Research Scientist
Thesis Supervisor

Certified by ...... [illegible] ....
Kent F. Hansen, Professor of Nuclear Engineering
Thesis Reader

Accepted by ....
Allan F. Henry
Chairman, Departmental Committee on Graduate Students

TASK AND CONTINGENCY PLANNING UNDER UNCERTAINTY

by

VOLKAN C. KUBALI

Submitted to the Department of Nuclear Engineering
on August 23, 1994, in partial fulfillment of the
requirements for the degree of
Doctor of Science

Abstract

The development of fault-tolerant plans in uncertain environments is the primary concern of this thesis. Toward this goal, the thesis investigates the general planning problems in the context of graph models and suggests the use of genetic algorithms for their solution. The method of genetic algorithms is compared to other classical search and optimization methods for a well-known deterministic path planning problem (TSP) and is found to exhibit reasonable performance. Applications of this method to stochastic planning problems indicates that the method is still successful when the fitness of individual solutions is sampled rather than evaluated exactly. Finally, the thesis introduces the concept of robust plans that can be modified or reoptimized with minimum cost in the case of contingencies. The robust planning examples are given in the context of path planning problems and the solutions are found with the applications of genetic algorithms. While genetic algorithms are satisfactory for those problems that are NP-hard, they are not directly applicable to those problems in which the evaluation of the fitness values takes time exponential in problem size. These complex problems are solved more efficiently by incorporating in the genetic algorithms problem-independent heuristics such as fitness sampling. The results indicate that the further scrutiny of the problems having dynamic optimization criteria may prove beneficial.

Thesis Supervisor: Lawrence M. Lidsky
Title: Professor of Nuclear Engineering

Thesis Supervisor: Andy B. Dobrzeniecki
Title: Research Scientist, Whitaker College

Thesis Reader: Kent F. Hansen
Title: Professor of Nuclear Engineering

Acknowledgments

I am delighted to find the opportunity to thank to those who helped me during this research in so many ways. First, many thanks to Prof. Larry Lidsky and Dr. Andy Dobrzeniecki for their continuous guidance. I would also like to thank to MIT International Nuclear Safety Program and U.S. Department of Energy for providing financial support. Thanks are also long due to Prof. Kent Hansen, the thesis reader and the director of the MIT Program, for providing support throughout the project.

My goal of getting a doctoral degree would never realize without Prof. Osman Kadiroğlu and Prof. Yalçın Sanalan of Hacettepe University. I strongly appreciate their assistance as well as their labor in making sure that their students get the best education possible. I could not even come to MIT without the support from Turkish Government. I appreciate their investment in me and hope to prove one day that its benefits to Turkish people will far exceed its costs.

My years at MIT were fun because of my friends. I wish everyone could have friends like Erol Çubukçu and Sinan Keskin whose joys are contagious, whose friendships are unfailing.

Without the great sacrifice made by parents, Meliha and Selahattin Kubalı, this endeavour would not be possible at all. I would like to thank my family for always providing the most important and irreplaceable support of them all: love.

Prologue

"The vast majority of the research in the fields of machine learning, artificial intelligence, self-improving systems, self-organizing systems, and induction is concentrated on approaches that are correct, consistent, justifiable, certain (i.e. deterministic), orderly, parsimonious, and decisive (i.e., have a well defined termination).

These seven principles of correctness, consistency, justifiability, certainty, orderliness, parsimony, and decisiveness have played such valuable roles in the successful solution of so many problems in science, mathematics, and engineering that they are virtually integral to our training and thinking.

It is hard to imagine that these seven guiding principles should not be used in solving every problem. Since computer science is founded on logic, it is especially difficult for practitioners of computer science to imagine that these seven guiding principles should not be used in solving every problem. As a result, it is easy to overlook the possibility that there may be an entirely different set of guiding principles that are appropriate for a problem such as getting computers to solve problems without being explicitly programmed."

J R Koza, *Genetic Programming, MIT Press, 1992*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

Plans are sequences of actions that are coordinated in order to achieve prespecified objectives. In regard to the complexity of their structures, they may range from a plain permutation of the activities that must be completed to a detailed specification of what activities will be processed by whom at what time using which resources. Planning process is directed at the determination of those plans that achieve the objectives with the minimum amount of resources and the maximum likelihood of success.

Difficulties in planning arise mainly from the inadequacy of the computational methods in addressing the real world problems. Such difficulties can be better appreciated within the context of an example. Consider the planning of a refueling outage of a power plant. While it is desired to perform as many maintenance tasks as possible during the refueling outage, the plant must be brought back to normal productive operation as soon as possible for economical purposes. These conflicting goals render it necessary to develop plans and schedules that would make the most efficient use of the resources (such as tools, manpower and time) while remaining within the established safety margins. The outage planners therefore must determine the best course of actions regarding at least a few thousand interdependent activities to be performed under various financial, safety, time, personnel, hardware, physical and technical constraints. In addition to facing such a challenging optimization problem, the planners must also consider that the plan is to be implemented in a working environment that is a dynamic and stochastic one, as opposed to a static one. For instance, it is always possible to diagnose new problems during maintenance that require immediate attention. As complex as it is, the outage management practice has been generally restricted to manual preparation of the outage schedules. For example, a state-of-the-art computer scheduling package, *Finest Hour*, has been deemed as inadequate [118] for use in outage planning. In general, the benefits of the conventional computerized techniques have been limited because of their limited modelling capability.

Another major concern of planning is the efficiency of the employed methodology. The most interesting planning problems have proven to be computationally demanding. Finding an optimum solution for these problems requires an inordinate amount of computing, necessitating a sacrifice from seeking perfection. The concern for efficiency dictates that a near-optimal solution must be found with reasonable computing resources. Any planning methodology, in order to be practical, must address this issue of efficiency.

This thesis investigates the development of efficient plan generation both in deterministic and nondeterministic settings in the context of search in graph models. The objective of the thesis is to provide computational planning methods by addressing such factors as efficiency, uncertainty management and flexibility of a plan to tailor to possible contingencies. Its primary concern is the development of *fault-tolerant* plans to be implemented in uncertain environments; these plans anticipate failures and are devised to minimize their adverse impacts on accomplishment of the objectives. Further, the thesis aims to produce a methodology for generation of plans that strike a balance between the probability of achieving the goals and the resources expended.

1.2 Motivations

1.2.1 Motivations for Nondeterministic Planning

In many planning problems, variables under consideration are usually subject to uncertainty. Yet, plans are generally designed under the assumptions that **i)** the environment is static, **ii)** there is perfect information about the world, and **iii)** the effects of the various actions in the world are fully predictable.

Nevertheless, the real world contains uncertainties and a dynamic, ever-changing environment that often violates these assumptions. The following quotation by French [49] emphasizes the importance of the consideration of uncertainty in scheduling: *"It may be argued that all practical (scheduling) problems are both dynamic and stochastic if for no other reason than that all quantities are subject to some uncertainty. In fact in many problems the randomness is quite obvious. For instance, it may be impossible to predict exactly when jobs will be available for processing, e.g., aircraft arriving at an airport space; it may be impossible to predict processing times exactly, e.g. during routine maintenance it will not be known which parts have to be replaced until they have been examined and that examination is one of the operations of the maintenance process; it may be impossible to predict the availability of machines, for some may have significant breakdown rates; and so on."*

Addressing a nondeterministic problem with a deterministic model may cause problems that cannot be ignored. Early methods of mitigating these potential problems have been based on sensitivity analysis. For example, a study summoned after the energy crisis of 1974 [104] urges businesses for incorporating uncertainty into their business plans by using a crude form of sensitivity analysis. Such an analysis follows a scheme described concisely in [148] : *"Projections of likely paths of exogenous variables are made and then the plan is formulated to fulfill some objectives given the assumed structure of the underlying system and the projected paths of the exogenous variables. The plan is solved for a few alternative scenarios and for a few combinations of high, medium, low forecasts of the most important exogenous variables. The plan is then based on the medium variant and the information on how the results differ under the alternative variants is used to educate the intuition to cope with different scenarios".* Later years have seen an increasing reliance on the use of simulation models that have also been increasingly more sophisticated.

Yet, such simulations did not fully address the problem of planning under uncertainty from the beginning. Given a structure or a plan, they could yield information about its performance; but finding a good solution structure remained a computationally difficult problem which was often treated independently from the simulation stage. As approximate solution techniques for computationally-hard deterministic problems have been developed and put into common use, it became increasingly clear that there was a need for systematically incorporating uncertainty in the construction of a plan.

1.2.2 Motivations for Flexible and Robust Plans

The need for developing efficient plans in a dynamic environment is an acute one that can be seen in almost every aspect of the human life. Given a probabilistic and an incomplete model of the world that is always subject to change, one wishes to obtain plans that guarantee the best possible performance. Indeed, it can also be argued that the capability of developing plans under uncertainty is an important attribute of intelligent behavior. It is difficult to see how intelligent behavior can be manifestated, unless one is able to develop plans that can work under limited perturbations and deviations.

Plans, even those that are optimum for the conditions stated in the model, may fail expectations. Plans are vulnerable to failure because the circumstances during the execution phase oftentimes diverge from the pre-determined conditions. Divergence between reality and the model may lead to unanticipated problems. For example, a project may be prolonged excessively or there may arise a need for extra resources such as money or manpower, which might not be easily available. Even if these required extra resources were available, their allocation in the plan would probably be inefficient from a global point of view and hence might inflate the project costs. What is worse than a waste of resources, such divergences might even make the achievement of the plan goals impossible because of the externally imposed constraints. Ideally, a plan should anticipate such divergences and minimize their impact by adapting to them in an efficient manner.

Common complaints on conventional planning software indicate that a plan possibly undergoes modifications which are often hard to accommodate with given deterministic models. A planner notes that: *"We fail miserably to integrate the 'unplanned' events into the existing schedule. All work activities tend to slip with the critical path work [134]"*. For example, a 1981 study [62] observes that : *"A frequent comment heard in many scheduling shops is that there is no scheduling problem but rather a rescheduling problem."* Even when these modifications are performed, the results are often too wasteful of resources and seriously undermine the optimality of the original plan. The following quote from a World Bank report [148] explains why deterministic plans have not been found valuable by practitioners: *"It is known a priori that the exogenous variables and the behavioral representation of the economy will deviate from their assumed values by some random magnitudes. But this information is not taken into account systematically in the initial preparation of the plan. Therefore, during implementation a plan is usually hastily adjusted to contingencies or even completely scrapped. (The plan has served its political function and attention of policymakers has shifted from it). ... The absence of good contingency planning - namely, a planning that allows for subsequent adaptation to random events - is a major reason why formal planning is perceived by the most policymakers and heads of the planning offices to be an essentially irrelevant exercise, useful mostly for window dressing and political mobilization. Under current planning practice, it is as if a flight engineer were taught how to fly an airplane only under ideal wind and visibility conditions and given no training in how to adjust to turbulence."*

The need for developing plans that are flexible enough to succeed in uncertain environments is by no means unique to specific areas. Planning under nondeterministic conditions is a subject of study for project, business, economy and finance planners. Almost all real world applications ranging from daily human activities to sophisticated industrial activities are performed with incomplete, inexact information of the world. Therefore, the development of planning methods that address the problem of producing optimum or at least near-optimal solutions under a dynamic environment may prove useful in many diverse areas.

1.3 An Example of a Robust Plan

Contingency planning concerns with plans that allow for subsequent adaptation to random events that might arise during the execution of a plan. We call those plans that minimize the adverse impacts of such dynamic, random events on plan goals as robust plans. This capability to accommodate contingencies is usually provided by built-in options in a plan, such as the capability of backtracking or choosing an alternative route. This is best clarified with an example :

Consider Figure 1-1 that shows two similar paths from a given start node s to a given terminal node t both through two distinct nodes. Each edge in this example has an associated cost and probability value. For simplicity, we assume that the probabilities of the initial and final edges are equal to unity

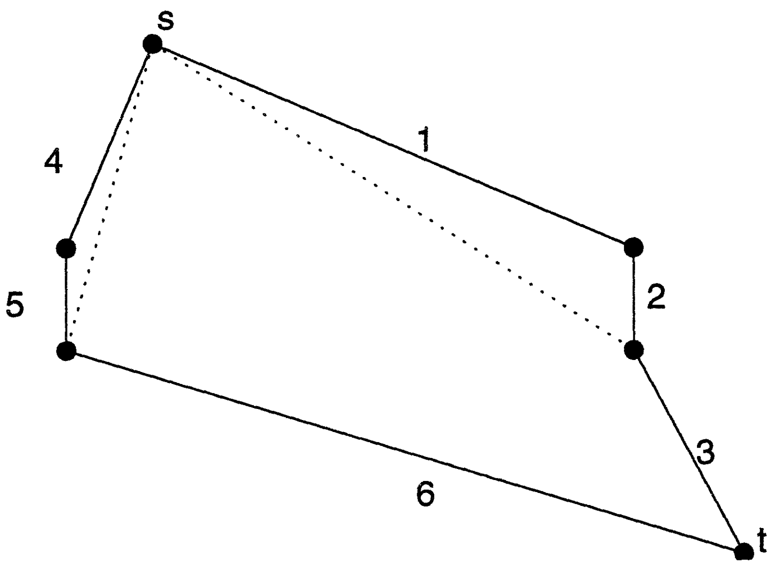


Figure 1-1: *Two different paths from s to t*

but the probabilities for the middle edges are smaller than unity for both of these paths. We also assume that a failure along the middle edges forces us to return to the starting point.

The first path **a** has a cost of $C_a = c_1 + c_2 + c_3$ and a probability of $R_a = p_2$ leading to an expected cost $E_a(c) = (c_1 + c_2 + c_3)p_2$. Note that this conventional definition of expected cost does not take the possible subsequent modifications into consideration; it is only a static measure. Similar equations can be written for the second path **b** : $E_b(c) = (c_4 + c_5 + c_6)p_5$. Suppose the cost and the probabilities for both the middle edges is the same, i.e.: $c_2 = c_5$ and $p_2 = p_5 = p$; but the following holds for the other edges:

$$c_1 > c_4$$

$$c_1 + c_3 < c_4 + c_6$$

Given these data, conventional methods suggest that we should choose the first path **a** over the second path **b** as it has a smaller cost.

This result may change drastically when the dynamics of the plan execution is included in the evaluation. Let us assume that upon failure along travel across the middle edge, one returns to the starting node but, because of a commitment constraint, one has to traverse only the same path. Clearly, if there is no factor that limits the number of repeated attempts, the overall probability R of reaching to terminal node t, for both of these paths, reaches unity as

$$R = p + (1-p)R \implies R = 1$$

Nevertheless, there is generally be an overall cost limit that limits the number of such attempts. Let us denote the number of repeated attempts with k_a and k_b for these paths **a** and **b**. For a given cost limit $C_{\max}$, this number k_a for the first path **a** can be calculated from the following budget equation:

$$C_{\max} - k_a c_1 - (c_2 + c_3) = 0 \tag{1.1}$$

as

$$k_a = \frac{C_{\max} - c_2 - c_3}{c_1} \tag{1.2}$$

Given this limit, the overall probability R_a of reaching to the terminal node t is equal to the sum of the probabilities for each attempt:

$$R_a = p\left(1 + (1-p) + (1-p)^2 + \cdots + (1-p)^{k_a - 1}\right) \tag{1.3}$$

It follows that

$$\begin{aligned} R_a &= p \sum_{i=0}^{k_a-1} (1-p)^i \\ &= p \frac{1-(1-p)^{k_a}}{1-(1-p)} \\ &= 1-(1-p)^{k_a} \end{aligned} \tag{1.4}$$

and the expected cost, now being dynamical, is equal to the sum of probability weighted costs of all distinct realizations.

$$\begin{aligned} E_a^*(c) &= \sum_{i=0}^{k} \Pr(i) C(i) \\ &= pC_a + p(1-p)(C_a + c_1)p(1-p)^2(C_a + 2c_1) + \cdots + p(1-p)^{k_a-1}(C_a + (k_a - 1)c_1) \\ &= \sum_{i=0}^{k_a-1} p(1-p)^i C_a + \sum_{i=0}^{k_a-1} ip(1-p)^i c_1 \\ &= C_a R_a + c_1 p \sum_{i=0}^{k_a-1} i(1-p)^i \\ &= C_a R_a + c_1 \frac{1-p}{p}\left(1 - k_a(1-p)^{k_a-1} + (k_a-1)(1-p)^{k_a}\right) \end{aligned} \tag{1.5}$$

Same values for the second path **b** are obtained in a similar manner:

The number of attempts k_b is :

$$k_b = \frac{C_{\max} - c_5 - c_6}{c_4} \tag{1.6}$$

The overall probability R_b is:

$$R_b = 1 - (1 - p_5)^{k_b} = 1 - (1 - p)^{k_b} \tag{1.7}$$

and the dynamic expected cost is:

$$E_b^*(c) = C_b R_b + c_4 p \sum_{i=0}^{k_b-1} i(1-p)^i \tag{1.8}$$

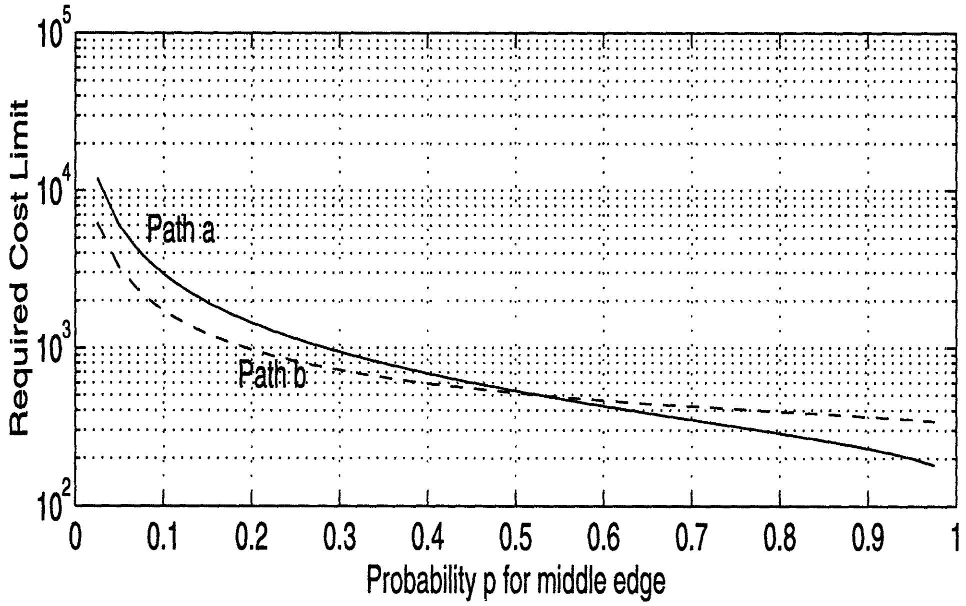
It is apparent that the second path has a lower cost for failure and therefore can be preferable over the first path depending on the probability p of the middle edge. For a comparison, we can fix the overall success probability to a certain value and then compare the required budget and/or the expected cost for either of these paths. The results of this computation are shown in Figure 1-2 for the assumption that an overall success probability of 95% is required. With this value, the required number of attempts for a given probabilistic path can be computed from the above equation 1.4. Figure 1-2 shows the expected cost and the required cost limit versus the probability p for both path **a** and path **b** for the assumed values $c_1 = 100$, $c_2 = 10$, $c_3 = 90$, $c_4 = 50$, $c_5 = 10$, and $c_6 = 290$. As can be seen from the figure, the second path b, which has 75% larger cost than the other path, has both a lower overall cost limit and a lower expected cost for small values of p. Within the range $0.4 < p < 0.5$, the path **b** has a lower cost limit but slightly higher expected cost than the first path **a**. Only at higher probabilities, the path **a** dominates the path **b** with respect to both measures.

The above example shows that the result of optimization depends on what criteria have been chosen in the beginning. More importantly, it shows that taking dynamic evolution and adaptability of a plan into account can produce results that may contradict those found with conventional methods.

1.4 Organization of the Thesis

We start in **Chapter 2** with a review of the planning methods from different perspectives, especially within the context of artificial intelligence studies. We also present the mathematical formulation of certain planning and scheduling problems. Chapter 2 also gives the reasons why we selected the graph search paradigm for handling general planning problems.

In **Chapter 3**, we examine the graph search problems in more detail. We review the complexity theory as it pertains to the planning problems in graph models. We discuss both the exact and the approximate methods for solving computationally hard problems encountered in graph models. Among these search methods, we analyze the genetic algorithm in detail as we suggest it as a general heuristic method for solving planning problems both in deterministic and nondeterministic settings. We develop a model of genetic algorithms that allows us to compare them with the Monte-Carlo method.



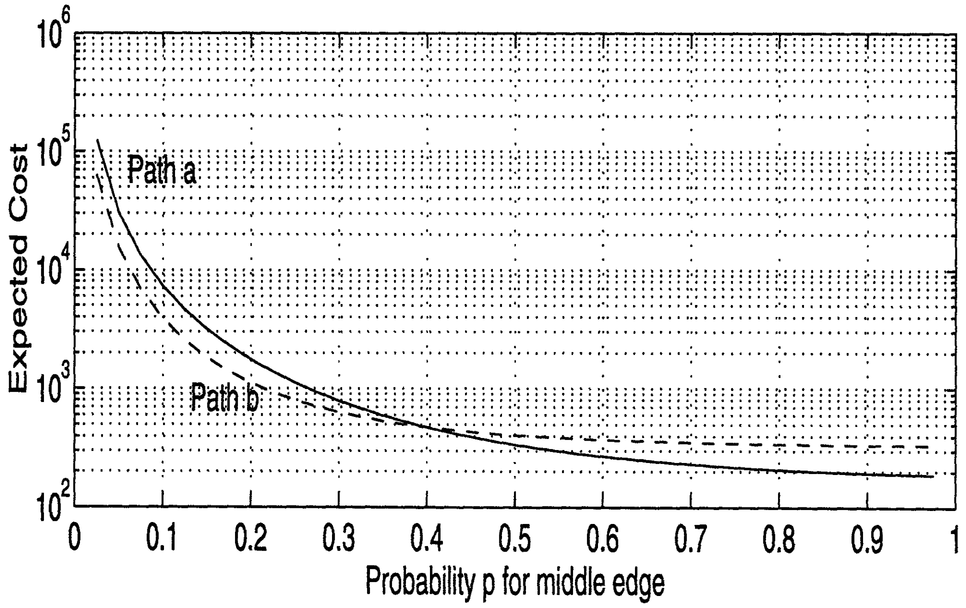


Figure 1-2: *Cost limit and expected cost versus probability for the paths of Figure 5.1 for fixed success probability=0.95*

In **Chapter 4**, we analyze the performance of genetic algorithms on a basic deterministic path planning problem, namely the traveling salesman problem. We observe the performance of genetic algorithms on certain benchmark problems and later compare it to the performance of different search methods.

In **Chapter 5**, we focus on stochastic planning problems. Stochastic problems involve models in which data are represented by probability distributions. After the examination of the complexity results and a review of the literature on these problems, we apply the genetic algorithms on a simple stochastic path planning problem and observe their performance.

In **Chapter 6** we review the literature on different areas that take the dynamics of a plan into consideration. After summarizing the related concepts, we introduce the concept of robust plans. We suggest different optimization criteria that can be used for measuring robustness and present a depth-first algorithm that can be used for this purpose.

Chapter 7 examines the development of robust plans in the context of path planning problems. We define certain problems, examine their complexity and apply the genetic algorithms. Later, we suggest various methods to cut down the computational effort required to solve them.

Finally, **Chapter 8** summarizes the results, reviews the problems encountered, and describe avenues for future research.

Chapter 2

Planning from Different Perspectives

The purpose of this chapter is to present a review of the Artificial Intelligence (AI) based approaches to planning. While planning techniques have generally been classified within the fields Operations Research (OR) or Project Planning, a convergence can be observed between these fields and artificial intelligence based studies. Planning in all these areas involve a discrete computation for determining the best coordination of tasks in order to achieve some pre-specified objectives. In all these areas, increasing the efficiency of solutions to computationally hard problems is a main concern. A review of different paradigms of AI may bring benefits for efficient planning by indicating what problems could be solved and what approaches are likely to be most fruitful.

This chapter starts with the examination of the goals of artificial intelligence and then reviews the developments in different AI approaches with regard to planning *(Section 1)*. These approaches are **logical formalism** *(Section 2)*, **knowledge-based systems** *(Section 3)*, **distributed computation** *(Section 4)* and **search and optimization** *(Section 5)*. Among these areas, search and optimization is examined in more detail, for our models will be developed on this area. We also present a review of the extension of these approaches into nondeterministic models. Then, in *Section 6*, we summarize the other AI-based approaches. The lessons drawn from these different approaches are summarized in *Section 7*. Thus, this chapter presents the context for the selection and methodology used in the course of this work.

2.1 Planning in Artificial Intelligence

Artificial Intelligence is an interdisciplinary area that draws from computer science, mathematics, logic, cognitive science and engineering. AI can be viewed as research for the realization of both of the following goals :

1. *The emulation of human capabilities that require intelligence in computers,*

2. *The extension of human problem solving capability by enriching our collection of problem solving techniques.*

AI cannot be defined more strictly because we cannot define intelligence. Minsky [98] notes that defining intelligence is like defining *"unexplored regions of Africa"*. As soon as they are explored, they do not anymore fit into the given definition. Intelligence, similarly, consists of not-well-understood problem-solving-techniques. As soon as we know how to solve a problem, it requires no more intelligence to us (Although, a person without this knowledge would probably not think so). How this knowledge is used in human problem solving constitutes the main concern of AI, because the basis of AI lies in the following premise (often called physical-symbol-system hypothesis): *Thinking is no more than a set of computational procedures that process symbols.*

The challenge of AI is therefore to convert obscure human problem-solving techniques to neat, well-understood, systemized algorithms. **Planning** *is one of the most important human capabilities that is responsible for most of intelligence-requiring actions.* Thus, a significant part of AI work has been on constructing automated computer planners. The **automated planning** part of AI, whose first aim is to add a planning capability to robots, constitutes the first branch of AI work that can help devising more efficient ways of planning. Another possibility for employing artificial intelligence, and indeed so far the only widely benefited AI spin-off, arises from a competing view. The **knowledge-based** approach assumes that every problem requires specific knowledge and intelligence is no more than learning and using the problem-specific knowledge. This approach has resulted in *expert systems* that digest the domain-knowledge of an expert in the area and make it available to others. The **distributed-computation** paradigm is a relatively new and promising branch of AI and has already achieved some success in areas where logical formalism failed. The most productive results of AI studies stem from the development of efficient search and optimization techniques for computationally hard problems that arise so often in the field of planning and scheduling. In fact, a competing view of AI is that all difficult problems that require intelligence are indeed hard computational procedures and they can only be solved by efficient **search**. The need for search arises most often in the context of games such as chess and othello, but it is also encountered in natural language understanding, pattern recognition (image, speech recognition etc.) and expert systems.

2.2 Automated Planning

In early days of AI, it was felt that there can be a single, generic reasoning mechanism for solving problems in diverse fields. This belief combined with a drive for emulating the common-sense planning has given birth to the automated planning field of AI. The researchers in automated planning rationalized their approach with the following hypothesis: Planning under deterministic conditions can be considered as a problem solving technique that involves determining a course of actions that takes a system from an initial state to a desired (or a goal) state. Then computer planners may generate plans with methods similar to those used for solving puzzles, i.e., they only need to find a path between a given pair of initial and final states. In this way, a group of researchers have strived to develop planning systems to capture the essence of this problem solving ability independent of the problem domain [2].

AI was mostly dominated by logical formalism approach in its initial stages. The main approach to automated planning was to develop plans from given causality relations by using logical inference rules. For example, in 1955, A. Newell and H. Simon, two pioneers of AI, observed that humans plan mostly by backward chaining. For achieving a goal, we try to achieve first its subgoals and later we try to achieve the subgoals of subgoals and so on. This observation has resulted in a planner with a fancy name (GPS) *General Problem Solver*. It was later to be discovered that this so-called "*means-ends analysis*" was limited and could easily fall into loops. However, the techniques dealing with such problems were not late to be forthcoming. The lessons from the GPS were:

1) The world state could be described by a set of logical propositions,

2) The actions are propositions that add and delete some of these propositions,

3) A plan is a transformation from initial state to goal state as a result of application of a series of actions,

4) The causality in plan could be captured by the use of logical inference rules.

2.2.1 Structure of Automated Planners

A domain-independent planner can be seen as a black-box that generates a plan which achieves the given goals when supplied with an initial state description, a goal description, and a list of operators available in the domain. Internally, the planner must have means to represent partial plans and intermediate states. Thus, a domain-independent planner specifies in advance **1)** the world representation formalism, **2)** operator representation formalisms, and **3)** an internal representation of partial plans [72]. Any representational formalism serves two functions: The formalism must be sufficiently rich to describe the types of changes that can go on in a domain of interest. The formalism must also supply the information needed for the planning algorithm to be efficient.

World Representation

Classical AI planners use a set of ground terms to represent the state of the world. Every state is a set of logical propositions about the world. The closed world assumption is used: the propositions that are true are specifically stated; what is not listed is considered false.

Operator Representation

It would be impossible to state all the facts that are true after an operator is applied. For example, it is true but quite trivial to state that after this page is read, it still is a black-and-white page! The need for finding a concise and efficient way to summarize the changes to the state of the world after an operator is executed has led to the development of the frame formalism. In automated planning, the frame formalism was introduced by STRIPS planner. STRIPS uses a set of preconditions and a set of add and a set of delete lists for application of each operator. For example, an operator MOVE may have the following specifications :

MOVE(`ObjectA, FromLocationX, ToLocationY`):
Preconditions: HOLD(`ObjectA`), CLEAR(`LocationY`)
Add-List : AT(`ObjectA, LocationY`)
Delete-List : AT(`ObjectA, LocationX`)

Plan Representation

A STRIPS planner tries to achieve a goal not found in the current state by finding an operator whose add list contains the goal and then recursively achieving the preconditions of that operator. When achieving a conjunction of goals, STRIPS sequentially considers each goal. In this way, a STRIPS planner needs to keep track of the steps executed, the current state of the world, and the goals yet unachieved.

This simple internal representation of plans does not account for possible interactions between goal achievements. For example, if we may want to achieve two goals, and the actions for the first goal can make it impossible to achieve the first goal. The linear planning technique of STRIPS does not allow interleaving of actions directed at achieving different goals. Therefore, **nonlinear planning technique** has been developed for solving the conjunctive goal achievement. Note that in most cases, the goal state can be reached through different paths. When the order of the actions is not strictly critical to the solution, planners should produce only partially-ordered (as needed) plans instead of a completely ordered list of actions. Such planners are called **nonlinear** planners and are generally more efficient and useful than linear planners because of their ability to limit the search space by considering the possible interactions among the steps of a plan from the very beginning.

Another modern approach to planning involves the hierarchical organization of the problem. The goals are decomposed into subgoals and these subgoals are grouped depending on the abstraction level. The top level consists of the most abstractive and general group of tasks. Each successive layer includes more detailed characterization of tasks. With this approach, it becomes possible to provide for increasingly detailed plan production. The recent examples of nonlinear hierarchical planners are TWEAK, NONLIN and DEVISER.

2.2.2 The Limitations of Automated Planners

Although many domain-independent planning systems have been developed, their power and utility turns out to be limited because of the serious limitations of their formalisms. First, such planners start with a static, closed world model: what is listed in the world model is true, what is not listed is false. Second, they assume that a single reasoning method would be adequate for deriving plans and choose the operator representation planning according to this assumption. For example, most planners based on GPS use a means-ends analysis or difference-reduction scheme [72]. The following is a criticism of GPS by Laurière: [89]

"First, the problems that GPS has dealt with are not a very representative set; there is something of a family resemblance among them all and none are really difficult. If GPS can solve them, it is to some extent because it 'knows' that it can solve such problems and therefore 'knows' that it will be able to reduce the search space to a very small number of elements.

Second, it seems that the method of differences is not appropriate when a global view of the task is necessary for finding a solution. The method takes a short-range view, assuming a certain continuity in the path to the solution, that no wide detour is needed and that the problem can be broken down into a sequence of elementary steps, all of which are effectively equal in importance. But there is a large class of problems for which this is far from the case, even among the formal problems and logical brain-teasers similar to those put in GPS.

Third and last, it is not at all easy to state a problem in way that can be put to GPS. In fact, the statement prejudges the solution: in every case the system has to be given an appropriate set of differences, the connection table and other information. Further, considerable intellectual effort is required in describing a situation in terms of a set of operators."

For applications in the real world, the automated planners face the following difficulties:
i) The facts often cannot be represented as logical predicates; they are fuzzy and uncertain.
ii) The world is far too complex to represent with simple data structures. A complex world model cannot be typed in either; the planner has to have a learning capability.

Even though these difficulties can be overcome or simply ignored, the following problems regarding the efficiency of logical formalism in defining the intelligent behavior still remain to be addressed:
iii) The closed-world assumption brings significant limitations.
iv) Using a single reasoning method can never be good enough for solving diverse problems; a combination of different forms of thinking are necessary.

Indeed, the all types of logical reasoning run across the notorious **frame problem**. The frame problem arises from the paradox on how to determine if a piece of knowledge is relevant or irrelevant to the problem at hand, *without* going through a problem solving phase that would make use of this piece of knowledge. It is said that logic does not tell us what to do; rather it tells us what not to do! The further limitations of logic used by automated planners can be specified as follows [103]:

1. Decidability: For some problems, the truth value of a goal -whether it is true or false- is not decidable, i.e., it does not follow from the axioms.

2. Intractability: It is sometimes too inefficient to use more expressive logic models having powerful inference rules, even though the problem could be decidable.

3. Monotonicity: Time is not involved in the standard logic, so any fact, once proven is valid forever.

4. Consistency: If one proves accidentally, both P and not(P), one can prove anything in logic. Hence, no contradiction can be allowed in logic models.

5. Uncertainty: There is not a consensus on how to handle uncertainty efficiently by using logic models.

2.2.3 The Lessons from Automated Planning

Even though these problems involve the core issues of AI, rigorous solutions have not been found yet. Past research in automated planning has kept the models simple in order to get around of such problems; their domains are limited and their formalisms are probably not yet sophisticated enough for applications in real world. Furthermore, the use of more powerful and expressive logic models seems to require a substantial computability tradeoff.

We should, however, be appreciative of the limited success of automated planning through logical formalism. Automated planning systems were able to generate plans in limited domains by using the built-in causality in a model. Moreover, nonlinear planners can produce plans with multi-branches that are very similar to PERT graphs. In fact, one of the main researchers in automated-planning, Austin Tate, who developed NONLIN, realized this and applied nonlinear planning techniques to produce PERT type plans for some industrial operations [133].

An unintended but provably useful product of automated reasoning has come from the new methods of programming. These methods that emerged from AI research have been designed from the start to be more clear, easier to understand and more efficient than the classical programming techniques. The most important of these is **logic-programming**. Another, **object-oriented programming**, make it easier to handle complex models.

Logic programming aims to hide the complex control aspects of a problem solving method. In idealized case, a program will consist of just the data-base and the logical formulas. The control will be handled by a theorem-proving module. This approach results in programs that are more transparent and easy to modify. For example, the following PROLOG program finds the critical path in a project for house-building. (obtained by modifying an example given by Lee [90]):

THE DATABASE:

```
    activity(start, event1, foundation, 5).
    activity(event1, event2, walls, 6).
    activity(event1, event3, plumbing, 4).
    activity(event2, event3, ceiling, 5).
    activity(event2, event3, electrical, 3).
    activity(event3, end, painting, 2).
```

THE FORMULA:

```
    critical-path(X,X,[],0).
    critical-path(X, Z, Total-Time, Activity-List) :-
        setof((T,Y,[A—L]), (T1,T2),^
          (activity(X,Y,A,T1), critical-path(Y,Z,L,T2), T is T1+T2), SubStates),
        maxof(SubStates,(Total-Time,_,Activity-List)).
```

Logic programming approach has also been useful in making it easier to handle **simulation** models in complex environments [50]. Object-oriented programming, too, has proven benefits, especially for simulation purposes. Fox [45] gives an example in his paper about improving the scheduling simulation.

Nevertheless, these programming methods are not exempt from the fundamental problems surrounding the automated planning techniques. The control mechanisms behind these methods have to address the fundamental issues such as computational complexity, decidability and model capacity. It is probably safe to say the difficulties arising with regard to these fundamental issues are the reasons that lie behind that that the automated reasoning and planning methods have not generally led to practical uses. Yet, the emergence of such fundamental problems and the spin-off of new programming techniques must be considered as quite useful products of automated planning.

2.2.4 Extension to Nondeterministic Models

The logical formalism approach encounters problems even in a deterministic world model. Extending the classical Boolean logic to include reasoning with a probabilistic world model turns out to be much more problematic. In fact, there is not yet a universally accepted theory of how to handle uncertainty in logic. One recently dominating approach is **fuzzy logic**. Fuzzy logic is based on a more comprehensive set theory than normal logic; in this theory, membership measure is a real value between 0 and 1, opposed to either 0 or 1 or classical logic. Through new definitions of **or** and **and** relations and new inference rules, fuzzy logic employs a reasoning process that inherently accounts for uncertainty. Fuzzy logic is commonly applied in control and production rule systems (as a result in expert systems) with considerable success; but its applications in automated planning have been limited. A similar model is **multi-valued logic** in which the degrees of truth are discrete values between 0 and 1. This logic has interesting truth tables but its applications have been similarly limited.

Non-monotonic logic, is a way of logical reasoning, which is designed specifically to overcome the limitations of the deductive logic when the conflicting propositions have to be made in a changing environment. Non-monotonic logic is a domain-dependent tool; one has to specify the knowledge-base beforehand. A more serious drawback of non-monotonic logic is that it does not accommodate utilities. Langlotz [88] notes that even though non-monotonic logic might be a useful tool if the utilities of actions are relatively unimportant, it cannot be used for optimization problems as one would not have a way of comparing the consequences of various actions.

Perhaps the most sound technique for planning under uncertainty is to use decision analysis methods coupled with Bayesian probability. There is only very primitive work in planning using **probability logic**; hence no applications in real world examples. Attempts have been made to further unify the field of probability logic; the Dempster-Shafer theory, which includes fuzzy logic as a subset [80] seems to successfully integrate belief measures with probability measures. Nevertheless, the present state of art in extending logic to nondeterministic models indicates that, before the new logic models can be applied to real world nondeterministic planning problems, the fundamental problems (such as computational tractability) have to be faced first.

2.3 Knowledge-Based Planning

According to some researchers, the core of intelligence is knowledge (*"Knowledge is power"*). It is, hence, claimed that, in AI, the priority must be given to the problem of knowledge acquisition and manipulation. This view has led to the development of knowledge-based systems. With the emergence of expert systems, it has been possible to model the heuristics and knowledge of the human experts. Unsurprisingly, most AI applications in literature are based on expert systems.

The expert systems are indeed one of the ironic successes of AI research. They are found to be very useful in duplicating of an expert's work and at sometimes enhancing and supporting a user's decision by making the application of domain-specific rules transparent. With expert systems, it has been possible to solve difficult calculus problems or other type of problems that require significant amount of knowledge. The irony, however, is that the knowledge-based methods have been quite unsuccessful in solving seemingly much simpler problems. Among these, common-sense reasoning comes first. The stories that tell how an expert system gives non-sense answers are well-known. Also, computation-intensive problems, such as language understanding, image recognition could not be handled by expert systems. The use of expert systems have been limited to specific domains in which the knowledge can be summarized by at most a few thousand rules and there does not exist a sound mathematical model behind these rules.

Having said this, the benefits of expert systems for capturing the knowledge of an expert and making it available to non-experts for support in their decision-making cannot be ignored. This potential, for example, has been recognized by project management community and as a result there has been a significant amount of work integrating the expert systems with project management. Frankel [47] and Hosley [76] present a general view of expert systems in project management. It is noted that a project planner can accumulate a large body of knowledge in his domain as result of previous work and this knowledge is precious for other planners who do not have the expertise. The expert systems can also help a planner by making transparent to him how a decision is arrived. The forward or backward chaining in reasoning can be made clear by showing the rules and the order in which they are used.

An interesting expert system example is so-called cognitive planning in which the different reasoning methods of an expert are tried to be emulated. Bernard *et. al.* [13] describes the use of a temporal reasoning model for aircraft maintenance planning. Another more conventional example is PROJCON in which the specific techniques for construction management planning is programmed [54]. Gudes *et. al.* [68] describes an expert systems based methodology for solving resource allocation problems by using expert heuristics. Thus, the best known applications of AI consist of using domain-specific knowledge and techniques for each distinct problem through expert systems.

Further uses of knowledge-based approach are directed at making the data-base more suitable for logical inference. For example, Elleby [39] develops a method, which he calls "Extended Relational Analysis" for better scheduling by making it clear how the derived facts arise from the data-base. Another paper by Kasahara *et. al.* [83] shows how interactions in complex systems and interference among tasks and operators can be modelled with a knowledge-based approach for automated planning and scheduling for maintenance work in a nuclear power plant.

2.3.1 Extension to Nondeterministic Models

Since most experts had to reason under uncertainty, the probabilistic modeling was used very early in the area of knowledge-based systems. Indeed, one of the pioneering expert systems, MYCIN, used a certainty factor for each rule by which the confidence in the associated rule is specified. Then, MYCIN was able to find out the most likely outcome by taking into account these belief measures. Kangari and Boyer [80] note that accommodation of risk management to expert systems must be particularly useful for project management purposes. Walmsley [142] gives examples of project planning problems in the domain of construction management that can be solved by expert system methodologies. He also notes that these techniques will be very useful when applied continuously as the project plan is being performed. Another system developed by Shaw and Whinston [127] uses a dynamically changing knowledge-base for flexible scheduling. One can also see many examples of knowledge-based project planning systems that use certainty factors and probabilistic calculations for comparing different project alternatives. It can be said that there is a considerable background in this area that may prove useful for drawing conclusions on what approaches should be avoided in accounting and managing uncertainties in planning. The best incorporation of probabilistic calculations to production rule systems seems to be based on the Dempster-Shafer theory [80, 117].

2.4 Planning as Distributed Computation

Yet another different approach used in AI problems is the simulation of an organization which uses different agents, hierarchy and negotiations for solving a problem. In fact, in view of the deficiencies of logical formalism and knowledge-based systems, there is an increasingly more accepted view in the AI community that intelligence is an emergent property that arises as a result of interactions of many independent problem-solving agents. Minsky's *Society of Mind* theory is the primary example [98] of this approach. According to this theory, brain consists of many agents, hierarchically organized like a bureaucracy, and both cooperation and competition is allowed.

In the application of distributed computation to planning, the subproblems are solved by different agents each of which can be considered as an expert in their limited area. The use of the experts can be controlled through a centralized blackboard and executive or in a more distributed fashion through pairwise execution [72]. A paper by Hadawi *et. al.* [69] specifies a distributed architecture, named REDS (Requirement Driven Scheduling), for the factory-level job-shop scheduling. They note in this paper that their system is superior to conventional systems because of its ability to look at a problem from different perspectives as well as its flexibility. Another supporter of this approach is Findler [42], who came to develop a computer-based theory of strategies as a result of his research experience in the field of distributed-computation. In his book [42], Findler gives an example of distributed-planning system with considerable flexibility. Another example, which is due to Phinhobmongkol and Chang [110], uses meta-knowledge on top of the knowledge of agents' planning knowledge. It is claimed that this method is especially useful for conflict resolution.

To sum, distributed-computation is perhaps the most promising branch of AI. Such systems can very easily be adapted to parallel computers and neural networks. They can work with less than idealized input; for example neural networks have the property of "graceful degradation". They are also more suitable for understanding of how human intelligence works as well as duplicating it exactly. The property of similarity with human intelligence, however, could also be the limitation for distributed-computation systems. These systems are difficult and less transparent to improve performance-wise due to their complexity, compared to the centralized approach. As a result, the benefits that could be obtained by using the distributed computation are not clear. However, it can be expected that this field will mature and will be put into more extensive use.

2.4.1 Extension to Nondeterministic Models

We have noted that *distributed-computation* has the advantage of being inherently applicable to nondeterministic conditions. For example, the neural networks used for character recognition can read quite distinct handwritings. The planning models that use distributed-computation will be able to produce results that are not optimal but still acceptable, even though some agents cannot get all the information they want. This flexibility is always emphasized by the researchers in this area. Burke *et. al.* [19] note that in their distributed-planning system, the external world is modelled as just another agent, but one with whom negotiation is disallowed. So, the distributed-planning systems do not need to make a clear-cut distinction between incomplete and complete knowledge. However, the limitations of this technique are not yet clarified. Findler [42] points out that the applications in this area are few and there is a need for a more general understanding of the method.

2.5 Search and Combinatorial Optimization in Planning

To sum, the use of symbolic reasoning from the first principles, has not been particularly useful for AI purposes unless the context-dependent, domain knowledge is used. Often, the largest impediment encountered in AI studies has been the lack of an efficient solution method for the problems being considered. In such problems, the set of candidate solutions grows exponentially as the problem size increases (this is also called combinatorial explosion); yet there does not exist a universal method that can find the desired solution efficiently. Many planning problems can be considered in this category because finding a best plan among a large set of candidates is often necessary in planning and scheduling. Because the process of searching for the best solution can also be considered as an optimization process, search is closely intertwined with a field known as **combinatorial optimization**. Search and combinatorial optimization problems are ubiquitous and hence have been scrutinized in many fields, including operations research. In this section, we examine the applications of search techniques to the problems of planning/scheduling.

2.5.1 Planning/Scheduling as a Mathematical Problem

It is possible to express the planning problem as a mathematical problem by limiting the attention to the particular aspects of the problem. This formalism helps to understand the complexity of the problem and the requirements for the existence of an efficient solution. Let us recall that we define a project as a coordinated set of activities and tasks designed to achieve an objective by the development of physical, service, or other capabilities, under conditions of defined schedules, budgets, and performance criteria [47]. This definition is similar to that of the automated planners but it avoids the problem of world modelling (the selection of the world and operator representation formalisms) by requiring a work-breakdown structure be already given by the project manager.

We can then define the deterministic planning problem [7, 12, 49] by using the five-tuple system $(\tau, \prec, [D_{ij}], [R_i], w_i)$, where :

1. $\tau = T_1, T_2, ..., T_n$: The set of tasks that can be executed.

2. $\prec$ is an irreflexive partial order defined on τ which specifies operational precedence constraints. That is, $T_i \prec T_j$ signifies that T_i must be completed before T_j can begin.

3. $[D_{ij}]$ is an mxn matrix of execution times, where $D_{ij} > 0$ is the time required to execute T_j, $\leq j \leq n$, on processor, P_i, $1 \leq i \leq m$.

4. $R_j = [R_1(T_j), ..., R_s(T_j)]$, $1 \leq j \leq n$ specifies the amount of resources required throughout the execution of T_j. Note that processors are not usually included in resources.

5. w_i, $1 \leq i \leq n$, specifies the weights which will be used in performance measures. They are interpreted as cost rates and may be arbitrary functions of scheduling properties influencing T_i.

The problem can be solved in two basic steps :

i) Specification of the subsets τ, $\prec$, D, R and w.

ii) The development of a schedule for this given five-tuple system. We must note that for realization of the project goal, an iterative process may be necessary, i.e. after having gone through step **ii**, one might find that no schedule for the subset determined in step **i** satisfies the given goals and therefore it may be necessary to select a different subset.

The first part of the above process is usually named planning process. In this process, a knowledge-base of world models and the methods of reasoning about the knowledge-base are required. It is also important to specify a time-range for the problem. Short-term planning assumes that a general set of activities, resources and policy constraints are given and thus it concentrates on the choice of specific activities and resources for finding a good plan. Only after learning from short-term planning, one can perform long-term planning, through which one may redefine goals or may change the context on which plans are based. We will confine this discussion to short-term planning.

The second step is usually called the scheduling process. A schedule can be defined as a suitable mapping that assigns a sequence of one or more disjoint execution intervals in $[0,\infty)$ to each task such that [12]:

1. Exactly one processor is assigned to each interval.
2. The sum of the intervals is precisely the execution time of the tasks, taking into account, different processing rates of different processors.
3. No two execution intervals of different tasks assigned to the same processor overlap.
4. Precedence and additional resource usage constraints are observed.
5. There is no interval in $[0, max\{f_i\})$ during which no processor is assigned to some task.

One can specify further constraints on the desired schedule. The two most common ones are :

1. *Nonpreemptive scheduling*: A task can not be interrupted once it has begun.
2. *List scheduling*: An ordered list of tasks in τ is assumed or constructed beforehand. This list is called priority list. Specifically, when a processor becomes free for assignment, the list is scanned until the first unexecuted task in list is found and done.

Given this general model for scheduling, one can define some performance measures as follows:

- **Makespan** (C_{max}) : The completion time of the schedule. Let S specify the schedule, and C_i the finish time for task T_i.Then,
  $Makespan(S) = C_{max}(S) = max_{1 \leq i \leq n} C_i(S)$
- **Maximum Flow Time** (F_{max}) : Flow time is defined as $F_i = C_i - r_i$, whereas r_i ready (or start) time for task T_i.
- **Maximum Lateness** (L_{max}) : If for a task T_i due date is given as d_i, $L_{max} = max_i(C_i - d_i)$
- **Maximum Tardiness** (Tr_{max}) : Tardiness is defined as $Tr_i = max(L_i, 0)$.

Often, the mean values such as $\overline{C}$, $\overline{L}$, $\overline{Tr}$, $\overline{F}$, (they could be weighted) can also be of interest to a planner. The cost of a particular schedule is given as:

Cost=Function (Performance-Measures)

The type of the function will depend on the specific problem. For example, in a project with a given due-date one would be interested in minimizing C_{max}, Tr_{max}; whereas in a project with expensive processing costs one could be interested in minimizing *both* the mean weighted flow time and C_{max}. In general, the planning/scheduling of a project can be considered an optimization problem in the space defined by desired performance measures subject to given constraints.

2.5.2 Mathematical Aspects of Scheduling Theory

The scheduling problem is not only a beautiful mathematical problem but also is one with numerous applications. It has been attacked by many researchers and a substantial amount of literature exists on its details. A quick and general survey of the field is as follows:

1- The scheduling problem can be classified into three major categories: the assembly-line balancing problem, the job-shop scheduling problem, the project scheduling problem [12].

Assembly line balancing is interested in deciding the minimum number of workstations that are assigned to work elements with some precedence relations, or minimizing the cycle time, that is the maximum among total working time at each station, under the given number of workstations.

Job-shop scheduling problem is to determine the sequence of n jobs on each of m machines in order to minimize a given objective function. A variant of this problem is flow-shop problem where each job has identical ordering.

Project scheduling is also called coordination problem and is concerned with the planning which consists of activities that must be processed by following the given precedence relations and resource constraints.

The multiproject scheduling problem with resource constraints is the most general scheduling problem and a correspondence between this problem and other problems can be established [12].

Multiproject-scheduling Problem	Job-Shop Scheduling Problem
Project	Job
Activity	Operation
Precedence Relation	Ordering
Resource	Machine
Resource availability	Number of identical machines
Resource requirement by each activity	Number of identical machines that can process each operation

2- There have been some advancements in the classification of problem with respect to definitions [49]:

Equivalence of Measures : Two performance measures are equivalent if a schedule which is optimal with respect to one is also optimal with respect to other and vice versa, e.g. $\bar{C} = \bar{F} = \bar{L}$

Efficiency : A schedule is efficient with respect to measures $\mu_1, \mu_2, \ldots \mu_l$, if there does not exist a schedule S such that $j, \mu_j \leq \mu_j^{'}$

Regular measure : One that is non-decreasing in the completion times. That is R is a function of $C_1, C_2 \ldots C_n$ such that

$R(C_1, C_2 \ldots > C_n) \leq R(C_1^{'}, C_2^{'} \ldots C_n^{'})$ if $C_i \leq C_i^{'}$.

By these definitions, it has been possible to show the limits and scopes of scheduling algorithms regarding different performance measures.

3- In very simple cases, it has been possible to give constructive algorithms, i.e. algorithms that will give the unique answer in time polynomial with the problem size. For example, consider the sequencing problem with n tasks, one machine, empty resources and empty precedence sets ($R = \phi, \prec = \phi$). For optimization with respect to different measures different constructive algorithms must be employed. Some of these are as follows: The tasks must be sequenced in an order of

i. non-decreasing processing-time (shortest-processing-time-first), for minimizing the mean flow time,

ii. non-decreasing due dates, for minimizing L_{max}, Tr_{max},

iii. non-decreasing slack times, for maximizing L_{min}, Tr_{min}.

Another example of a constructive algorithm is so-called Johnson's algorithm for minimizing F_{max} with 2 processor machines [12, 49]. Unfortunately, the domain of constructive algorithms does not currently extend to the problems with more than two different types of resources.

4- The general scheduling problem is found to be computationally intractable, i.e. the best solution algorithm to the problem is super-polynomial in running time (with respect to the problem size). For example, even when $R = \phi$ and $\prec = \phi$, a problem with n tasks and m machines requires a solution from the combination set with $(n!)^m$ cardinality. Since a brute search process for finding the schedule with minimum cost (the British Museum approach!) is intractable, algorithms that employ more efficient search methods have been developed. Note that these search methods are not unique to the scheduling problem, but, generally speaking, they are applicable to a whole class of problems, called NP-complete.

Owing to the complexity of the scheduling problem and unavailability of a universal solution procedure for NP-complete problems, the general scheduling problem is solved in most cases by algorithms that make use of special heuristics and give satisfactory solutions rather than the optimal answer.

2.5.3 OR-based Methods for Planning/Scheduling Problem

Because of the practical importance of the problem, OR methods have been developed for the general problem of scheduling. It must be noted that the theory generally assumes that the set of tasks and operations required for carrying out the plan are already determined. Then the general planning problem is reduced to that of finding an optimized schedule under given constraints. Generally, directed-acyclic-graphs (DAGs) are used for representing such problems with clear visualization of the precedence relations. The developed graph then can be analyzed by using the network analysis techniques. Well-known applications of DAGs in the project planning involve the PERT (Project Evaluation and Review Technique) and the CPM (Critical Path Method). The PERT solves the problem of finding the schedule for a single project that minimizes the objective function such as the project completion time. This technique is based on the calculation of early-start/early-finish times for specified tasks through a forward propagation and late-start/late-finish times through a backward-propagation. In this way, the slack times for each activity can be determined. Then, the paths from the start to the finish can be sorted according to their durations. The path with longest duration is named the critical path. This knowledge then can be used for focusing on the activities that are along the critical path. The CPM is a similar technique that is developed for finding the schedule with the minimum cost with a pre-specified project completion time in the case costs can be associated with each activity.

The PERT assumes that the resources are infinite and thus, all the tasks that can be performed in parallel, as long as they obey the precedence constraints. If the resources are limited, however, a path which otherwise would not be critical may turn out to be a critical path, provided that it employs minimally available resources. Thus when there is a problem with the availability of resources, the problem turns into a combinatorial resource allocation problem. For this reason, the project scheduling problem has always been a subject of interest for combinatorial optimization. The problem has been attacked by either one of the known techniques for solving combinatorial problems, such as general integer programming, or specialized algorithms for different problems which, in fact, resort to different versions of heuristics.

The classical algorithms employed for combinatorial optimization problems will generally fall into one of the following classes:
1. Dynamic Programming: Based on the concept that the subpaths of an optimal path would have to be optimal, this method enumerates options at each step of the scheduling and eliminates the non-optimal ones.
2. Branch & Bound : Generates tree schedules and eliminates those with higher costs.
3. Integer Programming : Recasts the problem as an integer programming problem and solves it in this domain.
4. Heuristic Approaches : In the process of scheduling, one has to make certain choices at each step. Heuristic approaches make use of some rules for these choices even though the rules may not have a mathematical basis. Some of the exemplary rules are **Random**, **First-Come-First-Served**, **Most-Work-Remaining**, **Least-Work-Remaining**, **Most Operations Remaining** or even a random choice from a set of such rules. Note that some of these rules are contradictory! However, these approaches are often found to be satisfactory in practice [28].

2.5.4 AI-based Methods for Planning/Scheduling

The development and analysis of heuristics for the scheduling problem is an area where both the OR and the AI approaches intersect. One can find in literature related to the AI applications many research papers that find particular heuristics to particular problems. For example, for job-shop scheduling, Cartesian-filling heuristic is developed by Pierce [111]. As a result of such studies, new heuristics are being continuously added to the repertoire of solution techniques for computationally hard problems. Further benefits of the AI come from the classification of the search methods and the development of problem independent search methods.

In addition to the classical search methods of OR, the past AI research in the search-related problems have produced new ways of attacking these problems. Recently, new powerful methods, such as **genetic algorithms** and **simulated annealing** have emerged. Applications of such techniques to the scheduling problem already appear in literature (see Keler [84], for a review). *Simulated annealing* can be seen a form of search that normally proceeds along the gradient of optimization, but at times take reverse directions so as not to be confined in a local optimum. *Genetic Algorithms* are modelled after evolution process. These techniques are examined in detail in Chapter 3.

Another new idea is the use of **constraint propagation** techniques for limiting the search under given constraints. Whereas logic is not always an answer to symbolic reasoning problems, it can help by reducing the search space. For example, given that $x + y \geq 2$ and $xy < 0$ for x and $y \in \{-3, -2, -1, 0, 1, 2, 3\}$, we can deduce that either $x = -1, y = 3$ or $x = 3, y = -1$ and can select either of these pairs according to other criteria. This computation is much more efficient than a brute search process that tests each possible pair and eliminates infeasible solutions only after they have been generated. The constraint-propagation procedure is based on the reduction of solution set by applying the constraints and making inferences first. Perhaps, the most noteworthy recent development in AI-based work is the adoption of logic programming to constraint propagation techniques. The paradigm of constraint-based logic programming is claimed to be an efficient computation model as well as an environment which allows more flexible and fluent programming environment. Van Hentenryck notes that [139] whereas specialized programs for scheduling may take months to be developed, most complex scheduling problems can be programmed in a few days and can be solved with almost the same efficiency with the constraint-based logic programming. A further claim of this paradigm is its ability to handle disjunctive constraints, so that precedence relations can be defined with more freedom. Constraint-propagation can be expected to increase the efficiency of search process in scheduling problems because such problems are generally subject to many constraints rather than being purely combinatorial. Finally, the method known as **lifting** can be promising for alleviating the search. In lifting, search is postponed as far as possible by keeping the variables unbound and applying the symbolic reasoning to the expressions containing these variables.

2.5.5 Extension to Nondeterministic Models

Inclusion of uncertainty can be relatively straightforward in search problems. Nevertheless, such inclusions generally increase the complexity of the problem being considered. Consider a case where we have stochastic activity durations. That is, an activity duration d is specified by a probability density function (**pdf**). Let us assume that the **pdf** of an activity can be approximated by a discretization scheme: An activity duration d can take n different values, where $d \epsilon D$ given that $D = \{d_1, d_2 \dots d_n\}$ and each d is associated with a unique probability value. The calculation of **pdf** for project duration can be accomplished by considering the realization of every alternative individual task duration value. Unfortunately, this approach is intractable due to its exponential complexity. In a given path with m activities, with each **pdf** being represented by n distinct duration values, $\mathbf{n^m}$ calculations are needed.

Thus, the nondeterministic problem can still be examined as a search problem, however, at the expense of increasing complexity. Therefore, it becomes more important to use powerful optimization or search methods and to develop efficient heuristics. This area will be examined in more detail later.

2.6 Other AI based Approaches

Various AI techniques could be beneficial for simulation purposes. Although they generally do not bring any new theoretical advantage to the Monte-Carlo simulation they can make the modelling and programming task easier [50]. Further approaches to the nondeterministic planning combine different methods. Levitt gives an example of how AI programming techniques plus expert systems can be combined to automate scheduling [92]. Tan describes a neural network optimization approach in which the data is obtained through Monte-Carlo simulations [132] for identifying a preventive maintenance policy which minimizes cost. Lozano-Perez *et. al.* [93] use a robot path planning algorithm based on *compliant motion* concept. An example of this type of motion, is to drag an object along the table until it meets with the hole where it is supposed to be placed in contrast to picking the object up, bringing it exactly on top of the hole and dropping it. This clever strategy is found to be very robust in the face of uncertainty. However, it is difficult to see how compliant motions can be discovered in the planning/scheduling problem by natural intelligence let alone artificial intelligence!

2.7 Summary and Conclusions

We observed that there exist different approaches to planning each of which has a different emphasis. Automated planning systems aim to emulate the common-sense planning based on the frame axioms and logical formalism. Knowledge based systems aim to make the most use of the particular rules that are specialized for particular planning domains. Distributed computation approach propagates

	FOCUS ON COMPLEXITY	FOCUS ON MODELLING
GRAPH SEARCH	xxxxxxxxxxxxxxxxxxxx	xxxxxx
AUTO. PLANNING	xxxxxxxxxxxx	xxxxxxxxxxxx
KNOWLEDGE-BASED	xxxxxxxx	xxxxxxxxxxxxxxx
DISTR. COM	xxxxx	xxxxxxxxxxxxxxxxxxxxxxxx

Table 2.1: The differences between AI approaches

the division of labor among specialized agents and suggestp that an overall planning capability may inherently emerge from such cooperation. Graph search suggests that plans can be generated through optimum path finding algorithms operating on the data that represent the states of the world. Despite their differences, all these approaches face similar issues related to complexity.

A planning system can focus either on such complexity issues or on the other issues such as the capacity for modelling. Some AI approaches are more focused on the capacity and the ease of modelling, while some others allow more attention to be paid to the complexity issues. Table 2.1 indicates these differences for different AI approaches. Distributed computation systems are based on an extensive modelling of a system but the complexity problems that may emerge in these models have rarely been paid attention. Knowledge-based systems use production rules that provide a high degree of freedom for modelling purposes; yet they have a relatively low emphasis on the complexity issues. Automated planning systems have limited their capacity to particular formalisms, for example, the model for STRIPS planning is known as STRIPS logic. Graph search problems generally employ simple data structures; yet the emphasis is on the complexity issues.

On the other hand, the complexity problems cannot be avoided if one wants to make sure that these approaches will result in practical uses. For example, it is proven that automated planning under a very general representation is an undecidable problem, see Chapman [21]. It is also proven that the STRIPS planning corresponds to the complexity class PSPACE. This indicates that the STRIPS planning can be solved with graph search through a free graph model in which the vertices are a collection of propositions [97]. The equivalence of graph search and STRIPS logic indicates that the representation issues are often a syntactic concern. While the logical formalism approach seems to be more rigorous than the other approaches, it, too, is subject to similar constraints on complexity. It is proven that a complete set of inference rules cannot be decided on polynomial time even in the simplest logic model (propositional logic) [97]. The method commonly used for drawing inferences in propositional logic is known as Boolean Constraint Propagation (BCP). BCP is linear time decidable; however it is not complete for it fails to incorporate the case analysis rule (therefore one cannot derive Q given that $P \rightarrow Q$ and $\overline{P} \rightarrow Q$). Higher order logics have larger representation capacities, but their determination may be computationally much harder or intractable.

The trade-off between focusing on complexity or modelling capacity in a given approach to planning indicates that one should choose a particular approach depending on what aspect of the problem will be emphasized. In this thesis, we are concerned with drawing general lessons for planning problems under uncertainty and therefore have chosen the graph search approach. This also makes it possible to benefit from the experience of OR studies on graph problems. While there is an extensive amount of work on planning under uncertainty in different paradigms, especially in knowledge-based system approach, these studies are generally *ad hoc* and have limited use. In contrast, the results of graph search may prove beneficial for different planning paradigms. The next chapter therefore reviews the graph search algorithms and the complexity issues in more detail.

Chapter 3

Graph Search, Complexity and Genetic Algorithms

In this chapter, we introduce the graph model paradigm for planning problems (*Section 1*), and we discuss the complexity of the problems often encountered in this paradigm (*Section 2*). We eamine both the exact (*Section 3*) and the approximate (*Section 4*) methods to solve the computationally hard problems. Later, we review and compare the graph search and optimization techniques generally employed in the solution of these particular problems (*Section 5*). We present a detailed analysis of genetic algorithms (*Section 6-9*). Finally, we produce a probabilistic model of genetic algorithms comparing them to a random search model (*Section 10*) and argue that (*Section 11-12*) the genetic algorithm is a successful, general heuristic for solving the complex search and optimization problems.

3.1 The Capability of Graph Models in Expressing Planning Problems

One of the first questions that should be addressed in any problem concerns the representation of the problem. The choice of representation or modelling methodology defines the range of solvable problems as well as the effectiveness of the solution techniques. The problem is especially acute in the planning arena for there are various paradigms of planning as the need for making *a priori* multistage decisions arises in various domains. For example, the paradigms of integer programming or linear programming that are used for minimizing some objective function can be generalized to cover multistage problems. If we are concerned with giving problem-solving capability to a robot, we prefer first-order logic for modelling the robot's environment and in the development of the planning algorithms. In business, the selection and the temporal ordering of business strategies in order to achieve preset goals constitutes an important problem which is often modelled within the framework of decision analysis or game theory. In project management, the problem manifests itself as the assignments of the activities to individuals and time-slots for the early completion of a project and is generally subjected to PERT type network methods. In short, we observe that many types of planning problems can be found in widely differing contexts, most of them being attacked with specialized methods. The selection of a methodology that could cover the common grounds of these contexts constitute an important problem in itself.

Because we hope to draw lessons regarding planning under uncertainty that might be of value in many areas, we have chosen the graph model for analyzing the planning problems. The representation power of the graph models as well as their adaptability to problems that belong to different contexts justifies our decision to some degree. For example, it has been shown that some logic models used in the robot planning are only a syntactic variation of the graph model [97]. Graph models have long been used in planning problems, especially in the context of project management. Furthermore, there is a large background work on the graph theory, both scrutinizing the theoretical problems and applying the theory to practical problems.[1] Therefore, graph models are a natural choice for use in planning problems.

Another potential of graph models arises from the opportunity to represent the basic types of planning problems with the classical problems of graph theory. For example, the most simple planning problem involves the sequencing of a set of tasks. A prime example of this *permutation* problem is so-called the Traveling Salesman Problem (TSP) of graph theory, in which one wants to obtain a particular permutation of all the cities to be visited such that the total travel cost is minimized. Some planning problems require in addition to ordering, the selection of tasks among a set of potential candidates. The selection process, too, must take into account the possible constraints of the problem. The *selection & permutation* problem has a natural counterpart in graph theory with the shortest path type of problems. If there exists no total ordering constraints (such as one activity at a time), a nonlinear plan (the one with parallel branches) can be represented with a set of partial paths minimally ordered. At the far end lies problems that also deal with resources in addition to sequencing and task selection. It is customary to call such type of problems as *resource allocation* problems. An example of resource allocation problems is the scheduling of projects subject to constraints on manpower and machine availability. The visualization of such problems in graph models is relatively difficult, but they too can be effectively dealt within that framework.

[1]Graph models are often criticized for being unsuitable for representing continuous processes. Some techniques to work around this problem have been suggested, but they are cumbersome to use [70]. However, similar critiques can be directed at many other models as well.

3.2 Complexity Theory on Planning Problems

Most planning problems can simply be expressed as a computational problem on graph models. Complexity theory provides a framework for classifying the planning problems according to their difficulty. In this section we examine what complexity theory has to say about planning problems. But first, we should clarify what is meant by the complexity of a problem.

3.2.1 Complexity of a Problem

Consider the problem of determining whether a given number n is prime. There are some general methods for answering this question theoretically but they are completely unsuited for practical computations. For example, a number n is prime if and only if $n|[(n-1)!+1]$, i.e., n divides $(n-1)!+1$ without remainder [129]. This is an interesting property of primes but it is totally useless for verifying that a 20-digit number like 435835313536579538727 is prime or is not prime. Twenty-five centuries ago, the Chinese gave what they believed was an infallible rule for determining primality. Their rule stated that n is prime if and only if $n|(2^n-2)$. This rule was believed to be true for twenty-three centuries. Further, Fermat showed that the Chinese were correct when n is a prime. Yet the rule fails for $n = 341(= 11 * 31)$. Indeed, there is no known algorithm - let alone a formula - today that will decide the primality of a number in running time bounded by a polynomial function depending on the data size of the number n, formally defined as the number of digits required to express it in the binary system [129].

If the maximum number of the elementary computational steps in an algorithm, (therefore the maximum time it takes to perform the algorithm) can be given as a polynomial function of the input data size, this algorithm is said to have a polynomial (**P**) *complexity.* If a problem has a solution procedure having **P** complexity, then we say that the problem belongs to class **P**. Since there is no known **P** algorithm for the determination of primality, it is believed that the problem of the determination of primality is not in **P**; although some suspect that a **P** algorithm for this problem, on whose difficulty modern cryptography depends, might be found [129].

Most of the algorithms in common use today have polynomial complexity. Examples include *sorting* ($O(n \log n)$), *finding an Eulerian cycle* in a graph with m edges ($O(m)$), *constructing the minimum-cost spanning tree* for a graph with m edges ($O(m \log m)$), *finding the shortest path* between given vertices in a graph with n vertices and m edges ($O(mn)$), *finding the transitive closure* (finding all vertices that are directly or indirectly connected to a given vertex in graph with n vertices) ($O(n^2)$), *testing a graph for planarity* ($O(n)$), *finding the maximum matching* ($O(n^{5/2})$) and *finding the maximum flow* in a network ($O(n^3)$). Some problems, on the other hand, can be solved only with algorithms having exponential complexity, i.e., they have **E** complexity. An obvious example is that of finding all subsets of a given set ($O(2^n)$). Many problems, however, do not quite fit into either class **P** or **E**. An example of these last class is the problem of scheduling about which Conway [28] observes that *"Many proficient people have considered this problem and all have come away essentially empty-handed. Since this frustration is not reported in the literature the problem continues to attract investigators who just cannot believe that a problem so simply structured can be so difficult until they have tried it!"*.

3.2.2 Complexity Class NP

Now suppose that we somehow know that a particular number n is prime. Can we prove this in a reasonable amount of time? The answer is yes thanks to an algorithm devised by V Pratt [113] based on a theorem that every prime number must satisfy (which in turn follows from the small Fermat theorem). The complexity of Pratt's algorithm is **P**. Nevertheless Pratt's algorithm only *certifies* (verifies) the primality of n; it does not solve the aforementioned problem of the *determination* of primality. If the solution of a problem can be verified in polynomial time if the answer is known beforehand, we classify this problem as *Nondeterministically Polynomial* or **NP** in short.

An interesting feature of the problem of determination of primality is the fact that problem itself also belongs to the class of **coNP**, which is the complementary class of **NP**, and consists, essentially, of those problems for which it is simple to check the correctness of a negative solution [87]. In other words, it is not difficult to prove that n is not a prime when we know that it is composite. A nice illustration is given in [113]; in order to disprove the hypothesis of Mersenne, 200 years old, that $2^{67}-1$ is prime, F. Cole needed in his own words "3 years of Sundays". However, when he lectured on his result at the meeting of American Mathematical Society in 1903, all he needed to do was to write down the following equality: $2^{67}-1 = 193707721 * 761838257287$.

For a rigorous definition of the class **NP**, we should first clarify what is meant by a nondeterministic algorithm. A nondeterministic algorithm differs from a deterministic one by the feature that it is possible to decide quite freely between several possible continuations of the computation. The computation is thus not determined by the initial configuration and instead of a computation we should rather speak about the system of all possible computations [87].

As an illustration, we describe a simple algorithm [87] for deciding whether there is an independent set X (such that no edge has both endpoints in X) of at least k elements in a given graph **G** of vertices $v_1, v_2, ..., v_n$.

1. [*Initiation*] $X := \emptyset$

2. [*Nondeterministic construction of* X] For $i = 1, 2, ..., n$ perform the following instruction: put either $X := X \cup v_i$ or $X := X$.

3. [*Verify independence of* X] If there exist vertices v_i and v_j in X such that v_i, v_j is an edge then *reject.*

Table 3.1: The Time It takes to Solve a Problem (*1 step = 1 μs*)

	n					
f(n)	10	20	30	40	50	60
n	1.10^{-5} s	2.10^{-5} s	3.10^{-5} s	4.10^{-5} s	5.10^{-5} s	6.10^{-5} s
n^2	1.10^{-4} s	4.10^{-4} s	9.10^{-4} s	16.10^{-4} s	25.10^{-4} s	36.10^{-4} s
n^5	0.1 s	3.2 s	24.3 s	1.7 min	5.2 min	13 min
n^{10}	2.7 h	118.5 days	18.7 years	3.3 cent.	30.9 cent.	192 cent.
2^n	0.001 s	1.0 s	17.9 min	12.7 days	35.7 years	366 cent.
3^n	0.59 s	58 min	6.5 years	3855 cent.	2.10^8 cent.	1.10^{13} cent.
$n!$	3.6 s	770 cent.	8.10^{16} cent.	2.10^{32} cent.	9.10^{49} cent.	3.19^{66} cent.

4. [*Count the number of vertices*] If X has at least k elements, then *YES*, else *NO*.

This algorithm should reply *YES* or *NO* according to whether the required independent set exists or not. The algorithm is naturally worthless when we actually have to find an independent set of the required size because we would have to go through all, or a majority, of the possible continuations of step 2, and since these possibilities correspond to all possible subsets of the vertex set of the graph G, their number is 2^n. This algorithm, however, can be used, after we managed to find a solution, to immediately prove the correctness of the solution.

Let us imagine a hypothetical random access machine that can unconditionally jump to one of many possible continuations in the program, yet somehow could process all admissible computations in parallel, i.e., without putting these continuations in order, process one by one. This machine either accepts the answer or rejects it after exhausting all possible computations in polynomial-bounded time. By **NP**, we denote the class of all problems **J** such that there exists a nondeterministic computing model which works in polynomial-bounded time and answers the problem **J**. The practical importance of these definitions arises from the fact that with the computers we have (sequential or parallel with a fixed number of processors), the problems that belong to **NP** cannot be answered with polynomial algorithms. As a result, we will have to use exponential algorithms for a complete determination of the answer, which would be intractable for all practical purposes if the input data size is large.

For a better appreciation of the awful slowness of an exponential algorithm, we include Table 3.1 that presents the time it takes to solve a problem depending on the complexity of the underlying solution algorithm for a computer that processes 1 million instructions per second (1 MIPS). Table 3.2 shows that, when dealing with algorithms having exponential complexity, switching to a 1000 times faster computer would not be helpful since the problem size that could be handled stays almost constant.

The relation between classes **P** and **NP** is subtle yet not completely understood. Since a deterministic

Table 3.2: Computational Gain in the Solvable Problem Size by a Faster Computer

f(n)	Benchmark Comp. The size of Data	1000 times Faster Comp. The size of Data
n	n_1	1000 n_1
n^2	n_2	31.62 n_2
n^5	n_3	3.98 n_3
n^{10}	n_4	1.99 n_4
2^n	n_5	n_5+10
3^n	n_6	n_6+6
n!	n_7	n_7+{1,2,3}

algorithm is a special case of a non-deterministic one, we have the following:

$$P \subset NP \tag{3.1}$$

It is not yet known that whether $P = NP$ or $P \neq NP$. This question is often considered to be one of the most important open problems of modern mathematics, and it has remained unsolvable despite an immense effort by a great number of mathematicians.

3.2.3 NP-complete problems

Another subtle property of class **NP** is due to the fact that many problems in **NP** are reducible to others, i.e., they can be answered by using the solutions of another problem. From the historical point of view, the following result of S A Cook [52] had been of major importance: Cook showed that every problem which is a member of the class **NP** can be reduced in polynomial-bounded time to the problem of determination of satisfiability of logical formulae in conjunctive normal form. This last problem tries to find whether there exists a model of Boolean variables x_{ij} yielding **true** for the formula

$$X = \bigwedge_i \bigvee_j^m x_{ij}, \qquad m \geq 3$$

If every problem that is in **NP** *can be reduced to a particular problem J in polynomial-bounded time, the problem J is said to be* **NP-hard***; further, if J too, is in* **NP***, it is called an* **NP-complete** *problem.* For example, the *satisfiability* problem, of which every *NP* problem is reducible in polynomial-bounded time, can be reduced to the aforementioned *NP* problem of *finding an independent set* via a quick algorithm, therefore the problem of finding an independent set is *NP-complete.* In a sense, an *NP-complete* problem serves to summarize the complexity of the entire class **NP**. At the present time, several hundreds of *NP-complete* problems are known.

Many such problems are found in the fields of number theory and set theory. Some examples are *quadratic diophantine equations* (do there exist natural numbers x and y with $ax^2 + by = c^2$ where a, b, and c are given natural numbers?), *quadratic congruence* (does there exist a natural number $x < c$ with $x^c = a \bmod b$ where a, b, and c are natural numbers?), *knapsack problem* (do there exist binary variables x_i that satisfy the equation $\sum_i a_i x_i = b$, where a and b are integers?), *binary partitioning* (does there exist a binary partition of set S of integers x_i into two subsets S_u and S_l such that $\sum_{i \in S_l} x_i = \sum_{i \in S_u} x_i$?).

Also, a large number of problems in *Artificial Intelligence* involve NP-complete problems. They are frequently encountered in the fields of diagnosis, image recognition, propositional calculus and language understanding [11]. For example, the following question is NP-complete: Given a monocular picture, what is the best three-dimensional description of the locations of the objects? Indeed, what are the objects? In planning, computing the correctness of a logical formula in a nonlinear automated planner using modal truth criteria has been shown to be NP-complete [146].

In addition, most of the problems of *Operations Research*, such as optimal factory location or optimal vehicle scheduling can be stated in terms of classical NP-complete problems. Integer programming, which expresses a large number of problems addressed in Operations Research, is NP-complete.

Furthermore, many NP-complete problems arise within the context of graph models and thus directly related to planning and scheduling. Some examples are as follows:

For a given graph $G(V, E)$ with n vertices and a given natural number K,

- *3-coloring:* Can the graph be colored by three colors?
- *Independent-set:* Does the graph contain an independent set of vertices of a prescribed size?
- *Hamiltonian-cycle:* Does the graph have a Hamiltonian cycle (a cycle passing through all of the vertices) ?
- *Comparative TSP:* Is there a TSP solution having a cost less than a preset threshold?
- *Width of a graph:* Does there exist a labelling F of the vertices by the natural numbers $1, ..., n$, such that for each edge $[u, v]$ we have $|F(u) - F(v)| \leq K$?
- *Optimum linearization of a graph:* Does there exist a labelling F of the vertices, such that $\sum |F(u) - F(v)| \leq K$?
- *Minimum cut of a graph:* Does there exist a disjoint composition of V into two non-empty sets X and Y such that the number of edges $u, v \in E$ with $u \in X$ and $v \in Y$ is smaller than or equal to K?

- *Graph thickness:* Do there exist sets $E_1, ..., E_k$ such that $E_1 \cup, ..., \cup E_k = E$ and the graph (V, E_i) is planar for each $i = 1, ..., k$?

- *The shortest path in a generally labelled graph:* Is there a path from u to v with the sum of labels of edges smaller than or or equal to K? (For non-negative labels the problem is easily solvable).

- *The longest path:* Is there a path from u to v with the sum of labels of edges larger than or or equal to K?

- *Partially determined path:* Is there a directed simple path from u to v that passes through w in a directed graph with specified distinct vertices u, v and w? [44]

- *Strongly connected subgraph:* Is there a subset of edges E' such that $|E'| \leq K$ and $G'(V, E')$ is strongly connected, i.e., for each pair of vertices u and v, there exists a directed path from u to v and a directed path from v to u?

- *Minimization of the fictitious PERT activities:* Is the number of fictitious activities (M) used in the procedure for representing a project with the PERT digraph less than a prescribed number, i.e., $M \leq K$? [38]

- *Existence of a path in a GERT network:* Is there a network realization in which the activation of the source implies the activation of the sink? [102]

- *One-processor scheduling:* Is it possible to use one processor for a given set of tasks and still comply with the running time, earliest-start time and latest-finish time constraints for each task? [53]

- *Scheduling with precedence:* Is it possible to process given tasks using the given processors and respecting both precedence and the time of finalization for each task? [137]

Kucera [87] uses a vivid metaphor to explain the present state of the art in dealing with *NP-complete* problems: *"In the 1960's it still seemed that the above problems resembled heavy boulders which we would have liked to move and for which there was hope that at least some of them would show a weak spot making such a move possible. By the beginning of 1970s, however, it had turned out that those boulders are interconnected in such a way that we are actually attempting to move a whole mountain. It is thus hardly surprising that no one has yet managed to move it in either direction, i.e., neither do we know a quick and precise algorithm for all of these problems (which would mean finding quick and precise algorithm for any of those problems at once), nor we do know a proof that such algorithms do not exist (which would show that none of the* NP-complete *problems can be coped with)."*

3.3 Solving NP-complete Problems

Today, it is believed that sufficiently fast algorithms for solving NP-complete problems, despite their importance in many fields, are very unlikely to be found. Further, complexity theory points out that some problems would be even more difficult to solve than NP-complete problems. For example, the computation of the optimum strategy for a player in a game belongs to a class called **PSPACE**, which stands for polynomial-bounded memory space, and it is known that $PSPACE \supset NPtime$ for a finite yet an exponential number of states can be expressed with even polynomially-bounded memory space.

These negative conclusions, however, should not mean that we are completely impotent to deal with such problems. By *setting less ambitious goals* than finding an algorithm which would always quickly find the exact answer of the problem at hand, more tractable algorithms can be obtained. Especially in the context of planning studies, where NP-complete problems often appear only as a part of the required optimization, we can be satisfied with an answer that is sufficiently close to the optimal answer. Nevertheless, this method of approximating the answer may not be of much value when the problem at hand is a decision problem rather than an optimization problem.

Even in this case, though, we can rely on *probabilistic algorithms* that might find an approximate or perhaps the exact answer but do not have the guarantees of the deterministic programs. Indeed, there exist some algorithms for the solution of some difficult problems which are not in **P**, although are not shown to be NP-complete either, such that their probability of yielding an erroneous answer can be exponentially reduced. For example, Rabin's algorithm [121] for the determination of the primality of n checks Miller's assertion[2] that holds for any pair (k_i, n), where k_i is a natural number smaller than the prime number n, for each of the m pairs containing the random and mutually independent natural numbers $k_1, k_2, ..., k_m$. Thus it yields the answer with the probability of a mistake (taking a composite number for prime) being at most 2^{-m}. An application of this algorithm was able to find the largest prime smaller than 2^{400} ($2^{400} - 593$) with a probability of error smaller than 0.1%.

Another front concerns the *average performance* of a suggested method by emphasizing that the results of the complexity theory hold only for the worst-case. Some algorithms which have exponential complexity may exhibit a performance having a polynomial complexity in the average case. A well-known example involves the simplex method for the problem of linear programming. It has been shown that linear programming belongs to class **P** and a solution to the problem with a polynomial complexity has been submitted by Karmarkar [130]. Yet, in practice the simplex method has been preferred over the Karmarkar's algorithm for its average-case complexity is less than that of the new method.

Yet another strategy applied in the solution of these problems is to rely on some rules of thumb that

[2]Miller's theorem states that the number n is a prime if, and only if, there does not exist a number k ($1 < k < n$), for which the following assertion is valid : *either* $k^{n-1} \neq 1 \bmod n$ *or* there exists i such that the number $m = \frac{(n-1)}{2^i}$ is an integer, and the greatest common divisor of the numbers k^{m-1} and n is larger than 1 but smaller than n.

produce satisfactory performance on the average. These rules of thumb or *heuristics* can be derived from many different sources and can be employed at any level of complexity. A relatively simple heuristic for TSP, *the nearest insertion rule*, which relies on the extension of a circuit through inclusion of a new node to the nearest node in the circuit, yields answers at worst twice larger than the optimal [24].

A promising approach to computationally hard problems, applicable when these problems involve some constraints, uses the method of propagating the constraints. In this approach the size of the search space is reduced as the search continues by applying the constraints at each step and thus eliminating the part of the search terrain where solutions can not be found. Constraint satisfaction methods have proved to be useful in the solution of many practical problems [139]. Nevertheless, these techniques are not of much value in purely combinatorial problems. Besides, they often involve a tradeoff between computational efficiency and efficiency in finding a good solution, i.e., stronger forms of the constraint satisfaction methods are computationally more expensive.

3.4 Approximating the NP-hard problems

The apparent computational intractability of optimization problems have pressed many researchers to design and use methods that find approximately optimal solutions instead of the optimal one. These studies culminated in various algorithms that can find solutions within a factor of the optimal solution. Unfortunately, most NP-hard optimization problems resisted to attempts to obtain algorithms with parametric error rates. Moreover, no explanations could be given, until recently, why finding good approximation algorithms is difficult (or at least, why significant amounts of research failed to find such algorithms). Recent studies yielded a simple explanation: many NP-hard problems are also NP-hard to approximate [5].

Let A be an approximation algorithm and $L(A)$ denote the value of the solution produces by this algorithm. Also, let L^* denote the value of the optimal solution for the corresponding optimization problem. It is customary to define the *performance ratio* as $\frac{L(A)}{L^*}$ for maximization problems and as $\frac{L^*}{L(A)}$ for minimization problems. There exist algorithms for various NP-hard problems that find solutions with a constant performance ratio. For example, polynomial time algorithms can produce solutions, for TSP with triangle inequality, node cover, maximum cut, and maximum satisfiability, with the performance ratios 3/2, 2, 1/2 and 3/4, respectively. However, it is often desirable to reduce such bounds on the performance ratios. For example, we might want to design an algorithm A such that the error in the nearness of the designated solution to the optimal one ($\epsilon = \frac{L(A)-L^*}{L^*}$) could be reduced arbitrarily by increasing the number of the calls made to this algorithm. If the algorithm runs in polynomial time, the result would be a *polynomial time approximation scheme* (PTAS) which produces a solution with a performance ratio of $1 + \epsilon$. Despite some positive results for special cases, such as the bin packing problem (which has a PTAS [36]), the goal of obtaining PTASs has generally remained elusive. Williamson [147] presents a general $\mathcal{H}(k)$-approximation algorithm (where $\mathcal{H}(n) = 1+\frac{1}{2}+\cdots+\frac{1}{n}$ and k is the highest connectivity requirement), which is applicable to a large class of graph problems.

Garey and Johnson [52] proved that many NP-hard optimization problems do not yield to approximation schemes. For example, they prove that **P**$\neq$ **NP** implies that we cannot obtain an approximation scheme that produces solutions with a bounded error such that $|L(A) - L^*| < K$ for any given K for the knapsack and the maximum independent set problems. Similar results have been obtained for many other NP-complete problems. For example, Arora [6] obtains that for some constant $\delta > 0$, there does not exist any polynomial time algorithm which will approximate the maximum clique within a ratio of n^δ unless **P=NP**. Karger and Motwani [82] show that, for any $\epsilon < 1$, the problem of finding a path of length $n - n^\epsilon$ in an n-vertex Hamiltonian graph is NP-hard. They generalize this result to the longest path problem, and thereby obtain that no polynomial time algorithm can find a constant factor approximation to the longest path problem unless **P=NP**.

Recently, it was proven that the problem of approximation is NP-hard for a large subset of NP-hard problems [5]. In specific, the results of Arora et. al. [5] establish that unless **P=NP**, there do not exist polynomial time approximation schemes for optimization problems which are MAX SNP-hard. The class **SNP** is a strict version of **NP** and was defined by Papadimitriou and Yannakakis [108] based on a syntactic characterization of **NP**. They also provided a notion of approximation-preserving reductions for problems in this class, and under this reduction, identified a large number of approximation problems that are MAX SNP-hard, and are therefore unlikely to have any polynomial time approximation schemes. These problems include such widely studied problems as 3-SAT, vertex cover, metric TSP, Steiner trees, independent set, chromatic number [94], set cover, vertex cover, max-cut, decoding to the nearest codeword, learning in the presence of errors and many others.

Further, Arora has proceeded to give a new definition of the class NP based on probabilistic checking of proofs. At the heart of these new results is a new type of NP-completeness reduction which acts globally (as opposed to the classical reductions, which used local transformations). A more interesting way to view this reduction is as a new probabilistic definition of **NP** : **NP** is exactly the class of languages for which membership proofs (i.e., certificates) can be checked in polynomial time by using only O(log n) random bits and examining only O(1) bits in them [5].

3.5 Graph Search Methods

3.5.1 Plain Methods

The preferred choice of the solution method for NP-hard problems historically has been a systematic search that followed a generally simple rule for converting the inherent nondeterministic step in a solution algorithm to a deterministic one. This plain search method has proved to be useful not only because it is problem independent - in contrast to most heuristic methods - but also it is valued as a benchmark useful in comparing different search methods. Moreover many other methods can be understood as the extensions of the plain search.

Let us call the naive search method *algorithm zero.* Algorithm zero uses a data structure called *FRINGE* which contains a set of pairs (n, P) where n is a node and P is a plan that leads from the given initial node s to the node n. For finding a plan that leads from s to g:

1. [*Initiate*] *FRINGE* $:= ((s, \emptyset))$
2. [*Select*] *Select a pair* (n, P) *from FRINGE*
3. [*Check goal*] *If* $n = g$*, then terminate and return* P
4. [*Extend*] *For each neighbor* j *of* n*, add the pair* $(j, P \oplus j)$ *to FRINGE*
5. [*Continue*] *Go to step 2*

If the step 2 always selects the most recently added pair (in which case *FRINGE* behaves as a stack), the algorithm zero executes a *depth-first* search. If the step 2 always selects the earliest added pair (in which case the *FRINGE* behaves as a queue), the algorithm zero executes a *breadth-first* search [97]. Note that the algorithm zero can be improved by using a marking scheme to recognize the previously visited nodes in order to prevent falling into cycles.

With the marking scheme, both methods systematically search whole solution space. In the worst-case, this would require going through all possibilities, i.e., both algorithms have exponential complexity. For example, the breadth-first search may take order m^d time to find a plan of length d in a graph with m outgoing edges per node. Their actual performance however depends on the structure of the particular search problem in question. Let $N(d)$ be the number of distinct nodes in the graph that can be reached by a plan of length d. The marking version of the breadth-first search takes order $O(N(d))$ time. $N(d)$ grows slowly in the graphs with sparse edges, yet it still follows an exponential law. If the solution is likely to be long, i.e. if d is large, the depth-first search may be a better choice than breadth-first, for it requires less memory resources. In general, depth-first strategy will need to store only $\log_2 n$ states to search a n-node tree, while breadth-first search needs to store $n/2$ states [103].

Both depth-first and breadth-first searches are blind in the sense that the selection they make in step 2 depends totally on the problem specification and is not grounded on the knowledge of the problem or the partial solution. In fact, by making this selection random, we obtain the pure **Monte-Carlo** method which relies on hitting the desired solution by luck. Thus, these searches do not use any knowledge obtained from the search process. These plain strategies constitute a benchmark that other solution methods must be judged against.

We can classify search methods according to the paths they take in the search space, under three different headings; *Monte-Carlo*, *gradient-descent* and *probabilistic* methods. While the Monte-Carlo method obviously follows a random path, the others include a systematic bias in the paths they take.

3.5.2 Gradient-descent Methods

Gradient-descent based methods use the local information obtained in search for choosing the next step to be taken. Newton's method for finding the roots of a function is a nice illustration of this technique. In symbolical processing this technique often appears as hill-climbing strategy; its variation that additionally incorporates the accumulated cost of a partial solution is known as best-first search. *Best-first* search sorts the *FRINGE* data structure appearing in step 2 of the algorithm zero with respect to the accumulated cost and then selects the best one to explore further. In most problems that aim to find a solution having the minimum cost, the consideration of the accumulated cost turns out to be quite useful as it bounds the the search space. As soon as a solution with cost C_0 is found, any partial plan having an accumulated cost C_p such that $C_p > C_0$, can be discarded if the further exploration would only add to C_p (*Branch & Bound*). Further, in a minimization problem with a monotonically increasing cost function, the sorting in the selector step 2 of algorithm zero can be replaced by the instruction "select the pair with minimal cost", that results in a polynomial complexity.

In the shortest-path problem in a graph with edges having non-negative labels, this variant of the best-first strategy (*Dynamic Programming*) produces an algorithm with order $O(mn)$. In general, however, the best-first search can finish processing plans of cost c in order $O(N(c) \log N(c))$. For $N(c)$ usually growing exponentially in c, the best-first search might take an exponential amount of time.

If one uses heuristic knowledge in the selection of the next step for the nondeterministic part, one obtains the famous *A** search. In finding a plan with minimal solution, A* search estimates the distance of a partial plan to the goal by using an underestimating heuristic function $h(n)$. Adding this estimate and the accumulated cost $C(p)$ of a partial plan, one obtains the *directly projected cost* of a partial plan to completion. A* search replaces the sorting measure of the best-first search by the directly projected cost. If the heuristic function is monotone and an underestimating one, i.e. $h(n) \leq h^*(n)$ where $h^*(n)$ is the cost of the minimal cost plan that leads from n to the goal node, A* is guaranteed to find a minimal cost solution. Even with heuristic functions that do not meet the monotonicity condition, A* search can find the solution if the directly projected cost is replaced by the projected cost, which is the maximum over all of the directly projected costs of the prefix plans [97]. Let $N(c)$ denote the number of nodes in a plan with cost no larger than c and $N^*(c)$ denote the number of those with projected cost no larger than c. A* search terminates in time proportional to $(N^*(c) \log N^*(c))$ where c is the cost of the minimal cost solution plan. If the values of h are large relative to accumulated costs, $N^*(c)$ would be smaller than $N(c)$ resulting in some gain in efficiency compared to best-first search.

Many variations of these classical search techniques can be found in the literature. The iterative versions of these have been especially successful in some problems arising in areas where time or memory resources are critical. *Iterative deepening*, for instance, starts with a certain depth limit and tries to find a solution within that limit; if it can not find a solution, it backtracks to the start, extend the depth limit, and resumes search, repeating this process until the time resource is exhausted. *Iterative widening* applies the same idea to the cardinality of the queue in the breadth-first search. These iterative methods have to walk back over the previously explored search spaces following a jump back to the start node. Yet, their complexity is only marginally larger than that of the corresponding plain search with built-in limits since the sum of a geometric series is dominated by its last term.

3.5.3 Probabilistic Methods

Probabilistic methods are those that try to draw the useful aspects of the both the Monte-Carlo method and the gradient-descent method by combining them in harmony. One can count *simulated annealing*, *genetic algorithms* and *tabu search* as the prime examples of this class. (Another often cited example *evolutionary strategy* relies only on mutation and can be considered as a variant of the random search). All these instances differ in the ways and in degrees they combine randomness and biased selection. For example *tabu search* adapts the mutation scheme of simulated annealing but also tries to make use of the information a local search provides through continuously updated memory structures. Another promising technique that has been suggested [105] changes the measure of desirability for a given state. It uses probabilities and a range of values as opposed to single value representation of the classical state search techniques.

Simulated annealing can be considered as both a variant of the Monte-Carlo method and a variant of the greedy search. It starts with a random solution and tentatively modifies it by using a user-specified mutation algorithm. When a mutant improves on the current solution, it replaces the original. So as not to be confined into a local optimum, though, the algorithm sometimes accepts a solution that is worse than the current solution at a given time. When that happens is decided by comparing a random-number with a threshold probability value which is continuously reduced according to an arbitrary annealing schedule (hence the name simulated annealing). In other words, if a proposed local change does not bring an improvement, it will not be taken - *unless* a random number (p) satisfies the condition

$$p < e^{-\delta E/T}$$

where δE is the change in energy that maps into the objective function and T is the current annealing temperature. By moving from high temperatures to low temperatures, a global optimum can be obtained [85]. With simulated annealing, it has been possible to obtain satisfactory answers to combinatorial problems having a cardinality as large as **300!**.

Genetic algorithms are modelled after the evolution process and thus apply genetic operators, most important of them being *crossover*, on a number of solutions in order to obtain the next set of solutions. *Crossover* creates a new solution by swapping the part of a solution with the part of another solution. The fitness-based selection effectively eliminates the low-fitness solutions whereas those with high-fitness solutions are either duplicated or cross-breeeded with the others. Thus, the next generation of the solutions are expected to be better on the average from the previous ones. Genetic algorithms are particularly amenable for implementation in parallel computers that could process and evolve a group of individuals in a single step.

Tabu Search is essentially a meta-heuristics that can be superimposed on other procedures to prevent them from becoming trapped at locally optimal solutions. Tabu search is founded on three primary themes [57] : **1)** the use of a flexible attribute based memory structure designed to permit evaluation criteria and historical search information to be exploited more thoroughly than by rigid memory structures (as in branch-and-bound and A* search) or memoryless systems (as in simulated annealing and pure random search) **2)** an associated mechanism of control - for employing the memory structures - based on the interplay between conditions that constrain and free the search process ; and **3)** the incorporation of memory functions of different time spans, from short term to long term, to implement strategies for intensifying and diversifying search.

The core of tabu search is embedded in its short-term memory process, which evaluates a candidate list of mutations to current solution and then chooses the best among them. In the probabilistic variant, tabu search generates probabilities for selecting mutations and, like the deterministic form, establishes priorities based both on move evaluations and the tabu search memory structures. The selection of the best candidate also checks for the solutions' admissibility. Admissibility is based on the tabu restrictions and aspiration criteria. These checklists are updated at every iteration of and thus serve as long-term memory. Further memory structures are incorporated by establishing a historical standard for differentiating the quality of alternative mutations [57]. This learning capability helps in intensifying and diversifying the search by reinforcing attributes historically found good and driving the search into new regions.

Tabu search has been found to be highly effective in the solution of many problems [57]. Further, it has been successfully integrated to other solution methods such as neural networks and branch-and-bound.

3.5.4 Comparison of Different Methods

Which technique is most suitable for planning problems? We can now reduce this question to the specific questions about the cardinality of the solution set, the complexity of the search space, the possibility of heuristic methods and so on.

Prior experience shows that the problem-customized heuristic methods should be embraced whenever possible, for the problem-independent search methods cannot compete with them in efficiency. In addition, *search strategies making use of heuristic information or at least incorporating it as much as possible should have an advantage over others.* The difficulty lies, however, in finding the necessary heuristics. For as it will be shown later, the problems we consider are inherently nonlinear, i.e., they do not generally accept monotonically increasing cost functions, and thus it is not easy to find and implement heuristics that would be general enough to be useful in different types of problems. However, we will perform experiments that can be used for inferring such heuristics.

Any comparison of the search methods must also take into account the structure of the problems that are being considered. If the problem lies in a very irregular search space such that there is no correlation between any pair of different solutions, the Monte-Carlo technique is perhaps the best we can hope for. The expected time it takes to hit a particular solution in this case is linearly proportional to the size of the solution set. If the search space is regular and smooth, the gradient-descent based methods that use local information would be quite effective. Nevertheless, if the search space contains local optima, these techniques may suffer from falling into local optima unless a way out of this problem has not been specifically incorporated to the method. Furthermore, all of these methods can suffer from the computational complexity of the problem. When the solutions proliferate, as it happens with the NP-complete problems, either memory or time resources might be depleted before a solution can be found.

If the search terrain lies between being smooth and wildly irregular, it is believed that the best search methods one can apply are the probabilistic methods. These methods can make use of local information while being able to explore the larger sections of search space. Davidor [33] notes that GAs are good for moderately complex, nonlinear spaces. He adds "*In fact, it is only worthwhile to apply GAs to spaces that are complex and nonlinear enough so that the use of targeted algorithms is unsatisfactory*".

Even though GAs and other stochastic methods cannot guarantee to find the best solution, the solutions they find, for most cases, can be quite close to the best solution. Many engineering design and optimization problems are modelled using approximations and fuzzy data. Hence, from an engineering point of view, little meaning can be attached to *best.* Hopfield [75] notes that

"Often, what is truly desired is a very good solution, which will be uniquely best only for simple tasks. In many situations, a very good answer computed in a time scale short enough so that the solution can be used in the choice of appropriate action is more important than a nominally-better best *solution."*

From this point of view, the classic graph search methods, such as A*, or best-first search, which guarantee to find the best solution, may be more powerful and time consuming than necessary. The class of problems that can be solved with these particular search methods corresponds to the complexity class PSPACE. Because they are contained in PSPACE, NP-complete problems can be solved with these methods as well as some other problems that are even harder. Yet, it appears that the wide-applicability of classic search methods results in a relative inefficiency in solving such problems as these methods do not exploit particular characteristics of NP-complete problems, in contrast to constraint satisfaction techniques. Further, these techniques construct a solution piece by piece and thus are unsuited for applying heuristics and cost functions that require a complete solution to evaluate.

Because of these disadvantages associated with A* type search techniques and limited benefits of exclusively relying on heuristics for the problem of most interest here, we have chosen genetic algorithms for further analysis.

3.6 Genetic Algorithms and Evolution

In nature, the struggle for existence among living beings results in, over the generations, a gradual accumulation of the features that increase the chances for successful next generations and a gradual elimination of those that reduce it. Making this observation, Darwin concluded that the species evolve to adapt to their environment under the pressure of natural selection. Today, the biological structures that determine the features of the living beings are called *genes* and evolution is viewed as the way genes live and die [34], rather than the way species change. The genes, while competing with different genes for the same natural resources, direct the bodies in which they are embodied for achieving their goal of reproducing. In other words, bodies are temporary vessels for those genes they carry within.

The process of evolution can be viewed as a procedure for finding better solutions to some externally imposed problem of fitness. What this fitness measures in natural evolution has been a topic of discussion, yet the most convincing explanation emphasizes success in reproduction (number of offsprings, each weighted by the probability that it will reproduce) [140]. When a group of individuals and their descendants have been subject to a continuous - stationary or changing - pressure forcing adaptation, it is observed that the population will generally evolve toward "greater fitness" determined by a particular set of genotypes. These genotypes are distinct from others that have not survived merely by the property of having better coped with the selection pressure. In this sense, fitness of an individual is a measure of the survival probability for her genes in a given environment.

The idea of modelling this phenomenon for generating better solutions to optimization problems has led to the birth of *Genetic Algorithms* (GAs). Today, GAs refer to a family of computational models inspired by evolution which are applied in a broad range of problems, such as program induction and cellular automata as well as function optimization. The first application of the GA appeared as early as in 1962 in a structural design study [17]. Later, various versions of simulations of evolution were evolved. As of today, though, most of these simulations still embody only a very simplified model of biological evolution.

In its most general form, a GA can be classified as a stochastic, inductive learning mechanism. A

```
1 - Generate a set of solutions for the seed generation
2 - Evaluate the fitness of each individual in the current generation
3 - Apply genetic operators to each individual
        In specific, decide on the basis of fitness and a probabilistic test, to apply either:
            a- Duplication
            b- Crossover
            c- Mutation
4 - Replace the population with new individuals
5 - Go to step 2
```

Figure 3-1: *A Canonical Genetic Algorithm*

GA can also be compared to a special type of iterative Monte-Carlo algorithm which tries to improve its solutions by preserving the better solutions while replacing the worse ones. Whereas the Monte-Carlo approach generates new solutions randomly, a GA recombines the previously obtained solutions in order to construct new solutions.

GAs can be best characterized as recursive search procedures modelled after natural evolution; they start with some initial solutions and proceed by recycling and refining them. Figure 3.6 presents a canonical representation of a Genetic Algorithm. The recycling process uses a biased selection and recombination of the solutions in which preferences are given to those that show more promise for satisfying the predetermined objectives of the simulation. This performance-oriented selection process is not deterministic, but it uses a probabilistic mechanism that makes decisions by comparing a random number to a scaled fitness value that measures the desirability of a solution.

Only the most obvious aspects of biological evolution have been modelled in GAs. The recombination of old solutions to produce new ones can be considered as a cursory simulation of sexual reproduction. More importantly, GA is considered to be simulating natural selection, for it involves a mechanism in which the fitness of an individual strongly influences its fate in the recycling process.

In GAs, the building blocks of a solution take the place of the genes and solutions become the entities to which the selection pressure is applied via genetic operators. The most important of these genetic operators is the simulation of sexual reproduction, which is often called *crossover*. A crossover operation usually produces two new individuals from two old ones by exchanging randomly selected subsets of their genes. Although it is employed in nature mostly by primitive life-forms, asexual reproduction is sometimes modelled in GAs through *duplication* operation. It has been found useful in most optimization problems for it transfers the better solutions to next generations without any degradation. Another basic genetic operator, *mutation*, is secondary in importance to both crossover and duplication, for it is effective in much longer time scales both in natural and artificial evolution. Practical implementations of GAs can also contain operations without equivalents in natural evolution.

The natural selection in the GA is fitness-driven. In contrast to ambiguity on what fitness measures in nature, the goal-oriented optimization studies employing the GA can apply a plain fitness measure: the value of the objective function that maps a solution to a real number. Because the fitness of a solution is used only to make comparisons, most implementations use a fitness function that is slightly different than the objective function. In such cases, the fitness function may rescale and shift the objective function or may even represent it in a coarse manner. Evaluation of the fitness function is often the most computationally expensive part of the GA. Sometimes, a limited form of fitness evaluation can be used through sampling or simulation or even a partial evaluation of the solution in the given environment. This use of GAs is especially common in artificial-life research.

The similarities between GAs and evolution provide some justification for the success of the algorithm in the optimization and adaptation studies. Yet, one needs a theoretical explanation of the algorithm's behavior for a better understanding. In the next section, we review the first developed theory of GAs, called the *schema theorem.*

3.7 Schema Theorem

When a particular individual in nature leaves many descendants and thus perhaps decreases the probability of extinction of its genes, we usually cannot pinpoint which features or which combination of features were responsible for this achievement. Similarly, when one solution to a fitness problem appears to be better than others in a trial method, it is often hard to know which specific features were responsible for its better fitness.

Yet, keeping track of the historical record of this individual and its descendants could yield many clues on the question of which features count in success. Both in natural and simulated evolution this question is posed implicitly due to the selection pressure and the answer is produced again implicitly, thanks to repeated applications of selection, reproduction and recombination over a set of individuals.

An explanation of how the process of evolution can achieve this feat is first given in Holland's book *Adaptation in Natural and Artificial Systems* [74]. Holland proved that GAs produce near-optimal strategies for multi-armed bandit problems [3] through a credit allocation mechanism. In a two-armed bandit problem, a rule suggesting that *"stay on the winner, switch on the loser"* yields better results, on average, than the 50-50 per cent random selection rule. However, more qualified information can be extracted from a record of the results of different strategies.

[3]A classic example of a two-armed bandit example is as follows:
Suppose two treatments are available for a certain disease. Patients arrive at a clinic one at a time and one of the treatments must be used on each. Information as to the effectiveness of the treatments accrues as they are used; no prior information exists. The overall objective is to treat as many patients as effectively possible. This problem of allocating treatments to patients over time is surprisingly difficult, even when the responses are dichotomous, either success or failure. [14]

For an effective solution algorithm involving sampling from a search space, one needs to find a delicate balance between further exploration of the search space (which can be considered as an oversimplified version of *"switch on the loser"* rule) and making best use of the current data (*"stay on the winner"*). In this search, one is always faced with a trade-off contrasting further exploration that could potentially bring gains against the cost of the search. More importantly, one must decide on how to proceed in the sampling process by choosing the areas to be further explored. Confining new search regions to the immediate vicinity of those that proved valuable in former trials can quickly lead to an overexploitation of statistical data. GAs are successful because they can exploit accumulating information about an initially unknown search space and bias subsequent search in order to determine the new search subspaces where the above-average payoffs are likely, all based on the database of tentative solutions. This learning capability enables the GA to search without being either too greedy or too conservative.

Genetic algorithms induce the features of the target answer from a set of trials by giving almost optimally each possible answer an opportunity to contribute to the next set of trials. The allocation proceeds by making sure that each explanation on what constitutes the best answer is accounted for approximately in balance with its average fitness. It gives credit to solutions but not explicitly to genes, because most of the time, due to nonlinearities, a particular combination of individual genes would be responsible for high-fitness.

The learning capability of GAs is often analyzed in terms of induction of concepts, called **schemata** in the parlance of GAs, that successfully explain why some solutions have higher fitness in a set of trials. A concept is a hypothesis or a rule that can differentiate between a set of instances in regard to certain features. Given an alphabet with K different symbols and a maximum object size of l symbols, we can list K^l instances of objects and $(K+1)^l$ different concepts for these instances. The concept space is larger than the object space because a concept can specify more than a single object simply by using an *or* relationship for certain features. Concepts are often expressed with the help of an extra symbol *, meaning *"don't care"*.

A GA converges to target concepts gradually by decreasing the probability of emergence of those with low-fitness and increasing the corresponding probability for those with high-fitness. Consider a binary alphabet with word length 3. In this alphabet, $(2+1)^3 = 27$ concepts exist. Tentative solutions such as 010, 011, 001, or 110 are members of many schemata. For example, the word 010 belongs to all of these schemata: 010, 01*, 0*0, *10, 0**, *1*, **0, ***. Therefore, even if 010 emerges as a very high-fitness solution, it should not necessarily qualify as the desired answer. On the other hand, the schemata *10, *1*, *** also contain the word 110. If 110 were a low fitness example, while 010 had high fitness, we should eliminate those schemata as the possible explanations for fitness. It is thus possible to discharge the impossible explanations and come up with plausible ones through a learning algorithm, although to narrow down the possible explanations to a single one usually requires an extremely large amount of samples.

The GA is not unique in its induction capability; there is a large family of algorithms that enable inductions. GAs, however, are not limited to a passive learning. Thanks to a fitness-driven sampling, they also actively seek those samples that can eliminate alternative explanations. Therefore, GAs can account for the conflicts appearing between different schemata which can result from the inherent nonlinearities in a problem, from errors introduced by statistical sampling, from the noise in the environment, or even from changes in the environment [86] that can be expressed with a non-stationary fitness function. GAs are particularly useful when applied to those problems that contain nonlinear relationships between various decision parameters. Such relationships are raised by often subtle linkage between various genes, such as the suppression of the effect of a gene by a totally different gene and they are called *epistasis* in the field of genetics.

GAs achieve all of this by giving weight to each schema in proportion to its average fitness. Although GAs do not calculate the average fitness of schemata nor even perform any explicit operation on those, the fitness-proportionate reproduction makes sure that the number of individuals belonging to a particular schema changes on average proportionally to the ratio of average fitness of the schema to that of the average fitness.

This fact is contained in the so-called *Schema Theorem*, due to Holland [74]. For the purpose of stating the theorem, let $f(H,t)$ be the *average fitness of a schema H*. That is,

$$f(H,t) = \frac{\sum_{x_i \in H} f(x_i,t)}{m(H,t)} \tag{3.2}$$

where $m(H,t)$ is the number of occurrences of schema H in generation t. This average fitness has an associated variance that depends on the number of items being summed to compute the answer.

Note that because GAs sample from a very small portion of the search space, all the averages they use are estimates and thus involve statistical noise. Conversion of the deterministic credit allocation mechanism to a stochastic one helps to minimize the errors due to statistical noise.

The schema theorem states that, for a genetic algorithm using Darwinian operation of fitness-proportionate reproduction and the genetic operations of crossover and mutation, the expected number of occurrences of every schema H in the next generation is approximately

$$m(H,t+1) \geq m(H,t)\frac{f(H,t)}{\overline{f(t)}}(1-\epsilon_c)(1-\epsilon_m) \tag{3.3}$$

where $\overline{f(t)}$ is the average fitness of the population at generation t, ϵ_c is the probability of disruption of the schema H due to crossover and ϵ_m is the probability of disruption of the schema H due to mutation.

The schema theorem implies, to the extent that ϵ_c and ϵ_m is small, that the change rate in the occurrence of a particular schema is proportional to its fitness ratio $\frac{f(H,t)}{\bar{f}(t)}$. If a schema has fitness ratio higher that 1, its occurrence should grow exponentially.

The allocation of trials is most nearly optimal when both ϵ_c and ϵ_m are small. Further modelling of these operators indicates that this would be the case for ϵ_c when the defining length (the distance between the outermost specific, non-* symbols) is short, and for ϵ_m when the number of non-* symbols is small.

In summary, the GA superficially seems to be concerned with the individuals to whom it applies the genetic operators but it actually process a large amount of useful information contained in unseen schemata (which is often referred to as *intrinsic* or *implicit parallelism*). Further, this computation is accomplished without any explicit memory beyond the population itself and without any explicit computation beyond the simple genetic operations acting on the individuals of the population [86]. The only memory involved in the genetic algorithm is the state of the system itself. Schaffer notes that [125] the possibility of processing a large number of explanations with only a few individuals represents the only known example of combinatorial explosion working to advantage rather than to disadvantage!

3.8 Schema Theorem Extended

The schema theorem implies that, in the generation of a new group of individuals, each schema would appear proportionally to its fitness. From this assumption, Goldberg [60] obtains an equation that gives the proportion $P_k(t)$ of a structure k in terms of its former proportion. *The proportion $P_k(t)$ of a special schema k would be approximately equal to its former proportion in the preceding generation weighted by the fitness values* as the GA tries to allocate new slots in the next generation proportional to the current fitness values. Following Goldberg's line of analysis [60] and modifying it for multi-structure cases, we obtain for a population size of n :

$$P_k(t) \approx \frac{f_k(t-1)P_k(t-1)}{\sum_{j=1}^{n} f_j(t-1)P_j(t-1)} \tag{3.4}$$

where f' is defined as a weighted average over the fitnesses of all individuals excluding the desired solution :

$$f' = \frac{\sum_{j=1,j\neq k}^{n} f_j P_j}{\sum_{j=1,j\neq k}^{n} P_j} \tag{3.5}$$

Because the sum of proportions must be equal to 1, i.e. :

$$\sum_{j=1,j\neq k}^{n} P_j + P_k = 1$$

we can make the following manipulation to the numerator of Equation 3.4 using Equation 3.5

$$\begin{aligned}\sum_{j=1}^{n} f_j P_j &= f_k P_k + \sum_{j=1,j\neq k}^{n} f_j P_j \\ &= f_k P_k + f'(1 - P_k) \end{aligned} \tag{3.6}$$

Defining r_k as the ratio of the fitness of the solution k to the others' average fitness

$$r_k = \frac{f_k}{f'} \tag{3.7}$$

we finally obtain :

$$\begin{aligned} P_k(t) &\approx \frac{f_k P_k(t-1)}{f_k P_k(t-1) + f'(1 - P_k(t-1))} \\ &\approx \frac{r_k P_k(t-1)}{r_k P_k(t-1) + (1 - P_k(t-1))} \end{aligned} \tag{3.8}$$

$P_k(t)$ can also be expressed in terms of the initial values :

$$P_k(t) \approx \frac{P_k(0)\prod_{l=1}^{t} r_{kl}}{(1 - P_k(0)) + P_k(0)\prod_{l=1}^{t} r_{kl}} \tag{3.9}$$

This formula indicates that the proportions of the high-fitness schemata with $r > 1$ would increase exponentially while the proportions of the low-fitness schemata would decay exponentially. Note that GA will always find at least one schema in each generation with a fitness ratio r_k greater than 1.

As different schemata are cross-bred with each other, a pure genetic algorithm without any side effects should converge eventually to a single schema. Competing schemata can cause the genetic algorithm to deviate from converging to an optimal structure, but, on the average, we can expect that a single good schema would cover an exponentially increasing fraction of the newly produced individuals. Thus, through the feedback of the fitness values as a selection pressure and cross-breeding, GA eventually can find the desired solutions, even though the initial probabilities of high-fitness solutions could be extremely miniscule.

3.9 Discussion of Theoretical Models of Genetic Algorithms

While the schema theorem can explain how GAs can converge to good solutions, it does not tell much about the specific concerns such as the complexity of the algorithm, the convergence rate, the best ways of implementing genetic operators, the best choice of parameters and so on. There are many sophisticated models of GAs that try to answer such questions by going into details in the modelling stage and in the assumptions they make. Unfortunately, most of these theories are often not realistic enough and their results may not be consolidated well with empirical data reported in literature. For example, Markov models of GAs have been gaining popularity recently [144, 141], yet they are found to be inadequate in resolving the issues regarding the optimal parameters for a GA. As a result, many problems that arise in the use of GAs, such as "what is the best parameter for the population size?", are still debated in the literature.

Furthermore, a substantial part of the later studies on theoretical explanation of GAs have been based on a special implementation. In this classic implementation, a gene consists of a binary number and a solution is an ordered list of such genes. Crossover selects two random points on each gene and exchanges the bits between them. Mutation consists of the inversion of a random bit. The goal is generally the optimization of a function that can be usefully expressed with such a representation.

While the classic implementation has been the preferred method of employing GAs, there is no requirement of using it; the algorithm in Figure 3.6 does not specify any details. The classic implementation is often criticized for being too tight in order to freely apply general optimization problems. For example, Goldberg claims that [59] "messy" GAs that use changing word lengths, in contrast to the "tidy" implementations, perform better in most problems. Koza diverges significantly from the classic paradigm [86] by replacing the binary words with programs in a user-defined functional language. Besides, it is often observed that using specialized versions of crossover or mutation often yields more efficient solution algorithms.

Diverging from the classical implementation has brought out two problems that are still unresolved. The first concerns what the best representation would be for a given problem. The second problem asks what the best implementation of the genetic operators should be.

Whether binary encoding is the most effective representation for GAs is currently a discussion topic in the literature [64, 61, 35]. While DeJong emphasizes that [35] there is nothing sacred about about traditional string-oriented genetic operators, others believe that [122] binary encoding can process schemata more effectively than other representations due to its compactness. As DeJong [35] points out, GAs work work best when the internal representation encourages the emergence of useful blocks that can subsequently be combined with each other to improve performance.

Along with the representation problem, problems in regard to implementation of genetic operators surface when one diverges from the classical implementation. Further, the classical method of implementation does not produce valid offspring in many constrained problems, prompting some researchers to implement new operators that overcome this difficulty and some others to use penalties in the selection process for constraint violations. The implementation of different crossover operators for particular problems is a favored study topic in GA community.

Also, questions concerning the mutation operator have been posed but not convincingly and completely answered yet. What is the best implementation for performing mutation? What should be the optimum frequency of mutation? Can GAs perform adequately enough without mutation? The debate continues on whether the most important feature of GAs is natural selection and hence mutation should be sufficient in simulating evolution or whether the important feature is the effective schemata sampling achieved naturally by crossover and therefore good performance can be obtained only when crossover is used. See [145] for a summary of this discussion or [100] for a defense of the first view that basically assess the GA as a hillclimber.

Another closely-related discussion topic considers whether GAs optimize often prematurely due to strong fitness-proportionate selection and whether this could be avoided. The effect of premature optimization could be minimized by incorporating mutation and performing multiple independent runs at the expense of increased computation.

Another question concerns the method of selecting and matching individual solutions for cross-breeding. Different algorithms for performing this task, such as tournament-based selection, rank-based selection or elitist selection, are suggested as being superior to others in the literature.

Also, some other questions arise on the mapping between the normalized fitness and the objective function. For example, shifting the fitness function by adding a constant to it would change the ratios of the fitnesses for different schemata. Even though the objective function is only superficially changed, the allocations of trials to schemata change, making the interpretation of the schema theorem problematical [66].

Despite the fact that complexity of the GA has important practical implications, there are only a limited number of results from this perspective. DeJong has examined GAs capability for solving NP-complete problems and found that some NP-complete problems were handled better than some others, Boolean satisfiability being a good performance example. DeJong also finds that the relationship between the logarithm of the number of evaluations required and the logarithm of the search space is sub-linear for such problems.

Finally, a very important question on simulating evolution asks the best ways of improving the simple model of simulation. One might suspect that more realistic modelling of biological evolution can help improve the performance of classic GAs. A study by Hillis [73], for example, demonstrates that the addition of co-evolving parasites is helpful in preventing the system from sticking at local maxima. Different niches interact with each other in nature. A simulation of this phenomena could be accomplished by running multi GAs in parallel with different niches linked via periodical emigration and immigration. Goldberg states the idea, based also on distributed computation, with a lively metaphor [58]:

> The communities consist of a set of homes connected to the centralized, interconnected towns. Parents give birth to offsprings in their homes and performs ... evaluations there. The children are sent to centralized singles bars in town, where they meet up with prospective mates. After mating, the couples go to the town's real estate broker to find a home. Homes are auctioned off to competing couples. If the town is currently crowded, the couples may also consult the broker about homes in other communities...

Discussions abound on these questions regarding the details of GAs. Further, in implementations of genetic algorithms, different routes are taken frequently. For example, while some use meta-GAs to obtain the optimum parameters, some others claim that their effect is not really significant anyway and go ahead with arbitrarily selected parameters.

For all these reasons, as well as for the reason that the software we developed is substantially different than the classical implementation, we do not attempt to replicate the results found for the classical implementation. Rather we develop a simple probabilistic model comparing GAs and the Monte-Carlo approach that will be useful for answering those questions on the convergence rate, the complexity of a generic GA and the optimum population size.

3.10 A Probabilistic Model Comparing GAs and Random Search

Given the unresolved issues surrounding GAs, it should not be surprising that the following question is posed: *"Do genetic algorithms really perform well?"*. This has been a topic of discussion in optimization and computation community due to the method's wide popularity. Upon listening to the sides of this discussion holding extreme views, one searcher for truth might believe that GAs are a sort of elixir while another might think that whole thing is nothing more than hype. The question is especially justified when we consider that a similar phenomenon has been experienced on neural networks: Artificial neural networks too, have been modelled after nature, have not gained popularity for a long time their emergence, and have been praised for their success in solving many types of computationally difficult problems (as GA's). Yet, there have been many applications of neural networks in areas in which they do not perform as well as some of the well-known techniques, an obvious example being curve fitting through regression. Finally, one might suspect that GAs as well as simulated annealing have been named after their counterpart processes occurring in nature perhaps in an effort to rationalize their performance. Certainly, there is a lack of rigorous theoretical models to explain the reasons for these methods' success. In the following, we give a simple probabilistic model of the gentic algorithm that allows us to compare it with the Monte-Carlo approach and resolve some issues surrounding the GAs.

3.10.1 Definitions

The following definitions will be useful in analyzing evolution of the probabilities of finding the desired solutions, in both the GAs and pure random search (called Monte-Carlo from now on):

Let $\alpha_i(t)$ stand for the probability that an individual i of generation t meets the success criterion. The success criterion is easy to determine once the optimal solution is known; it can be checked whether a solution within close proximity of the optimal solution is obtained. Suppose that the distance of a given solution to the optimal solution can be quantitatively measured as the ratio of the corresponding values of the objective function. Then success can be measured in terms of obtaining a solution with a convergence ratio less than a prescribed value. In other words,

$$\alpha_i(t) = \Pr(y_i^{(t)} \leq y^*) \tag{3.10}$$

where y^* is the required convergence ratio and $y_i^{(t)}$ is the convergence ratio attained by the individual i of the generation t.

Let $\beta(t)$ be the probability of finding at least one individual among a generation t of n individuals meeting the success criterion :

$$\beta(t) = \Pr\left((y_1^{(t)} \leq y^*) \vee (y_2^{(t)} \leq y^*) \vee \ldots \vee (y_n^{(t)} \leq y^*)\right) \tag{3.11}$$

Applying the laws of probability and invoking the assumption of independence between individual solutions, we obtain

$$\begin{aligned} \beta(t) &= 1 - \left[\Pr(y_1^{(t)} > y^*)\Pr(y_2^{(t)} > y^*)\cdots\Pr(y_n^{(t)} > y^*)\right] \\ &= 1 - \left[(1-\Pr(y_1^{(t)} \leq y^*))(1-\Pr(y_2^{(t)} \leq y^*))\cdots(1-\Pr(y_n^{(t)} \leq y^*))\right] \\ &= 1 - \prod_{i=1}^{n}(1-\alpha_i(t)) \end{aligned} \tag{3.12}$$

Now, let Γ_M define the probability of at least an individual meeting the success criterion **after** $M-1$ generations, at Mth generation.

$$\begin{aligned} \Gamma_M &= (1-\beta(1))(1-\beta(2))\ldots(1-\beta(M-1))\beta(M) \\ &= \left(\prod_{t=1}^{M-1}\prod_{i=1}^{n}(1-\alpha_i(t))\right)\left(1-\prod_{i=1}^{n}(1-\alpha_i(M))\right) \end{aligned} \tag{3.13}$$

We also define a *cumulative probability* Ω_M as the probability of at least an individual meeting the success criterion **within** M generations. Then,

$$\begin{aligned}\Omega_M &= \beta(1) \vee \beta(2) \vee \cdots \vee \beta(M-1) \vee \beta(M) \\ &= 1 - [(1-\beta(1))(1-\beta(2)) \cdots (1-\beta(M))] \\ &= 1 - \prod_{t=1}^{M} \prod_{i=1}^{n} (1-\alpha_i(t))\end{aligned} \tag{3.14}$$

It follows from Equation 3.13 and Equation 3.14 :

$$\Gamma_M = \Omega_M - \Omega_{M-1} \tag{3.15}$$

Finally we define a complementary probability to Ω as

$$\Theta = 1 - \Omega \tag{3.16}$$

Note that these equations yield only the probabilities corresponding to the criterion of finding *at least one* acceptable solution. More information in regard to the different criteria can be obtained by using the corresponding probability distribution which a set of trials obey.

Now, we can make either of the following assumptions leading to an expression for the cumulative probabilites for either pure random (Monte-Carlo) search or the GA.

- The trials in each generation are independent.
- The trials in each generation depend on the preceding generations.

Below we examine each case separately.

3.10.2 The Random Search Model

A simple method for finding a solution would be to generate an independent set of solutions and to repeat this process when these trials do not satisfy the desired criteria. The independence assumption leads to a constant probability α of finding an acceptable solution throughout a set of trials.

$$\alpha_i(t) = \alpha_0 \tag{3.17}$$

Given α_0 a constant, we obtain :

$$\beta(t) = \beta_0 = 1 - (1 - \alpha_0)^n \tag{3.18}$$

$$\Gamma(t) = (1 - \alpha_0)^{n(t-1)} \left(1 - (1 - \alpha_0)^n\right) \tag{3.19}$$

And, for the cumulative probabilities, we obtain :

$$\Omega(t) = 1 - (1 - \alpha_0)^{nt}, \qquad t = 1, 2, 3, \ldots \tag{3.20}$$

$$\Theta(t) = (1 - \alpha_0)^{nt} \approx e^{-\alpha_0 nt} \tag{3.21}$$

This last equation 3.21 indicates that the nonconvergence probability Θ decays exponentially with the number of independent experiments performed (nt) wherein α_0 corresponds to the decay constant. Despite this exponential behavior, the Monte-Carlo method may not be valuable when α_0 is an extremely small number. For example, if there is a set of permutations for possible solutions but there is only one acceptable solution $\alpha_0 = \frac{1}{N!}$ and hence the decay rate for the nonconvergence probability may never be fast enough.

3.10.3 The GA Model

For the GA, neither we can invoke the independence assumption nor we can determine the evolution of the individual probabilities $\alpha_i(t)$ from Equation 3.9 or from the schema theorem. Despite such shortfalls, the following argument makes it possible to develop an expression for the nonconvergence probability Θ for GA. Based on the extension of the schema theorem, we can expect that the probability β of finding at least one individual among the members of a population meeting the particular success criterion should grow in accordance with the growing proportion of high-fitness schema.

Thus, using Equation 3.9 that expresses the proportion of good schema in terms of the fitness ratio r of the best schema to the average fitness of the other schema :

$$\beta(t) \approx \frac{r\beta(t-1)}{(1 - \beta(t-1)) + r\beta(t-1)} \tag{3.22}$$

It is plausible to express the evolution of the probability $\beta(t)$ in terms of the initial values :

$$\beta(t) \approx \frac{\beta_0 \prod_{i=1}^{t} r_i}{(1-\beta_0) + \beta_0 \prod_{i=1}^{t} r_i} \tag{3.23}$$

The complementary probability to β would be :

$$1 - \beta(t) \approx \frac{1-\beta_0}{(1-\beta_0) + \beta_0 \prod_{i=1}^{t} r_i} \tag{3.24}$$

The above equation can be substituted into Equation 3.14 to obtain an expression for the nonconvergence probability $\Theta = 1 - \Omega(t)$:

$$\begin{aligned} \Theta(t) &\approx \prod_{i}^{t}(1-\beta_i) \\ &\approx (1-\beta_0)^t \frac{1}{(1-\beta_0+\beta_0\prod_{j=1}^{1} r_j)(1-\beta_0+\beta_0\prod_{j=1}^{2} r_j)\cdots(1-\beta_0+\beta_0\prod_{j=1}^{t-1} r_j)} \end{aligned} \tag{3.25}$$

We can further simplify this equation by noting that the first term on the right hand side expresses the corresponding probability Θ_{MC} of the Monte-Carlo approach obtained with the independence assumption. Also by defining a function F as

$$F(\beta_0, r, t) = \frac{1}{(1-\beta_0+\beta_0\prod_{j=1}^{1} r_j)(1-\beta_0+\beta_0\prod_{j=1}^{2} r_j)\cdots(1-\beta_0+\beta_0\prod_{j=1}^{t-1} r_j)} \tag{3.26}$$

and by defining function G such that

$$F(\alpha_0, n, r, t) = \exp\left(-G(\alpha_0, r, t, n)nt\right) \tag{3.27}$$

the nonconvergence probability Θ for the genetic algorithm is obtained as :

$$\Theta_{GA}(t) = \Theta_{MC}(t)e^{-G(\alpha_0,n,r,t)nt} \tag{3.28}$$

This equation allows us to compare the performance of the genetic algorithm to that of the Monte-Carlo approach.

The function $F(t)$, which is strongly determined by the factors r, depends also on the initial success probability, which, in turn, is determined by the population size, the required convergence ratio and the problem size. F decreases sharply after a certain threshold value of the generation number.

When the fitness function maps directly from the objective function, $r(t)$ should not be taken as a constant because it should decrease as the solutions converge to good schemata. Even with exponentially decreasing values of r, the cumulative success probability of a GA still increases much faster than corresponding probability for the Monte-Carlo approach.

These results indicate that the probability of not finding an answer in GA is an exponentially decreasing fraction of that of the Monte-Carlo approach.

3.10.4 Comparison of the Convergence Time

The above formulas, which give us the probabilities of finding an answer in terms of the initial probabilities of the Monte-Carlo approach can also be useful in giving information about the convergence time of the algorithm.

A Single Run

The expected convergence time for a single run as a function of the failure probability δ, can be obtained from the following equality :

$$T(\delta) = H(n)t(\delta) \tag{3.29}$$

where $H(n)$ is the time required to process a single generation of n individuals. $t(\delta)$ denotes the expected number of generations for a given failure probability such that $\Theta(t) = \delta$ and δ is generally a small number.

From Equation 3.28, one can find the expected number of generations to obtain a solution with a confidence probability $1 - \delta$

$$t(\delta) = \Theta^{-1}(\delta) \tag{3.30}$$

It follows that for the Monte-Carlo approach:

$$t(\delta)_{MC} \approx \frac{1}{n\alpha_0} \log \frac{1}{\delta} \tag{3.31}$$

For the GA:

$$t(\delta)_{GA} = \Theta_{GA}^{-1}(\delta) \tag{3.32}$$

Assuming that the dependence of the function $G(\alpha_0, r, t, n)$ defined in Equation 3.27 to variable t is weak:

$$t(\delta)_{GA} \approx \frac{1}{n(\alpha_0 + G)} \log \frac{1}{\delta} \tag{3.33}$$

Both Goldberg [60] and Ankenbrandt [4] find similar results by making sole use of Equation 3.4. Manipulating Equation 3.9 yields

$$t = \frac{1}{\log r} \log\left(\frac{1-P_0}{P_0}\frac{P_t}{1-P_t}\right) \tag{3.34}$$

Assuming that the solution exists initially at a proportion $P_0 = \gamma$ and that one lets the run proceed until a final proportion $P_t = (1-\gamma)$, the following result is obtained:

$$t = \frac{2}{\log r} \log \frac{1-\gamma}{\gamma} \tag{3.35}$$

If the proportion γ is assumed to be a constant independent of the population size, this equation tells us that the number of generations to convergence is of order $O(1)$ [60]. If the population is permitted to converge to bitter end, we can assume that that we start with one better individual and end with all but one of the slots in the population filled by the better individuals. In this case, $\gamma = 1/n$ and

$$t = \frac{2}{\log r} \log n - 1 \tag{3.36}$$

In other words, these equations also tell that for a particular convergence ratio, the number of generations to convergence would be a logarithmic function of the population size n.

Multiple Runs

A more informative picture is obtained from the following analysis that enable us to compare the GA approach with the Monte-Carlo approach by looking at the number of required runs in order to have a certain confidence in obtaining the solution.

The benefits of multiple runs arise from minimizing the effects of various random events and initial conditions and especially of premature optimization. Koza notes that GAs are prone to premature optimization [86], which also manifests itself in nature as the so-called *niche preemption* principle. According to this principle, a biological niche in nature tends to be dominated by a single species. Nature carries out its genetic experiments in parallel in numerous niches. The species that ultimately dominates any given niche may be decided by unique initial conditions of that niche and the subsequent unique history of probabilistic events in that niche [86]. Hence, performing multiple independent runs bring advantages over a single run with a larger population size or a longer generation time for better optimization.

Now, let us assume that we fix the probability of succeeding at least once in R runs in M generations as z. Because the runs are independent,

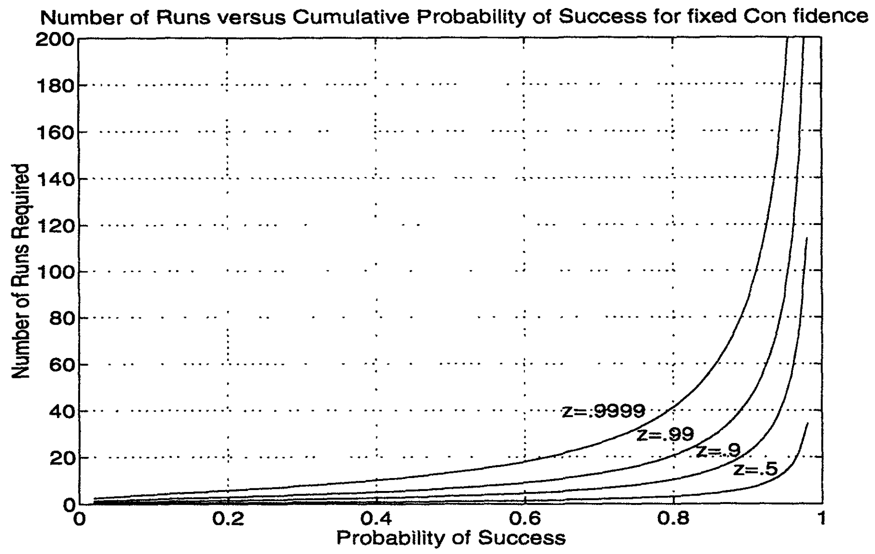


Figure 3-2: *Number of Independent Runs versus Cumulative Probability of Success for Different Confidence Probabilities*

$$z = 1 - (1 - \Omega)^R \tag{3.37}$$

What is the required number of runs in order to have a certain confidence value? From the above equation, we obtain that

$$R = \frac{\log(1-z)}{\log(1-\Omega)} = \frac{\log(1-z)}{\log\Theta} \tag{3.38}$$

The following figure, Figure 3-2, shows the behavior of the number of the independent runs $R(z)$ required to yield a success with probability $z = 0.9999$, $z = 0.99$, $z = 0.9$, and $z = 0.5$ versus the cumulative probability of success Ω. One can compare the required amount of computation for a certain confidence ratio by using previously developed expressions for the convergence probabilities.

When the independence assumption holds:

$$\begin{aligned} R_{MC} &= \frac{\log(1-z)}{\log\Theta_{MC}} \\ &= \frac{\log\delta}{nt\log(1-\alpha_0)} \end{aligned} \tag{3.39}$$

The required computation time is therefore

$$T_{MC}(z) = ntR_{MC} = \frac{\log(1-z)}{\log(1-\alpha_0)} \tag{3.40}$$

For the GA model

$$\begin{aligned} R_{GA} &= \frac{\log(1-z)}{\log \Theta_{GA}} \\ &= \frac{\log(1-z)}{\log \Theta_{MC} + \log F(\alpha_0, n, r, t)} \\ &\approx \frac{\log(1-z)}{nt \log(1-\alpha_0) - ntG(\alpha_0, r, t, n)} \end{aligned} \tag{3.41}$$

It follows that

$$\begin{aligned} T_{GA}(z) &= ntR_{GA} \\ &\approx \frac{\log(1-z)}{\log(1-\alpha_0) - G(\alpha_0, r, t, n)} \\ &\approx \frac{T_{MC}}{1+\frac{G}{\alpha_0}} \end{aligned} \tag{3.42}$$

We observe that GA can perform much better than a pure Monte-Carlo approach because the fraction $\frac{G}{\alpha_0}$ is always positive. The degree of improvement obviously is determined by the function $G(\alpha_0, n, r, t)$ which strongly depends on the fitness ratios r.

3.11 Advantages of GA

GAs are very efficient in finding suboptimal solutions, yet these can also be obtained via heuristic techniques with often significantly less computation. The efficient utilization area of GAs may be limited to those cases requiring solutions quite nearly optimal such that they cannot be provided by heuristic methods. Besides, it is possible to find other algorithms that can outperform GAs in any given problem; for the TSP, simulated annealing is one of them. These results indicate that GAs fall short of completely solving computationally hard problems and should be used when the other methods may not perform as effectively as them in the particular problem being examined. Despite these shortcomings, GAs have some special advantages that can be useful in some problems.

First of all these is the possibility of using GAs in **hybrid** algorithms. Heuristic techniques can easily be incorporated into GAs method of building solutions. As DeJong points out [35] if one has a strong domain theory to guide the process of structural change, one would be foolish not to use it. The most obvious way of taking advantage of an heuristic rule which suggests some complete solutions consists of using these suggested solutions in the seed generation of the GA and to observe whether the GA can improve on them. On the other hand, some heuristics do not produce complete solutions but only help construct a solution by advising on the steps to be taken while extending a partial solution. These type of heuristics can be incorporated in the implementation of genetic operators. Both Grefenstette [65] and Jog *et.al.* [79], who used this method for TSP, obtained encouraging results indicating a fast convergence to high-quality solutions. Further, one can introduce a bias in the selection scheme of the GA by incorporating heuristics in the stage of fitness-proportionate selection. Tidor and de la Maza [136] suggest an example wherein the fitness-oriented selection is combined with weighting factors that are gradually reduced according to the Boltzmann equation, in a quite similar fashion to the simulated annealing algorithm.

Second, GAs are robust algorithms. Koza expresses this with the term *ruggedness* and notes that [86]

> In nature, evolution proceeds in the presence of noise and incomplete information in a changing environment in which the population may occasionally be subjected to catastrophic damage. Similarly, noise, incomplete information, unannounced changes in the rules of the game, and malfunctioning machinery are features of many real-world problems.
>
> From the early days of computing, von Neumann recognized the importance of synthesizing reliable organisms from unreliable components. Since complex computations require large numbers of computing devices wherein each device is unreliable to some degree, complex calculations inherently raise the issue of probabilistic logic. Genetic methods offer a way of to overcome the unreliability inherent in complex calculations.

Koza gives five examples of adaptation occurring in the presence of these interfering factors. His first example is an experiment where evolution proceeds using only inaccurate information (i.e., there is noise in the enviornment). In the second example evolution proceeds using only incomplete information about the environment (i.e. there is sampling). The evaluation of the individual solutions must be approximate and noisy if a complete evolution requires examination of an exponentially large number of possible states. A further example in which the state space must be sampled in a limited fashion is given by Fitzpatrick and Grefenstette [43] for a control application where the GA searches for an optimum control strategy. These examples are especially important for our purposes becase they indicate that a considerable amount of computer time can be saved by sampling the fitness or using partially correct fitness values. With these methods, the convergence rate may not be as fast as one would expect if the complete information were to be used.

Koza's other examples pinpoint the fault-tolerance of the GA (that was demonstrated by convergence despite the fact that a part of the population was killed in random intervals), adaptation to a changing environment (that was demonstrated by convergence in spite of the fact that the objective function to be learned was repeatedly modified), adaptation to a even more extremely changing environment in which the problem was changed halfway through computation. In all these examples, the GA achieves to converge to optimal or slightly suboptimal solutions.

As these examples show, the GA has an extra characteristic that can be useful in optimization. By keeping a set of diverse solutions and relying to fitness values probabilistically, GA is able to cope with inexact information and a changing environment. This property constitutes one of the unique and strong features of GAs in contrast to those other optimization methods which keep track of only a single solution and therefore become vulnerable to problems raised by such imperfections.

3.12 Summary and Conclusions

This chapter started with justifying the selection of the graph search approach for analysis of planning problems. Then, we presented a brief review of the complexity theory which indicates that certain classes of problems are intractable with conventional computers [4]. Among these problems lie many that are related to planning and graph models. We presented a list of these examples and summarized the approaches suggested for solving them. We especially concentrated on the recent results that indicate that the most NP-hard problems are also NP-hard to approximate and therefore there can be given no guarantees or bounds by an approximation scheme.

In the second part of this chapter, we classified and reviewed the methods that are used for graph search problems. The first class includes the enumerative techniques that search systematically. The second class consists of the gradient-descent based methods. The third class contains the newly popular probabilistic search mechanisms. We argued that this last class of search methods is most suitable to those problem spaces that are nonlinear and complex.

In the last part of this chapter, we presented a detailed analysis of genetic algorithms. We have produced, based on the schema theorem, a theoretical model that yields the evolution of probabilities for obtaining satisfactory solutions. This model was useful for comparing the GA with the Monte-Carlo approach. We argued that the GA is a general heuristic that is best suitable for finding suboptimal solutions.

Finally, we argued that GAs have extra advantages that make them superior to other methods. In addition to their ability to handle imperfections and incorporation of other heuristics, they can accept incomplete information and process those solutions that are only probably good, yet eventually converge to a desired solution by gradually refining them.

[4] Preliminary results on the theory of quantum computers indicate that some problems intractable with present computers (such as factoring) may be solved in linear time with these theoretical computers [128]

Chapter 4

Deterministic Planning Problems and Genetic Algorithms

In this chapter, we describe two simple, well-known, deterministic path planning problems and examine their complexity (*Section 1*). Then, we examine the performance of genetic algorithms on a selected test problem (namely the Traveling Salesman Problem) (*Section 2*). The empirical results obtained are used to characterize the results of genetic algorithms. The results on benchmark problems also validate our genetic algorithms-based software. The performance of genetic algorithms is compared to those of other methods (*Section 3*).

4.1 Deterministic Path Planning Problems

4.1.1 The Shortest Path Problem

Most common planning problems can be better understood by comparing them to the the shortest path problem (SPP) in a deterministic model. The SPP is a basic problem of combinatorial optimization, with many applications. The SPP can be defined as follows: We are given a digraph $G(V, E)$ where each edge e_{ij} connecting a node i with a node j has an associated cost d_{ij}. Further a pair of nodes (s, t) are specified as source and sink nodes. We are asked to find a path from s to t with the minimum cumulative cost.

Among many solution algorithms that can solve this problem, Dijkstra's algorithm is the most prominent one, even though it is neither the most universal nor the most efficient one. Dijsktra's algorithm can be used when there are no cycles and when $d_{ij} \geq 0$ for all edges. This algorithm can be classified within the category of best-first search method which was described in Chapter 3. The complexity of Dijkstra's algorithm is $O(n^2)$ or $O(m \log n)$, whichever is smaller (We assume that $|V| = n$ and $|E| = m$). If the negative edge costs are allowed, the Bellman-Ford algorithm can be substituted for Dijsktra's algorithm. A recent algorithm suggested by Fredman and Tarjan [48] improves the worst case complexity to the largest of $O(n \log n)$ or $O(m)$.

4.1.2 The Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a well-known and well-defined problem of combinatorial optimization. It is easy to state but difficult to solve. Informally, it asks for the shortest tour for a list of cities to be visited. More formally, it asks for the specific permutation v_π of n vertices such that the sum

$$\left(\sum_{i=1}^{n-1} d(v_{\pi(i)}, v_{\pi(i+1)})\right) + d(v_{\pi(n)}, v_{\pi(1)}) \tag{4.1}$$

of the labels in an edge-labelled graph is minimum in the space of all possible permutations.

TSP is very often investigated under the following restriction: the nodes are points on the plane, and the distance of the points u and v of coordinates u_1, u_2, and v_1, v_2, respectively, is determined in one of the following ways :

Maximum Metric: $d[u, v] = \max(|u_1 - v_1|, |u_2 - v_2|)$
Manhattan Metric: $d[u, v] = |u_1 - v_1| + |u_2 - v_2|$
Euclidean metric: $d[u, v] = \sqrt{(u_1 - v_1)^2 + (u_2 - v_2)^2}$

It has been proved that if the coordinates are integers, and if we ask whether there exists a Hamiltonian circuit which is not longer than a given integer K, the problem is NP-complete under either maximum or manhattan metric.

For the euclidean metric it has only been proved that the problem is NP-hard. It is, in fact, not clear [87] whether the euclidean variant belongs to the class NP because it is not known whether it is possible to decide in polynomial-bounded time whether the following inequality:

$$\sqrt{m_1} + \sqrt{m_2} + \cdots + \sqrt{m_n} \leq K$$

is satisfied for given natural numbers $m_1, \ldots, m_n$ and K (in the case the time bound is related to the number of digits needed to write down the input data).

A person looking at a TSP problem in a planar drawing with relatively limited number of cities can quickly find a very good path, and one might therefore feel that it is an easy problem. Our ability to do so is based on the fact that all the relevant relationships can be seen in a two-dimensional drawing [75]. For example, one can easily use the heuristic that if nodes A and B are as far apart as possible, they will tend to occur near opposite sites. Nevertheless, when the problem does not correspond to a two-dimensional drawing, our ability to solve the problem visually disappears.

Despite such reliefs, obtaining a good solution to TSP still requires powerful optimization methods. Enumeration of possible solutions is an hopeless attempt for there are $(n-1)!$ solutions, where n is the number of nodes in the problem ($(n-1)!/2$ solutions in a nondirected graph).

It is important to recognize that TSP, being essentially a permutation problem, corresponds to many practical planning problems, and thus is encountered in seemingly unrelated subjects. For example, in engineering design, the question **"Given a circuit board, what is the best wiring layout?"** is an easily recognizable form of TSP. TSP also appears in job-shop scheduling and in transportation planning. Besides these direct applications, many NP-hard problems can also be solved easily once TSP is solved. That is, many combinatoric problems can be mapped onto TSP relatively easily.

Because of its widespread occurrence, TSP has been attacked vigorously with many different types of procedures. These studies culminated in a very large database of solution methods, excellent heuristics as well as many benchmark problems. These studies also prove useful in comparing different optimization procedures. Since TSP is both an important planning problem and an important benchmark, it was chosen for the subsequent testing of our GA software.

4.1.3 Dynamic Programming Applications

The SPP can be efficiently solved because it has the property of optimal substructure. The exploitation of this property by solution procedures is best observed within the framework of dynamic programming. For the shortest path problem, assuming that we are given a graph with the set of nodes $1, 2, ..., N$, the dynamic programming equation reduces to :

$$J_k(i) = \min_{j \in S_{k+1}} \left(d_{ij} + J_{k+1}(j)\right), \quad i \in S_k, \quad k = 0, 1, \ldots, N-1 \tag{4.2}$$

with

$$J_N(i) = d_{iN} \tag{4.3}$$

This equation can be intuitively interpreted as :

Optimal cost to go from i to t = $\min_j(c_{ij}+$ Optimal cost to go from j to t).

Bellman [12] expresses the principle of optimality as follows: *"An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."* For the shortest path problem the optimal substructure property corresponds to the fact that the subpaths of the shortest paths become the shortest paths. This is easily proven by the contrary argument that if a shortest path were to have an suboptimal substructure, it would be possible to construct a different path that would be shorter only by replacing the suboptimal subpath with the optimal one.

Dynamic programming can also be applied to the TSP. However, in this case the number of states that are needed to be examined grows exponentially and therefore dynamic programming does not lead to a polynomial time algorithm. When applied to the TSP, the amount of computation required by the dynamic programming algorithm is in the order of $O(n^2 2^n)$ [107]. Despite the fact that the problem space is smaller for the TSP compared to the shortest path problem [1] , dynamic programming does not lead to a polynomial time algorithm because even a partial path specification implicates a global constraint in the TSP.

4.1.4 Closed Semirings

The dynamic programming algorithm for the SPP produces a polynomial time algorithm because the operators used satisfy certain properties. These properties are formalized by Ullman and Hopcroft [137] who showed that a general algorithm can solve the path finding problems in certain structures. Their general path finding algorithm works for any closed semiring. A closed semiring is a system $(S, \oplus, \odot, \bar{0}, \bar{1})$ where S is a set of elements, $\oplus$ (the **summary** operator), and $\odot$ (the **extension** operator) are binary operations on S, and $\bar{0}$, $\bar{1}$ are elements of S. For the shortest path, we use the closed semiring $(R^{\geq 0} \cup \infty, \min, +, \infty, 0)$.

The properties of the closed semirings can be found in [137]. Some important properties of closed semirings are:

1) Both binary operations $\odot$ and $\oplus$ are associative and commutative,

2) The extension operator $\odot$ distributes over the summary operator $\oplus$, i.e.

$$a \odot (b \oplus c) = (a \odot c) \oplus (a \odot c) \tag{4.4}$$

For the above described closed semiring for the shortest path problem this is easy to observe:

$$a + \min(b, c) = \min(a + b, a + c), \quad a, b, c \geq 0 \tag{4.5}$$

Note that this result can be generalized over the monotonically increasing cost functions (and monotonically decreasing cost functions in maximization problems). These properties lead to optimal substruc-

[1]The number of Hamiltonian paths in a graph with n variable node is equal to the the space of all permutations, i.e. it is $n!$. The number of all paths in a graph having n nodes (assuming that any node can be reached from any other node) is equal to the following sum:

$$\sum_{k=0}^{n} k! \left(\begin{array}{c} n \\ k \end{array} \right) = \sum_{k=0}^{n} \frac{n!}{(n-k)!} = \sum_{k=0}^{n} \frac{n!}{k!} = n! \sum_{k=0}^{n} \frac{1}{k!} \approx en!$$

ture.

On the other hand, the property of optimal substructure does not hold for a different problem in which we want to maximize the cost (the *longest path problem*, LPP) because the addition operator does not distribute over the operator max, i.e.:

$$a + \max(b, c) \neq \max(a, b) + \max(a, c), \quad a, b, c \geq 0$$

Because we cannot partition the state space, the LPP turns out to be an NP-complete problem. (This can be deduced from its conversion to Hamiltonian path problem). Finally, the problem of finding a path with the maximum utility U such that $U(P) = f(C(P))$ can be solved with the dynamic programming in polynomial time only when the utility function is monotonically decreasing with greater cost, such as $f(C) = -C$, $f(C) = e^{-C}$ or $f(C) = 1/C$.

4.2 Genetic Algorithms and TSP

Genetic algorithms constitute an important solution technique for search and optimization problems. The quickly growing literature on GAs indicate a strong performance in many areas, if not all. Davidor [33] comes to this conclusion after he tried many problems of robotics with GAs. Goldberg [58] gives many examples where GAs were useful in various search and optimization problems. Koza [86] extends the classical genetic algorithms with the paradigm of genetic programming. In this paradigm the problems are represented by a set of instructions and solutions consist of functional programs. The paradigm has been applied to a wide and diversified set of problems quite successfully [86]. GAs are successfully applied in image registration, machine learning, scheduling and in many other computationally-hard problems [59]. Interesting applications include music composition, finding the face of a criminal suspect with the help of a witness, producing random-number generators, and finding good structures for artificial neural networks.

Because of their special advantages, we suggested the genetic algorithms as a general method that can solve both deterministic and nondeterministic planning problems. Hence, we developed an object-oriented software based on genetic algorithms and tested it through the applications on the *Traveling Salesman Problem* (TSP), which is the central problem of discrete optimization. Indeed, it is frequently cited that while GAs perform well on an average problem, TSP is one of the rare examples where they can fail expectations [64]. These observations are generally explained as due to the difficulty of representing a TSP tour with classic binary encoding. Successful applications of genetic algorithms to TSP has recently started to appear in literature, see [18] for an example.

The benchmark problems we have examined have been obtained from the TSP Library developed at the University of Augsburg [123] We have selected five problems containing 51, 76, 96, 105 and 202 nodes (in the files **eil51, eil76, gr96, lin105, gr202**), which will be denoted as P51, P76, P96, P105 and P202 respectively. The optimal solutions of these problems are also provided by the library. An example problem is shown in Figure 4-1 for P96. The distances in all these examples are calculated by using the standard Euclidean 2-D metric.

4.2.1 Implementation of Genetic Operators for TSP

Koza's examples [86] as well as further recent work [35] demonstrate that binary encoding is only one among many other possible representation schemes which can produce satisfactory performance for genetic algorithms. It must be emphasized that the efficiency of a representation can be defined only over a class of problems. Our implementation, therefore, uses a natural object type encoding and relies on specific implementations of GA operators for each distinct class of objects.

The solution to TSP is a cyclic ordering of the vertices. This particular structure prevents the use of well-known implementations of genetic operators and make it necessary to use particular implementations of genetic operators such that the solutions generated by these operators are feasible. Our object-oriented software uses specific methods for the genetic operators being applied to TSP. Mutation is performed by choosing at random two vertices along a path and by reversing the order of the vertices along these paths. This implementation is originally suggested for application of simulated annealing to TSP [85]. Implementation of crossover is slightly more complicated. First, it is selected, at random, a break point along each parent path, yielding four subsequences. After determining the common subsets, these subsequences are modified by trying to protect as much as possible the longest common subsets without any repetition of the vertices. The children paths are then generated by interchanging the modified subsequences. This implementation differs from the methods suggested in literature (those are mostly either locus-based or order-based encodings); see [61], [64] for a review of these different approaches and see [18] for a particular implementation of crossover with a superimposed heuristic.

4.2.2 GA Run Results

Running the GA software repeatedly and using each time a different seed for the Park-Miller randomizer,[2] we obtained the data on the evolution of the GA solutions for each version of the problem. Figure 4-2 shows the performance of four different GA runs for P51. The y-axis denotes the ratio of the best solution cost found by GA to that of the best given in the library and thus signifies the quality of the solution. All the examples in this suite are run for 500 individuals through at least 300 gener-

[2]Park-Miller randomizer uses the recursion $x_i = 7^5 x_{i-1} \bmod (2^{31} - 1)$. It is found to be superior to other random number generators in many measures; for example its results have lower informational entropy than those of most well-known methods, although GAs have succeeded in producing random number generators with lower informational entropy [86].

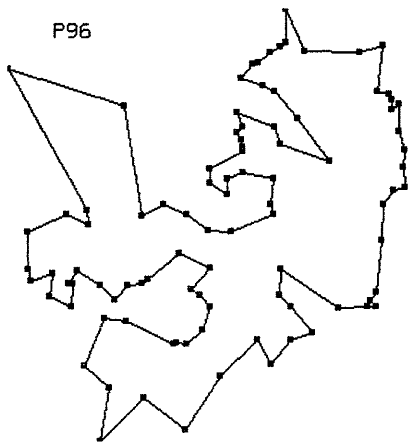


Figure 4-1: *TSP Problem P96 and Its Optimum Tour*

ations. Further examples show that a result that is *at most* 3% larger than the optimum is obtained for the specific problem P51 with 500 individuals and 300 generations, approximately in one out of six independent runs.

Similar performance curves are obtained for other problems P76, P96, P105 and P202 as well (each with 500 individuals per generation) and shown, respectively, in Figure 4-3, Figure 4-4, Figure 4-5, and Figure 4-6. One can observe from these figures that as the size of the problem increases, the convergence to a particular ratio requires a larger number of generations. For instance, convergence to a solution with a cost that is only 50% larger than that of the optimal solution is reached within 50 generation in the case for P51, whereas it takes about 400 generations in the case for P202.

A comparison of the selected runs for each of the problems P51, P76, P96, P105 and P202 is presented in Figure 4-7. The figure suggests that problems having larger sizes might have a smaller convergence rate. In addition, we observe that the initial values of the runs are different for each problem. Can this last factor explain the lag in the performance that appears as the problem size is increased? We could perhaps get a better picture for comparison by normalizing the results with respect to the initial values. We postpone the discussion of this question and turn our attention into developing a model which can be useful in answering this question as well as some other questions that one might pose.

4.2.3 Modelling of Performance Data

For drawing practical conclusions from these data, we first simplify the results by developing an empirical model of the observed performance curves. Let us denote with y_0 the cost of the minimal-

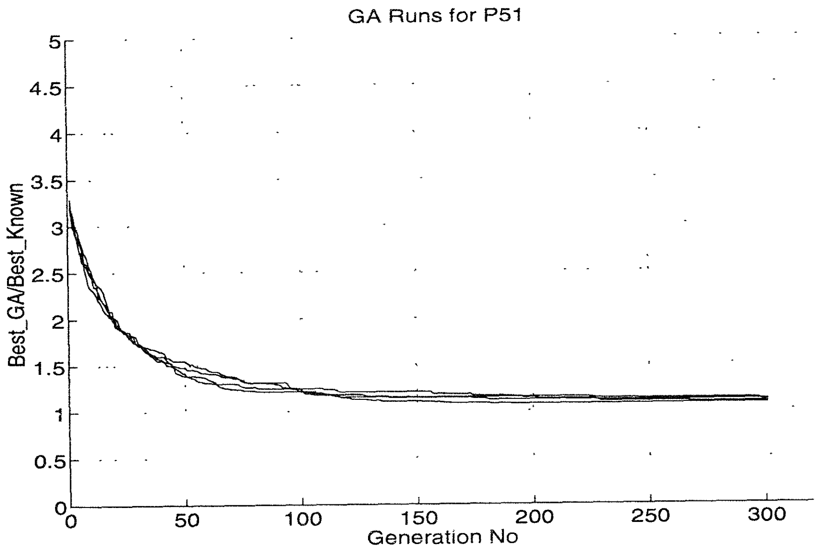


Figure 4-2: *GA Runs for P51*

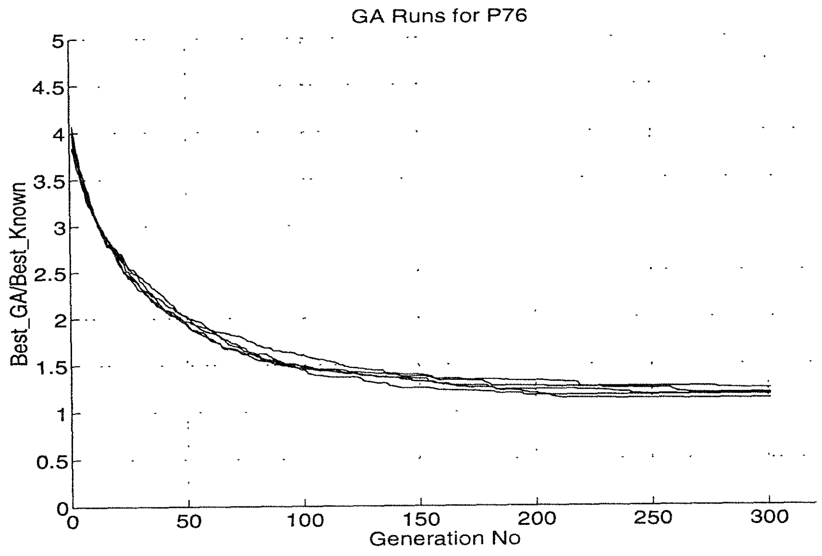


Figure 4-3: *GA Runs for P76*

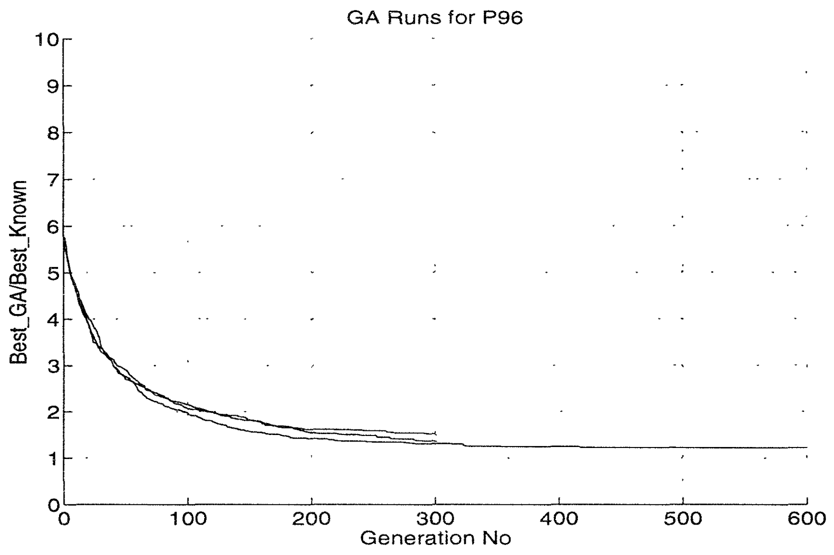


Figure 4-4: *GA Runs for P96*

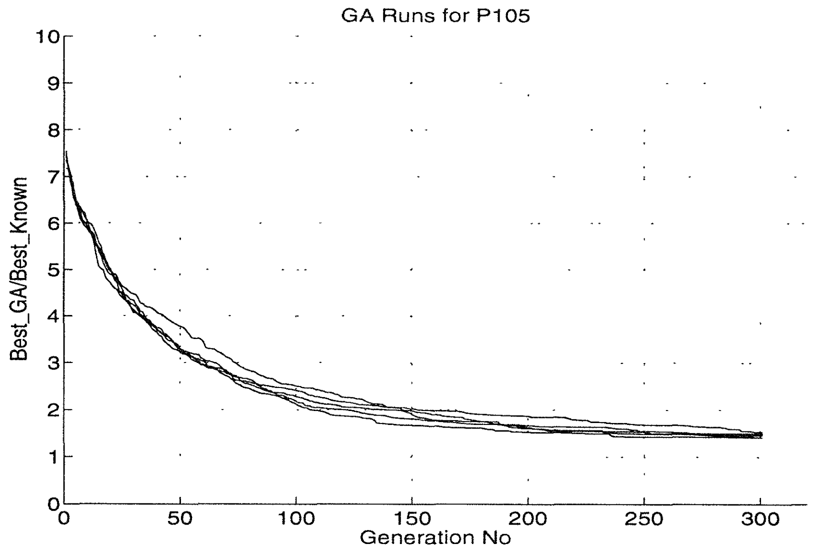


Figure 4-5: *GA Runs for P105*

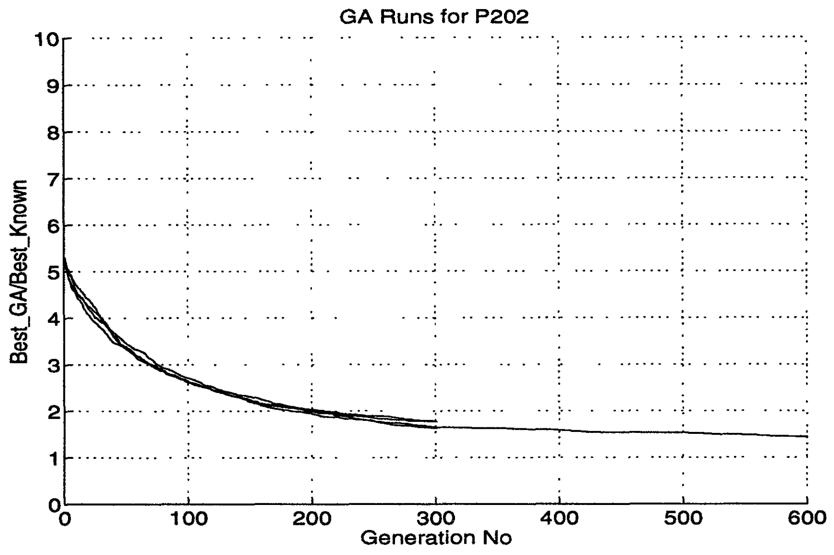


Figure 4-6: *GA Runs for P202*

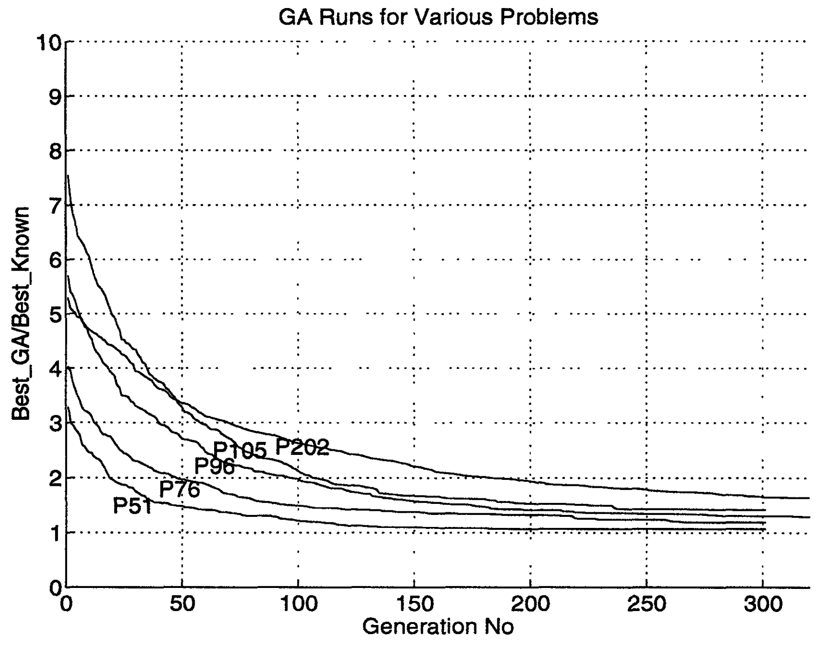


Figure 4-7: *Comparisons of GA Performance for P51, P76, P96, P105 and P202*

Table 4.1: *The Parameters of the Performance Model of GA for Benchmark Problems (500 individuals)*

Problem Size	α	y_0	Average Correlation
51	0.209	3.22	0.984
76	0.171	4.00	0.994
96	0.148	5.70	0.994
105	0.155	7.32	0.993
202	0.101	5.24	0.996

cost solution in the first generation of a GA run. Clearly, y_0 depends on the population size and the problem structure. Further, let us denote with $y(t)$ *the expected value* for the cost of the minimal-cost solution in the t-th generation of a given GA run. (Previous figures can be interpreted as showing the evolution of $y(t)/y_0$ with each new generation). Since GAs are stochastic algorithms the observations of $y(t)/y_0$ fluctuate around a mean value. Nonetheless, we can approximate their average behavior with an empirical formula. In fact, a quite good fit is obtained for these problems with an empirical function in the following form:

$$y(t) = 1 + (y_0 - 1)\exp(-\alpha\sqrt{t}) \tag{4.6}$$

Note that both y and y_0 are the expected values of the corresponding random variable; thus Equation 4.6 must be interpreted as the maximum likelihood estimate of the instantiations of the random variable for $y(t)$. Table 4.1 presents, for all the problems, the y_0 and α values that appear in Equation 4.6, found by a linear regression, as well as the average correlation coefficients of the regression. The results of the empirical formulas diverge from the data with at most 9% error. In addition, the large values of the correlation coefficients point to a good fit between the data and the empirical formula. It should be noted that the empirical formula is arbitrary in the sense that the exponent of the generation parameter t has been chosen (.5) before the regression analysis.

Analyzing the limit cases, we observe that:

$$y(t = 0) = y_0$$

and,

$$\lim_{t\to\infty} y(t) = 1$$

Further, we have the following for the convergence rate

$$\frac{d}{dt}(y - 1) = -\frac{\alpha}{2\sqrt{t}}(y - 1) \tag{4.7}$$

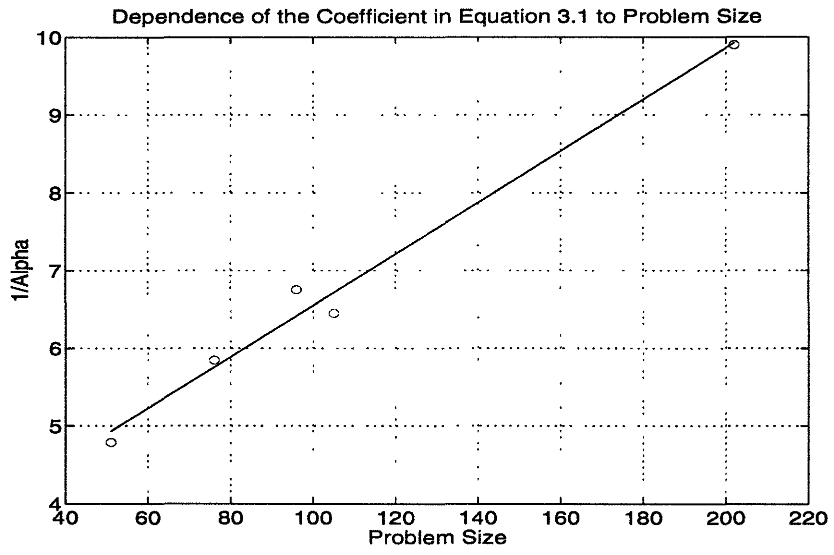


Figure 4-8: α^{-1} *versus Problem Size for P51, P76, P96, P105 and P202 for 500 individuals*

To answer the question on the effect of the problem size on GA performance, we first use the predictions of the empirical models. Figure 4-8 presents how α^{-1} values depend on the problem size (S). We find a first-order polynomial relation between these parameters. In other words,

$$\alpha^{-1} = \beta^* \, S + \gamma^* \tag{4.8}$$

where $\beta^* = 0.033$ and $\gamma^* = 3.24$.

Equation 4.8 is tested by plotting the fraction

$$\frac{y(t) - 1}{y_0 - 1} = \exp\left(-\frac{\sqrt{t}}{\beta^* \, S + \gamma^*}\right) \tag{4.9}$$

versus the problem size S and comparing the prediction to the empirical data as presented in Figure 4-9. Although discrepancies between the data of the new test-suit and the predictions of the extended model now appear larger, the results and the empirical model are still relatively congruent.

4.2.4 Effects of the Population Size

Another important parameter affecting the performance of a GA run is the size of the population that undergoes the genetic operations. Generosity in the choice of this parameter usually leads to a faster convergence rate per generation. But, this advantage is obtained at the expense of a larger processing time for each generation. Thus, one suspects that an optimum choice of the generation size exists. Beside determining the convergence rate, the population size also affects the initial value (y_0) in a given run, which is a minimum over all individuals in the seed generation.

We can make a further extension to the empirical model by analyzing the effect of changing population size. Performances of several runs with different population sizes (20,50,100 and 250) are shown in Figure 4-10 for P51. Figure 4-11 shows the same comparison for the problem P202. In addition to the aforementioned manifestation (that is, the performance of GA suffers as the population size is reduced) we also observe that this effect is more dominant in the problems having larger sizes.

Employing both Equation 4.6 and Equation 4.8 and regressing on the results, we obtain α values versus problem size for each of the problems P51, P96 and P202. Upon examination of the results, we propose a logarithmic relation as a likely candidate:

$$\alpha(S,n) = \frac{1}{(\beta^*\, S + \gamma^*)} \frac{\log n}{\log 500} \tag{4.10}$$

Indeed, the plot of this function and the observed values show good agreement (Figure 4-12).

We can simplify the formula appearing in Equation 4.9 by defining $\beta = \beta^* \log 500$ and $\gamma = \gamma^* \log 500$. Then,

$$\frac{y(t)-1}{y_0-1} = \exp\left(-\frac{\sqrt{t}}{\beta S + \gamma} \log n\right)$$
$$\frac{y(t)-1}{y_0-1} = n^{-\frac{1}{\beta S+\gamma}\sqrt{t}} \tag{4.11}$$

4.2.5 Effects of the Initial Values

One can also ask how the initial values affect the results of the GA runs. First let us recall that the initial best convergence ratio y_0 depends on the population size and the probability distribution function that a random set of solutions follow. Note that y_0 is a *minimum* over all n individuals in the seed generation :

$$y_0 = \min_n \{y_1^{(0)}, y_2^{(0)}, \ldots, y_n^{(0)}\} \tag{4.12}$$

The distribution that $y^{(0)}$ follows is determined by the method used for the generation of the individuals for the seed generation. The usual method practiced draws these individuals randomly from the problem space, although one is free to introduce any systematic bias in the construction stage.

For a randomly-created generation $y^{(0)}$ follows a probability distribution, which can be approximated quite well with a normal distribution $\mathcal{N}(\bar{y}, \sigma)$. Figure 4-13 compares the observed instantiations of $y^{(0)}$ to a normal distribution. Table 4.2 presents the expected values (*means*) and standard deviations for the problems being examined.

Once the probability density function for a set of random variables $y^{(0)}$ is known, the corresponding

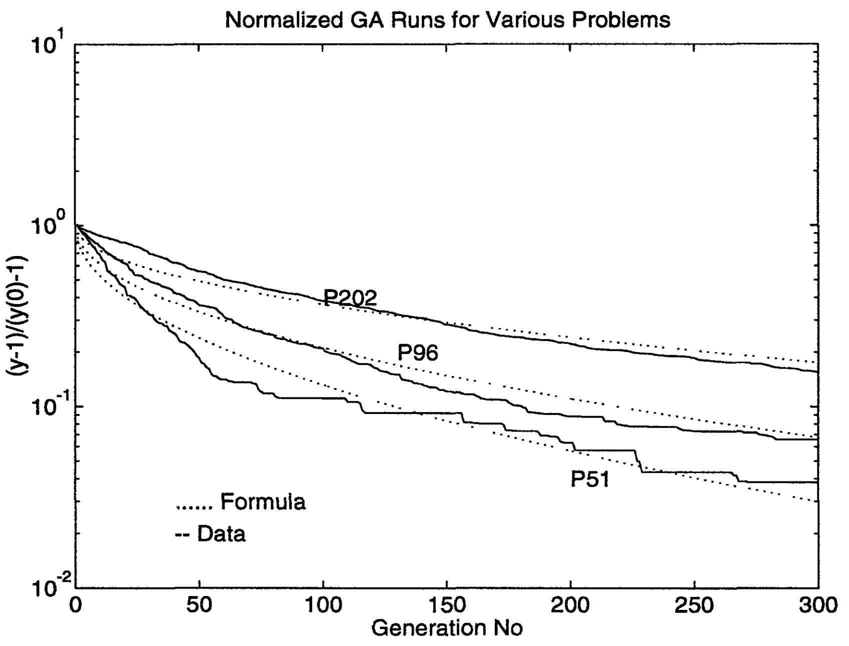


Figure 4-9: *Normalized Comparisons of GA Runs for P51, P76, P96, P105 and P202*

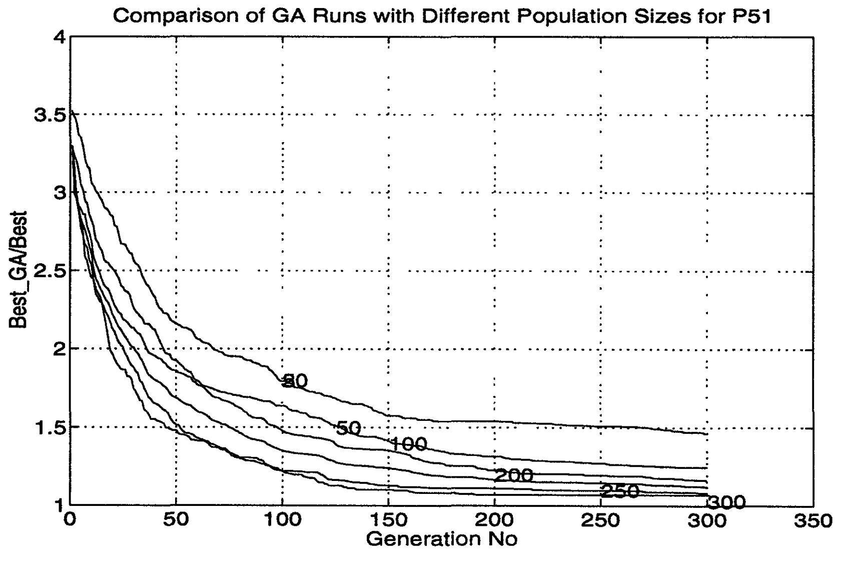


Figure 4-10: *Comparison of GA Performance for Runs with Different Population Sizes for P51*

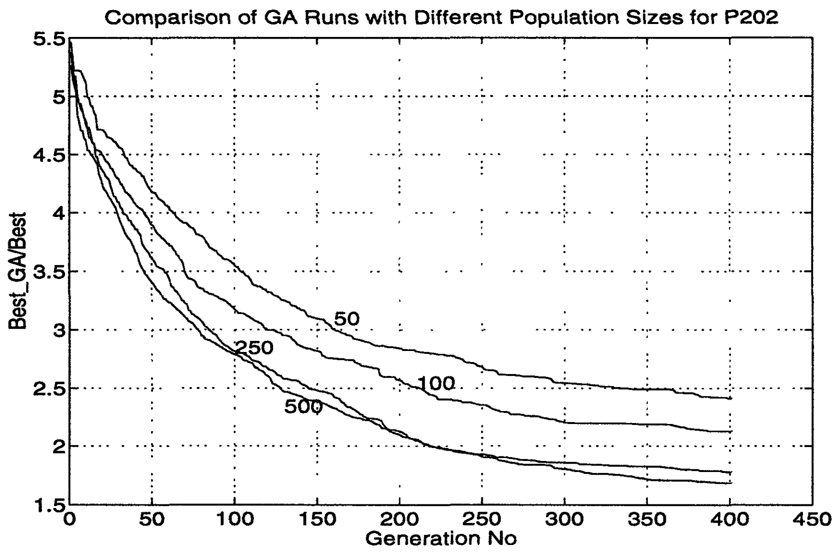


Figure 4-11: *Comparison of GA Performance for Runs with Different Population Sizes for P202*

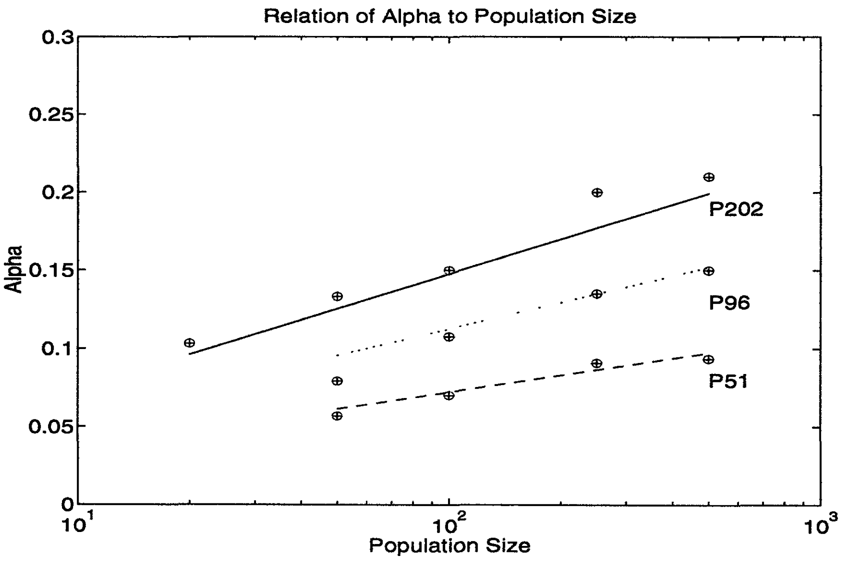


Figure 4-12: *Alpha values versus Population Sizes*

Table 4.2: Parameters for the Distribution of GA Initial Values

Problem Size	Mean ($\bar{y}$)	Standard-Dev. (σ)
51	3.851	0.2039
76	4.552	0.2025
96	6.586	0.2923
105	8.607	0.4137
202	5.707	0.1518

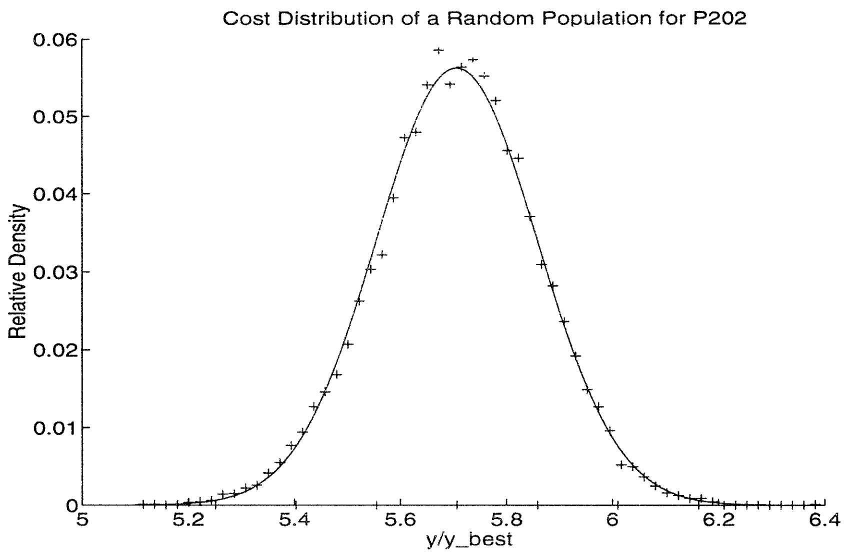


Figure 4-13: *Distribution of Initial Solutions for P202*

cumulative probability distribution function for the minimum $y_0^{(0)}$ for the given set of random variables can be found easily from the laws of probability. (See [91] for the asymptotic values of the extreme-value distributions). From this p.d.f., the mean value of the minimum-value distribution can easily be evaluated. These evaluations indicate that the corresponding probability density function has a lower mean value and a higher standard deviation than the initial distribution.

We do not delve into details of these evaluations for the results of the GA are not very sensitive to how the initial population was constructed. This has been observed by performing an experiment wherein the initial population is selectively constructed. For P51, the GA search has been performed with 100 individuals throughout 250 generations with biased initial populations for several times. Figure 4-14 shows that the range of the results are quite similar even though the initial populations were different. In this figure, G0 indicates a randomly selected set of 100 individuals. G1, G2, G3 G4 and G5 indicate the corresponding rank of a 100 individual group among 500 randomly-created individuals sorted according to their fitness. The initial best convergence ratio differs approximately 3% among these groups. Repeating this experiment for other problems P76, P96, P105 and P202 has produced similar results.

On the other hand, reducing diversity by using structurally very close or replicated solutions have generally adverse effects on convergence. Researchers have observed that introducing good solutions are helpful, but not too much and if overcarried might even be harmful for it could limit diversity and lead to premature localized optimization [145, 35].

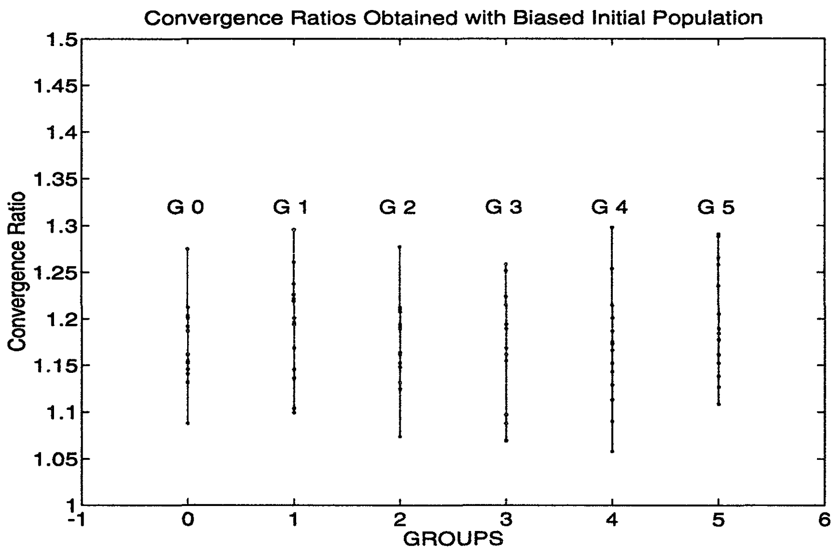


Figure 4-14: *Effect of Initial Populations for P51*

4.2.6 Optimum Population Sizing

A frequent question in the use of GAs is what values should be used for the population size for minimizing runtime. While increasing the population size enables a faster convergence rate, the amount of computation per generation is also increased. In a serial computer that process each individual separately this trade-off results in an optimum population size. In this section we examine what the empirical results indicate on the optimal value of population size.

Rearranging Equation 4.11, we obtain the expected number of generations t^* for attaining a certain convergence ratio y^* as

$$t^{*1/2} = (\beta S + \gamma)\frac{\log \frac{y_0 - 1}{y^* - 1}}{\log n} \tag{4.13}$$

To find the required time T^*_{exp} for a given number of generations, we assume that the time it takes to process a population of n individuals will be linearly proportional to the population size n for computation on a serial computer. Then

$$T^*_{\text{exp}} \sim \frac{n}{\log^2 n} \log^2 \frac{y_0 - 1}{y^* - 1} \tag{4.14}$$

If we neglect the logarithmic term that includes both y^* and $y_0(n)$, $T_{\text{exp}} \approx n/\log^2 n$ and we obtain the optimum population size that minimizes the expected runtime, by using the property

$$\frac{d}{dn} T_{\text{exp}}|_{n=n_{\min}} = 0$$

as $n_{\min} = e^2 = 7.34$.

The effect of the logarithmic term containing $y_0(n)$ is beneficial in reducing the expected runtime for the initial convergence ratio is more likely to be smaller with increasing population size. Nevertheless, as we have seen before this effect is quite small, and can easily be ignored. Thus, the empirical results indicate an optimum population size between 7 and 8.

This results can be compared to other results in the literature. Goldberg [60], who examines the same question by using the probabilistic models of classic GA, arrives at these conclusions:

1) Assuming a logarithmic convergence with population size n, the optimum population size is $n_{\min} = 3$, regardless of the string length l used in the representation of problem.

2) Assuming a constant convergence time, optimum population size grows exponentially with increasing string length.

Our results indicate a square-logarithmic convergence, which is different yet comparable to the result of Goldberg's first assumption. Other studies [35, 64] also suggest relatively small population sizes for computations with serial computers. Also note that both Goldberg's and our results indicate a population size as large as possible for a perfectly parallel machine.

Nevertheless, we should note, caution should be exercised in the interpretation of our results. The empirical results are obtained with a minimum population size of 50 and perhaps should not be extrapolated to smaller population sizes. With a smaller number of individuals,

- Reduced diversity leads to a greedy optimization that converges quickly to a local optimum.
- The probability of convergence to target solutions decreases sharply.

The second point is especially worth emphasizing. In both probabilistic algorithms, GAs and Monte-Carlo methods, having a larger number of individuals in a given experiment increases the probability of finding an acceptable solution.

Suppose that one requires a certain confidence probability for attaining an acceptable solution. Because reducing the population size will decrease the confidence ratio, when using a smaller number of individuals, one will have to increase the number of experiments performed in order to keep the confidence ratio constant. In the case of Monte-Carlo runs with independent trials, these counter-effects balance each other and therefore the expected completion time becomes independent of the population size. It was found that (Equation 3.40) the runtime would be proportional to the ratio $\frac{\log 1-z}{\log 1-\alpha_0}$ where z was the required confidence ratio and α_0 was the probability of finding a random individual satisfying the success criterion.

In the case of GAs, the optimum population size can be found once we know the corresponding probability distribution of the runtime as a function of the population size. Nonetheless, if we fix the confidence ratio in obtaining an acceptable answer, we observe that we will have to repeat the number of experiments (GA runs) to find a solution. Because one can restart the GA with small population size many times in the time it takes completing a GA with a larger population, there exists a trade-off between increased number of experiments and the decreased number of individuals. The optimum value for population size for a fixed confidence ratio can be explicitly found from Equation 3.43, that gives the runtime as a function of the ratio of the nonconvergence probability of GA to that of Monte-Carlo method, which includes the population size as a parameter.

These concerns with the empirical results indicate that the best use of GAs in a serial computer is achieved by a small population size and by running the algorithm repeatedly until a nominal convergence. Further, as Goldberg suggests [60], one can introduce diversity by transferring the best individuals of the converged population to the new population and then generating the remaining individuals randomly.

4.2.7 Completion Time

Finally, we examine the question regarding the observed runtimes and their scaling with problem size. Let τ_0 stand for the processing time required for a single generation of n individuals for problem size S. These individuals first are evaluated with a fitness function that maps their fitness to a real number. The fitness evaluation takes time proportional to the problem size S for it simply sums up the costs of edges appearing in a tour. In addition, each of these individuals is processed separately by genetic operators (*duplication, crossover,* and *mutation*). The time taken by these operators depends on the particular implementation but it is almost universally a polynomial function of the problem size. If one requires an algorithm having an exponential complexity for these operations, one is probably using a wrong representation.

Ignoring the time taken by sorting and other overhead costs between generations, for computation time on a *serial computer*, we have

$$\tau_0 = c\, n\, h(S) \tag{4.15}$$

where c is a constant that depends on the processor speed and $h(S)$ is a polynomial measuring the computation time per generation per individual for a problem with data size S. $h(S)$ includes both the time taken by fitness evaluation and the time taken by genetic operations; in other words

$$h(S) = f(S) + g_{dup}(S) + g_{cros}(S) + g_{mut}(S)$$

In our implementation, the overall time taken by genetic operators is approximately linearly proportional with the problem size. This results from the following facts : **1)** problems are represented with ordered list of nodes, **2)** some of the steps in genetic operations, such as the instructions `copy` and `reverse`, take time proportional to the length of the list they operate on, and **3)** these instructions dominate over the rest timewise. Because the fitness evaluation $f(S)$ is also linearly proportional with the problem size, we have $h(S) \sim S$. In fact, for the TSP problems being examined, using $c = 0.00145\ s$ and $\tau_0 = cnS$ yields a very good approximation for the observed runtimes for a SunSPARC-4 workstation running LUCID-Lisp 4.1 with ~80% of CPU time.

Once again by ignoring the overhead costs, we can easily use the generation parameter t to obtain the runtime T :

$$T = \tau_0\, t = c\, n\, h(S)\, t \tag{4.16}$$

Substituting this into Equation 4.11, we obtain

$$y(T, n, S) = 1 + (y_0(n) - 1)\, n^{-\frac{1}{\beta S+\gamma}\sqrt{\frac{T}{cnh(S)}}} \tag{4.17}$$

The expected value of the of the runtime (T_{exp}) for achieving a particular convergence ratio (y^*) for a problem of size S, with a population size n:

$$\begin{aligned} T_{\text{exp}}(y^*, n, S) &= \frac{cnh(S)(\beta S + \gamma)^2}{\log^2 n} \log^2 \frac{y_0 - 1}{y^* - 1} \\ &\sim S^{2+m} \log^2 \frac{y_0 - 1}{y^* - 1} \end{aligned} \tag{4.18}$$

where m is the degree of polynomial $h(S)$.

Note that the required time is proportional to the square of the logarithm of the inverse of the error term $\epsilon^* = y^* - 1$. In other words, the completion time increases exponentially with ever-decreasing error ratio. More importantly, the time required is only a **polynomial** function of the problem size. ($T_{\text{exp}} \sim S^3$). This proves that the GA method is easily scaleable. Note that this formula gives only the expected complexity as a function of the problem size for asymptotical convergence.

Sometimes it is important to find *the best* solution to a problem. GAs can also be employed for this purpose. Because a GA most often converges to a suboptimal solution, one must perform a number of independent, time-limited runs in order to evade the suboptimal solutions. If there were no limit in the number of experiments to be performed and in the computation resources, the optimum solution will eventually be found in one of these test runs. Runtime for obtaining the optimal solution obeys a probability distribution. Therefore, one can talk about the expected runtime or a runtime with a pre-set confidence probability.

In order to determine the behavior of runtime for obtaining the perfect answer versus problem size, we performed a series of new experiments involving the TSP problems having a number of nodes between 6 and 14. These experiments were performed by running the GA with 20 individuals and for a maximum

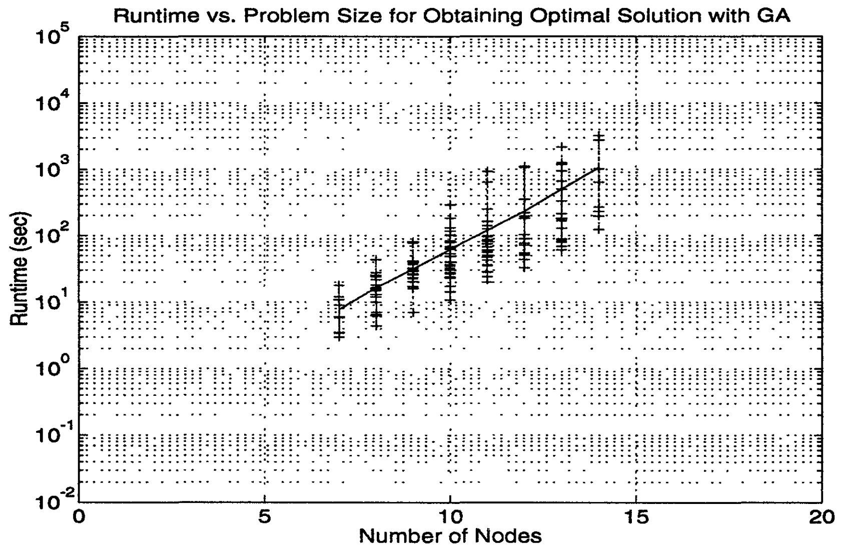


Figure 4-15: *Runtime versus problem size for obtaining the optimal solution with GA*

of 25 generations until the target answer is attained. If the optimum solution was not found with a single computation, the process was repeated again by transferring the best 3 solutions to the seed generation of the next run. Since experiments run until the target solution would be diagnosed, one must resort to different methods for finding the optimum solution. Although the problems chosen have small sizes their sizes were still large enough to make it impractical to apply enumerative techniques. We circumvented this problem by designing a particular TSP such that the optimal answer is known beforehand. In these experiments, all nodes have been located along a circle; therefore the optimal answer was a polygon.

The results of the experiments, which were performed in a SunSPARC-1 workstation, is presented in in the next figure 4-15. As can be seen from the figure, expected runtime increases exponentially with larger problem size. In fact, the following empirical equation yields the values for the expected runtime, with an almost perfect correlation:

$$T_{\text{exp}} = 0.075e^{0.67S} \quad sec \tag{4.19}$$

where S is the number of nodes in a given problem.

For runtime has a probability distribution, GA will obtain the target answer in a much shorter time than what the equation suggests at times and at some other times this will take much longer. Nevertheless, the expected runtime has an exponential complexity and we observe that GAs become intractable algorithms for combinatorial problems once the perfect answer is required of them.

In view of these results, we can view a GA as a heuristic method that can yield good results easily but as a method that will take exponentially increasing time as the criterion for suboptimality is stricktened.

4.3 Comparison of Experimental Results

In this section, we compare the GA method using TSP as the benchmark problem, with *Monte-Carlo*, *simulated annealing*, *best-first* and *A** search methods, both with respect to observed runtime and accuracy in obtaining a good solution.

4.3.1 Comparison to Random Search

The first evaluation of GAs perhaps must be made in comparison to Monte-Carlo method, which is their forefather in a sense. The experimental results of GA can be compared to what should be expected from a set of Monte-Carlo runs by expressing the results in terms of the standard deviations. For obtaining the expected values for the Monte-Carlo approach without resorting to determination of the corresponding extreme-value distribution, we can use the property that the cumulative probability must be equal to 0.5 for the mean value. For the probability of not obtaining a single solution within t generation with n trials in each, we have

$$\Theta_{MC} = (1 - \alpha_0)^{nt} = 0.5 \tag{4.20}$$

It follows that

$$\alpha_0 = 1 - 0.5^{\frac{1}{nt}} \tag{4.21}$$

As the number of experiments (nt) increases, the value of α_0 asymptotically reaches 0. This indicates that one can obtain a solution with the Monte-Carlo method with 50% probability even for those correspondingly small α_0 values as the sample size increases. As a first-order approximation, we can use the fact that the solutions approximately follow a normal distribution and from this fact we can find the corresponding convergence ratio or the standard deviation from the mean value for a given number of experiments.

Let the random variable X denote the number of standard deviations from the mean value of all solutions. In other words $X = \frac{Y - \bar{y}}{\sigma}$ where Y corresponds to a random variable for the convergence ratio y. For y follows a normal distribution, the probability α_0 follows a cumulative normal distribution:

$$\alpha_0 = \Pr(y \le y^*) = \Phi(X^*) \tag{4.22}$$

where X^* is found from the required convergence ratio as

$$X^* = \frac{y^* - \bar{y}}{\sigma} \tag{4.23}$$

Table 4.3: *The Convergence Ratios Expected from Monte-Carlo versus GA results*

Problem Size	MC(500,0) with z=0.5	MC(500,250) with z=0.5	GA(500,250) Obtained
51	3.24	2.95	1.08
76	3.95	3.66	1.19
96	5.71	5.30	1.29
105	7.37	6.79	1.41
202	5.25	5.04	2.28

It follows that

$$\begin{aligned} X^* &= \Phi^{-1}(\alpha_0) \\ &= \Phi^{-1}(1 - 0.5^{1/nt})) \end{aligned} \tag{4.24}$$

and

$$\begin{aligned} y_0 &= \bar{y} + \sigma X^* \\ &= \bar{y} + \sigma\Phi^{-1}(1 - 0.5^{1/nt}) \end{aligned} \tag{4.25}$$

The above equation can be used for both evaluating the Monte-Carlo method and comparing the results of a randomly-created seed generation to later generations in GA. These equations indicate that if one is looking at a population of 500 independent trials, one can expect, with 50% probability, to find a solution that is 2.99σ away from the mean. Similarly, repeating this experiment 250 times with 500 trials in each, increases the expected deviation to 4.4σ. Table 4.3 presents the expected convergence values for the Monte-Carlo search and compares them to the results of the GA run with 500 individuals and for 250 generations. Note that the evaluated expected values of the best convergence ratios for the seed generation are congruent with the experimental observations presented in Table 4.1. As can be seen from Table 4.3, the GA runs attained results with much larger deviations than those values expected from random search. These results prove that GA results cannot be attributed to chance.

4.3.2 Comparison with Simulated Annnealing

Because *Simulated Annealing* (SA) was successfully applied first to TSP [85], it should be interesting to compare the results of SAs with those of GAs. For this comparison, we used the same mutation scheme which was used before for the GA runs. By tuning algorithm one can get quite impressive results with this method. For example, the optimum solution for P202 was found in 1 hour 20 minutes in one run. The temporal evolution of the minimum energy (cost) is shown in Figure 4-16 for 3

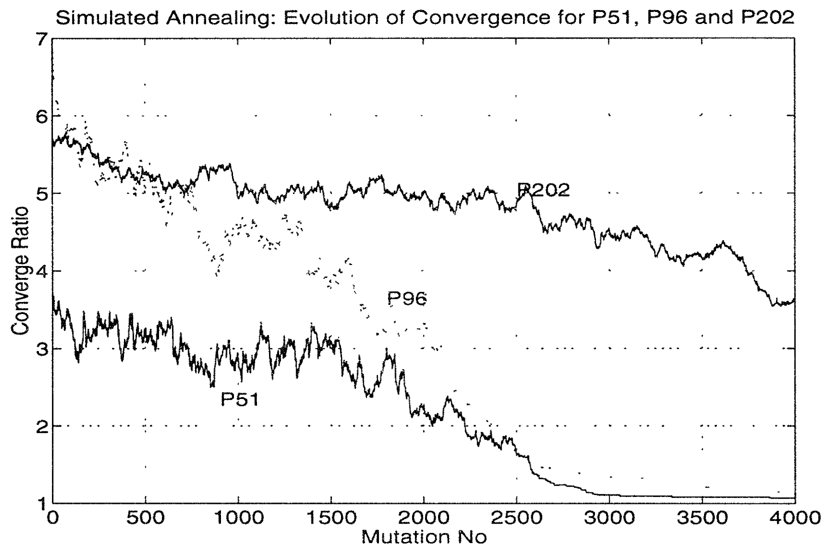


Figure 4-16: *Simulated Annealing Runs for P51, P96 and P202*

Table 4.4: The Results of Simulated Annealing Runs for Test Problems

Problem Size	SimAnneal Convergence
51	1.01
76	1.05
96	1.11
105	1.12
202	1.00

problems P51, P96 and P202. The local maxima indicate the times when an adversary mutation was accepted according to the result of a random-number comparison experiment. Because the temperature is decreased exponentially according to Boltzmann equation, the probability of accepting an adversary mutation decreases quickly. Excursions that prevent being trapped in local optima gradually die out and the system generally converges to very good local solutions. The following table (table 4.4) presents the best results obtained with simulated annealing algorithm, by tuning algorithm separately for each case.

Despite these encouraging results, simulated annealing might turn out to be computationally expensive once strict convergence criteria are applied, because in this case one has to tune the algorithm by changing the parameters or the mutation scheme such that the most effective annealing schedule is obtained. For there is no known clues as to what constitutes the best annealing schedule, the tuning may require performing many experiments separately for each problem. On the other hand, one could be content with suboptimal solutions and may prefer not to tune the algorithm for each version of a given problem but tune it only for a single version and use the same annealing schedule for the rest. The

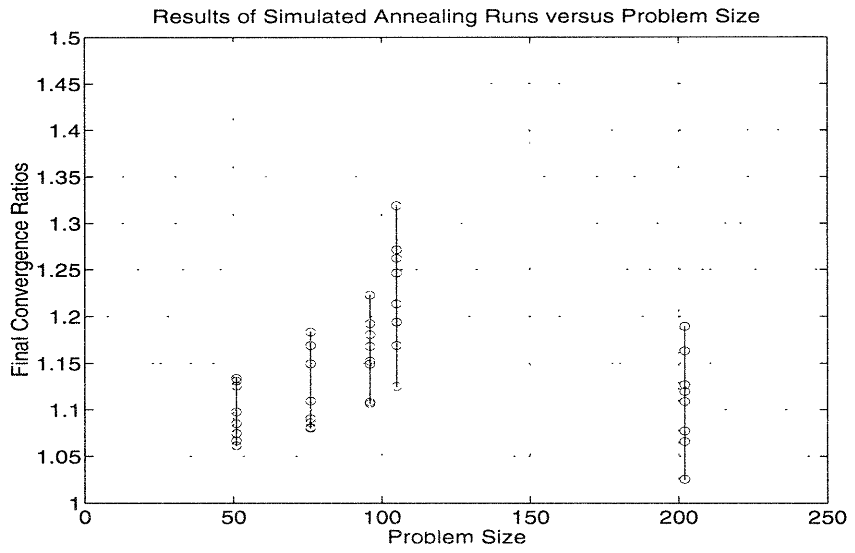


Figure 4-17: *Results of Simulated Annealing Runs versus Problem Size. Annealing Schedule is the same in all these runs.*

following figure shows the results when we use the algorithm with the same parameters and therefore the same annealing schedule. For this parameters, we have chosen the annealing schedule that worked best for P96, which proved to be the hardest to improve. As can be seen from the figure, we now can obtain results that are slightly worse those presented in Table 4.4, but they may still be good enough for most purposes.

In effect, simulated annealing was able to find solutions quite close to optimal and within a much shorter time period compared to GAs. Runtimes of Simulated Annealing results were found to be linearly proportional to problem size. On the other hand simulated annealing is claimed not to do well on most other problems. In fact, if mutation is computationally expensive, simulated annealing algorithm may not be a good choice.

4.3.3 Comparison to Artificial Neural Networks

Another method that is often used in optimization problems involves artificial neural networks. A study by Hopfield [75] demonstrates that good solutions to TSP can be obtained with appropriately designed neural networks. Hopfield uses n^2 neurons for a problem with n nodes and expresses the tour distance as the energy to be minimized. Nevertheless, the solutions found by Hopfield are not of high quality as those found by GA and simulated annealing algorithms. Hopfield indicates a solution for a TSP problem involving 30 cities is obtained easily with his network; and the cost of this solution is only %40 larger than that of the best solution. On the other hand, our experiments with GA applied to the same problem obtained a convergence ratio of 1.02 within a reasonable computation time of around 50

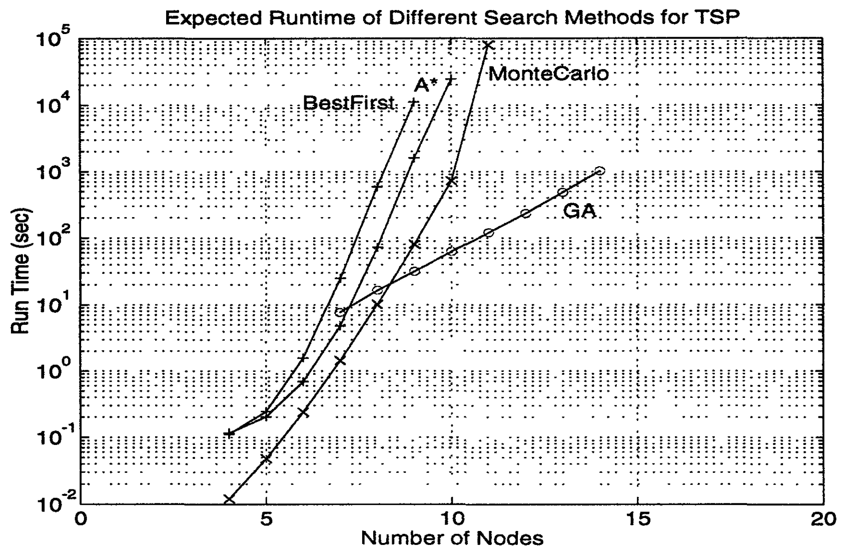


Figure 4-18: *Comparison of Runtimes for Different Search Procedures*

minutes in a SunSPARC-1 workstation. A further disadvantage associated with neural network method lies in the fact that one should build a new architecture for each new problem.

4.3.4 Comparison to Classical Search Methods

We also tested the classic search methods in TSP problems by implementing the best-first and A* state search running them for a set of randomly-created problems. We observed that as the problem size increases these classic search methods suffer from computational complexity. In fact, before one runs out of patience, the programs based on these methods exhaust any reasonable dynamic memory limit and collapse. These programs are required to keep track of a large queue which grows exponentially as the problem size increases. Thus, a significant portion of the runtime in these problems are wasted in rearranging the memory. This inefficiency arises because we use a a simple heuristic that does not help much in branching and bounding of new solutions. Nonetheless, unless a limit on the queue size is set, the queue will grow quickly with increasing problem size and exhaust the physical memory, making the search methods unsuitable to handle larger problems even if the time resource were not a factor.

Figure 4-18 shows how these search procedures compare to each other with respect to runtime over a set of random problems. We assumed the perfect answer is required for both GAs and Monte-Carlo as well as for other methods.

Overall, Figure 4-18 suggests that GAs compare favorably with other methods. Whereas in practice all of these methods have exponential complexity for obtaining the optimum value, the growth coefficient of the exponent is less for GAs unless the problem size is extremely small.

4.4 Summary and Conclusions

In this chapter, we examined the complexity of two well-known deterministic path planning problems, namely the shortest path problem and the TSP. Then, we applied the GAs to the TSP. Our results demonstrate that GAs are quite successful for optimization problems, especially when finding the perfect solution is not required of them. In solving the TSP without relying on problem-specific information, GAs were outperformed in our experiments only by the simulated annealing algorithm. The results in the literature [75] indicate that another paradigm modelled after nature, neural networks, can produce good solutions to TSP, but not as efficiently as GAs. While we found that the classical search methods are quite inefficient in handling the TSP, all of these methods suffer from combinatorial explosion once the optimal solution is required. Even in this case, however, GAs can find the solution faster, and therefore they should be preferred over other methods excluding simulated annealing.

We have also shown that the GAs can also be used as a reliable approximation scheme. Although there can be given no guarantees, the expected value of the time to converge to a desired solution is polynomial with the problem size for polynomial-time fitness evaluation. Further, this time is proportional with the logarithmic error term: $\log \frac{c}{\epsilon}$.

It follows that GAs are very efficient in finding suboptimal solutions, yet these can also be obtained via heuristic techniques with often significantly less computation. The efficient utilization area of GAs may be limited to those cases requiring solutions so nearly optimal that they cannot be provided by heuristic methods. Besides, it is often possible to find other algorithms that can outperform GAs in any given problem; for the TSP, simulated annealing is one of them. Indeed, excellent heuristics exist for various versions of the TSP. Yet, these results should not discourage the users of GAs. Applying GAs in a special problem requires only the coding of the special methods for the genetic operators. In many cases, heuristic information can be directly sumperimposed onto these operators or to the selection mechanism. Thus, it is often possible to improve the performance of a GA by utilizing the context-dependent information. Finally, GAs have some special advantages that can be useful in problems with incomplete, inaccurate information. Because GAs are probabilistic algorithms, they can achieve convergence even under stochastic data. Note that this reasoning is also valid for the simulated annealing method (SA). GAs, however, may prove more robust than the SA for they keep track of a set of solutions in contrast to a single one by simulated annealing. GAs may also be less expensive than the SA because they normally converge to an acceptable solution with 3000-4000 fitness evaluations, whereas the SA requires around 100,000 evaluations. For the TSP, the SA was significantly faster than the GA, but this can be attributed to a cheap (linear time) fitness evaluation and and a significant overhead in GA. In problems with expensive fitness evaluation, the GA may outperform the SA.

Finally, we note that the GA optimization technique can be easily extended to other planning problems that may be quite distinct from TSP. Well-optimized plans can be developed easily once an easy-to-compute objective function is described and special methods for the genetic operators are implemented. If the goal of finding the optimum plan is computationally intractable, we may have to settle for an approximate answer. When a planning model involves many decision variables, the problem becomes harder since the process may now require the consideration of possible trade-offs between distinct variables. Yet, this problem too can be solved in a manner similar to the single-variable optimization problem by employing an objective function that contains all the decision variables with properly chosen weights. If one cannot decide on what values are appropriate for these weights, one can alternatively opt for a solution well-optimized with respect to a single measure, yet satisficing with respect to others. In these problems, the set of acceptable-solutions are said to be Pareto-optimal, meaning that they do not perform any worse than other unacceptable solutions while performing better than those according to at least a single measure.

In sum, these results indicate that GAs converge to near-optimal solutions fast, but fall short of being a complete or the most efficent solution method. At times, the other methods may outperform the GAs. Despite these shortcomings, GAs produce satisfactory performance for the TSP, even though this is cited as one of the most difficult problems for GAs. Most importantly, because GAs do not require heuristics and can converge to a desired solution under inexact information, satisfactory performance can be expected for their applications in problems with uncertain data.

Chapter 5

Stochastic Planning Problems

Most practical plan problems can be addressed only through incorporation of nondeterministic models. Although there is a formidable amount of literature on deterministic planning problems, relatively little attention has been devoted to their nondeterministic versions. Further, the existing studies on these probabilistic problems usually disagree on the assumptions and the models they use, as they generally emphasize a different nondeterministic aspect of the given problem.

In this chapter, we examine the complexity of the probabilistic planning problems under various optimization criteria and under various assumptions (*Section 1* and *2*). We indicate the reasons why the complexity results may change drastically as soon the objective functions do not satisfy certain criteria. Then, we concentrate on the stochastic path planning problems where the edge costs are described by probability distributions (*Section 3*). We examine different solution methods for the stochastic longest and shortest path problems under safety-oriented optimization criteria (*Section 4* and *5*). We present in *Section 6* the results of applying the GAs for a special version of the stochastic shortest path problem where we are concerned only with Hamiltonian paths.

5.1 Probabilistic Models with Additive Objective Function

The theory of dynamic programming can easily be extended to nondeterministic models. Further, when an additive objective (cost) function is used, the principle of optimality is still valid. Recall from the previous chapter that a path planning problem is efficiently solved if it is a closed semiring.

Stochastic Dynamic Programming

The basic problem of dynamic programming is stated as follows [15] :

Given a dynamic system of the form

$$x_{k+1} = f_k(x_k, u_k, w_k), \quad k = 0, 1, \ldots, N-1 \tag{5.1}$$

where

k indexes discrete time

x_k is the state of the system and summarizes past information relevant for future optimization

u_k is the decision variable to be selected at time k with knowledge of the state x_k

w_k is a random parameter (also called disturbance or noise)

N is the horizon or number of times decision is made

and given an additive cost function $g_k(x_k, u_k, w_k)$ such that the total cost along any system sample trajectory is

$$g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) \tag{5.2}$$

and given an initial state x_0, *find an admissible policy* $\pi = \mu_0, \mu_1, \ldots, \mu_{N-1}$ *that minimizes the cost functional*

$$J_\pi(x_0) = E_{w_k} \left\{ g_N(X_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k) \right\} \tag{5.3}$$

It is found that under these conditions the solution (the optimal policy J^* is given as:

$$J_N^*(x_N) = g_N(x_N), \tag{5.4}$$

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k) w_k} E\left\{ g_k(x_k, u_k, w_k) + J_{k+1}^*[f_k(x_k, u_k, w_k)] \right\} \tag{5.5}$$

where $k = 0, 1, \ldots, N-1$.

Dynamic programming is particularly efficient when the cost function is monotonically increasing and additive. For example, in some applications the cost of reaching a state is only a function of that state. Then, the cost function we want to optimize can be expressed as

$$E_k = E_{k-1} + p_0 p_1 p_2 \ldots p_k D_k \tag{5.6}$$

where D_k is the reward (or penalty) attained at state k. In some applications it may be more natural to model the cost of a transition from state i to state j as a scalar d_{ij} that also depends on j. Bertsekas suggests that [16] D_k can be viewed as an expected cost of arriving that state, given by $D_k = \sum_{j=1}^{n} p_{ij} d_{ij}$. A particular version of the application of dynamic programming with scalar state cost values is known as *policy iteration.* The policy iteration algorithm is guaranteed to converge in a number of iterations polynomial in the cardinality of state space. Dean and Kaelbling *et. al.* [37] take advantage of this formulation in developing a robot planner for a stochastic domain.

Further, with this model, the problem becomes very easy to solve for its TSP-like constrained version. In this case, the optimal solution is arrived through by comprising a path from the nodes ordered according to decreasing fraction $\frac{p_i D_i}{(1-p_i)}$. The optimality of this solution is shown through a simple interchange argument [15]:

Let i and j be two adjacent nodes in an optimally ordered solution $L = (i_0, i_1, \ldots, i_{k-1}, i, j, i_{k+2}, \ldots, i_N)$. Consider the solution $L' = (i_0, \ldots, i_k - 1, j, i, i_{k+2}, \ldots, i_N)$ obtained from L by interchanging the order of the kth and $(k+1)$th nodes i and j. We compare the expected rewards of L and L'. We have

E(reward of L)=E(reward of $i_0 .... i_{k-1}$) $+p_{i_0} ... p_{i_{k-1}} (p_i D_i + p_i p_j D_j) + p_{i_0} ... p_{i_{k-1}} p_i p_j$E(reward of $i_{k+2} ... i_N$)

E(reward of L')=E(reward of $i_0 .... i_{k-1}$) $+p_{i_0} ... p_{i_{k-1}} (p_j D_j + p_j p_i D_i) + p_{i_0} ... p_{i_{k-1}} p_j p_i$E(reward of $i_{k+2} ... i_N$)

The difference is

$$E(L) - E(L') = p_{i_0} ... p_{i_{k-1}} (p_i D_i + p_i p_j D_j - p_j D_j - p_j p_i D_i) \tag{5.7}$$

Because of the condition that $\frac{p_i D_i}{(1-p_i)} \geq \frac{p_j D_j}{(1-p_j)}$, the difference term is always positive, i.e.,

$$(p_i D_i + p_i p_j D_j - p_j D_j - p_j p_i D_i) \geq 0 \tag{5.8}$$

It follows that the expected reward of L is always greater than an alternative solution L' obtained by an interchange. Hence L is an optimal solution.

5.2 Probabilistic Models with Unreliable Edge Travel

In general, however, stochastic planning problems may be computationally expensive. Consider a simple modelling of the nondeterministic problems that is developed by ascribing a unique cost and probability value to each edge in a given graph. The probability of an edge can be considered as its reliability, i.e. the probability of the event of having successfully traversed the given edge. The complementary probability value therefore implies that the given edge cannot be traversed: it might be blocked, non-existent or failed in some another way. This model is especially useful in describing contingent events. It is also relatively simpler then another commonly used nondeterministic model assigning a probability distribution to the cost of each edge (stochastic edge costs). Further, we note that the above mentioned model with stochastic edge costs can be converted into this model easily. Assume that we are given an edge e_{ij} connecting the node i with the node j having a cost distribution which is characterized by the vector $D = \{d_{ij_1}, d_{ij_2}, \ldots, d_{ij_r}\}$ with the corresponding probability values

$P = \{p_{ij_1}, p_{ij_2}, \ldots, p_{ij_r}\}$. By replacing the given edge e_{ij} with r new edges e_{ij_k} each having an associated cost value d_{ij_k} and a probability value p_{ij_k}, we obtain a new graph wherein edges have a certain cost and a probability value of being successfully traversed. Under the condition that a path uses only one of the of r edges with the same arrival and departure nodes, this model will be equivalent to the above described model with stochastic edge costs.

A variety of optimization measures can be associated with paths within the context of this model. For example, we may want to optimize the path probability or expected cost beside the usual optimization measures cost or utility. We might also want to optimize on both probability and cost at the same time or we might try to optimize on one measure while ensuring a bound on another one.

Let us start by examining the complexity of **maximizing the probability** of a path. Maximizing the probability of a path can be solved with transformation through the equivalent problem

$$\text{minimize} \sum -\log p_{ij}$$

with the same techniques that find the minimum cost path. This scheme is used in the Viterbi decoder for convolutionally coded data and in speech recognition. It also follows from the same argument that minimizing the probability is NP-hard.

In another type of optimization, we might want to **minimize cost and maximize probability**. Unfortunately, the problem now turns out be NP-hard for general cost and probability values (if all edges have uniform cost or uniform probability, it would be easy to solve). The recognition version of this problem is proven to be NP-complete [52]:

Given a pair of source and sink nodes in a graph $G(V, E)$ where each edge has an associated cost value d_{ij} and an associated weight w_{ij}, is there a path with total cost less than $C_{\max}$ and a total weight less than $W_{\max}$?

We might also want to **minimize the expected cost** or to **maximize the expected utility**. The expected utility is the expected value of the utility of a given path: $EU(P) = \Pr(P) * U(P) = (\prod p_{ij}) f(C(P))$. The expected cost is the expected value of the cost of a given path: $EC(P) = \Pr(P) * C(P)$. (In shortest path problems, a more sensible alternative could be to minimize the average expected cost defined as the ratio of the expected cost to the number of edges that comprise the given path).

Under these criteria, the objective functions are neither additive nor monotonically increasing. The recurrence relation for the expected utility, for example, becomes

$$EU_k = p_k EU_{k-1} + p_0 p_1 p_2 \ldots p_k U(d_{k-1,k}) \tag{5.9}$$

It follows from this equation that

$$EU(L_1 \odot L_2) \neq EU(L_1) + EU(L_2)$$

Therefore the principle of suboptimality is not valid, and the dynamic programming or branch & bound search methods cannot narrow down the search space efficiently. Under special conditions, these problems can be solved with polynomial time algorithms. For example, if the utility function is of the exponential form, the objective function becomes separable under multiplication and we can find the optimal solution with the standard dynamic programming. Also, under uniform edge costs or uniform edge probabilities, these problems can be solved in polynomial time. In short, when the path concatenation operator $\odot$ distributes over the summary operator (in this case the expected cost), we can use dynamic programming principle which exploits this fact to eliminate from the consideration any path segment that cannot be a part of the optimum path. Thus, Dijkstra's algorithm can be adopted for the linear or exponential utility function, under independence assumption, but not for a general utility function. Nevertheless, under general conditions, these problems are NP-hard.

5.3 Stochastic Path Planning Models with Stochastic Edge Costs

Among many different variations on probabilistic path planning problems, the ones that has attracted most attention are the stochastic shortest path problem (SSPP) and stochastic longest path problem (SLPP) in a graph wherein the edge costs are specified as stochastic random variables. In these models, each edge e_{ij} has an associated cost distribution function $f(d)$. Because the individual edge costs are stochastic, the cumulative cost of a path becomes a random variable. Note that even if the edge costs are independent random variables, the path costs are, in general, dependent owing to shared edges.

These problems are often studied with the following objective function: From the set of all paths Π, find the path $P^* \in \Pi$ having an expected cost $E(L^*)$ less (or larger) than the expected costs of all other paths. Objective functions of more general nature can also be defined. For example, we can ask for a path P^* such that the probability of its cost L^* exceeding a threshold $L_{\max}$ is less than a specified probability value α_0, i.e. :

$$\Pr(L^* \geq L_{\max}) < \alpha_0 \tag{5.10}$$

A similarly encompassing objective function was considered by Frank [46]. He considered the determination of the cumulative probability distribution function PDF of the cost of the shortest path P^* from s to t

$$PDF(l) = \Pr(L^* < l) \tag{5.11}$$

which can be found from the probability density function $pdf(x_1, \ldots, x_q)$ (where $q = |\Pi|$) as

$$PDF(l) = 1 - \int_l^\infty \cdots \int_l^\infty p(x_1, \ldots, x_q) dx_1 \cdots dx_q \tag{5.12}$$

The *pdf* can be expressed as a vector and can be discretized given discrete cost values and their associated probabilities. The evaluation of this multiple integral, however, is an extremely arduous task, as it covers the whole state space.

In general, safety-oriented optimization criteria can be described by considering the reliabilities. Evaluating the reliability of a plan involves the computation of the probability for achieving a specified success predicate. Three distinct criteria can be associated with this probability for reliability-based optimization:

- *Safety-first:* Find the solution with the *maximum* success probability.
- *Strict Safety-first:* Find a solution with a success probability larger than a prescribed value.
- *Safety-fixed:* Find the solution with the *maximum* utility **and** a success probability larger than a prescribed value.

Note that, in analyzing stochastic problems, it is usual to ask for those solutions with maximal *expected utility* which is calculated normally as the probability of a given solution multiplied by its associated utility. This criteria, derived from the assumption of rationality, is well-suited for most problems of stochastic optimization. Yet, when there is a particular concern about risk, it could be more appropriate to employ criteria which treat risk in a more obvious way by concentrating on probabilities. These criteria are especially of interest when the problem being examined can lead to dire consequences for in this case one prefers plans whose failure probability is minimal or bounded.

5.4 Methods for the Stochastic Longest Path Problem

The stochastic longest path problem is of particular interest to project planners. The overall goal of a project planner is to minimize the cost of a project by allocating the resources efficiently. Often, the most restrictive resource is time and therefore one wants to minimize the project duration by limiting the longest path. This goal requires first the determination of the longest path in a project.

5.4.1 Discretization technique

Let us assume that the **pdf** of an activity is approximated by a discretization scheme: An activity duration d can take n different values, where $d \in D$ given $D = \{d_1, d_2 \ldots d_n\}$ and each value is associated by a unique probability. The calculation of **pdf** for project duration can be accomplished by considering the realization of every alternative individual task duration value. This approach unfortunately has an exponential complexity. In a given path with m activities, with each **pdf** being represented by n distinct duration values, we would need n^m calculations.

5.4.2 Analytical technique

Assume the activity durations are independent of each other. Then, we can use the approach suggested by Martin [119]: The project network is first reduced to a series-parallel form. The distribution function of parallel activities is derived by multiplication of the individual activity distribution functions. For activities in series, convolution of the functions is employed. By proceeding systematically in this way, the network is eventually reduced to a single arc, whose distribution function is that of the project. To employ the method, the distribution functions of the activities have to be approximated by polynomials. It is noted that the amount of computation can be reduced if, instead of trying to derive the project distribution function, one merely attempts to put bounds on it. One ambiguity with this approach is its handling of N type subgraphs which can not be reduced to either parallel or series form.

5.4.3 Stochastic PERT

The most well-known method of determining the longest path in a stochastic models is incorporation of the probabilistic task durations to the PERT paradigm. The stochastic PERT makes the following assumptions:

- The activity durations are modelled via certain probability distributions. The most preferred distribution is the beta distribution with the parameters being computed from the most-pessimistic, most-optimistic and most likely times.
- The activity durations of independent of each other and one can use the central limit theorem for finding the **pdf** of a path. The result is a Gaussian distribution with a mean value equal to the sum of the mean values of individual activity duration mean values and with a variance equal to the sum of the variances of of the tasks along the path. Note that the use of central limit theorem relies on the assumption that there is a sufficiently large number of activities in each path.
- After finding the probabilistic distributions for each path, PERT assumes that the path with the longest completion time will always be the critical path. Then, the probability of a given completion time for a network is given as

$$Pr(L^* \leq L_{max}) = \int^{L_{max}} N_\pi(t)dt \tag{5.13}$$

where N_π is the normal distribution of the critical path π.

Although most PERT programs make use of this final assumption, it is easy to see that the assumption is quite misleading as pointed out by several researchers [63, 7]. A path with smaller mean value than that of the critical path but with a larger variance can turn out to be more limiting in determining the probability of project completion time. Therefore, it is meaningless to talk about a single critical path and one should take into account the inherent uncertainty in all paths.

It is also easy to show that the PERT method is optimistic: The true project duration is given as $E[max_p(L_p)]$, the PERT gives as $max_p(E[L_p]) = E[L_\pi]$ (where π is the critical path). Since $max_p(L_p) \geq L_\pi$, it follows that true mean is always larger than the PERT mean.

5.4.4 Other Methods

Although the PERT method have been applied for most applications, it is simply incorrect to assume that the statistics of the project would be influenced only by the critical path. For correct calculation of overall **pdf** of the project completion time, there have been several suggestions besides the one that is based on the calculation of joint probability of all paths, which is computationally expensive. One is to consider only a few near-critical paths besides the critical path. Another is to use the Monte-Carlo techniques to sample the **pdf** of the project completion time. A criticality index for activities based on their probability of being on a critical path can also be used as guidance [63]. For a complete examination of a probabilistic network with many paths, a multi-variate statistical examination that would specify the correlation among different paths is also suggested.

Another method, developed by Elmagrahby [119], is similar to dynamic programming. The method involves calculating, node by node, the expected value of the maximum path length to each node. Let this be f_j, for node j, where $f_1 = 0$. Then f_j is defined recursively as follows:

$$f_j = E[max_i(f_i + d_{ij}^{(k)})] \tag{5.14}$$

where nodes i are the immediate predecessor nodes of j, $E[]$ denotes the expected value and $d_{ij}^{(k)}$ denotes the kth combination of duration values for the activity $i - j$. The value of f_n for the final node n is the estimate of expected project duration. This value can be shown to be optimistically biased but it is a big improvement on the PERT estimates. This method in contrast to the discretization technique does not become more complicated as the network size increases, and this is its greatest virtue [119].

5.5 Methods for the Stochastic Shortest Path Problem

The methods for determining the stochastic shortest path is, in general, parallel to those methods used for determination of the longest path. Frank discusses [46] various methods of determination of the joint probability distribution function integral (appearing in Equation 5.12), among them Monte-Carlo methods, nonparametric and parametric analysis. Through examples, he shows that the cumulative *PDF* can be approximated well with the normal distribution function. Alternative methods for the stochastic shortest path problem have also been suggested by various researchers. Martin [96] presents a technique for computing the distribution function of the length of the shortest (s, t) path that can be readily adapted for the shortest path, in a directed, acyclic network, whose arc lengths are independent and have a finite range. Hayhurst and Schier [71] use a factoring approach for evaluating the multiple integral appearing above. Mirchandani and Soroush [99] proposed a method of solution that completely avoids the use of multiple integrals. They assume that the edge costs are independent and that comparison between different paths is performed according to some utility function. Their method relies on eliminating from consideration any path segment that is *dominated* (i.e. at least another path segment exists which is permanently preferred to it). They give efficient solution algorithms for the case of linear and exponential utility functions. These functions are respectively additive and multiplicative separable under their statistical independence assumption. They also present an algorithm for the quadratic utility case, for which in efficient algorithm cannot be established. They claim that even though the worst case complexity of this algorithm is exponential in problem size, its running time is reasonable in practice.

The SSPP has also been presented within the context of different models, such as stochastic linear programming, stochastic dynamic programming, and general Markov chains. Corea and Kulkarni [29] convert the problem to a discrete time Markov chain (DTMC) with a finite state space and a single absorbing state and an associated cost of zero or one with every transition. The states of this DTMC can be ordered such that its transition probability matrix is strictly upper triangular. This structure of the transition probability matrix enables them to develop simple recursive algorithms for the exact computation of the distribution function and moments of the total costs incurred until absorption. Unfortunately, the size of the DTMC grows exponentially with the size of the graph and the size of the support of the random variables. Bertsekas and Tsitsiklis [16] study a most general stochastic shortest path model. In their model, it is asked to select a probability distribution over the set of successor nodes so as to reach a certain destination node with minimum expected cost. Their model too can be viewed as a general Markov decision problem. Owing to the difficulties encountered in the general case, some researchers have focused attention on Monte-Carlo methods for this problem (see [46] or [10]).

Most studies on the SSPP concentrate on finding a solution with the minimum expected cost. When the solutions have large variance or when there is a particular concern about risk, it is generally more appropriate to use safety oriented criteria. For example, we might be concerned with guaranteeing that extremely *costly* paths are avoided as often as possible. Therefore, instead of asking for the path with the minimum expected cost, we would like to find a path with the minimum probability of exceeding the cost limit, i.e., we would like to *minimize* $\Pr(L^* \geq L_{\max})$. A mirror (equivalent) version of this condition is to *maximize* the probability of not exceeding the cost limit, i.e., we would like to maximize $\Pr(L^* < L_{\max})$. Below, we examine various approaches suggested for this purpose:

5.5.1 Approximation

Under certain conditions, the following approximation can be used: Assume that the edge costs are mutually independent. Under certain conditions, sums of the random variables are approximately normally distributed. Then, we can approximate the distribution of the cost of a path P_i with a normal distribution function $\Phi(\mu, \sigma)$. Because $\Phi(\mu, \sigma)$ is a monotonically increasing function, we only need to perform the following test

$$\frac{L_{\max} - \mu_i}{\sigma_i} < \frac{L_{\max} - \mu_j}{\sigma_j} \tag{5.15}$$

for choosing between two distinct candidate paths P_i and P_j.

Note that this approach is closely related to the PERT approach.

5.5.2 Dynamic Programming

It is easy to observe that we can use the dynamic programming whenever the edge costs are independent and the path costs can be approximated with a monotonically increasing function. The techniques of dynamic programming have been extended to the general cases by Kao et. al. [81]. Their preference order dynamic programming is stated as follows: For an optimal path yielding the maximum probability that τ or less cost will be expended $f_k(i, S_k)$, let the distribution function of the corresponding total cost be denoted by $G_k(i, S_k)$. Note that

$$f_k(i, S_k) = [G_k(i, S_k)(\tau)] \quad \forall i, k \in S \tag{5.16}$$

By the principle of optimality

$$G_k(i, S_k) = \perp_{j \in S_k} \{F_{ij} \odot G(j, S_k - j)\} \quad i = 1, 2, \ldots, n \tag{5.17}$$

with $G_0(i, \emptyset) = F_{i0}, i = \{1, 2, ..., n\}$ and $k \in \{1, 2, \ldots, n-1\}$ and $G_n(0, S_n) = \perp_{j \in S_n} \{F_0 \odot G(j, S_n - j)\}$ and $\odot$, the compositor operator [81]. The preference order operator $\perp$ is a mapping which chooses the i^*th order distribution function Ω_{i^*} from the set of distribution functions $\{\Omega_1, \Omega_2, \ldots, \Omega_l\}$ via the criterion

$$[\Omega_{i^*} Z](\tau) \geq [\Omega_i \odot Z](\tau) \quad \forall i \in \{1, 2, \ldots, l\} \tag{5.18}$$

with $\Omega_i, Z \in \Upsilon$ -the set of all distribution functions.

5.5.3 Monte-Carlo Approach

Frank suggests [46] the use of a statistical test for the selection of a path with the associated probability of satisfying the success criteria larger than a preset threshold. Notice that in any statistical test, two types of mistakes can be made. First, one can accept the hypothesis even if it were wrong due to the misleading sample (this is a type II error) or one can reject the hypothesis even if it were correct (this is a type I error). The probability of making a wrong decision can be minimized by a special test based on Neyman-Pearson theory [135]. The test described below uniformly most powerful; in other words, it is optimum in the sense that the probability of making a type I error is less than a fixed size β and the probability of making a type II error is minimum. Nevertheless, the test does not give a bound on the magnitude of the probability of making type II error. (If we want to minimize the type II error, we can formulate the dual test, however, in that case we cannot bound the type I error; there is a tradeoff between these two types of errors). The test is as follows:

Let H_1 stand for the hypothesis that the probability α is equal to or larger than a given threshold value α_0. Let H_2 stand for the alternative (null) hypothesis that $\alpha < \alpha_0$. Then, if k and γ are predetermined constants, we perform the following [46] :

Reject hypothesis H_1 if $m > k$
Reject hypothesis H_1 with probability γ if $m = k$
Accept hypothesis H_1 if $m < k$

The constants k and γ are determined from

$$\sum_{i=k+1}^{n} \binom{n}{i} (1-\alpha_0) i \alpha_0^{n-i} + \gamma \binom{n}{k} (1-\alpha_0)^k \alpha_0^{n-k} = \beta \tag{5.19}$$

where β is the probability of Type I error. Note that in this equation the left hand side is the probability $\Pr(X > k) + \gamma \Pr(X = k)$, where X is a binomial random variable with parameters $1 - \alpha_0$ and k.

5.6 Application of GAs on the Stochastic Shortest Hamiltonian Path Problem

The SSP also provides a suitable context for applying the GAs. We apply the GA to a particular case of the SPP in which all paths have be Hamiltonian in order to be feasible. We assume that the individual edge costs are specified with certain probability distribution functions. We also assume that these edge costs are discretized with a sampling process before the GA is started. Let k denote the size of sample data for each edge. For a path with m edges, therefore, we have k^m different combinations for the total cost value of the path. Let n be the number of these combinations exceeding the preset cost limit. Then, the probability of the path exceeding the total cost limit is simply given by the fraction:

$$\alpha_{ki} = \Pr(L_i \geq L_{\max}) \approx \frac{n}{k^m} \tag{5.20}$$

As the discretization becomes finer, this probability value will approach to the theoretical value which can be determined by evaluating the integral. The problem, however, is not how fine the discretization is but rather how large the number of combinations is. Even with a moderate number of discrete data and a moderate number of edges, we will have a large number of possible combinations (for example $k = 4$, $m = 10$ leads to 4^{10} combinations). Therefore, the only feasible alternative is to use a sampling process that draws randomly a value for the cost of each edge and uses this in evaluating the total cost value of the path. As the sampling size n is increased, the estimate of the probability must approach the actual probability value:

$$\lim_{n \to k^m} \hat{\alpha_k}(n) = \alpha_k \tag{5.21}$$

It must be emphasized that the simulation approach does not yield a lower or upper bound on the magnitude of the probability α. In other words, $\hat{\alpha} = \alpha + \epsilon$.

Nevertheless, it is possible to obtain the near-optimal solutions through GAs even *under noise.* Indeed, the GA can be considered as a feature extractor that uses statistical tests to classify probabilistic patterns. The question on whether the GA would converge to the vicinity of the optimal solutions or if it would diverge because of the statistical errors introduced through the sampling of the fitness function can be answered through theory or tests.

5.6.1 Theory

Recall from Chapter 4 that the expected convergence ratio in a GA run can be given by

$$y - 1 = (y_0 - 1)e^{-\alpha t^a} \tag{5.22}$$

where α and a are constants, t is the generation number and y is the convergence ratio of the designated best solution in generation t. Then, the expected convergence rate is given as:

$$\frac{d}{dt}y = (y_0 - 1)\alpha a t^{a-1} e^{-\alpha t^a} = \alpha a t^{a-1}(y - 1) \tag{5.23}$$

We can add a noise term for the cases in which the fitness function is being sampled and therefore contains some statistical noise. Given that the observed value of the convergence ratio for a solution i is

$$y_o(i) = y(i)(1 + \epsilon_i(y(i))) \tag{5.24}$$

and noting that in an unbiased sampling the error term has zero mean, we obtain the following differential equation for the *expected* convergence rate z under noise:

$$\frac{dz}{dt} = \alpha a t^{a-1}(z - 1) + z\frac{d\epsilon(z)}{dt} \tag{5.25}$$

The expected convergence ratio under noise can be found from Equation 5.25 once the error term $\epsilon(z)$ is specified as a function of the convergence rate. If we assume that the error term is independent of the expected convergence rate and therefore time, we obtain that the expected result is the same as that of the GA without noise, i.e.

$$z - 1 = (z_0 - 1)e^{-\alpha t^a} \tag{5.26}$$

Our computational experiments with randomness introduced to the fitness function of the TSP (such that $f(P) = g(r)C(P)$ where $g(r)$ is the white noise with a mean equal to 1) indicate that the GA performance could even be improved by rescaling the white noise! This is not surprising since the white noise can be considered as an additional parameter helping to optimize the GA. Recall that the GA chooses the genetic operators and selects the individuals to whom these operators to be applied in a *probabilistic* manner. Since the GA already incorporates statistical tests in the fitness based selection, the introduction of white noise does not affect the GA performance in a major way.

Another plausible assumption is that the magnitude of the error term is inversely proportional to the expected convergence ratio, that is, the error term increases with better convergence. This could be the case when there is no Bayesian updating of the probabilities and a fixed number of measurements are used per sampling. For this case, it is increasingly likely that the GA may be confused as the solutions get closer to each other and therefore the magnitude of the error term ratio increases. (Imagine that we are required to classify a fixed number of measurements obtained from a sampling process into two distinct probability distributions. If the distributions are quite far apart, the probability of error (an incorrect classification) is low. If the solutions are quite close, the probability must be correspondingly high. In a GA run, the solutions are initially distinct but increasingly resemble each other as the GA finds the optimal substructures; hence the error probability increases).

Assuming that $\epsilon(z) = \epsilon_0 z^{-b}$ where b is a constant larger than zero ($b > 0$), leads to the following result:

$$dz = \alpha a t^{a-1}(z-1)dt - b\epsilon_0 z^{-b} dt \tag{5.27}$$

The expected convergence ratio $z(t)$ can be obtained by solving the above differential equation. It is, however, obvious from the above argument that the expected convergence ratio for this specific case will grow slower (recall that $b > 0$) than the case without noise. Further if the error term is sufficiently large (such that the second term is larger than the first term in the RHS of equation 5.27, convergence cannot be achieved at all.

The above arguments demonstrate that convergence of the GAs are particularly robust to errors introduced through sampling. More rigorous models of the performance of GAs under noise may be developed with a Markov process with random drift. The optimal parameters for the population size, the size of sampling etc. can be inferred from the developed model.

5.6.2 Tests with Normal Distribution

Figure 5-1 shows the results of four different GA runs for a specific stochastic problem. In this problem, the graph has 20 nodes all located along a circle and the distances between the nodes (edge costs) are given with a normal distribution with its mean equal to the Euclidean distance and its standard deviation equal to 40% of the mean. Then these edge costs are discretized to 100 different values with corresponding probabilities. (Thus an edge has associated values: $\{(d_1,p_1),(d_2,p_2),\ldots,(d_{100},p_{100})\}$). The maximum cost limit $C_{\max}$ for a path is chosen as the 1.5 times of the cost of the most likely shortest path (a circle). The GA is run for the following purpose: *Find a Hamiltonian path with maximum probability of visiting all nodes under the given fixed budget.*

Fitness function samples 100 times the cost of a path by randomly selecting an edge cost value among given sets. The results – shown in Figure 5-1 – indicate that there is a wide deviation between different GA runs because of the noise inherent in the sampling process. In all ten cases the GA found increasingly improved on solutions but in only one case it was able to find the optimum solution.

Noise term inherent in sampling can be reduced by increasing the number of sampling data. However, there is a tradeoff between the error and the computation time. Figure 5-2 shows the GA runs for the same problem with the number of sampling being increased twice to 200. As can be seen from Figure 5-2, the GA run results are more smooth and have lower deviations.

Figure 5-3 shows the GA runs when the sample size increased to 400 and
Figure 5-4 shows the GA runs when the sample size increased to 800.

It can be seen from these figures that the expected convergence rate does not significantly improve

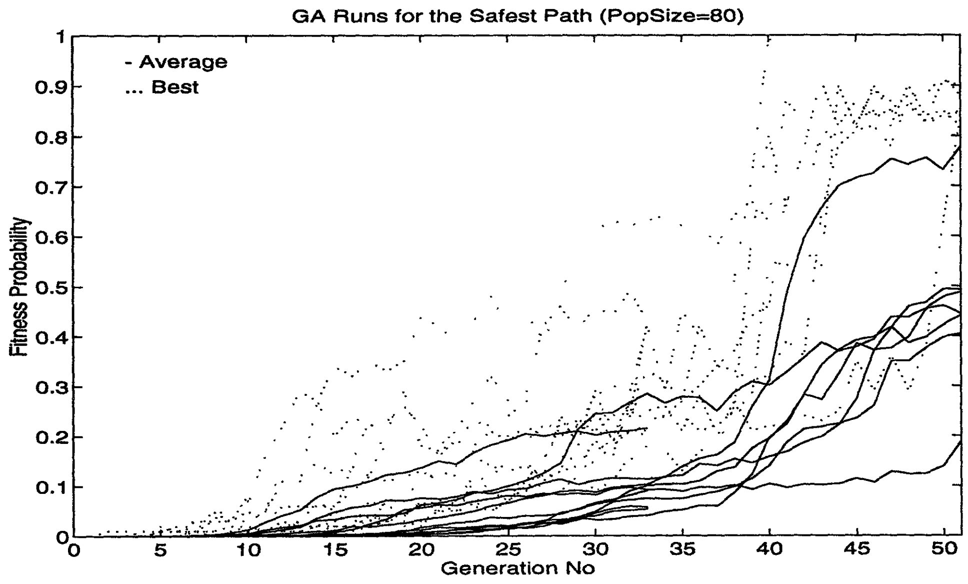


Figure 5-1: *GA runs with 100 times sampled fitness for the safest path in a graph with 20 nodes (PopSize=80)*

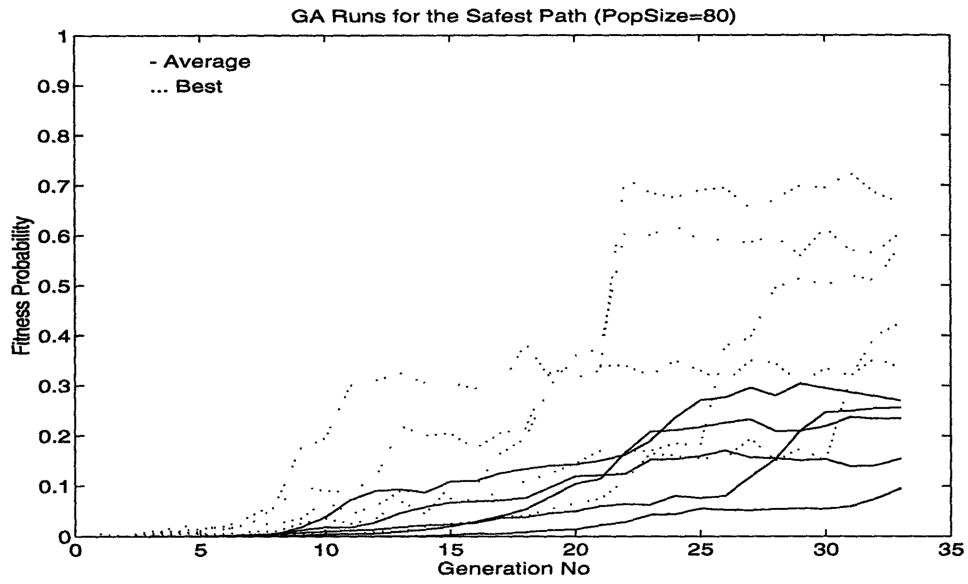


Figure 5-2: *GA runs with 200 times sampled fitness for the safest path in a graph with 20 nodes (PopSize=80)*

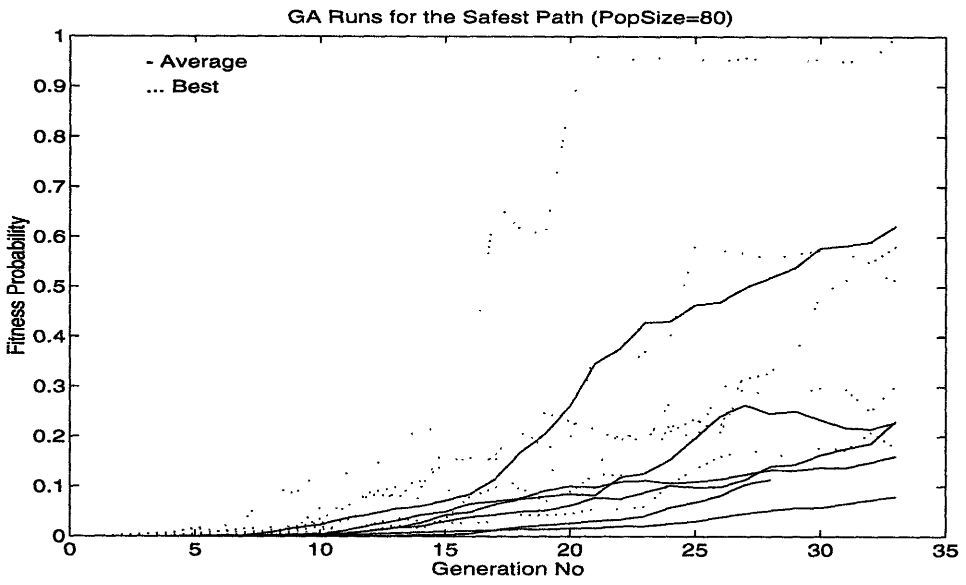


Figure 5-3: *GA runs with 400 times sampled fitness for the safest path in a graph with 20 nodes (PopSize=80)*

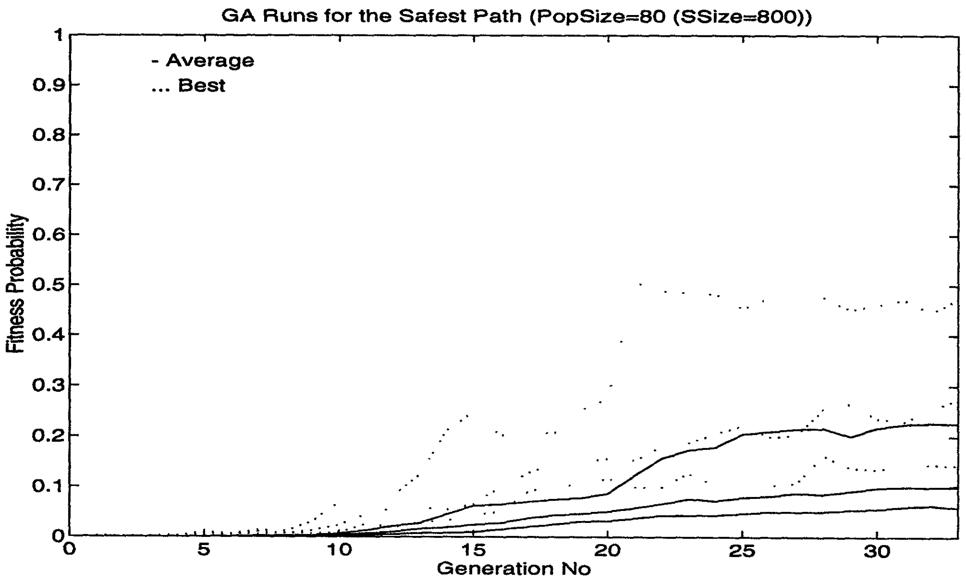


Figure 5-4: *GA runs with 800 times sampled fitness for the safest path in a graph with 20 nodes (PopSize=80)*

with the increasing number of measurements in the sampling process. While the average runtime increases linearly proportional with the increasing sample size, the expected convergence rate increases only marginally, changes generally remaining within the standard deviation of different runs.

In summary, we observed that the GAs can converge to the optimal solutions in polynomial time, however **1)** there are no guarantees in convergence time (we only know that the expected convergence time is of the form $e^{ct^{a}}$) and **2)** using simulation for fitness cannot guarantee a lower or an upper bound and **3)** a low number of measurements is usually enough to propel the GA toward the vicinity of optimum solution.

Increasing Efficiency of Fitness Simulation

Note that significant savings from runtime can be achieved by using more efficient methods of sampling. A very obvious method is to save the data from each sampling process and to use Bayesian updating of a priori values. Thus we can sample only those individual edge costs that have a high standard deviation in relative to those other edges. Because the effective sampling size is much larger, the GA results will have lower standard deviations in this case. Another possibility is that in each generation a single value can be assigned to the edges that are common in all solutions. These edge cost values need not be sampled because they appear as additive terms in all cost expressions.

Further, more elaborate methods of sampling can be adopted. For example, if the number of measurements per sample is variable, one can apply the sequential probability ratio test [135] in order to classify the patterns with minimum number of observation under specified error probabilities. Since our goal is to calculate through Monte-Carlo methods a multiple convolution integral, one can also adopt the advanced Monte-Carlo integration techniques (such as importance sampling or stratified sampling or VEGAS algorithm (see [114]) for efficient evaluation of integrals).

5.6.3 Tests with Beta Distribution

The above SSHP problem is similar to the well studied problem of project planning, namely finding the longest path in a graph with stochastic edge costs. The differences are : **1)** we restrict out attention to the set of Hamiltonian paths, and **2)** we look for the shortest path in contrast to the longest path. Nevertheless, the parallels between these problems indicate that the method of using the GA can be substituted for the PERT approach which is commonly used for finding the longest path.

In most project planning problems, the activity durations are represented by beta distribution. Therefore, we perform a new set of runs where the edge costs have beta distribution. In contrast to normal distribution, beta distribution is bounded in both ends. See Appendix A for a description of the beta distribution. In our calculations, we assume that the lower (upper) bound on an activity duration is 80% lower (larger) than the most likely value (which is taken as the Euclidean distance). Further,

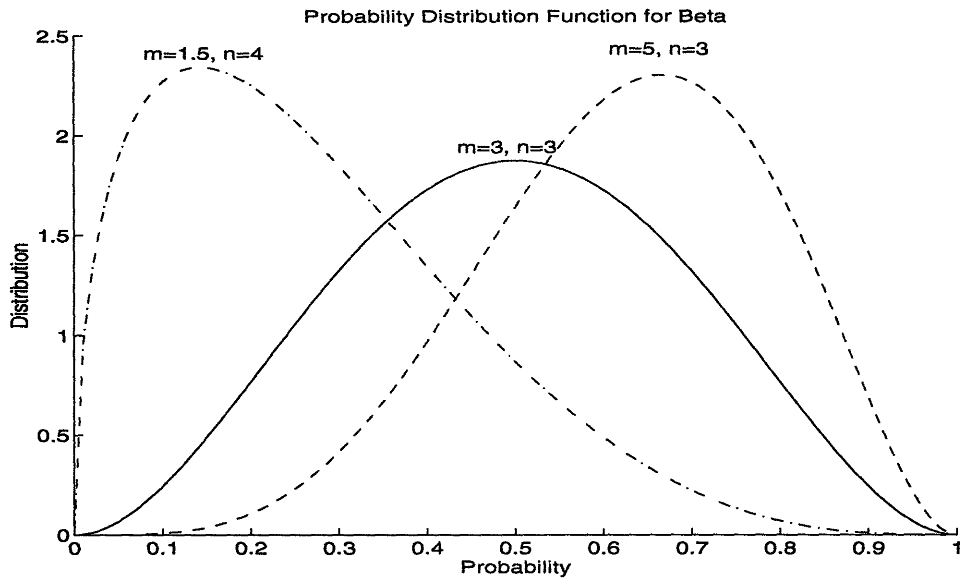


Figure 5-5: *Beta distribution function for three different parameter pairs*

we select three different beta distributions for activity durations limited between these lower and upper bounds; these distributions are shown in Figure 5-5.

Obviously, employing a symmetric beta distribution function for all edge cost values will produce results similar to those found by using Gaussian (normal) distribution. Therefore, we make the following assumption: Let $t_m(i)$ be the most likely value for the cost of edge i. Let T_L be a specified lower limit for the cost value and T_U be a specified upper limit. Assign the distribution for the cost value of edge i as

- $\beta(5,3)$ if $t_m(i) < T_L$
- $\beta(3,3)$ if $T_U \geq t_m(i) \geq T_L$
- $\beta(1.5,4)$ if $t_m(i) > T_U$

This assumption is pessimistic for edge costs whose most likely values are lower than T_L and is optimistic for those whose most likely values are higher than T_U.

The results of four GA runs are shown in Figure 5-6 for a graph with 20 nodes. It is observed that the GA can converge realatively quickly to the optimal solutions compared to those runs performed for cost values with normal distribution. This arises because a large number of solutions can now satisfy the optimization criteria. The size of the solution set can be reduced by hardening the optimization criteria (either by lowering $L_{\max}$ the maximum expendable cost or by changing the distributions). In any case, the best solution for the case with beta distribution on edge cost values is not necessarily same with the case with normal distribution on edge cost values.

Sensitivity Analysis

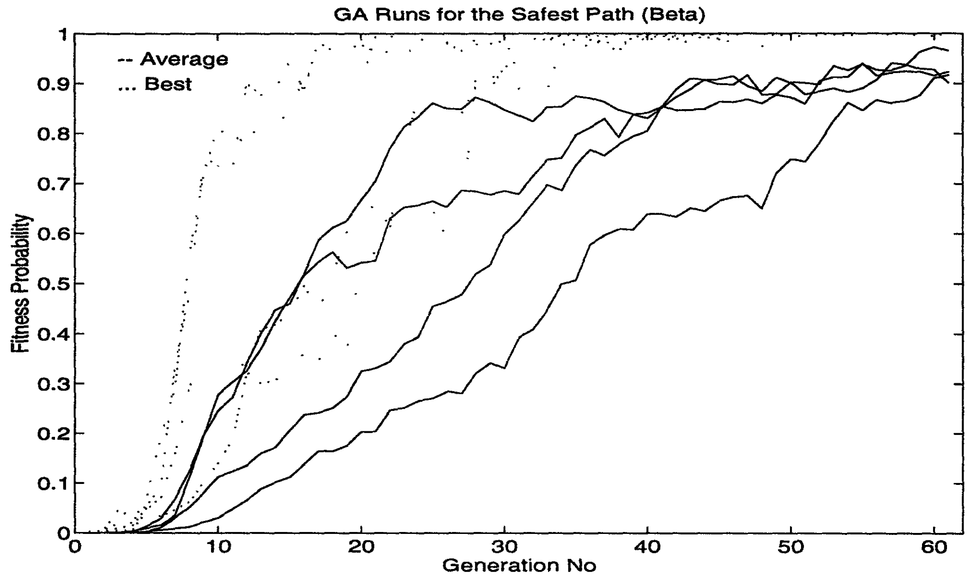


Figure 5-6: *GA runs for the safest path in a graph with 20 nodes (PopSize=50) for edge costs have Beta distribution*

The assumptions on probability distributions of edge cost values are often updated during plan execution. Since this information is not *a priori*, it cannot be incorporated directly into the initial model. Yet, it may prove useful to learn about the sensitivity of optimal solutions obtained to the assumptions made in the planning model. This problem is examined in detail in next chapters where we develop the concept of robust plans explicitly for this purpose. Nevertheless, we can point out that possible future changes in plan execution can be incorporated into the GA model by changing the the fitness function halfway during a GA run. The GA can often recover from the impacts of such changes and converge to the vicinity of the new optimum in the remaining time because it keeps track of a set of diverse solutions, some of which are probably close to the new optimum. Obviously, the changes can be accommodated easier if the GA run is in an early stage such that the probability of being trapped in a valley of the problem space would be low.

5.7 Summary and Conclusions

In this chapter, we examined specific examples of path planning problems under uncertainty. We observed the reasons why the introduction of uncertainty often increases the complexity of the problem under examination. We applied the GAs for a specific version of stochastic shortest path problem in which feasible paths are Hamiltonian and the criteria for optimization is minimizing the probability that the path cost exceeds an upper bound. The theoretical and experimental results have shown that GAs can be applied satisfactorily to this problem through evaluation of fitness by simulation. A practical use of this method is its application in the problem of finding the critical path in a stochastic project planning model, creating an alternative for the PERT.

Chapter 6

Robust Planning

Many formal structures, such as the layout of a microchip, cannot be changed after their design has solidified. As opposed to these, a plan is a structure that can be changed or dynamically modified as found necessary during the execution stage, provided that circumstances allow these modifications. In the execution of a plan, upon encountering failures or other contingencies, one either repeats one's attempts or resorts to alternative courses of action in order to adapt the plan to new circumstances. Efficient plans minimize the probability that a dynamic event will adversely affect the accomplishment of the plan goals. In other words, efficient plans should achieve maximum *adaptability* and *responsiveness* to possible circumstances that could occur during plan execution. Therefore, plan evaluations should involve the consideration of provided options and alternatives to be used when contingencies are encountered in the original intended plan. Integrating the adaptability to contingencies with the conventional plan evaluation measures leads us to the notion of *robust plans*. Robust plans minimize the cost of adapting to random events whose outcome become clear only after embarking on the plan.

This chapter introduces and examines the concept of robust plans. We review the related literature in *Section 6.1* and and especially concentrate on dynamic path planning problems in *Section 6.2*. We present a general view of the complexity of the planning problems with dynamic optimization criteria (*Section 6.3*). Later we focus on the reliability problem which is a good starting point for rigorously analyzing the plan performance in nondeterministic models (*Section 6.4*). Then, we introduce certain robustness measures and examine the algorithms for their determination (*Section 6.5*). We present a depth-first based algorithm in *Section 6.6* for evaluating the robustness measures and methods to make this algorithm tractable.

6.1 Background Work

Strong incentives existed for incorporating uncertainty into planning activities in many fields, such as economics, operations research and artificial intelligence. All these fields developed their own ways of addressing the problem in various ways. Nevertheless, there did not appear a systematic study that could benefit all of these fields by drawing on the strength of each particular approach. Besides, there was little or no emphasis on contingency or robust planning. For example, the need to control uncertaintied has led to the use of stochastic programming in the fields of economics and control; but these domains have paid attention mostly to rigorously formulated problems of their own disciplines and hence were too limited to be of practical value in most planning problems.

Yet, some of the recent developments, especially in the field of financial analysis have been inspiring for emphasizing the robustness and the flexibility of plans. For example, it is now commonplace to value financial investments by considering the worth of the *real options* built-in in the investment in conjunction with the classical evaluation. Real options are those alternatives that can be selected during the course of a project as found necessary. These options could be a choice on the timing of production or a choice on the production capacity or even a possibility to modify the end product. Option valuation methods sum up the net worth of these alternatives in order to reach a total project value.

In the field of Operations Research, Dantzig has started a program to generalize the field of linear programming to solve multi-stage decision problems subject to uncertainty [32]. Whereas linear programming finds values for a set of decision variables X that would minimize a linear objective function $Z = cX$ subject to constraints $b = Ax$, the generalized linear programming solves a multi-stage decision problem in which some constraints are dependent on earlier decisions. In this model, a planner wants to make a decision X_1; let random events happen; make a decision in period $t = 2$; let random events happen and so forth. The problem still consists of minimization of a given linear objective function Z of decision variables:

$$\min Z = c_1X_1 + c_2X_2 + c_3X_3 + \cdots + c_tX_T$$

But now the constraints follow a successive waterfall structure wherein the decision variables determine in part the constraints of the following stages:

$$\begin{aligned}
b_1 &= A_1X_1 \\
b_2 &= -B_1X_1 + A_2X_2 \\
\vdots &\quad \vdots \\
b_T &= \ldots\ldots\ldots\ldots - B_{T-1}X_{T-1} + A_TX_T
\end{aligned}$$

It is assumed that c_1, b_1, A_1 are known a priori, but other values are contingent upon the choices made and probability distributions being followed. For example, consider a two-stage problem where the variables of the second stage can take 3 discrete values with the corresponding probabilities $p_2(1)$, $p_2(2)$ and $p_2(3)$. The problem is then converted to

$$\min Z = c_1X_1 + p_2(1)c_2(1)X_2(1) + p_2(2)c_2(2)X_2(2) + p_3(3)c_3(3)X_3(3)$$

subject to constraints

$$\begin{aligned} b_1 &= A_1X_1 \\ b_2(1) &= -B_1(1)X_1 + A_2(1)X_2(1) \\ b_2(2) &= -B_1(2)X_1 + A_2(2)X_2(2) \\ b_2(2) &= -B_1(3)X_1 + A_2(3)X_2(3) \end{aligned}$$

To solve these equations, Dantzig adopts Bender's decomposition technique, which is a frequently used method for solving integer programming problems and suggests the use of parallel computing. Nevertheless, the integer programming itself is an NP-hard problem and this model too suffers from the ubiquitous complexity problem.

In the field of AI, the problem of planning under uncertainty has also attracted attention relatively recently. The problem was defined formally in 1985 [88]: *We are interested in a particular set of planning problems distinguished by the following characteristics: (1) the current situation is not known with certainty (2) the consequences of action are not known with certainty (3 the goals of the planning process are conflicting, and therefore, are not completely satisfiable. These types of problems are referred as planning under uncertainty."* Langlotz and Shortliffe [88] compared the use of different methodologies in these problems and concluded that a purely deductive approach is difficult to consider because these planning tasks entail uncertainty and tradeoffs. They point that both decision-theory and non-monotonic logic have similar worst-case complexity, $O(2^n)$ for n states; yet the use of decision theory is preferable over the use of non-monotonic logic which does not include utilities and is domain dependent.

As researchers in AI have been increasingly more busy with implementation of plans by robots, they needed to address the question of what to do when things do not go as expected. While an early study by Fikes [41], the author of STRIPS, made use of a method called triangle tables to produce plans that could cope with any order of a specific set of situations in contrast to more specific ordering requirements of the previous planners, the question was not particularly answered except the development of nonlinear planning which could be seen as a general heuristic for developing flexible plans. Nonlinear planning emphasized that plans should make minimal commitments such as specifying only absolutely necessary precedence relations among some actions. Nevertheless, acting on incomplete plans was unsatisfactory for some researchers and the urgency for addressing the question remained. From this point on, generally speaking, research on planning in AI has branched out to two different directions. The first direction is based on monitoring the plan progress for "failures"; when failures are found, the agent generates a new plan in reaction. This strategy of *ad hoc* plan generation is often called **reactive planning**. The following quote from Georgeff [55] describes the motivations for reactive planning: *In many domains much of the information about how best to achieve a given goal is acquired during plan execution. For example, in planning to get home from airport, the particular sequence of actions to be performed depends on information acquired on the way -such as which turnoff to take, which lane to get into, when to slow down or speed up, and so on....Decisions are deferred until they have to be made. The reason for deferring decisions is that an agent can acquire more information as time passes; thus, the quality of its decisions can be expected only to improve. Of course, because of the need to coordinate some activities in advance and because of practical restrictions on the amount of decision-making that can be accommodated during task execution, there are limitations on the degree to which such decisions may be deferred.* Later, Schoppers continued this tradition by developing universal planning based on *a priori* consideration of all possible scenarios. Schoppers [126], defining planning as the goal directed selection of reactions to possible situations, gives a method that prescribes a reaction for every possible situation that could take place during plan execution; these plans are robust and fast to execute, but can be very large and expensive. Nevertheless, Ginsberg [56] in *"Universal Planning An (Almost) Universally Bad Idea"* points out that this sort of planning would be too time consuming to generate. He points out that for n sensors and for a primitive actions, there exist $(2^a)^{2^n}$ distinct universal plans as there are 2^n set of situations and 2^a possible combinations of actions to be taken. For another example, consider the interaction of 100 decision variables leading to 2^{100} distinct events. Even when a recipe with 2^{90} items is given by the universal planner, it covers only one-thousandth of possible situations. The tradition of reactive planning has been continued especially in Stanford University. A recent example in [23] interleaves local reaction within the general framework of global plans. Finally, some researchers has carried the idea of reactive planning to extreme by dropping the planning stage altogether and produced systems that can only react. Two proponents of this camp, Chapman and Agre [22] defend their system PENGI as *"PENGI is not a planner; Pengi is designed to lead a life. Pengi's world, like ours, is not a problem to be be solved but an ongoing web of recurring opportunities to engage in sorts of activity. Planners, being designed to solve problems, are not good at leading lives."*

It is arguable that these last approaches do not capture the essence of planning but rather they are based on avoidance of planning activity. Therefore, they may suffer from the syndrome of *painting oneself into a corner*, which is why plans are needed at first. The second direction continued the classical planning tradition by developing the models with some classes of failures included. Yet, as Kaelbling points out [37] *"There is an inherent contradiction in all of these approaches. The world is assumed to be deterministic for the purpose of planning, but its nondeterminism is accounted for by performing execution monitoring or by generating reactions for world states not on the nominal planned trajectory."*

Kaelbling *et. al.* produce a planner for a nondeterministic model by making use of Markov decision models and dynamic programming techniques. They alleviate the complexity problem by focusing on a narrow section of the state space instead of a global analysis [37].

6.2 Dynamic Path Planning Problems

Some dynamic planning problems have recently been formulated in the context of path planning. These problems consider the adaptation of a path to events that may arise while the path is being travelled. An elementary understanding of the effect of new information on plans can be gained through these simple planning problems. *"It is a common experience while driving to come upon a road interruption and have to make a detour [3]".* Upon encountering such contingencies, a path must be adjusted or substitute paths must be used. Therefore, a priori solution to a path planning problem is not a static path but rather has the character of a policy. Asking for the optimal policies that not only indicate a path to follow but also how that path is to be modified in the event of new information naturally follows from the paradigm of robust planning. Models that include this dynamic response have recently been forthcoming in the literature. Dynamic models for path planning problems have been constructed for both deterministic and probabilistic environments.

6.2.1 Dynamic Deterministic Models

Most work concerning dynamic environments (subject to change) has been focused on **on-line** algorithms that solve deterministic problems. An example of an on-line algorithm is the best-fit heuristic for the bin-packing problem. This algorithm takes an item at a time out of a given list and sort this within the already decided packing scheme. Further interesting problems are considered by Papadimitriou and Yannakakis [109].

Consider the problem of navigation in which the map is dynamically revealed to the vehicle as it moves along without a priori knowledge about the planning environment. Papadimitriou and Yannakakis [109] consider *policies* that minimize the *worst-case ratio* of the maximum distance traveled under these policies to the true shortest distance if the graph were known ahead of time. They first consider the problem of finding an optimal policy (minimum worst-case ratio) for layered graphs. A **layered graph** is a graph whose nodes are partitioned in layers $L_1, L_2, \ldots, L_n$ and all edges are between nodes in adjacent layers. They assume that when the vehicle arrives at a node layer L_i, it learns the nodes of layer L_{i+1} and the costs of the edges leading to these nodes. They furthermore assume that the vehicle does not know when the terminal node will appear. They prove that if the width of the layered graph is 2, an optimal policy is to start by following the edge with the minimum cost, and continue on that path until the total distance on that path is greater than twice the distance on the alternate path; when this occurs the vehicle should go to the node on the alternate path [112]. They prove that this simple policy achieves a worst-case ratio of 9, which is the best possible. They generalize this result to layered graphs with width w.

In the same paper, they also introduce the *Canadian Traveller Problem* (CTP), in which the graph with associated costs are known a priori, however some of the edges might be unsuitable for travel at certain times, and such blockage is revealed only upon reaching an adjacent node. This problem can be viewed as a two-person game, between a searcher and a malicious adversary, who sets the weather conditions so as to maximize the ratio. They show that the problem of finding an optimal policy is PSPACE-complete. The same problem with a different objective function has been analyzed by Bar-Noy and Schieber [9]. They consider the minimization of the worst-case cost and provide algorithms for its solution. They also define the *Recoverable Canadian Traveller Problem* (RCTP) in which blocked roads may be reopened after a certain period of time.

6.2.2 Dynamic Probabilistic Models

Both of the above papers suggest generalizing their problems to stochastic models. Papadimitriou and Yannakis [109] mention that the CTP is #P-hard when each edge cost has a discrete probability distribution. Bar-Noy and Schieber [9] define the stochastic RCTP, in which every edge e_{ij} is blocked with some a priori probability p_{ij}. The traveler finds out whether an edge e_{ij} is blocked when he visits node i. They solve this problem with a label-setting algorithm. Probabilistic models are often more natural and useful than deterministic models.

One of the earliest papers that considered dynamical response to events by a navigator in a nondeterministic model is due to Croucher [31]. Croucher assigns a probability value to the event of finding a given edge active. He assumes that if an edge is found inactive, one chooses randomly another node to follow among its adjacent nodes. This model can also be considered as accounting for the uncertainties in control; one might accidentally (with probability p_{ij}) implement a wrong action that would take him to an unintended state. Croucher applies the dynamic programming algorithm to this problem. He expresses the shortest expected distance from node i (with n_i successor nodes) to the node N as

$$g(i) = p_{ij} D_{ij} + \frac{1 - p_{ij}}{n_i - 1} \sum_{k, k \neq j}^{n_i} D_{ik} \quad if \quad n_i > 1$$

$$g(i) = D_{ij} \quad if \quad n_i = 1$$

where D_{ij} is the shortest distance when the edge e_{ij} is actually traversed and given as:

$$D_{ij} = d_{ij} + g(j).$$

Thus, from node i a succeeding node j can be found with the following expression which minimizes $g(i)$ over all possible j:

$$If \quad n_i > 1: \quad g(i) = \min_j \left\{ p_{ij}[d_{ij} + g(j)] + \left(\frac{1 - p_{ij}}{n_i - 1}\right) \sum_{k,k \neq j}^{n_i} [d_{ij} + g(j)] \right\}$$

$$If \quad n_i = 1: \quad g(i) = d_{ij} + g(j)$$

While Croucher assumes a random selection among adjacent nodes is made upon encountering an inactive node, Andreatta and Romeo [3] employ an optimal selection for the new instance. They also redefine this problem as the *stochastic shortest path with recourse.* Polychronopoulos notes that [112] their problem is effectively the stochastic CTP with the distinct objective of minimizing the expected cost. This model, too, assumes that the edge costs are deterministic but there is some discrete conditional probability that a given edge would be active. In particular, they assume that there could be r realizations of the edge set E such that E_1, E_2, ..., E_r and their disjoint probabilities are specified a priori. By first defining the expected length of a recourse path from a node j to the destination node, they give an expression of the expected cost of a given path. Further, they give the following results for this problem :

1. The expected cost of a deterministic shortest path can be arbitrarily worse than that of a stochastic shortest path (SSP).

2. A SSP may contain a cycle, even if the distance function is nonnegative.

3. The expected cost of a SSP may be shorter than that of one of its subpaths, even if the distance function d is nonnegative.

4. A subpath of a SSP is not necessarily a SSP for the corresponding subproblem. That is, the Bellmann's principle of optimality does not hold here and, therefore, the SSP cannot be found by standard dynamic programming [3].

Furthermore, they give a recurrence relation for the expected length of the optimal SSP, in the form of

$$L^*(i) = \min_j \left\{ \Pr() \, [d_{ij} + L^*(j)] + (1 - \Pr()) L'^*(i) \right\}$$

They note that this equation is a version of the so called stochastic dynamic programming recurrence relation and therefore can be solved by applying any stochastic dynamic programming algorithm. They also note that, as is typical in stochastic dynamic programming, the number of states that need to be considered may grow exponentially with the number of stochastic edges in the graph. They also note that the problem of finding a SSP can be formulated as an optimal decision Markov process with a finite horizon.

Andreatta and Romeo's model has been extended to graphs with stochastic edge costs by Polychronopoulos and Tsitsiklis [112]. They claim that assigning distributions to edge costs help their model encompass the Andreatte model as a special case (one can assign a very large cost to an edge with a certain probability such that it will not be used on the optimal policy). They too define this problem as the *stochastic shortest path problem with recourse* (SSPPR). Further they assume that the realization of the costs of the uncertain edges are learned once these edges have been traversed. According to the observed cost realizations, the information of the decision maker is updated dynamically. They also make a distinction between two special cases: They call the version with dependent, general edge costs as R-SSPPR (from the R possible realizations of the graph) and the version with mutually independent edge costs as i-SSPPR. Note that if each edge can take k possible values and there exists m edges, the total number of realizations can be very large, namely $R = k^m$. They give a dynamic programming recursion for the solution of R-SSPPR, in the form of

$$V(i, I) = \min_j \{d_{ij} + E(V(j, h(j, I)))\}$$

where $V(i, I)$ is the optimal cost-to-go starting from state (i, I) at node i with information I and where $h(j, I)$ is the information set of the vehicle when it gets to node j given that it already has the information described by I. They provide a label-setting algorithm as a subroutine in the determination of the term $E(V(j, h(j, I)))$. Their algorithm solves the R-SSPPR in $O(2^R(Rn + m + n\log n))$ time. They also give a similar yet simpler algorithm for $i - SSPPR$ that runs in $O(k_{\max}^m(k_{\max}n + n\log n + m))$ time. Further they prove the following: **1)** The recognition version of the R-SSPPR is NP-complete and **2)** The problem i-SSPPR is #P-hard and i-SSPPR $\in$ PSPACE [112].

Finally, another model that considers the dynamics of plan execution has been developed by Jaillet and Odoni [77]. They modelled the problem of minimizing the expected cost of a TSP-tour in which the set of nodes to be visited is given with a priori probability vector P. Later, Jaillet extended this work to the shortest path problem and defined the *probabilistic shortest path problem* (PSPP). Jaillet gives the following motivations for these problems [78]: *"Our main concern is to define and analyze probabilistic versions of well-known combinatorial optimization problems while keeping their original combinatorial flavor. There are several motivations behind this work. ... The first one is the desire to formulate and analyze models that are more appropriate for real-world problems where randomness is present. The second motivation is an attempt to analyze the robustness of optimal solutions for deterministic problems when the network for which the problem has been solved is modified."* He gives a polynomial expression for the length of the path and proves that the PSPP is NP-hard by reducing it to Hamiltonian path problem.

6.3 Complexity of Dynamic Planning Problems

Robust planning problems consider the effects of possible contingencies to a given plan and aim to find those that are most resistant to adverse impacts. These problems are clearly parallel to the problems of game theory where decisions are to be made while facing an adversary. Consider the game of chess where a move's value depends on the opponent's moves. Good chess moves generally apply the *min-max* strategy: minimize the maximum gain of the opponent. In robust plannning problems, nature can be considered as the adversary and we aim to minimize the maximum gain of the nature (by definition our maximum loss). Since the general game theoretic problems belong to the complexity class PSPACE, it can be claimed that the general planning problems under dynamic optimization criteria too lie in PSPACE. Problems in this class are usually characterized by a discrete-time random process, the parameters of which can be influenced by dynamic decisions. Decisions are based on the current state, and the next state is a random variable with a distribution that depends on the current decision and state. The goal is to minimize the expectation, of some cost functional of the history of states and decisions.

Specific instances of decision making problems under uncertainty have been defined by Papadimitriou [106]. Papadimitriou has defined the following problems: *stochastic satisfiability, stochastic scheduling, dynamic graph reliability, general Markov decision processes and optimal control.* These problems are parallel to the classic NP-complete problems. In **stochastic satisfiability (RSAT)**, we are given a Boolean formula F involving variables $x_1, x_2, \ldots, x_n$. We are asked whether there is a choice of Boolean variables for $x_1, x_3, x_5, \ldots, x_{n-1}$ for a random choice of the truth value for variables $x_2, x_4, \ldots, x_n$ such that the probability that F comes out true under these choices is more than a given value p_0:

$$\Pr(\exists x_1 \sharp x_2 \exists x_3 \sharp x_4 \ldots \exists x_{n-1} \sharp x_n F(x_1, x2, \ldots, x_n)) \geq p_0$$

In **stochastic scheduling**, we are given a tree of precedence constraints amony tasks with execution times that are random variables with identical exponential distributions (Also known as Poisson tree scheduling). The problem is to find a strategy for scheduling these tasks on m processors so as to minimize the makespan.

These examples characterize both the dynamic nature and the stochastic nature of planning problems when uncertainty prevails. Papadimitriou has shown that for the general cases, these problems belong to the complexity class PSPACE; moreover they are PSPACE-complete (i.e; the other PSPACE problems are reducible to these in polynomial time). Their belonging to the class PSPACE implies that these problems are intractable -even harder than those problems that are NP-hard-.

6.4 Reliability

The concept of reliability is a good reference point for analyzing the concept of robustness. Reliability of a given system is the probability that the system will perform its intended function under stated conditions in a stated time interval. In the planning arena, the reliability of a plan can be interpreted as its overall success probability. Although this definition may sound straightforward, it does not clarify what is meant by failure because a plan has often many goals to be achieved. For instance, a research project can succeed in its goal of solving a new problem yet fail in its goal of completion on time. A practical way of dealing with this problem is to distinguish one goal and define failure and success according to whether this goal is attained or not. Other optimization measures must also be considered in addition to a plan's reliability with respect to this goal. One may evaluate then for each alternative plan the behavior of its reliability versus other measures (e.g. the value of the required resources) and thus may determine the plan that achieves the optimum configuration. In general, planners need to trade off the potential of falling short from their objectives against the extra costs required to increase the plan reliability.

Because we use graphs for modelling of plans, we now examine the reliability problem in the context of graph models, which have been extensively studied. Many parameters have been used as measures of reliability; these roughly divide into two: *deterministic measures* such as edge and vertex connectivity and *nondeterministic* measures such as probabilistic connectedness. Although the deterministic measures can be computed more easily relative to those that are nondeterministic, they provide only poor estimates of reliability. Three types of problems that deal with both deterministic and nondeterministic versions of reliability are overviewed below:

6.4.1 Survival Network Design Problem

A typical example concerning reliability arises in the design of telecommunication networks. Telecommunication networks are vulnerable to failure; they can be disrupted by accidents and other natural causes destroying the cables or the transmission nodes. Talluri notes that [131] the problem has taken on even more importance as copper cables are replaced by fiber-optic cables that can carry much more traffic; as a result, failures in transmission links can have catastrophic consequences. In order to maintain communication despite failures, one should design a network with alternate routes for communication that are to be used when some equipment fails. Such redundancy in the system reduces the chances of failure, but at greater overall network cost. Thus, we are led to the problem of designing a minimum-cost network that meets certain connectivity requirements.

The survival network design problem is as follows: **Given a graph and a survivability requirement r_i for each node i, find the minimum-cost subnetwork that has r_{ij} edge-disjoint paths**

between every pair of nodes i and j. (A set of paths are edge-disjoint when no edge appears more than once.) Typically in practice r_i is one of $(0, 1, 2)$. When $r_i = 1$ for all nodes, the problem becomes the minimum-cost spanning tree problem, which is a member of $\mathbf{P}$. When $r_i = 1$ for some nodes and $r_i = 0$ for others the problem becomes the network Steiner-tree problem, which is NP-hard.

6.4.2 Most Vital Links Problem

Another problem that is relevant to reliability and robustness in planning is the most vital links problem. The *n most vital links* in a network are those n links whose removal from the network results in the greatest increase in the shortest distance between two specified nodes s and t. The solution of the most vital links problem thus gives information on the sensitivity of the connection between the specified nodes s and t to removal or elimination of certain links of the given graph. A similar problem can be posed by considering the most vital *nodes* instead of *links.* A solution algorithm for this problem is given by Carley in [20]. For a summary of the problem, see [95]. Ball *et. al.* [8], who examined the problem with a removal cost $c(e)$ for each edge and a budget constraint prove that this version of the problem is NP-hard.

6.4.3 Network Reliability Problem

In contrast to examples that employ deterministic models, the *network reliability problem* is concerned with the probability that a given network is able to carry out some desired operation. While the survivable network design problem is concerned with building a network from scratch, the reliability problem analyzes an existing network to determine its reliability. In this problem, it is assumed that each edge of the network fails with a certain probability. Moreover, it is often assumed that all of the probabilities specified are statistically independent. The following list describes some common reliability measures:

1) For two specified nodes s and t, the two-terminal reliability denoted by $Rel_2(G)$ is the probability that the network G contains a $[s, t]$ path.

2) The all-terminal reliability $Rel_A(G)$ is the probability that for every pair of nodes v_1, v_2, the network contains a path from v_1 to v_2; equivalently, the all- terminal reliability is the probability that the graph contains at least one spanning tree.

3) The final network operation generalizes both of the previous two and involves pairwise communication of k specified nodes, $2 < k < n$. The k-terminal reliability $Rel_k(G)$ is the probability that for k-specified target nodes, the graph contains a path between each pair of the k-nodes. This is equivalent to asking for the probability that the graph contains a Steiner tree.

In addition to these three measures, one might define a number of other reasonable reliability measures. For general graphs, almost all of these measures are too difficult to calculate exactly.

6.4.4 Computing Reliability

The problem of combinatorial explosion is often encountered in reasoning with uncertainty. Because there are 2^n combinations for n discrete events, there are 2^n joint probability measures often needed in calculations involving interaction of these discrete events. Unfortunately, most of the general techniques for evaluating the reliability of complex systems suffer from combinatorial explosion and therefore are not suitable for automation. Colbourn [26] notes that calculation of most of the interesting reliabilities has been proven to be #P-hard, a complexity result roughly equivalent to being NP-hard for enumeration problems. Typical members of the class of **#P-complete** problems are those problems that ask for the number of solutions to an NP-complete problem [138]. Nevertheless, some counting problems that have no relation to NP-complete problems also fall into this class; reliability problem is an example of those. Counting number of trees in a directed graph constitutes another example.

Despite these negative results, evaluating reliability is not computationally hard for some restricted versions of the problem. An important class of such problems involves *serial-parallel* networks and a given pair of nodes with source s and destination t. These networks do not contain complicated links; the paths from source to destination include either a series of successive links or a set of parallel links or involve a combination of these. It is not difficult to express the reliability for these networks:

For a serial system

$$\begin{aligned} \Pr(S) &= \Pr(x_1 \cap x_2 \cap \cdots \cap x_n) \qquad (6.1) \\ &= \Pr(x_1) * \Pr(x_2|x_1) * \cdots * \Pr(x_n|x_1 \cap x_2 \cap \ldots \cap x_{n-1}) \end{aligned}$$

When the individual units are mutually independent of each other:

$$\begin{aligned} \Pr(S) &= \Pr(x_1) * \Pr(x_2) * \cdots * \Pr(x_n) \\ &= \prod_i \Pr(x_i) \qquad (6.2) \end{aligned}$$

For a parallel system

$$\begin{aligned} \Pr(S) &= \Pr(x_1 \cup x_2 \cup \cdots \cup x_n) \qquad (6.3) \\ &= 1 - \Pr(\overline{x}_1 \cap \overline{x}_2 \cap \cdots \cap \overline{x}_n) \end{aligned}$$

When the individual units are independent of each other

$$\begin{aligned}\Pr(S) &= 1-(1-\Pr(x_1))*(1-\Pr(x_2))*\cdots*(1-\Pr(x_n)) \\ &= 1-\prod_i(1-\Pr(x_i)) \qquad (6.4)\end{aligned}$$

Many models composed of serial and parallel links can be analyzed with the above equations by using the appropriate reduction techniques. Unfortunately, the problem is computationally hard for its general version. Thus, much of the research in this area is geared towards obtaining good, quickly computable bounds for the measures. The following is an example of an upper bound for $Rel_2(G)$. Suppose edge e of the graph fails with a probability p_e and is operational with a probability $1-p_e$. G is operational if there exists at least one path from node s to node t. G is operational only if every (s,t)-cut in G has at least one operational edge (A cut is a set of arcs that disconnects the graph). Therefore, the probability that G contains an (s,t) path is bounded above by

$$\prod_{i=1}^{k}(1-\prod_{e\in C_i}p_e)$$

where $C_1, C_2, ..., C_k$ are edge-disjoint (s,t) cuts in G. This bound is very easy to compute once the (s,t) cuts have been determined and has been found to be competitive with more sophisticated bounds [27].

The technique described in the above example falls under the category of minimal-cut set methods which is one of the frequently used method for evaluating reliability. A minimal cut set in a system is defined as those units that have to fail all at once for the system to fail. Recall from Chapter 3 that the evaluation of the minimal-cut in a graph itself is an NP-complete problem. Another frequently used method for evaluating reliability is based on fault trees. As with other methods, this method too becomes quickly unwieldy as the problem size grows, forcing to apply approximation techniques. Some work in assessing reliability has concentrated on using problem-specific information and model-based reasoning; a recent study that demonstrates their use in limited domains [67] can be accessed via Mosaic. Unfortunately, there are no general, tractable algorithms for evaluating reliability unless the system can be decomposed into serial-parallel chunks.

To sum, reliability has an important role in the evaluation of performance of nondeterministic formal structures. Its determination is computationally hard but satisfactory approximations are often available. Improving reliability of a plan generally involves a trade-off with other resources which should be taken into account by the plan developer. Diversity and redundancy used for improving reliability may also prove to be important concepts that could provide important information for heuristic rules for generating flexible plans. Finally, reliability, as traditionally defined, does not capture some of the ideas that are based on measuring the values of the options that a plan may have. This leads us to define robustness.

6.5 Robustness

6.5.1 Description of Robustness

What the above measures of reliability lacks is recognizing that plans are often modified as they are being executed. In a nondeterministic world, one is almost always confronted with new possibilities. The exercise of a single action, the occurrence of a new event can correspond to closing off of certain possibilities and opening up of some new possibilities. In order to address contingencies that may emerge during plan execution, plans should carry options for adaptation to events whose outcomes are critical to plan success. This notion of adaptability can easily be integrated in evaluation of the reliability. Because the new resultant concept encompasses the concept of reliability, it is suitable to give it a new name; we have chosen the name **robustness** for this concept.

A convenient measure of robustness can be developed directly from the two-terminal reliability, which is the success probability of a plan given a pair of source and destination nodes. Because plans are often modified, the success probability of a given plan will be a disjunction of all possible modifications. That is, for a plan p that satisfies certain objectives:

$$R(p) = \Pr(p \cup p^{'} \cup p^{''} \cup p^{'''} \cup p^{''''} \cup \cdots) \tag{6.5}$$

where $p^{'}, p^{''}, \ldots$ show the possible modifications that still satisfy the same objectives. Since the modifications are not subsequent but performed dynamically upon failure as the plan progresses, this probability is not equal to :

$$R(p) \neq \Pr(p) + \Pr(p^{'} \mid p\,\text{fails}) + \Pr(p^{''} \mid p\,\text{and}\,p^{'}\ \text{fail}) + \cdots$$

Rather, to express this probability, we resort to a recursive equation. Recursive evaluation provides a convenient way of incorporating the fact that one is always confronted with new possibilities during plan execution. An example of the recursive evaluation of the probability R is given below for a binary case where an action can either fail or succeed. For a shorthand, let us denote with x the probability that a state transition is made from the state a to the state b by exercising the corresponding action of the plan p.

$$x = \Pr(\mathit{State}\ a\ \rightarrow\ \mathit{State}\ b \mid p(a)) \tag{6.6}$$

The success probability then can be written as:

$$R(p) = x\, R(p^{*}(b)) + (1 - x)\, R(p^{'}(a^{*})) \tag{6.7}$$

When the exercise of an action results as expected, we are taken to state b, whereafter the rest of the plan $p^{*}(b)$ can be applied. If the result turns out to be different (with the probability $1 - x$), we are taken to the state a^{*}, which could be different than state a due to depletion of the resources because of the exercised action. If the exercised action had no side effects then the new state a^* is equivalent to the original state a: $a^* \equiv a$. There are two general options open to us at state a^*: We can either **modify** the plan by applying a strategy $\mathcal{F}(p)$ such that a new plan $p^{'}$ is obtained or **try again** the same action that had previously failed. This second option can also be considered as a null modification such that

$$p^{'} = \mathcal{F}_N(p) \ \equiv p$$

It is often the case, however, that one will select a strategy different than the obvious null modification. Ideally this modification strategy should be such that the probability $R(p)$ is maximized. Because the contribution of the second term to the overall probability is always positive, this last criteria corresponds to determination of a new plan which is optimum for the present circumstances. We call this strategy as *replanning*. Because replanning may be quite expensive, one may prefer to use strategies that are limited and use simple modifications such as heuristic repair rules.

The above equation 6.7 can also be used for modelling the effects of new information. Consider an event that imposes new circumstances during plan execution and therefore may take us from the state a to a^* even though there were no actions exercised by the planner. The effects of such environmental impacts on the success probability $R(p)$ can be taken into account via the second term of the equation 6.7. Further, the same ideas can still be applied to generalize from the binary case to those cases in which the results of an action or an event are not dichotomous, (i.e. more than two distinct states can be accessed from a starting state with corresponding transition probabilities). For example, with three different state transitions having probabilities x_1, x_2 and x_3

$$R(p) = x_1\, R(p_1) + x_2\, R(p_2) + x_3\, R(p_3)$$

where p_1, p_2 and p_3 signify the corresponding versions of the plan for these new states. Note that $x_1 + x_2 + x_3 = 1$. It is also possible to treat failures in nodes within this framework simply by duplicating the failing node and creating a synthetic probabilistic link between the original node and its duplicate.

Another possible measure of robustness can be derived from the concept of expected utility. In contrast to conventional methods, the expected utility for each specified plan must be calculated dynamically by considering all possible discourses for a given plan. The dynamic expected utility $E^*(p)$ for a given plan p is equal to the sum of probability weighted utilities for all possible variations:

$$E^*(p) = \sum_{i \in V_p} \Pr(p^{(i)}) U(p^{(i)}) \tag{6.8}$$

where the set V_p consists of all distinct variations for a given plan p and $U(p^{(k)})$ signifies the utility of kth version of the plan p. The dynamic expected probability reduces to the conventional (static) expected utility only if no changes can be made after embarking on the plan.

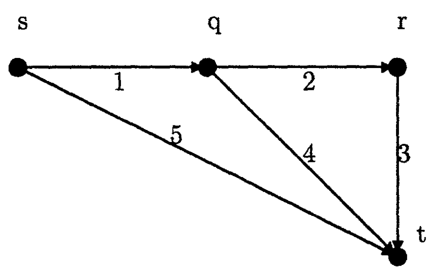


Figure 6-1: *A Simple Graph*

6.5.2 Recursive Evaluation

The recursive equation 6.7 that gives the success probability R can be used either with a forward induction or a backward induction. In either case, one must compute a new modified plan for each step that could go wrong. Assume that this computation is performed through a heuristic rule that takes a negligible amount of computational effort. Even for this given simplified case, the induction algorithm is computationally intractable as one encounters *bifurcations* at every level of a given plan. Consider an extreme case: There are no resources, no constraints and no limit on the precision of the probability R. Since there are no bounds, it is inevitable that almost all of the state space will have to be considered.

If there are no resources and constraints, the success probability can be simply calculated as the probability of reaching to the terminal node (or nodes). In other words, $R = \Pr(\text{State}= t)$. The following is an example of evaluation of the probability R for a plan p that normally follows a path starting from the node s and reaching to the node t through nodes q and r. The graph is depicted in Figure 6-1. Let the probabilities for each edge (or action) to be denoted as x_1, x_2, x_3, x_4 and x_5. If the first action fails (with probability $1 - x_1$), the alternate route from s to t is taken. If the second action fails, the alternate route from q to t is taken. There is no alternative to be used if the third action fails. Then, by using the recursive equation

$$R_{st} = x_1 R_{qt} + (1 - x_1) R_{s't}$$

$$R_{qt} = x_2 R_{rt} + (1 - x_2) R_{q't}$$

$$R_{rt} = x_3 R_{tt}$$

$$R_{s't} = x_5 R_{tt}$$

$$R_{q't} = x_4 R_{tt}$$

and by noting that $R_{tt} = 1$ and making the substitutions, we obtain

$$R_{st} = x_1 x_2 x_3 + x_1(1 - x_2) x_4 + (1 - x_1) x_5$$

In most cases, however, evaluation of the probability would be much more complex than this example. In general, for m edges we have to consider 2^m different events, causing the algorithm to run in exponential time. The recursive equation can be adapted to take into account the constraints to which a given problem may be subjected. For example, resources may be depleted by exercised actions and there may be limits on those resources. Alternative paths may have to be eliminated from consideration due to such constraints. Using advance lookup and constraint-propagation techniques, one can truncate the search space; yet these will not eliminate anxiety over the worst-case exponential running time.

6.5.3 Markov Model

Given these problems with computational complexity, an obvious question to be asked is whether there exists an alternative evaluation method which is less demanding for evaluating the success probability R. A candidate is the *Markov chain* that is often used in modelling stochastic processes. The Markov chain uses a state transition probabilities matrix P whose (i,j)th element is the probability of making a transition from state s_i to state s_j. These transitions are achieved by the application of an action from the action set $\mathcal{A}$. Thus,

$$p_{ij} = \Pr(s_i, a_{ij}, s_j)$$

where a_{ij} is the action that takes the system from state s_i to state s_j. These transition probabilities are used to update the current probability density vector π whose jth element expresses the probability of being in state s_j :

$$\pi(k+1) = \pi(k) \times P \tag{6.9}$$

In a similar manner to our recursive equation that tracks the evolution of state transitions, repeated applications of this equation 6.9 will yield the eventual probability density vector. For the asymptotic value,

$$\pi(\infty) = \pi(0) \times P^{\infty} \tag{6.10}$$

Another way of calculating the asymptotic probability density vector is due to the fact that the system eventually reaches a stable equilibrium :

$$\pi(\infty) = \pi(\infty) \times P \tag{6.11}$$

It is clear that the asymptotic probability of reaching any state can be found easily with this model unless the size of the state transition probabilities matrix is very large. Nevertheless, problems arise when the model is to be adopted for evaluating the robustness of plans.

First, the determination of the matrix elements corresponding to transition probabilities requires taking into account the priorities and orders prescribed by the plan as well as the modification strategy.

This is in contrast to most classical studies in which the state transition probabilities are constant and given a priori by the model. For example, plan normally leading from s to t through q may have to resort an alternative path through node r. Since the alternative is exercised after the original path has failed, the transition probability from state s to state r must be computed by multiplying the initial probability of this trial by the probability of this alternative path found from the corresponding edge probability in the model.

Second, problems arise due to resources and constraints. If the success probability is not simply the probability of reaching to the destination node but it also involves reaching to it under certain constraints, the Markov model's update of the probability density vector has to be applied step by step by taking note of these constraints, by applying the modification strategy in each step and by determining the current state transition probabilities. This is because the probabilities may change at each step in contrast to classical Markov models where they are static. This forced dynamic evaluation results in losing the advantages of Markov model; indeed, it differs from the recursive evaluation model mostly in evaluation of the probabilities of reaching to all other states beside the terminal state. Introduction of resources in this problem requires a memory preservation structure which contrasts with the spirit of Markov models. Also note that the introduction resources precludes the direct use of dynamic programming techniques.

6.6 An Algorithm For Evaluating Robustness

6.6.1 Implementation

Evaluation of the robustness measure is based on the generation of the possible variants of a given plan. We call these variants as *realizations.* A realization models the trajectory of a plan as it is being implemented -complete with the modifications performed on it. The following gives a PROLOG version of the algorithm for the realization model that has been used for measuring robustness :

```
Realization(Realization?, Plan, Constraints) :-
    Choose(ActionExercised?, Plan),
    Either                                                    ;Action Succeeds
        Update(ModifiedPlan?, Plan, ActionExercised?, Success),
        WhenViolates(ModifiedPlan?, Constraints) (Fail),
        Realization([Realization? | ActionExercised?], ModifiedPlan?, Constraints);
    Or                                                        ;Action Fails
        Update(ModifiedPlan?, Plan, ActionExercised?, Failure),
        WhenViolates(ModifiedPlan?, Constraints) (Fail),
        FindOption(Option?, ModifiedPlan?, ActionExercised?, Constraints),
```

RestorePlan*(AlternativePlan?,Option?,ModifiedPlan?),*
Realization*(Realization?, AlternativePlan?, Constraints).*

Given a plan, this algorithm produces all possible realizations of a given plan as allowed by the constraints. These realizations can be used to evaluate either the overall success probability or the dynamic expected utility for the given plan. In other words, either of the following robustness measures can be calculated recursively by the above algorithm :

$$R(p) = \Pr(p \cup p^{(1)} \cup p^{(2)} \cup \cdots \cup p^{(i)} \cup \cdots \cup p^{(N)}) \tag{6.12}$$

$$E^*(p) = \sum_{i=1}^{N} \Pr(p^{(i)}) C(p^{(i)}) \tag{6.13}$$

where $p^{(i)}$ signifies the present version of the plan p for the ith version of the state space and $C(p^{(i)})$ stands for the cost of the corresponding plan version. For a problem with n stochastic and binary variables, the state space contains 2^n possible instances, resulting in a maximum number of plan versions or realizations N equal to 2^n.

6.6.2 Strategies for Modification

For each of these instances, the algorithm requires a new version of the plan as the current specification of how to continue within the present state of affairs to attain the desired goals. This evaluation is performed within the subroutine FindOption through the application of a certain modification strategy $\mathcal{F}$, which will output the new plan version $p^{(i)}$ given the current state space and the current plan specification. There are various candidates for these modification strategies. An obvious strategy consists of simply ignoring the problem and insisting on using the original plan specification. This strategy has been called the **null** strategy and is denoted with $\mathcal{F}_n$. Another strategy can be based on the determination of a new optimum plan continuation for the current conditions, based on a reoptimization study. This strategy has been called the **replannning** strategy and is denoted with $\mathcal{F}_r$. Replanning produces the maximally obtainable robustness measures but it could be computationally expensive. Between these two extremes lie a variety of other candidates which demand less computational resources and therefore can be more suitable for the above algorithm. For instance, the **commitment** strategy is based on the continuation of the original plan specification with a simple adaptation to the present state. Ideally, such strategies should produce results that are close to those of the replanning yet should be easy to compute especially in real time during plan implementation. The selection of these strategies ultimately depends on the computational demands of the problem under consideration.

6.6.3 Runtime for the Algorithm

The time it takes to complete the evaluation of a plan through the above algorithm can be calculated recursively as follows: Let $t(n)$ denote the time required for evaluation of a given plan with n actions. Let $a(n)$ denote the time it takes to carry out all the calculations of the first part of the above algorithm and $b(n)$ denote the corresponding time for the second part which includes the determination of the new version of the plan after encountering a failure. Because of recursion,

$$t(n) = (a(n) + b(n))t(n-1). \tag{6.14}$$

It follows that,

$$t(n) = \prod_{i=1}^{n}(a(i) + b(i)). \tag{6.15}$$

If both $a(n)$ and $b(n)$ are constants such that $a(n) + b(n) = T_0$ then $t(n) = T_0^n$. In some problems, however, the time $b(n)$ to complete the evaluation of the second part may not be a constant. For this case, assuming that $a(n) = a_0 << b(n)$, we obtain that $t(n) \approx \prod_i b(i)^i$. The function $b(n)$ depends on the selection of the modification strategy for the subroutine FindOption. Nevertheless, it is apparent that even with a polynomial-time modification strategy, the algorithm produces an exponential number of realizations and therefore takes exponential time with increasing plan size in the absence of any constraints. Incorporation of constraints helps to restrain the exponential number of possible realizations which will otherwise make the algorithm intractable. The algorithm is therefore implemented both with a built-in constraint propagation scheme and cutoff schemes for those realizations whose values exceed certain pre-set thresholds. Below, we examine various versions of such cutoffs.

6.6.4 Cost Cutoff

First of these constraints is a limit on the total cost which simply measures the amount of expendable resources. It is assumed that any action that has been exercised, regardless of the result being success or failure will deplete a certain amount of cost associated with it. It is also assumed that costs are additive. For example, for the concatenation of edges e_1 and e_2 with the corresponding individual costs $c(e_1)$ and $c(e_2)$, the total cost will be $c(e_1 \oplus e_2) = c(e_1) + c(e_2)$. A larger limit on the total cost (expendable resources) provides a larger margin for recovering from errors. Therefore, the robustnessss measures vary monotonically with increased cost limit. Figure 6-2 shows an example of this for the success probability of a typical path. Note that as the cost limit is increased the success probability approaches an asymptotic value.

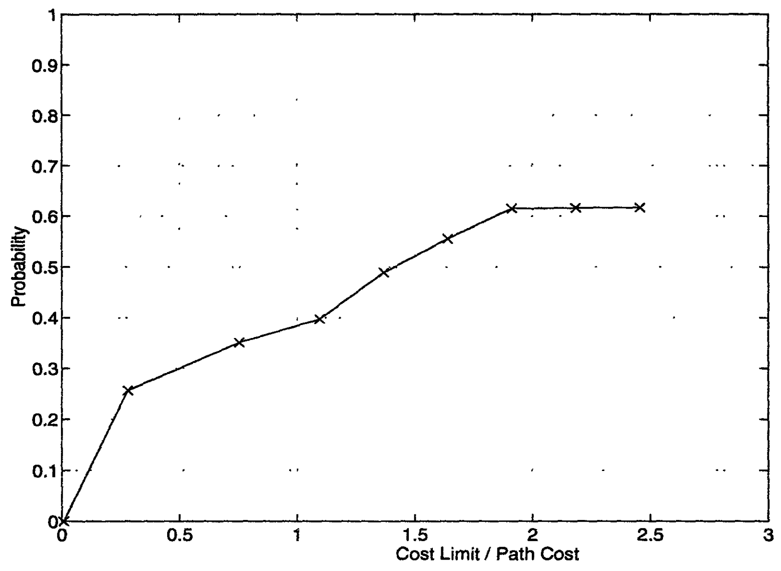


Figure 6-2: *Success probability versus cost limit for a typical path*

6.6.5 Probability Cutoff

Another natural constraint that is used to reduce the computational effort involves a probability cutoff. Because the main goal of the algorithm is to produce an overall probability value or an overall expected utility, it is not desired to examine those realizations with a small marginal utility. Indeed, it is quite impractical to evaluate the success probability or the expected utility to a very fine precision. Generally, when one reaches a few levels down in the event tree, initial probabilities of the alternative paths become very small and contribute very little to the overall success probability of the original plan. Ignoring those alternatives having small initial probability helps reduce the computational effort. Figure 6-3 shows an example of how the overall success probability varies with varying cutoff probability.

Both cost and probability cutoffs result in an underestimation of the desired probability value but they also significantly reduce the amount of computation required. Figure 6-4 shows how the runtime for measuring robustness through the above algorithm increases exponentially with increasing cutoff values in a typical problem.

6.6.6 Depth Cutoff

Beside those constraints such as an upper limit on the total expendable cost and a lower limit on the initial probability of an alternative path, we impose another constraint on the depth of the algorithm as a concession to complexity. Obviously, both cost and probability cutoff methods work similarly to depth cutoff but in a more natural way, for they allow us to probe interesting or important alternatives deeper. Nevertheless, the computational time increases exponentially with increasing depth

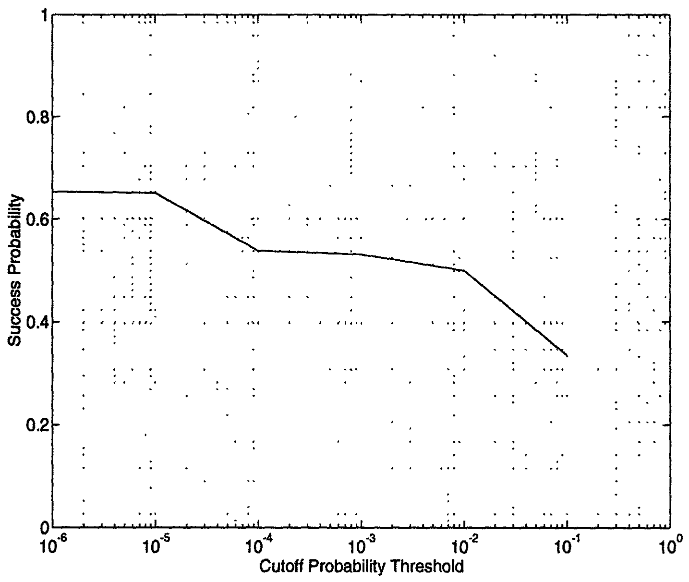


Figure 6-3: *Success probability versus probability cutoff threshold for a typical path*

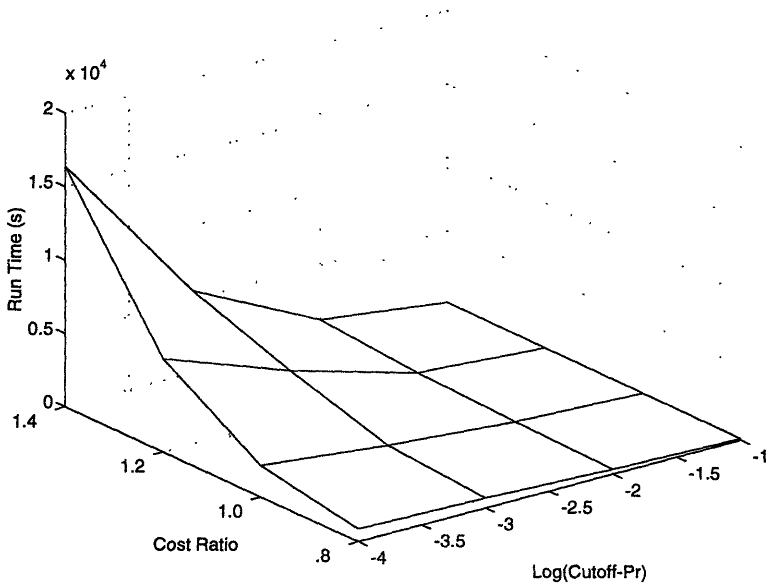


Figure 6-4: *Runtime versus cost and probability cutoff values*

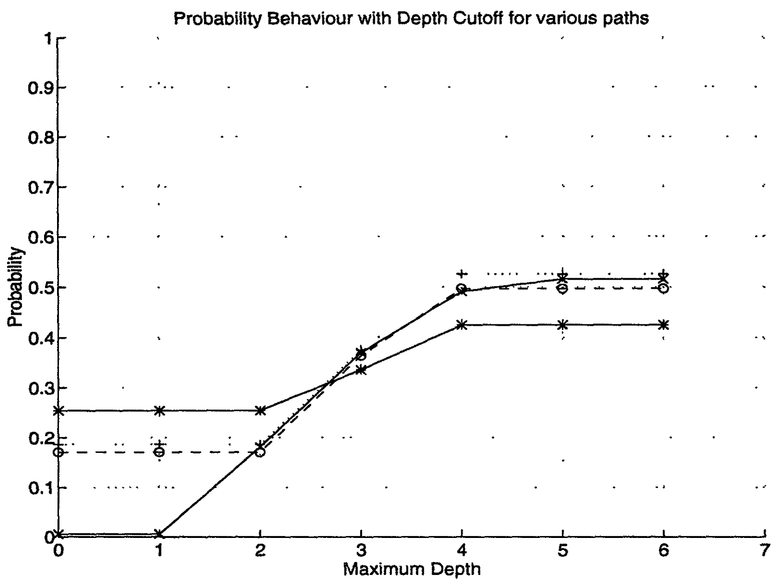


Figure 6-5: *Probability behavior with depth cutoff for various paths*

and therefore a depth limit may be necessary for guaranteeing a bound on the computation time. The following figure 6-5 shows how a depth cutoff affects the overall probability value for various paths in a random problem. It should be noted while a very limited depth value significantly helps in reducing computation time it may severely underestimate the required probability value. On the other hand a large depth cutoff value may not be useful at all because the marginal utilities of modifications carried out in deep levels of a plan (attempting to correct failures attempted for correcting previous failures...) is ignorable whereas the amount of computation for these modifications is tremendous. Note that a depth cutoff limit equal to k implies that there are at most k failures that are to be accounted in the evaluation of the robustness of a given plan.

6.7 Summary and Conclusions

We introduced the concept of robust plans by emphasizing the dynamic nature of planning. We reviewed previous work on diverse areas that could be useful for the development of robust plans. We especially concentrated on dynamic path planning problems. We reviewed the results that show that such problems of dynamic nature generally belong to the complexity class PSPACE. Then, we introduced the robustness measures based on the reliability and the expected cost of a plan; the difference between these measures and conventional ones are due to consideration of adaptation to dynamic events in the robust plans. Later, we showed that these measures can be determined exactly with a depth-first algorithm. Further, this evaluation can be made less computationally demanding by using various cutoff schemes.

Chapter 7

Robust Planning Examples

This chapter examines the generation of robust plans within the context of general path finding problems. We describe in *Section 7.1* four versions of robust path finding problems subject to dynamic optimization criteria. The complexity and the solution methods for these problems are examined in *Section 7.2* and *Section 7.3*. In *Section 7.4*, we compare the robust plans to the solutions optimal under conventional static optimization criteria. GAs, with their property of being particularly efficient for search in moderately complex and moderately nonlinear problem spaces, provide a relatively attractive method for the solution of robust planning problems. We present the results of genetic algorithms in finding solutions to some examples of robust planning problems in path planning. *Section 7.5* examines an NP-hard problem instance while *Section 7.6* examines the application of genetic algorithms to a PSPACE-hard problem instance. *Section 7.7* presents the heuristics that are developed to increase the efficiency of this solution process.

7.1 Problems Considered

Robust planning problems involve the determination of plans optimal with respect to metrics such as the success probability or the expected cost, evaluated over possible realizations of the given plan in a ever-changing environment. Because robustness measures are inherently probabilistic, these problems are posed naturally within the context of probabilistic graph models. Conceptually simple versions of robust planning problems can be derived directly from the most studied deterministic path finding problems, namely the shortest path problem and the traveling salesman problem.

We will refer to the probabilistic version of the shortest path problem that asks for the most robust path in a given graph as the *Probabilistic Shortest Path Problem* and we will denote it with PSPP. This problem can often be made more interesting and more difficult by requiring that some fixed fraction of all the nodes must be visited for a path to be feasible. By requiring that any path specification must include all the nodes of the given graph, we obtain the probabilistic version of the familiar TSP. This problem has been first referred as the *Probabilistic Traveling Salesman Problem (PTSP)* by Jaillet who analyzed it exhaustively in his PhD thesis [77], suggested adaptation of branch & bound scheme for its solution and provided several bounds for the cost of the optimal solution in the case where the nodes are located randomly on a Euclidean plane. Jaillet also [78] describes the *Probabilistic Shortest Path Problem (PSPP)* as follows: *"Consider the problem of finding a shortest path between a node source s and a node sink t in a complete network having a length associated with each arc. On any given instance of the problem, only a subset among intermediate nodes can be used to go from s to t, the subset being chosen according to a given probability law. We wish to construct an a priori path such that, on any given instance of the problem, the sequence of nodes defining the path is preserved but only the permissible nodes are traversed, the others being skipped. The problem of finding a priori path of minimum expected length is defined as a PSPP. "*

The problems we consider are similar to those originally suggested by Jaillet. Our models differ from Jaillet's model in that we ascribe probabilities to the edges of the given graph instead of its nodes. These probabilities signify the probability of reaching from a source node to a given destination node while the edges of graph may be failing. Furthermore, we also allow that a failing edge in a path may still add to the cumulative cost of the path. We also assume that a pair of source and destination nodes are always specified as well as a limit on the maximum expendable cost (i.e. there is a fixed budget). We are interested in finding optimal solutions with respect to both the expected cost and the success probability (of reaching to destination node starting from the source) criteria. Further, we also make a distinction between the **strategies** to be applied when contingencies arise. A strategy dictates how to modify a plan that deviated from its original course.

Adaptation to contingencies generally require a modification strategy that can be quickly implemented. A particularly fast modification strategy is the commitment modification strategy originally suggested by Jaillet. The commitment modification strategy dictates to follow, as long as possible, the order, the original course of the plan. This strategy can be implemented for both problems (SP and TSP) in polynomial time ($O(n)$). A particularly desirable modification strategy is *reoptimization* for the each possible instance of the problem environment.

In the light of this information, we define the following distinct robust path finding problems :

1. PSPP-E : Given a probabilistic graph $G(V, E)$ and commitment modification strategy, find the shortest path having the minimum *dynamically calculated* expected cost. The corresponding recognition problem asks whether there exists a path with the expected cost less than a preset limit.

2. PSPP-R : Given a probabilistic graph $G(V, E)$, the commitment modification strategy and a cost (budget) limit $C_{\max}$ find the path with the maximal probability of succeeding to reach from the

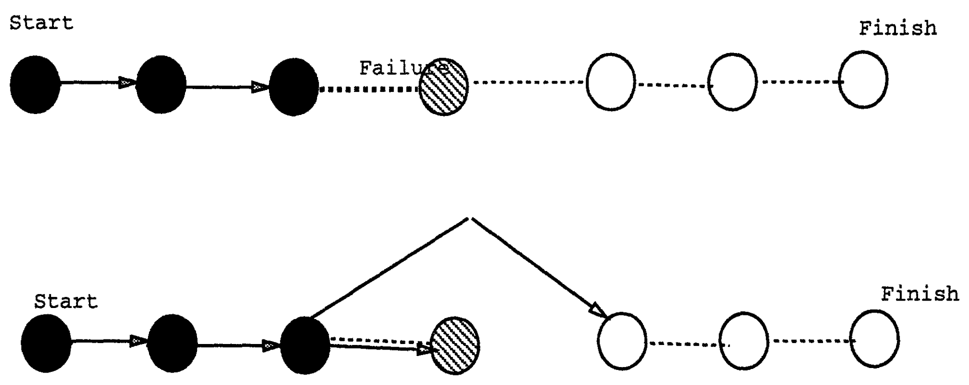


Figure 7-1: *Commitment modification strategy is applied after a failure in a simpel path*

source node to the destination node under the given cost limit under all instantianations. In other words,

$$\max \Pr(A \mid \forall\, i\, C_i < C_{\max}) \tag{7.1}$$

where A denotes the event of reaching to the destination node and C_i denotes the budget spent under the instantianation i. The corresponding recognition problem asks whether there exists a path with the success probability R larger than a preset limit.

3. PTSP-E : This asks the same questions as the above problem PSSP-E; the only difference is that we restrict our attention to Hamiltonian paths.

4. PTSP-R : The same question as the above problem PSPP-R, but again we require that a path must visit all the nodes of the given graph.

Changing the modification strategy from commitment strategy to reoptimization, we obtain a new set of problems. These would be called the *Reoptimized Shortest Path* and *Reoptimized Traveling Salesman Problems*: **RSPP-E**, **RSPP-R**, **RTSP-E** and **RTSP-R**.

7.2 Problems Under Commitment Modification Strategy

Both the PSPP and the PTSP employ the same commitment modification strategy for recovering from failures. This strategy consists of skipping to the next node after an action fails to take the agent to the intended node and leaves her back in the starting node (possibly with irreversible effects on the resources). Commitment modification strategy allows a quick adaptation to failures and simplifies the evaluation of the robustness measures. Figure 7-1 gives an example of the application of the commitment modification strategy for a simple path. In the following, we develop formulas for efficient evaluation of the robustness measures under this strategy.

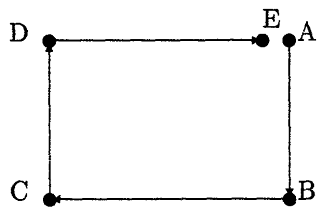


Figure 7-2: *A simple graph*

Table 7.1: Paths of Figure 6.1 classified

0 Nodes	1 Node	2 Nodes	3 Nodes
AE	ABE	ABCE	ABCDE
	ACE	ABDE	ABDCE
	ADE	ACBE	ACBDE
		ACDE	ACDBE
		ADBE	ADBCE
		ADCE	ADCBE

7.2.1 Realizations

Consider the graph in Figure 7.2.1. We assume that the nodes A and E have been specified as the start and the finish nodes. The appearance order of the nodes B, C, and D distinguishes a path from the others. For the PTSP, all of these nodes have to be visited and therefore there exists 3!=6 distinct Hamiltonian paths. For the PSPP, there exists 16 distinct paths; these are shown in Table 7.1 classified with respect to the number of nodes that they visit.

Applying the enumeration technique for finding the most robust path requires evaluation of the robustness measure for each candidate path. This evaluation, when performed through the recursive algorithm, is itself computationally expensive for it explicitly considers all possible realizations of a given path. Consider the path $L_0 \equiv AB - BC - CD - DE$ of the graph in Figure 7.2.1. For the Hamiltonian path L_0 there are 16 distinct realizations corresponding to combinations of 4 different edges each with two different outcomes (*success* or *failure*). Half of these combinations are infeasible as they contain a failure in their last link. The other 8 different realizations can be classified according to the number of failing links that they contain. These realizations are shown in Table 7.2.

Table 7.2: For Tour AB-BC-CD-DE : Realized Tours

0 Failure	1 Failure	2 Failures	3 Failures
AB-BC-CD-DE	AB-BC-(CDC)-CE	AB-(BCB)-(BDB)-BE	(ABA)-(ACA)-
	AB-BC-(CBD)-DE	(ABA)-AC-(CDC)-CE	-(ADA)-AE
	(ABA)-AC-CD-DE	(ABA)-(ACA)-AD-DE	

Suppose that we consider a graph having $n+2$ nodes (wherein two nodes are fixed as start and terminal points and n nodes are variable). For such a graph, there exists $n+1$ edges and 2^{n+1} distinct edge combinations. Half of these combinations comprises infeasible paths as they contain a failure in their last link that arrives at the terminal node. Thus there are 2^n distinct realizations that must be taken into consideration. The number of realizations for a given number k of link failures is equal to $\begin{pmatrix} n \\ k \end{pmatrix}$ where n is the number of variable nodes in the given path. For example, in the above problem there are $\begin{pmatrix} 3 \\ 2 \end{pmatrix}$=3 distinct realizations with double failures. In general, these realizations are distributed binomially such that

$$2^n = \begin{pmatrix} n \\ 0 \end{pmatrix} + \begin{pmatrix} n \\ 1 \end{pmatrix} + \begin{pmatrix} n \\ 2 \end{pmatrix} + \cdots + \begin{pmatrix} n \\ n-1 \end{pmatrix} + \begin{pmatrix} n \\ n \end{pmatrix}$$

This binomial expansion can be useful in evaluating the expected cost or the success probability of a given path.

7.2.2 Success Probability

The success probability for a given path is equal to the sum of the path probabilities for its disjoint realizations. For a path L, the probability R is:

$$R(L) = \sum_i \Pr(L^{(i)}) \tag{7.2}$$

where $L^{(i)}$ represents a possible realization of the path L. These realizations can be obtained through the above described scheme of binomial expansion.

For example, let us assume that all the edges have a uniform edge probability p. The overall success probability R of a path L with $n+2$ nodes, can be written by summing up each element of the binomial expansion:

$$R(L) = \sum_{i=0}^{n} \begin{pmatrix} n \\ i \end{pmatrix} p^{n-i}(1-p)^i \tag{7.3}$$

This expression can be simplified by noting that:

$$\begin{aligned} R(L) &= p\left(\binom{n}{0}p^n + \binom{n}{1}p^{n-1}(1-p)^1 + \cdots + \binom{n}{n-1}p^1(1-p)^{n-1} + \binom{n}{n}(1-p)^n\right) \\ &= p\,(p+(1-p))^n \\ &= p \end{aligned}$$

For example, the success probability $R(L_0)$ for the path L_0 for the above graph of Figure 7.2.1 can be written as the sum :

$$R(L_0) = p^4 + 3p^3(1-p) + 3p^2(1-p)^2 + p(1-p)^3 = p.$$

This reduction is achieved because all the possible realizations (except those half that never reach the terminal node) are feasible in the absence of constraints. Intuitvely, if the budget limit is infinite ($C_{\max} = \infty$), any path can be adjusted so that it eventually arrives at the terminal node. Hence, the overall success probability of a path equals the probability of its last link, which has to be successfully traversed in order to arrive at the terminal node. It turns out that, in this case, there is no need to distinguish among different paths since the probabilities of the edges arriving at the terminal node are equal. For example, for the above example graph, all 6 possible tours have the same probability $R = p$.

An alternative method for obtaining the success probability $R(L)$ relies on the recursive evaluation scheme. The probability $R_{i,j,l}$ of arriving at the node j starting from the node i through a path that skips the nodes $i+1, i+2, \ldots i+l-1$ and visits in order the nodes $i+l+1, i+l+2, \ldots j-1, j$ is equal to the sum of the following terms: the probability that the first edge is successfully traversed, multiplied by the probability of the remaining path and the probability that the first edge fails, multiplied by the probability of the alternative path. Because of the commitment strategy, the alternative path follows the original path only by skipping over the arrival nodes of the formerly failed edges. Then,

$$R_{i,j,l} = p_{i,i+l+1}R_{i+l+1,j,0} + (1 - p_{i,i+1})R_{i,j,l+1} \tag{7.4}$$

with the following boundary conditions:

$$R_{i,j,l} = p_{i,j} \quad l = j = i - 1 \tag{7.5}$$

for l such that all the nodes between the node i and the node j are to be skipped. As the special cases of this boundary condition,

$$R_{i,i+1,0} = p_{i,i+1} \tag{7.6}$$

and

$$R_{i,i,0} = 1. \tag{7.7}$$

The success probability $R_{0,n+1} = R_{0,n+1,0}$ can be easily evaluated with this scheme; however the evaluation takes exponential time. In the course of this evaluation, all the probability values $R_{i,j}$ for $j > i$ are determined. Although there are only $\sum_i^n i = \frac{n(n+1)}{2}$ of these terms, the evaluation of all these terms and therefore the determination of $R_{0,n+1}$ takes time in order 2^n.

The expression for the succes probability is considerably simplified if the edge probability values satisfy the following Bayesian property:

$$p_{ij} = p_{(i \vee j)} = p_{j|i} p_i \tag{7.8}$$

In other words, an edge probability is given by the probability of $p_{j|i}$ reaching to the arrival node j from node i, multiplied by the prior probability p_i of being at node i. It can be shown that, in this case, the success probability of any path between a node i and a node j is simply equal to the multiplication of the probability values p_i and p_j :

$$R_{ij} = p_i p_j \tag{7.9}$$

regardless of the nodes that lie between the node i and the node j. This simply follows from the fact that all the realizations are taken into account in the evaluation of R_{ij}.

If the problem includes constraints on resources or is there exists a limit on the total expendable cost (a fixed budget), the expressions for the success probability cannot be simplified. The above described construction schemes, both the binomial expansion method and the recursive evaluation, however, can be modified for efficient evaluation of the success probability. For example, in the evaluation of the binomial expansion, the contributions by the individual realizations can be summed up only after those combinations having a cost over the limit have been eliminated. Some amount of computation may be saved by discarding those paths with additional failures if they already include a certain combination of k edge failures that will result in exceeding the cost constraint. Thus, we obtain :

$$R = \biguplus_{C < C_{\max}, i} \binom{n}{i} p^{n-i} (1-p)^i \tag{7.10}$$

where $\biguplus_{C<C_{\max}}$ is an operator that sums after eliminating those variations exceeding the cost limit. Also, for the case in which the edge probabilities are different:

$$R = \biguplus_{C < C_{\max}, i} \Pr(L^{(i)}) \tag{7.11}$$

Further, the above construction indicates that the static path probability always constitutes a lower bound on the overall success probability if the resources spent on failed edges are irrecoverable:

$$R \geq \Pr(L_0) \tag{7.12}$$

7.2.3 Expected Cost

The expected cost of a plan is evaluated by summing up the probability weighted costs of each possible realization. For a plan L, the expected cost $E(L)$ is therefore equal to :

$$E(L) = \sum_{i} \Pr(L^{(i)})C(L^{(i)}) \tag{7.13}$$

where $L^{(i)}$ is the ith realization of the given plan.

An easily comprehensible expression of the expected cost follows from the binomial expansion. For the following, we assume that there are no constraints on the expendable cost and all the edges have the same probability value p. Then, the expected cost for a path L is equal to

$$E(L) = \sum_{i=0}^{n} \binom{n}{i} p^{n-i}(1-p)^{i}C(L(i)) \tag{7.14}$$

where $C(L(i))$ is the sum of the individual edge values for the path version with i link failures. For example, for the above graph, the following values are obtained with the assumption that all the edges have the same cost d and there is no cost penalty for the edges that failed: $C(L(0)) = 4d$, $C(L(1)) = 3d$, $C(L(2)) = 2d$, $C(L(3)) = d$. Then, the expected cost for the path L_0 is equal to :

$$E(L_0) = p^4(4d) + 3p^3(1-p)(3d) + 3p^2(1-p)^2(3d) + p(1-p)^3(1d)$$

The same equations are obtained by rearranging these terms in groups of the individual edges. For a general graph with $n+2$ nodes, the expected cost can be calculated as the sum of the cost of each edge multiplied by its appearance probability. The appearance probability of an edge between the node i and the node j is equal to the multiplication of the following terms :

- The probability that the node i is reached by starting from the start node (the node 0). This probability is equal to $R_{0,i}(L)$.
- The probability that the node j is reached by starting from the node i. This probability is equal to $p_{i,j}\prod_{k=i+1}^{j-1}(1-p_{i,k})$ as the nodes between the node i and the node j will have to be skipped.
- The probability that the terminal node (the node $n+1$) is reached from the node j. This probability is equal to $R_{j,n+1}(L)$.

Therefore the expected cost is obtained as:

$$E(L) = \sum_{i=0}^{n} \sum_{j=i+1}^{n+1} d_{i,j}R_{0,i}(L)p_{i,j} \prod_{k=i+1}^{j-1} (1-p_{i,k})R_{j,n+1}(L) \tag{7.15}$$

These equations do not take into account the possibility of constraints on resources, such as a fixed budget. But they allow us to evaluate the expression for the expected cost in polynomial time in order $O(n^3)$ once the dynamic reliability values are determined.

We also develop a recursive equation which can be used for evaluating the expected cost of a partial path. For a path that visits, in order, the nodes 0, 1, 2, ..., k-1, the expected costs at each step are:

$$
\begin{aligned}
E(L_0) &= 0 \\[2ex]
E(L_1) &= (E(L_0) + R_{0,0}d_{0,1})p_{0,1} \\[2ex]
E(L_2) &= (E(L_1) + R_{0,1}d_{1,2})p_{1,2} + \\
&\quad (E(L_0) + R_{0,0}d_{0,2})(1 - p_{0,1})p_{0,2} \\[2ex]
E(L_3) &= (E(L_2) + R_{0,2}d_{2,3})p_{2,3} + \\
&\quad (E(L_1) + R_{0,1}d_{1,3})(1 - p_{1,2})p_{1,3} + \\
&\quad (E(L_0) + R_{0,0}d_{0,3})(1 - p_{0,1})(1 - p_{0,2})p_{0,3} \\[2ex]
E(L_k) &= (E(L_{k-1}) + R_{0,k-1}d_{k-1,k})p_{k-1,k} + \\
&\quad (E(L_{k-2}) + R_{0,k-2}d_{k-2,k})(1 - p_{k-2,k-1})p_{k-2,k} + \\
&\quad (E(L_{k-3}) + R_{0,k-3}d_{k-3,k})(1 - p_{k-3,k-2})(1 - p_{k-3,k-1})p_{k-3,k} + \\
&\quad \ldots \\
&\quad (E(L_0) + R_{0,0}d_{0,k})(1 - p_{0,1})(1 - p_{0,2}) \cdots (1 - p_{0,k-1})p_{0,k}
\end{aligned}
$$

It follows that for a partial path, the expected cost at the node k is equal to:

$$E(L_k) = \sum_{i=1}^{k} (E(L_{k-i}) + R_{0,k-i}d_{k-i,k}) \prod_{j=k-i}^{k-1} (1 - p_{k-i,j})p_{k-i,k} \qquad (7.16)$$

This recursive equation indicates that the expected cost is not a monotonic function, therefore a dynamic programming formulation is not useful for this problem.

7.2.4 Complexity of the PSPP and the PTSP

The complexity of the problems PSPP and PTSP under commitment modification strategy can now be examined in the light of above-developed equations.

First, we assume that there are no constraints on the expendable total cost or on other resources. The elimination of such constraints simplifies the problems for it allows that all the realizations of a given plan are feasible. Hence, the success probability of any plan is determined by its initial conditions. In particular, the success probability of a path is determined by the probability of its terminal nodes, as shown in Equation 7.9, when there are no global constraints. If the edge probabilities are not uniform, the problems PSPP-R and PTSP-R are not trivial either.

The problem is more interesting when we use the other metric, expected cost, for optimization. The expected cost of a path can be found either from Equation 7.15 or recursively from Equation 7.16. Given that the probability values R_{ij} can be evaluated in constant time, both of these equations take running time in order $O(n^3)$, as the total sum of the elementary operations performed is equal to:

$$\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} \sum_{k=i+1}^{j-1} 1 = \frac{1}{6}(n^3 - n). \tag{7.17}$$

These equations are useful in evaluating the complexity of the problems PSPP-E and PTSP-E. By rewriting Equation 7.16 as

$$\begin{aligned} E(L_k) &= E(L_{k-1})p_{k-1,k} + R_{0,k-1}d_{k-1,k} + \sum_{i=2}^{k} (E(L_{k-i}) + R_{0,k-i}d_{k-i,k}) \prod_{j=k-i}^{k-1} (1 - p_{k-i,j})p_{k-i,k} \\ &= E(L_{k-1})p_{k-1,k} + R_{0,k-1}d_{k-1,k} + \gamma(k) \end{aligned}$$

we observe that the expected cost can be stated in terms of the expected cost of the immediatiely preceding state. Note that this equation will reduce to the normal (static) expected cost evaluation when $\gamma(k) = 0$. Therefore, complexity of the PSPP-E is the same as the complexity of the shortest path problem under statically calculated expected cost optimization measure. This last problem, which can be solved with stochastic dynamic programming, is an NP-hard problem under the given model with non-additive objective function as shown in the previous chapter. It follows that the PSPP-E without constraints is NP-hard for the general case.

Because of the global constraint (all nodes must be visited) on TSP type problems, neither the greedy search, nor the dynamic programming algorithm will always yield the globally optimal solution, despite the the $O(n^3)$ recursive equation 7.16. Therefore the PTSP-E is an NP-complete problem as shown below:

Lemma 7.1: PTSP-E is an NP problem.

Proof: We assume that the problem is posed as a query, e.g., *Does a given solution instance L satisfy the condition that $E(L) < E_{\max}$?* The answer can be given in polynomial time by virtue of the equations that evaluate the expected cost in running time $O(n^3)$. Because the solutions can be verified in polynomial time, the PTSP-E is an NP problem.

Lemma 7.2: The TSP is reducible from the PTSP-E.

Proof:

A solution algorithm to the PTSP-E can be used to solve the TSP. In order to obtain the solution to the TSP from the PTSP-E solution algorithm, we only need to assign unity to the edge probability values. Rewriting Equation 7.15 as

$$\begin{aligned} E(L) &= \sum_{i=0}^{n} \sum_{j=i+1}^{n+1} d_{i,j} R_{0,i}(L) p_{i,j} \prod_{k=i+1}^{j-1} (1 - p_{i,k}) R_{j,n+1}(L) \\ &= \sum_{i=0}^{n} \sum_{j=i+1}^{n+1} d_{i,j} \alpha(L,i,j) \\ &= \sum_{i=0}^{n} d_{i,i+1} \alpha(L,i,i+1) + \sum_{i=0}^{n} \sum_{j=i+2}^{n+1} d_{i,j} \alpha(L,i,j) \end{aligned} \tag{7.18}$$

we can observe that the TSP is a special instance of the PTSP-E in which all the probabilities are equal to 1. In this case, $\alpha(L,i,i+1) = 1$ and $\alpha(L,i,j) = 0$ for $j > i+1$ and therefore,

$$E(L) = \sum_{i=0}^{n} d_{i,i+1} \tag{7.19}$$

which is the cost metric of the TSP.

It follows from Lemma 7.1 and Lemma 7.2 that

Theorem 7.1:

The PTSP-E is an NP-complete problem.

Complexity of constrained problems

Finally, we examine the complexity of these problems under constraints on the total expendable cost and non-uniform edge probabilities. Under such constraints, the probability values R_{ij}, hence the expected cost values, take exponential time to evaluate because all the possible realizations will have to be examined (in the worst case). As the number of these realizations is equal to 2^n, determining which alternatives are feasible and which ones are not is an intractable task. Because the robustness measures cannot be evaluated in polynomial time, these problems do not belong to NP. Papadimitriou [106] shows that the recognition versions of many dynamic optimization problems can be reduced to RSAT (stochastic satisfiability) in polynomial time and therefore they belong to the complexity class PSPACE. The problems in PSPACE are distinguished by their requirements of time exponential yet memory (space) polynomial in problem size. Following Papadimitriou's conclusions, it can be shown that these dynamic optimization problems are as hard as RSAT. In these problems, there is an exponential number of realization for any given path, yet any alternative can be discarded from the physical memory after its contribution is taken into account in the sum operator (for an overall probability or expected cost value). Because the evaluation of fitness function takes exponential time and polynomial space in computer resources, the general dynamic optimization problems are PSPACE-hard.

7.3 Problems Under Reoptimization Strategy

While the commitment strategy expects loyalty to be shown to the original plan as much as possible, the reoptimization strategy assumes that an optimum continuation should be determined after each failure. The equivalent problems to the PSPP and the PTSP under the reoptimization strategy have been called the RSPP and the RTSP. An example for this problem is an instance of TSP in which the points are located along the corners of a convex polygon on a Euclidean plane and a uniform probability is associated with all the edges. With this model, the conevex polygon is the optimum solution for the TSP, the PTSP-E and the RTSP-E. The convex polygon is an optimum solution to the RTSP-E because all the instances in which some nodes are dropped out are still optimal for the corresponding subproblem.

The RSPP has the same complexity with the PSPP (both are NP-hard), because one can find the reoptimized path after each failure in polynomial time simply by using Dijkstra's algorithm. On the other hand, the RTSP is a significantly harder problem than the PTSP. The evaluation of the fitness values for each solution instance takes exponential time because a new optimum TSP tour has to be evaluated for each realization. Although, it is intractable to prove that a solution is optimum for the RTSP, it is easy to diagnose (verify) that a solution to the RTSP does not satisfy the optimality criteria. This would require only finding a counterexample that has a smaller cost for a possible realization of the plan. Therefore, the recognition version of the RTSP belongs to the complexity class **co-NP** which consists of problems in which the negation of an hypothesis is verifiable in polynomial time. Because of the doubly exponential time required to evaluate the fitness of a solution, the RTSP is an infeasible problem.

Fortunately, good solutions to the RSPP and the RTSP can be obtained by solving the PSPP and the PTSP. It is clear from the definitions of the RTSP and the RSPP that the optimal solutions for these problems also constitute the optimal solutions for the corresponding problems under the commitment modification strategy. Evaluating the sub-optimal solutions under the commitment strategy results in an underestimation of the values that would be found under the reoptimization strategy. However, the amount of underestimation decreases as these solutions "get closer" to the optimum solution. Therefore, the reasonably good solutions to general reoptimization problems can be found by solving those problems under the commitment strategy.

In order to observe the convergence between the commitment and the reoptimization strategies, we

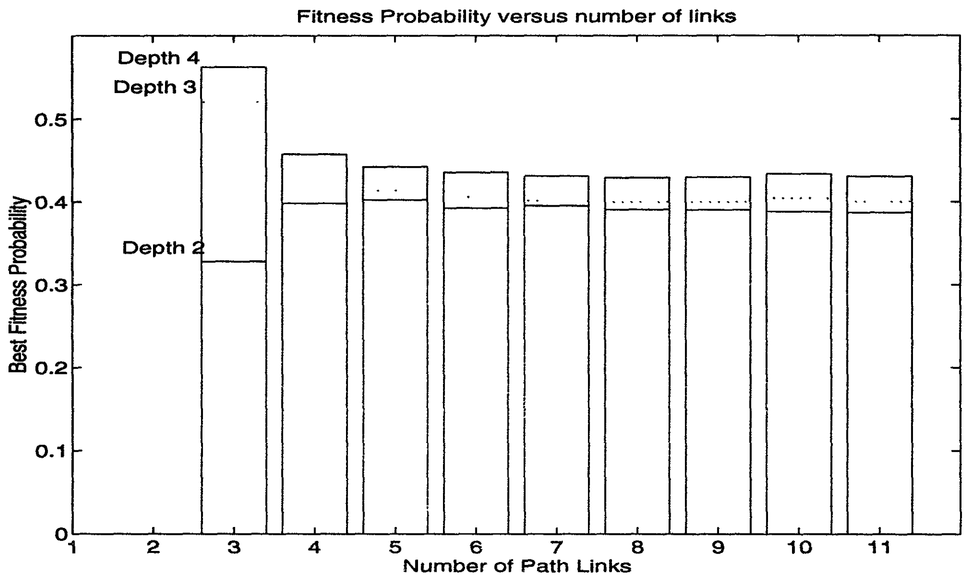


Figure 7-3: *Fitness probability versus number of links*

solved the RSPP and the PSPP for the same graphs. The use of randomly constructed graphs, however, created some unanticipated problems. In these examples, the best solutions were very easily obtainable because of the following special characteristics of the examined problems : (1) In these graphs, the best paths are generally those with a minimum number of edges. (2) The generation of a set of paths is highly biased towards those paths with a minimum number of edges. Figure 7-3 shows the distribution of the best fitness for paths classified according to the number of edges they contain. As can be seen from the figure, for a depth limit larger than three, shorther paths have higher fitness values. Figure 7-4 shows the density distribution of all paths versus the number of edges a given path contains. If a set of paths were to be generated in accordance with this distribution it would be unlikely to obtain those paths with a small number of edges and a high fitness. Fortunately, the construction of a path through depth search is biased toward those paths with a minimum number of edges (minimum depth). It is also shown in Figure 7-4 that Monte-Carlo construction methods produce a set of paths following a distribution that favors paths with shorter edges. Thus, it becomes quite likely that the best path will be among a set of randomly constructed individuals. Because of this condition, the Monte-Carlo method could find the best paths without much difficulty. Thus, we concluded that the unconstrained path finding problems for the above type of graphs are not well-suited for optimization purposes.

On the other hand, the above results cannot be generalized as there exist many types of graphs for which the Monte-Carlo method may not be efficient because the robust paths are not relatively simple and therefore have a low probability of being drawn out of the set of all paths. For example, for a randomly constructed graph having 15 nodes and 11061 paths, the most robust path contains 12 links and is not found easily with random generation.

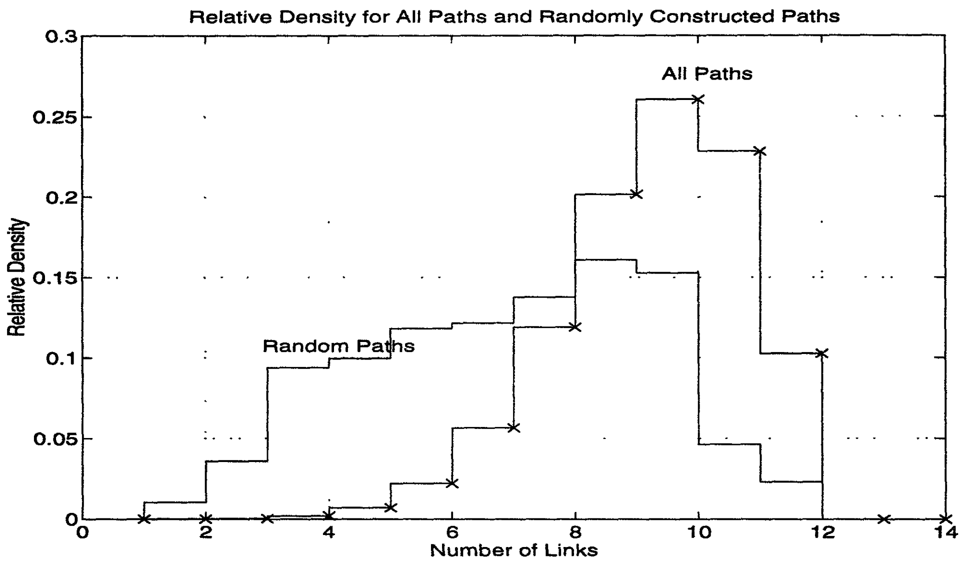


Figure 7-4: *The density distribution of all paths and a set of randomly constructed paths classified according to the number of links*

The above comparisons lead us to the following conclusions: **(1)** The GA may be over-powerful for some type of problems for which the robust paths have a relatively simple structure and therefore can be found more efficiently with the Monte-Carlo method. **(2)** It could be more appropriate to employ the GA for those versions of the path finding problems structured such that simple paths are eliminated from consideration. These problems could be those that include a constraint on the number of nodes visited. A typical example is the TSP problem in which all the nodes have to be visited.

In order to continue with our comparison of the PSPP and the RSPP, we restricted our attention to those problem instances in which at least half of the nodes must appear in a path. The experiments were performed for 3 distinct, randomly-constructed graphs with 10, 16 and 20 nodes with a uniform edge probability value. The GA was run for each of these graphs 10 times under the commitment strategy and 10 times under the reoptimization strategy. For the graph with 10 nodes, a unique solution instance has been converged in all cases whether under the reoptimization or the commitment strategy. For the graph with 16 nodes, the best solution obtained under the reoptimization strategy has also been obtained under the commitment strategy 8 times. The other two solutions were near-optimal. However, for the graph with 20 nodes, the optimum solution has been obtained only 3 times under the commitment strategy. These experiments show that, as a graph gets more complex, the divergence between the solutions of the PSPP-constrained and the RSPP-constrained may increase.

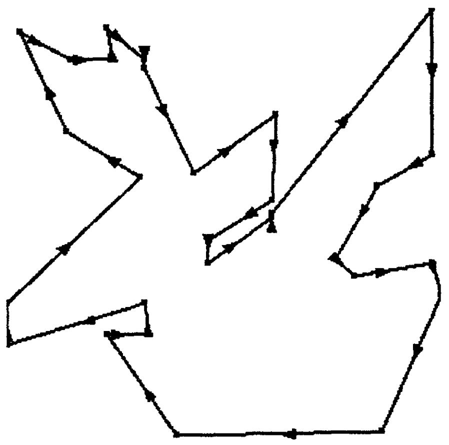


Figure 7-5: *Optimal solution to a random TSP with 30 nodes. Note that the tour may cross itself.*

7.4 Robust paths versus statically optimal paths

Optimization over robustness criteria may result in solutions that could be distinctly marked from those found under conventional optimization criteria (such as *static* expected cost or *static* reliability).

Figure 7-5 shows the optimal solution to a TSP problem with 30 nodes located randomly on a plane. By assigning a value p to the probability of each edge, a PTSP-E problem can be posed. The optimal solution to the PTSP-E is shown in Figure 7-6 for $p = 0.01$. Contrasting these solutions indicates that the PTSP-E solution could be quite different than that of the TSP.

The difference between the TSP and the PTSP-E may be observed better through a well-structured problem. An example is shown shown in Figure 7-7 with the optimal TSP solution for a *concentric* graph with 30 nodes. Let us denote the *dynamic* expected cost of this solution $E(1)$. Another possible tour (which is neither the optimal solution for the TSP nor for the PTSP-E) is shown in Figure 7-8. Let us denote the *dynamic* expected cost of this solution $E(2)$. The ratio of the expected costs depend on the probability values of the edges of the graph. We assume that all edges have the same uniform probability value. The ratio $\frac{E(1)}{E(2)}$ is plotted on Figure 7-9 for varying edge probability and graph size. As can be seen from Figure 7-9 the optimal TSP solution (1) performs quite well for the PTSP-E problem if the edge probability values are large. However at lower probability values the expected cost of the solution 1 is larger than the expected cost of solution 2. The difference is not large (less than 10%) and decreases again when the probability values get smaller; however it is striking in the sense that a bad TSP solution may yield good solutions for the PTSP-E.

It is important to distinguish those cases when the optimal solution to a statical problem constitute a good solution for the corresponding dynamic problem. Because optimization under conventional static criteria may be computationally less demanding than the dynamic optimization, it could be useful whether we can rely on optimal solutions of a static problem. If the solutions found under such criteria may be satisfactorily robust, there would be no need for a a detailed dynamic evalaution for robustness. By rewriting the robustness criteria as

$$\begin{aligned} R(L) &= \sum_{i=0}^{2^n-1} \Pr(L^{(i)}) \\ &= \Pr(L^{(0)}) + \sum_{i=1} \Pr(L^{(i)}) \end{aligned} \tag{7.20}$$

$$\begin{aligned} E(L) &= \sum_{i=0}^{2^n-1} \Pr(L^{(i)})C(L^{(i)}) \\ &= \Pr(L^{(0)})C(L^{(0)}) + \sum_{i=1}^{2^n-1} \Pr(L^{(i)})C(L^{(i)}) \end{aligned} \tag{7.21}$$

we observe that the first term of the sum in the RHS of these equations are the terms used for determination of the value of a solution in a static calculation. The other terms take into account the contribution of the realizations having failures. When the first term dominates over the rest, we may expect that the static optimization criteria can produce solutions that are satisfactorily robust. This condition is satisfied when the probability of the divergent realizations are small or the cost of these realizations are ignorable compared to the cost of the original plan.

On the other hand, solutions found under static optimization criteria could be arbitrarily bad under robustness criteria. Consider a case in which a backtracking from a failed edge e_{ij} has a cost equal to $d_{ji} = M$. It is obvious that the expected cost of any path (even if it is optimal under static optimization criteria) that includes such a failing edge may be arbitrarily increased by choosing for M a large value.

A comparison between the dynamic and static optimization criteria can be easily observed by viewing the whole problem space. We perform this comparison for both the success probability and the expected cost criteria for purposely constructed (regular) graphs and a set of random graphs.

7.4.1 Comparison of Solutions for Success Probability

It was found out that under distinct edge probabilities and the resource constraints, the success probability must be found by summing up all the probability terms belonging to the possible realizations of a given plan. The number of these realizations grows exponentially with the problem size. On the other hand, the contribution by the realizations that include a large number of failures are usually ignorable. Using a depth limit in the recursive algorithm can result in a significant reduction in the number of realizations that are accounted for. In particular, for the depth limit k, we account for

$\begin{pmatrix} n \\ k \end{pmatrix} = \frac{n!}{(n-k)!k!}$ distinct realizations in the calculation of the robustness measure for a given plan. This depth limit also corresponds to the number of failures and backtrackings that are allowed. *A depth 0 calculation yields the static probability of a given plan.* Figure 7-10 shows the behavior of the success probability versus depth limit for all paths in a small graph. These paths are sorted according to their static probability. The following observations can be made about the success probability:0 **1)** The success probability increases with increasing depth limit. Further it converges to an asymptotical value, **2)** This increase is not proportional with the probability values found with lower depth threshold. As a result, the shape of the solution space may change considerably. Some paths that previously ranked well perform worse and some others that previously did not rank well perform better when more failures are accounted for. And, **3)** The differences between the values of the success probability of distinct solutions become less marked at the converged values. We can expect this in most graphs with normally distributed data, but we must note that it is always possible to assign special values such that significant deviations can be observed.

7.4.2 Comparison of Solutions for Expected Cost

We compare the dynamic and static expected cost values of a set of Hamiltonian paths for a regular graph. The graph used is depicted in Figure 7-11 for 12 nodes. These graphs have their nodes located along a circle and are called circular graphs. They have the special property that the optimum path under static optimization criteria also constitutes the most robust path. This property allows us to test the correctness of the the GA algorithm. Even though the optimum path is the same for both dynamic and static cases, there is a wide difference between these two as can be seen from Figure 7-12. Figure 7-12 presents the expected cost (utility) values of all paths sorted in decreasing order with respect to their static expected costs. Note that deviations for the dynamic case are relatively smaller, as can be expected because of recovery from failures.

7.5 GA Runs for the PTSP-E

In the PTSP-E, the aim is to find a Hamiltonian tour or path with the minimum *dynamic* expected cost. This objective function can be evaluated in polynomial time through Equation 7.15, once the reliability values between the node pairs (s, v) and (v, t) (where v is any node in the graph and s and t denote the source and sink nodes) are known. In the following, we present the results of GA runs for the PTSP-E for a set of special problems with 15, 20, 25, 30 and 40 nodes. In all these examples, the graph is circular (i.e. the nodes are located uniformly along a circle) (see Figure 7-11) and the edge probability values are equal to 0.5. Because the edge probability values are uniform and no constraint is specified, we can simplify the reliability values between any given pair of nodes. These values are

Table 7.3: *The Parameters of the Performance Model of the GA for PTSP-E Problems (50 individuals)*

Problem Size	α	$y(0)$	Average Correlation
15	0.2560	1.79	0.956
20	0.1630	2.26	0.989
25	0.1314	2.48	0.979
30	0.0933	3.23	0.996
40	0.0660	4.23	0.995

determined at the outset of GA run and saved in a table for further reference. The expected cost of a path is found through Equation 7.15. Because of the special structure of the graph, the optimal solution to the PTSP-E is identical to the TSP, *it is a circle.* Thus, the GA results can be denoted by the convergence ratio (the ratio of the present expected cost to that of the optimal solution).

Figure 7-13 shows the GA results for the problem where the circular graph has 15 nodes,
Figure 7-14 shows the GA results for 20 nodes,
Figure 7-15 shows the GA results for 25 nodes,
Figure 7-16 shows the GA results for 30 nodes,
Figure 7-17 shows the GA results for 40 nodes.

The average expected cost of a generation in these runs agrees well with an exponential decay. The following equation yields satisfactory correlation values for the observed data:

$$\frac{\bar{y}(t)-1}{\bar{y}(0)-1} = e^{-\alpha t^{2/3}} \tag{7.22}$$

where t is the generation number, $\bar{y}(t)$ is the average convergence ratio expected for generation t and α is a parameter that depends on the problem size and the population size.

Figure 7-18 shows comparisons of the expected convergence ratio to the best-fit exponential decay curve. The numbers in paranthesis denote the correlation coefficient between the data and the curve. The correlation coefficients as well as the α and $\bar{y}(0)$ values are also shown in Table 7.3:

The parameter α appearing in the exponential decay curves depends on the problem size. Figure 7-19 suggests that it is inversely linearly proportional with the problem size, i.e.,

$$\frac{1}{\alpha} = c_1 n + c_0 \tag{7.23}$$

where n is the problem size (the number of nodes in the graph).

The parameter α also depends on the population size. A new set of experiments that were performed with a twice larger population (PopSize=100) indicate that α depends on the population size S as

$$\alpha(S) = \alpha(S_0)\frac{\log S}{\log S_0} \tag{7.24}$$

We spare the reader from a new set of figures of GA runs performed with a population size of 100, since these run results are quite similar to the previous set of figures (7-13, 7-14, 7-15, 7-16, 7-17) calculated for a popluation size of 50. The same performance model (Equation 7.22) still yields a good fit to the observed data (although an exponent equal to 0.5 for the generation number t will yield better correlation values in larger problem sizes). The ratio of the new set of α values ($\alpha(100, n)$) to the previous set of α values ($\alpha(50, n)$) are plotted in Figure 7-20 and compared to the $\frac{\log 100}{\log 50}$. The figure suggests that

$$\frac{\alpha(100)}{\alpha(50)} \approx \frac{\log 100}{\log 50}$$

and thus confirms that Equation 7.24 is reasonable.

Combining these results, we can rewrite the performance model for the PTSP-E as:

$$\frac{\bar{y}(t) - 1}{\bar{y}(0) - 1} = S^{-\frac{t^{0.66}}{c_1 n + c_2}} \tag{7.25}$$

where S is the population size used, n is the problem size and t is the number of generations. This result can be compared to the performance model developed for GAs for the benchmark TSP problems (Equation 4.11). (Note that in that model S denoted the problem size and n denoted the population size: opposite of the present model !). The differences between these two models are small: **1)** For the benchmark TSP problems, the exponent of generation number t was 0.5, while in the present model for the PTSP-E problems it equals to 0.66. **2)** On the other hand, that model (Equation 4.11) yields the expected value of the convergence ratio for the designated *best* individual while this model (Equation 7.25) yields the expected value *averaged* over the population at generation t. Futher experiments show that the best designated individual can also be modelled in the same way. This value, while at the first few generation of a GA run could have large deviations from the average expected value, eventually settles at a convergence ratio 40%-50% lower than that of the average.

Finally, we examine the time required for evaluationg the fitness of an individual solution in these GA runs. Figure 7-21 indicates that a perfect correlation is obtained when a polynomial of the third degree $O(n^3)$ is used for the runtime of the fitness function (where n is the problem size). This result is consistent with the previously developed formula (Equation 7.17).

In summary, the PTSP-E with the uniform edge probability values can be solved with the GA in a similar fashion to the process used for solving the TSP. Both of these problems are NP-hard and the differences arise basically because of the time required evaluating the fitness of an individual. Whereas a solution can be evaluated in $O(n)$ time in the TSP, it takes $O(n^3)$ time in the above instance of

PTSP-E. Thus, the expected time to reach a certain convergence ratio through the GA is $O(n^2)$ longer for the case of the PTSP-E in relative to the TSP case.

7.6 GA Runs for PTSP-R

This section presents the use of GAs for the PTSP-R in which evaluating the objective function takes time exponential to the problem size. While this evaluation can be performed in polynomial time for special cases, such as when all edge probabilities are the same, we concentrate on the general case for which the success probability values are calculated through the depth-first algorithm of Chapter 6. This allows us to examine the performance of GAs for problems that are even harder than NP-hard problems.

We examine four PTSP-R problems in circular graphs with stochastic edges where the edge probabilities are set randomly. These problems are denoted with PTSP-R-9, PTSP-R-10, PTSP-R-11 and PTSP-R-12, the digits representing the number of nodes contained in the respective problem. Since both the start-node and the finish-node are specifed, the number of tours in any of the above problems with n nodes is equal to $(n-2)!$. Relatively low numbers of nodes in these problems have been chosen to reduce the computational time and also to find the most robust solution through enumeration. The goal in these problems is to find a path with maximum success probability under a fixed budget. The convergence ratios for these problems have been calculated by finding the best solution through exhaustive enumeration (hence the small problem sizes). The results of the GA runs performed with a population size of 70 are shown in the following figures:

Figure 7-22 for PTSP-R-9,

Figure 7-23 for PTSP-R-10,

Figure 7-24 for PTSP-R-11,

Figure 7-25 for PTSP-R-12.

These figures indicate **1)** the GA results exponentially converge to toward the best solutions and **2)** and their performance superior to the Monte-Carlo technique which was performed by removing the crossover operation of the GA.

Figure 7-26 shows the distribution of the fitness probability for these problems as they change with the increasing generation number.

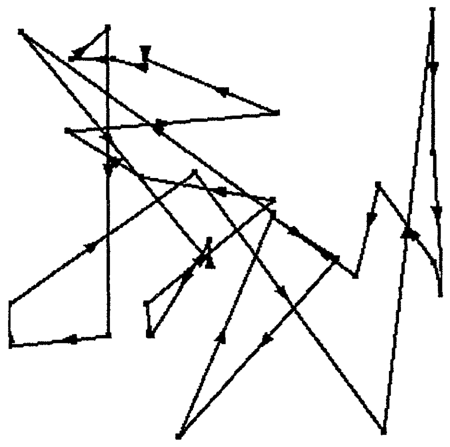


Figure 7-6: *Optimal solution to a random PTSP-E with 30 nodes*

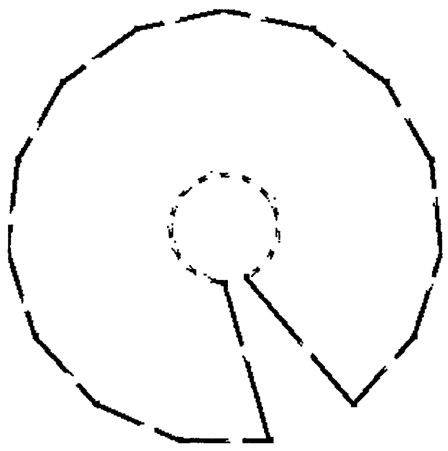


Figure 7-7: *Optimal solution to a concentric TSP with 30 nodes (Solution 1)*

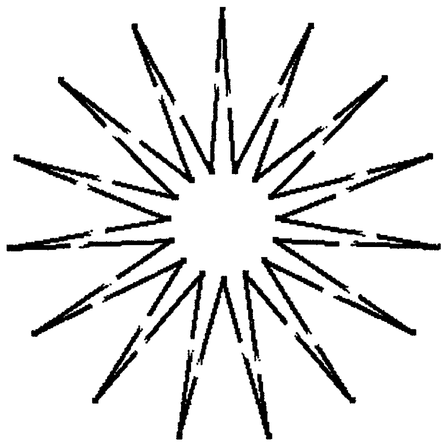


Figure 7-8: *A tour for the concentric graph with 30 nodes (Solution 2) that is a good solution to PTSP but a bad one for TSP.*

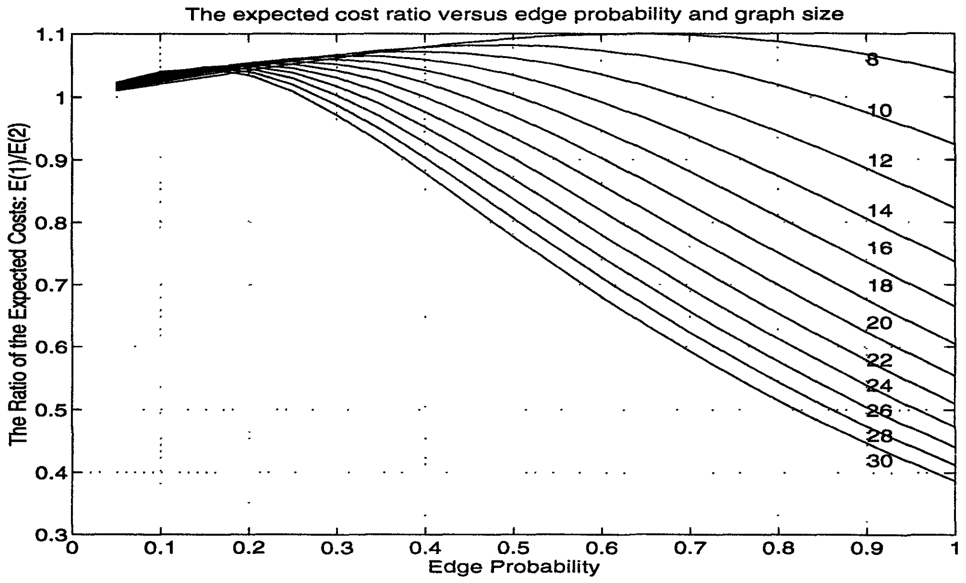


Figure 7-9: *The ratio of the dynamic expected cost of solutions 1 and 2 for the concentric graph.*

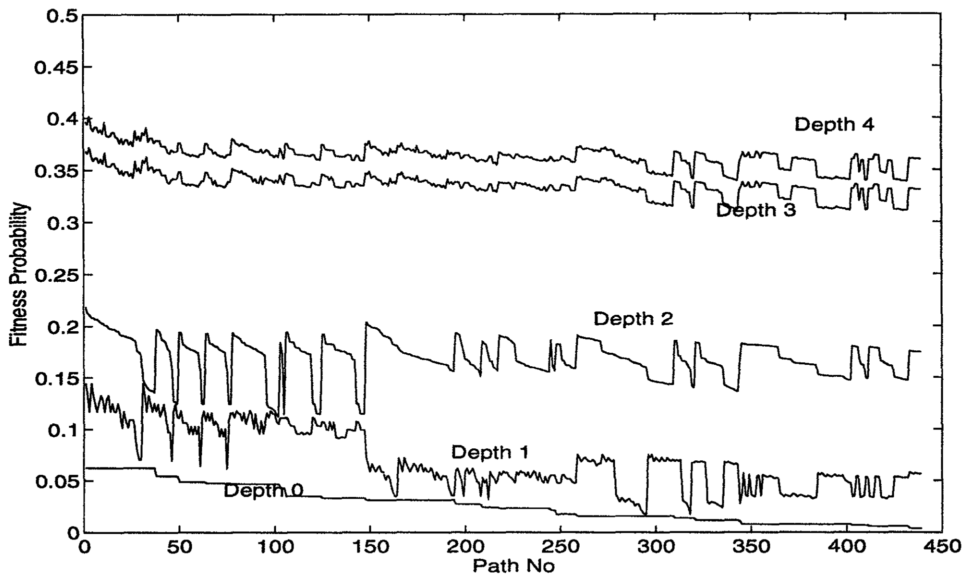


Figure 7-10: *Change in success probability with increasing depth limit*

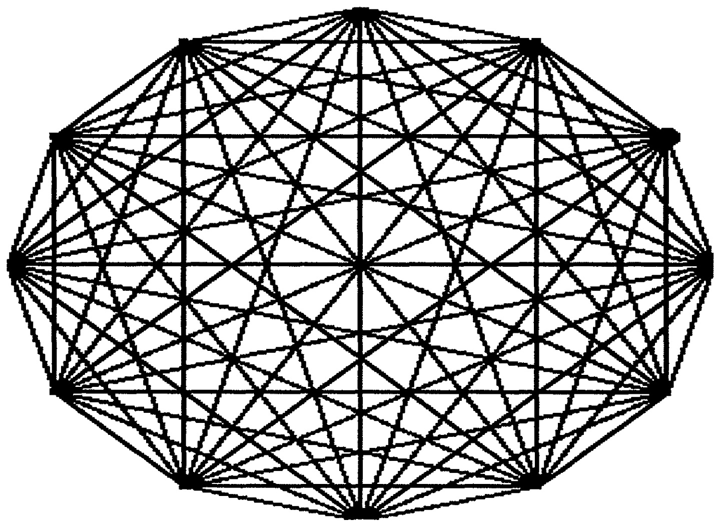


Figure 7-11: *A circular graph used in robust path problems*

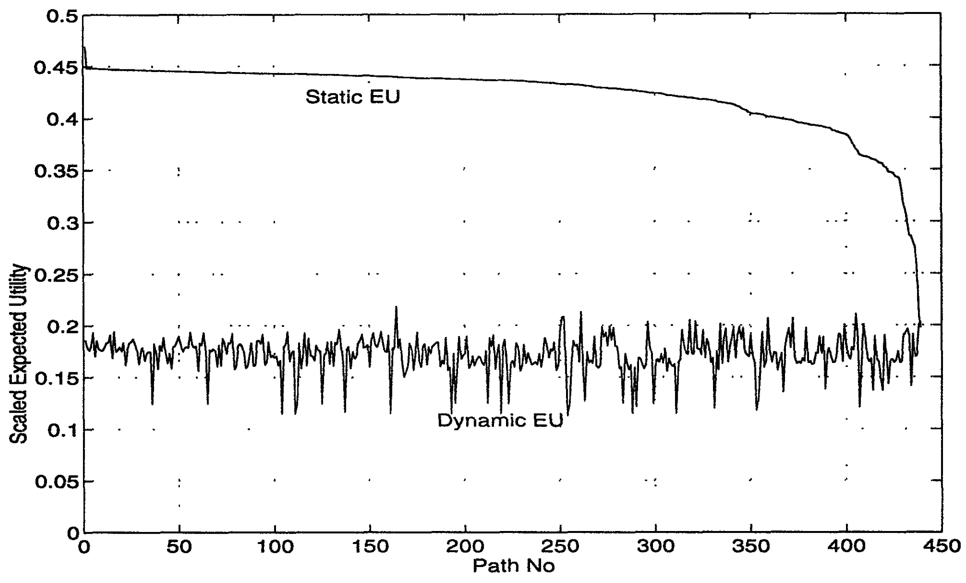


Figure 7-12: *Comparison of normalized expected utility values for dynamic and static cases*

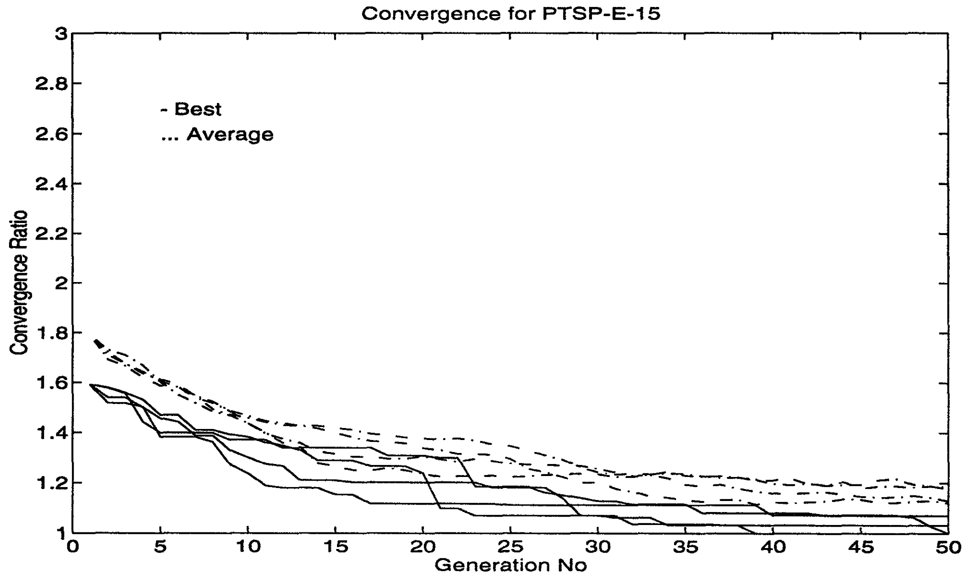


Figure 7-13: *GA runs (PopSize=50) for finding a Hamiltonian path with minimum dynamic expected cost in a circular graph with 15 nodes (PTSP-E-15)*

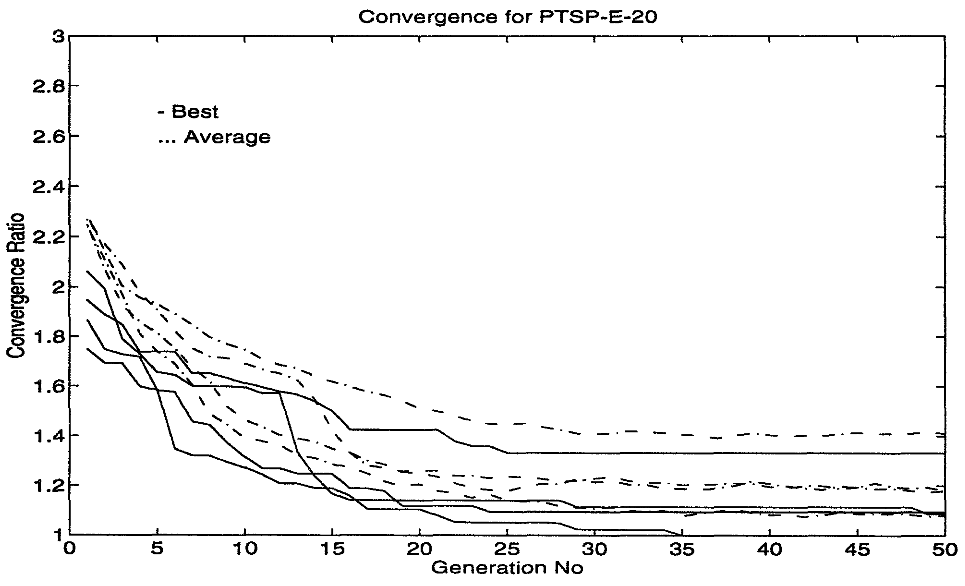


Figure 7-14: *GA runs (PopSize=50) for finding a Hamiltonian path with minimum dynamic expected cost in a circular graph with 20 nodes (PTSP-E-20)*

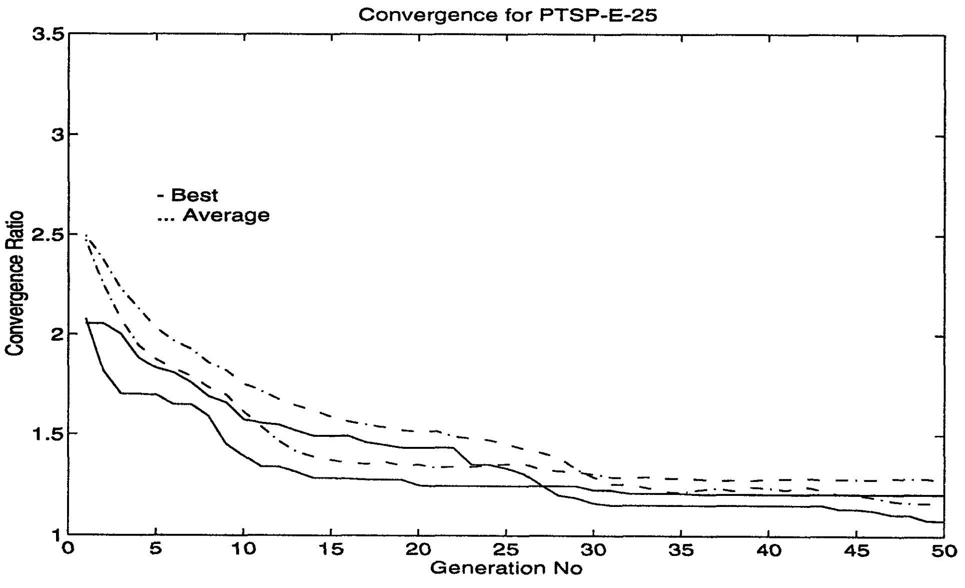


Figure 7-15: *GA runs (PopSize=50) for finding a Hamiltonian path with minimum dynamic expected cost in a circular graph with 25 nodes (PTSP-E-25)*

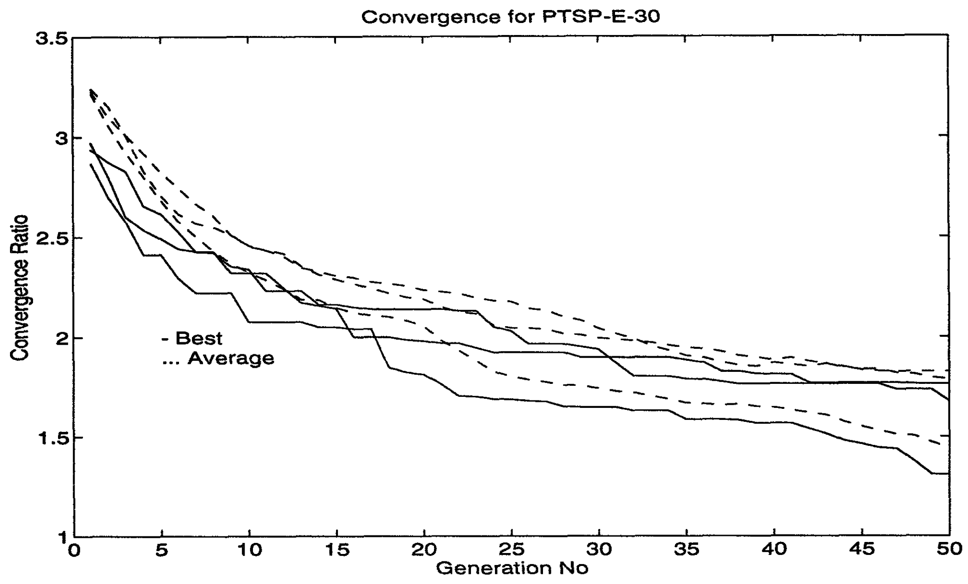


Figure 7-16: *GA runs (PopSize=50) for finding a Hamiltonian path with minimum dynamic expected cost in a circular graph with 30 nodes (PTSP-E-30)*

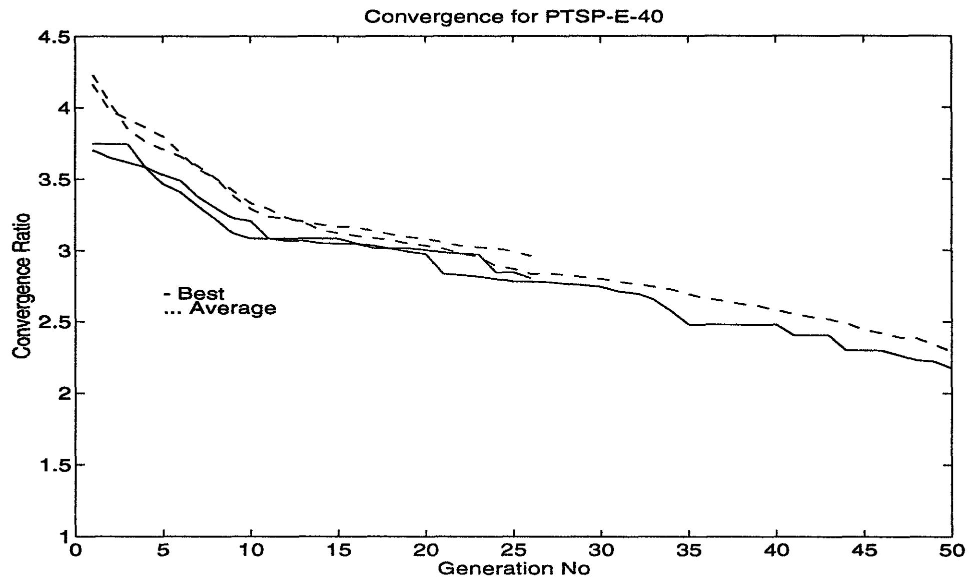


Figure 7-17: *GA runs (PopSize=50) for finding a Hamiltonian path with minimum dynamic expected cost in a circular graph with 40 nodes (PTSP-E-40)*

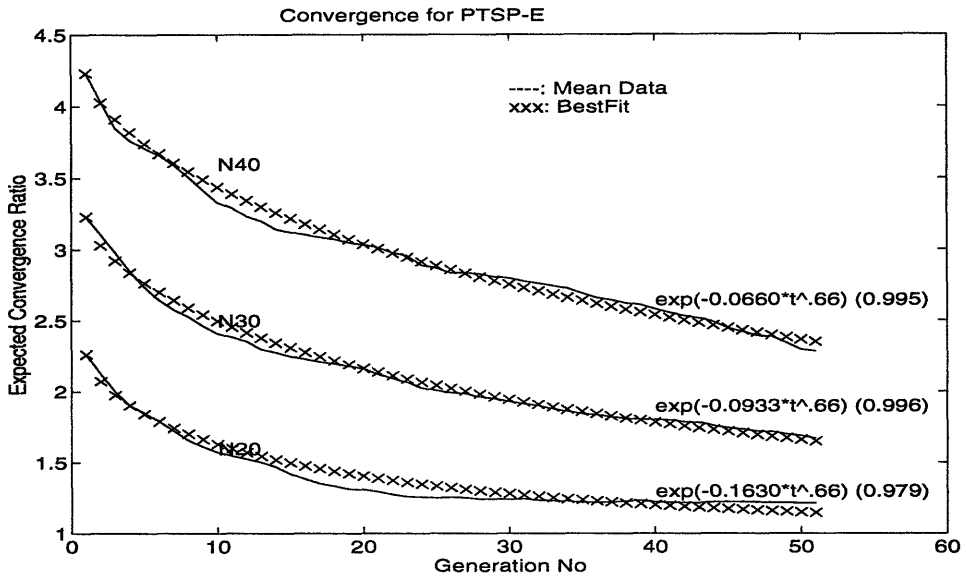


Figure 7-18: *Comparison of expected convergence rate (average of a generation) to the best fit curves for different problem sizes*

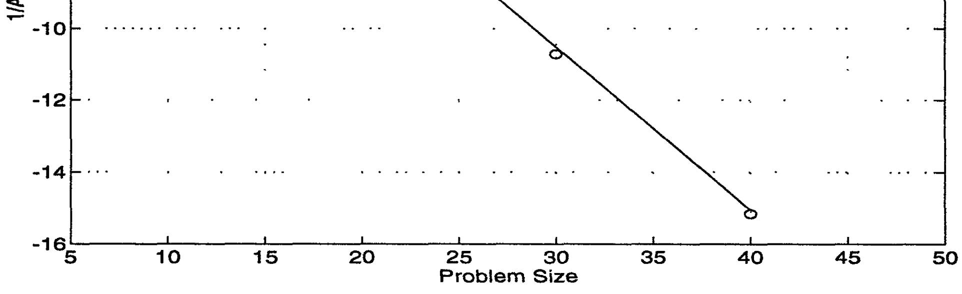


Figure 7-19: *Dependence of the parameter α to the problem size in GA runs for PTSP-E*

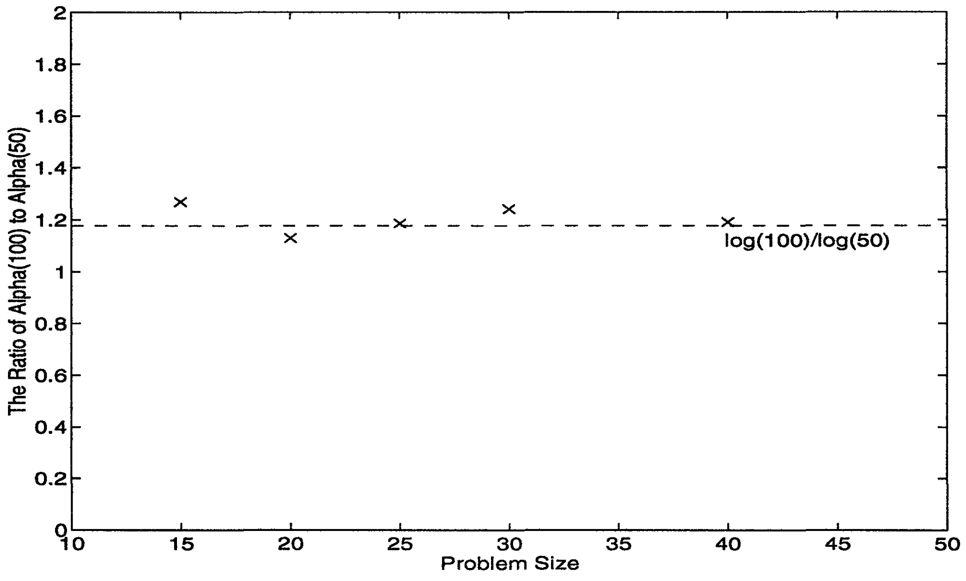


Figure 7-20: *The ratio of* $\frac{\alpha(100)}{\alpha(50)}$ *in GA runs for PTSP-E*

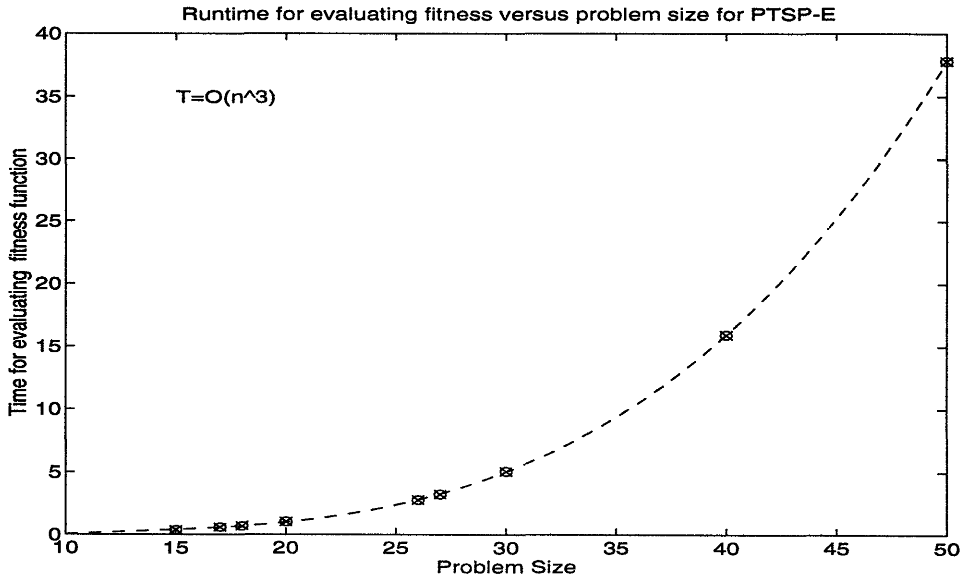


Figure 7-21: *Runtime for the fitness function for PTSP-E*

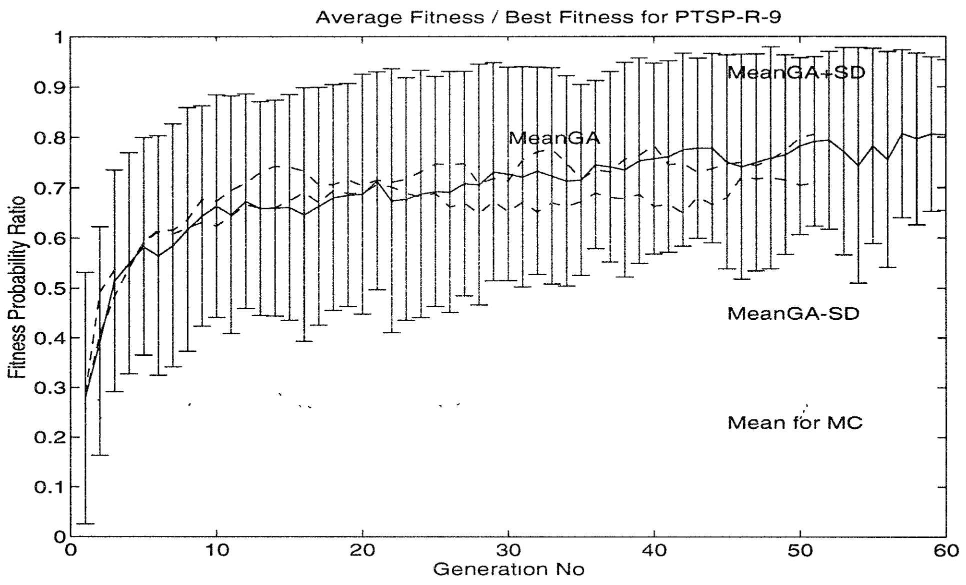


Figure 7-22: *The GA convergence compared to Monte-Carlo for maximizing the success probability in regular graphs with 9 nodes and probabilistic edges*

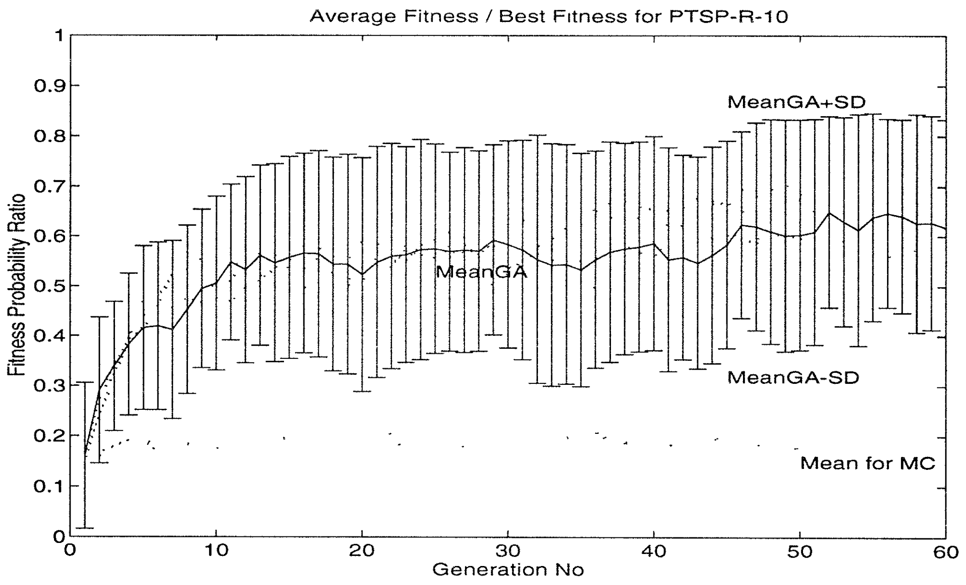


Figure 7-23: *The GA convergence compared to Monte-Carlo for maximizing the success probability in regular graphs with 10 nodes and probabilistic edges*

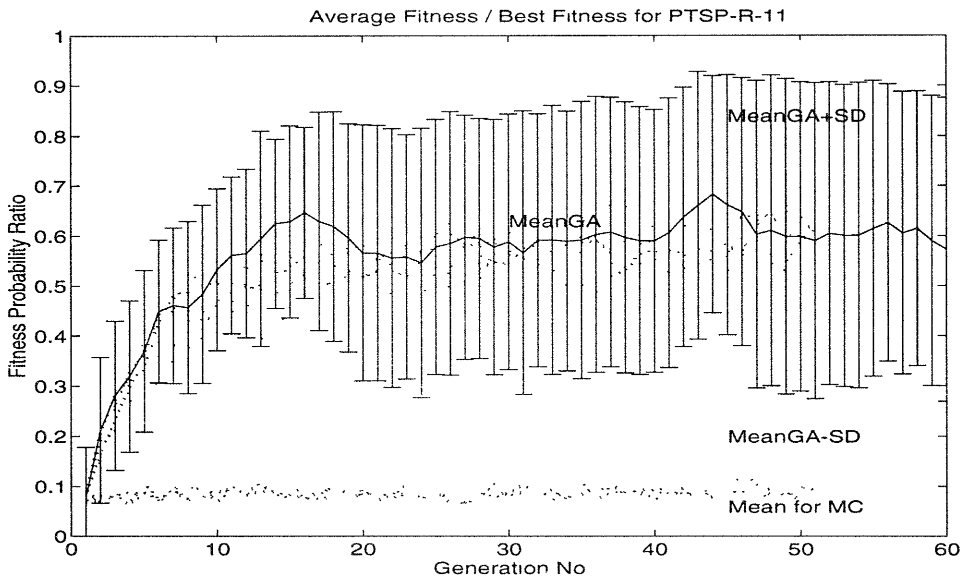


Figure 7-24: *The GA convergence compared to Monte-Carlo for maximizing the success probability in regular graphs with 11 nodes and probabilistic edges*

As was shown in Chapter 4, the GA performance can be improved by using a smaller population size and by performing successive runs in which the best solutions are transferred to the next run. By using a population size of 30 and by transferring the best 3 individuals to a next generation for a total of seven runs, we obtain the GA results that perform better than the above runs. Figure 7-27 shows the results for the multiple successive GA runs for the PTSP-R-11. The best designated solution of the GA exponentially converges to the optimum solution, as indicated in the figure which also indicates the exponential fitting curve (dotted line).

These results indicate that for the successive GA runs the following performance model is reasonable:

$$\frac{y-1}{y_0-1} = S^{\alpha t} \tag{7.26}$$

where y is the convergence ratio of the best solution at generation t, S is the population size and α is a constant. The only inconsistency between this model and the previously developed models arise because of exponent of the generation number t. While we used t^a with $a \leq 1$ in the RHS of the performance model, in this model we assume that $a = 1$. Although we found out that at times an exponent coefficient $a > 1$ may give a better fit in this set of problems, we assume that $a = 1$ to be on the conservative side.

Runtime For Evaluating Fitness

The results mentioned above are obtained through a fitness function that takes time exponentially proportional to the problem size. Figure 7-28 shows the time required for evaluating the fitness of an individual for two different cost ratios. Cost ratio is the ratio of the maximum cost limit to the cost of the shortest Hamiltonian path (circle). This figure suggests that the time τ required evaluating the

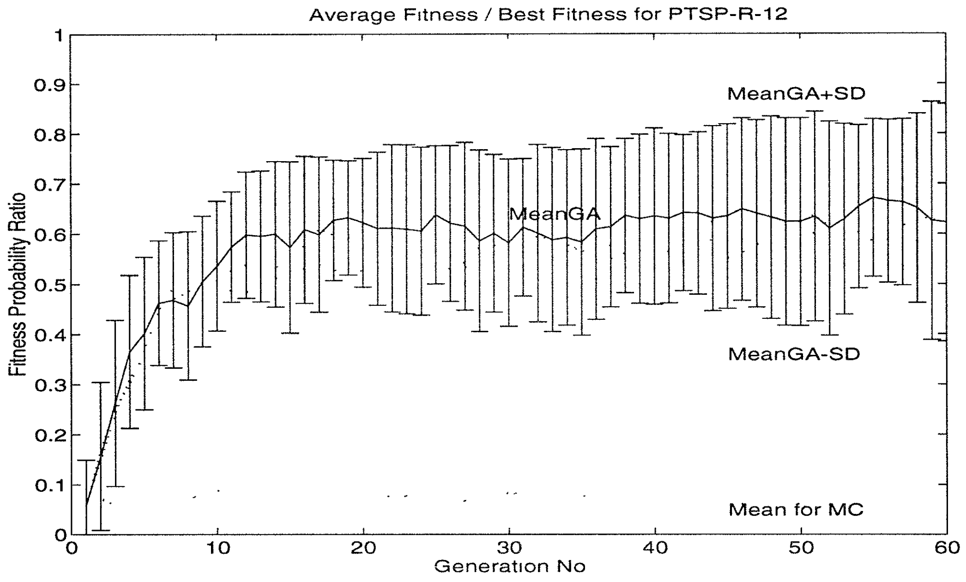


Figure 7-25: *The GA convergence compared to Monte-Carlo for maximizing the success probability in regular graphs with 12 nodes and probabilistic edges*

fitness of individual has the following form :

$$\tau \approx e^n$$

Figure 7-29 shows the time required evaluating the fitness of an individual versus cost ratio for different graph sizes. The figure suggest that

$$\tau \approx (1 - e^{-\frac{C}{C_0}})$$

The total time spent on evaluating the fitness values in a GA run in t generations with the population size S is simply the sum:

$$T = St\tau$$

Assume that we fix a certain converge ratio y^* to terminate the GA run. Assuming that $y_0(S) \approx y_0$ and denoting that

$$\Omega^* = \frac{y^* - 1}{y_0 - 1}$$

we obtain from the performance model (Equation 7.26) the value of the number of generations to convergence as

$$t^* = \frac{1}{\alpha}\frac{\log \Omega}{\log S}$$

Combining these results, we obtain that the time spent on fitness evaluation is roughly equal to

$$T^* = S\frac{1}{\alpha}\frac{\log \Omega}{\log S}e^n(1 - e^{-\frac{C}{C_0}}) \tag{7.27}$$

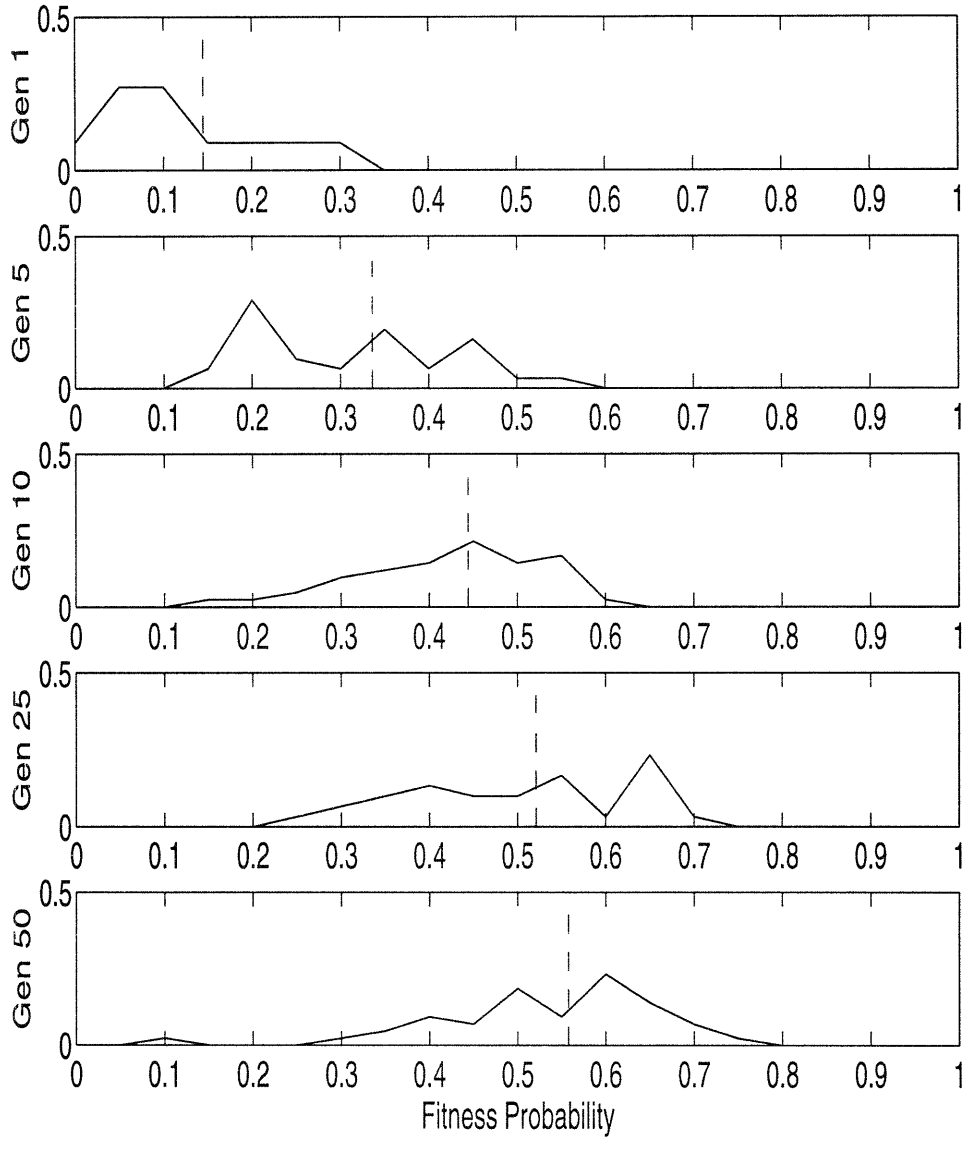


Figure 7-26: *Fitness Distribution with increasing generation for maximizing the success probability in a graph with probabilistic edges*

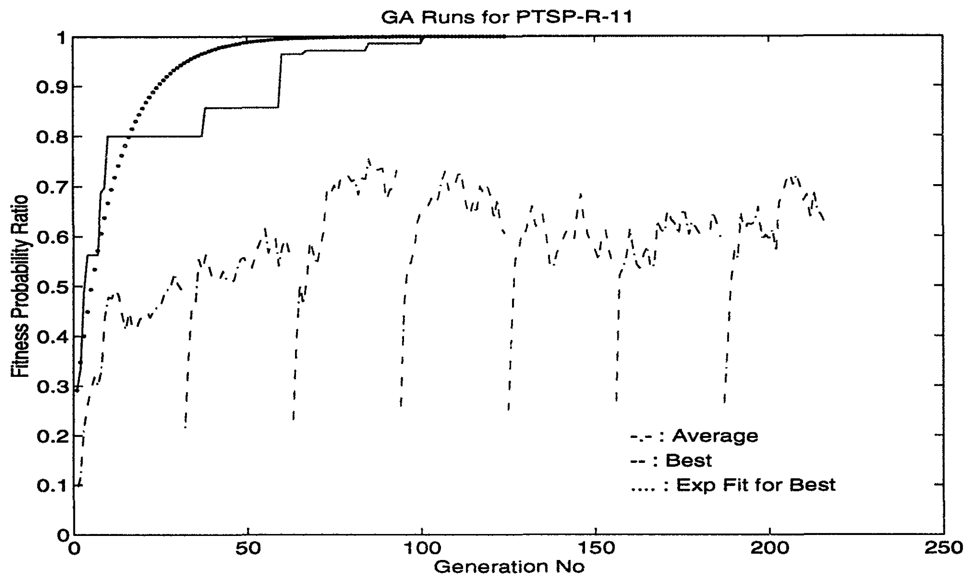


Figure 7-27: *Convergence for the successive GA runs for maximizing the success probability for PTSP-R-11*

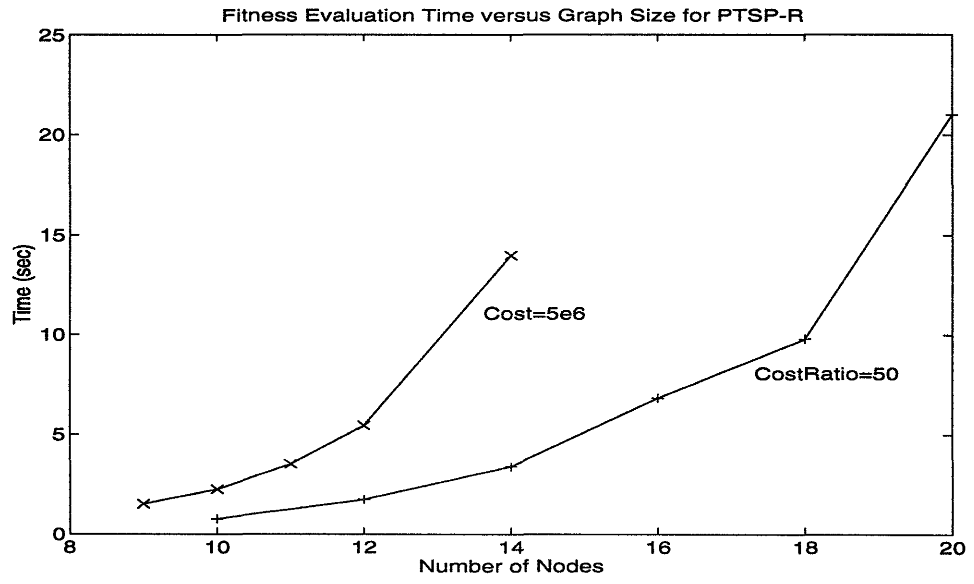


Figure 7-28: *The time required evaluating the fitness of an individual solution in PTSP-R versus the graph size*

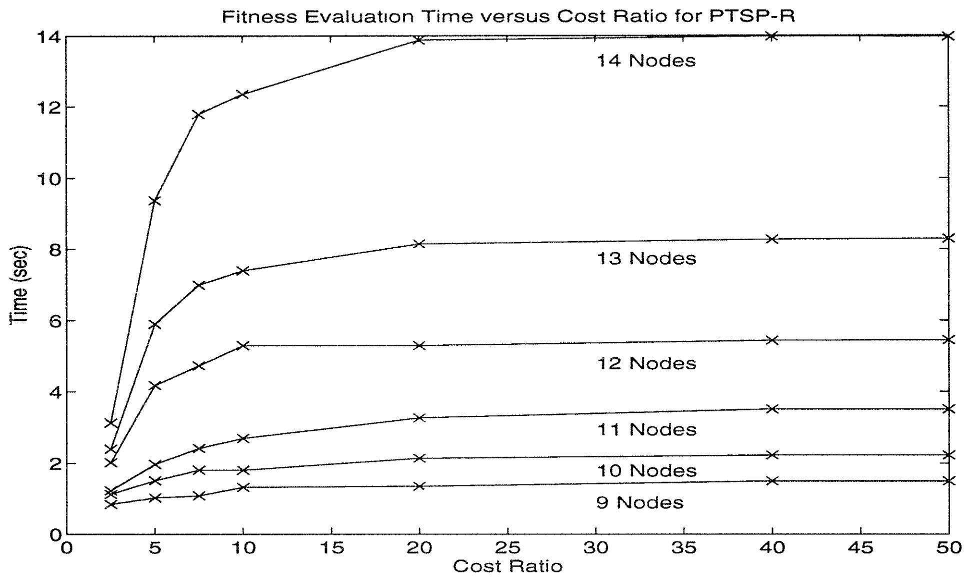


Figure 7-29: *The time required evaluating the fitness of an individual solution in PTSP-R versus the cost ratio*

While this time is logarithmic with the required convergence ratio, it is exponential with the problem size. These results indicate that a plain application of the GA for problems belonging to the complexity class PSPACE is intractable, even when an approximation would be acceptable. However, it must be noted that the GAs can converge exponentially to good solutions with increasing number of generations; therefore it is much superior to the plain random search.

7.7 Heuristics

Although no guarantees could be given on the error bounds, the performance of approximation methods for NP-hard problems is often more than satisfactory. Unfortunately, most interesting dynamic optimization problems belong to the complexity class PSPACE which contains problems even harder than those lying in NP. Because evaluation of a single solution takes time exponential with the problem size in PSPACE-hard problems, it is necessary for solving them to develop methods that are more efficient than the GA with the exact fitness evaluation. These methods (*heuristics*) demand less computation by sacrificing from accuracy.

Our goal for the efficient solution of the robust path planning problems is to develop heuristic methods **1**) *that are problem-independent to a large extent and* **2**) *can be incorporated directly into the GA*. These preconditions allow us to apply the same GA procedure for those planning problems that may be distinct yet belong to the PSPACE complexity class through only minor modifications. Although specialized heuristics can easily be developed for each problem type, we are not concerned with such heuristics because their results will not generally be extendable to distinct planning problems.

We suggest the following heuristics for the probabilistic planning problems under dynamic optimization criteria. These heuristics are designed to reduce the complexity of the objective function evaluation without relying on problem-specific information.

1. Simulation of Fitness
2. Terminating Constraints
3. Plan Repair
4. Memorization
5. Edge Vitality

Like any algorithm, there are two issues that must be addressed with any heuristic approach. The first is the accuracy of the solution method and the second is its runtime characteristics. Below, we adress these concerns separately for each suggested method.

7.7.1 Simulation of Fitness

When the evaluation of dynamic reliability values takes exponential time, it is natural to substitute this intractable exact evaluation with an approximation method. A well-known approximation technique is the Monte-Carlo or **simulation** method which samples only random branches of a given event tree. In other words, only those realizations that are drawn randomly out of the set of all possible realizations for a plan will be accounted in the evaluation of fitness.

The gain in runtime efficiency, however, is traded off with the decreased probability of convergence. Nevertheless, this tradeoff should not affect the expected results significantly because GAs already incorporate statistical tests. The theoretical results obtained (see 5.6) show that the expected convergence time increases under noisy fitness data but only under certain assumptions. The previous experimental results indicated that slightly noisy data may even result in a faster convergence rate.

GA runs with sampled fitness measurements are shown in Figure 7-30 for PTSP-E-16. Note that the evolution of the convergence ratio is not as smooth as the other runs with deterministic evaluation. Table 7.4 compares the quality of these solutions to the deterministic case for two particular cases distinguished by the number of measurements (sample size) made. These results have been obtained by averaging the results of a set of 10 different GA runs. As can be seen from the table, the results are of comparable quality as the differences remain within the standard deviations.

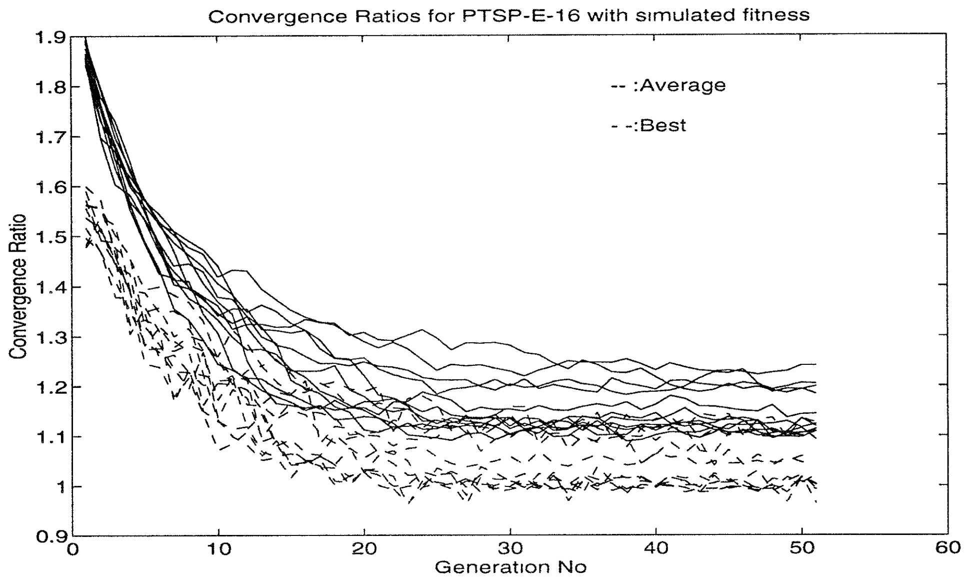


Figure 7-30: *Convergence ratios in GA runs for PTSP-E-16 with simulated fitness (# of measurements: 500)*

Table 7.4: Comparison of the convergence ratios for with exact fitness evaluation versus sampled fitness for PTSP-E-16

Gen No	Conv. Ratio (Exact) Determined	Conv. Ratio (Avg.-S.D.) Simulation (k=500)	Conv. Ratio (Avg.-S.D.) Simulation (k=1000)
1	1.59	1.61 (0.09)	1.56 (0.07)
10	1.22	1.29 (0.08)	1.22 (0.07)
20	1.09	1.16 (0.06)	1.07 (0.06)
30	1.05	1.09 (0.05)	1.05 (0.05)
40	1.03	1.05 (0.04)	1.04 (0.04)
50	1.03	1.04 (0.04)	1.03 (0.03)

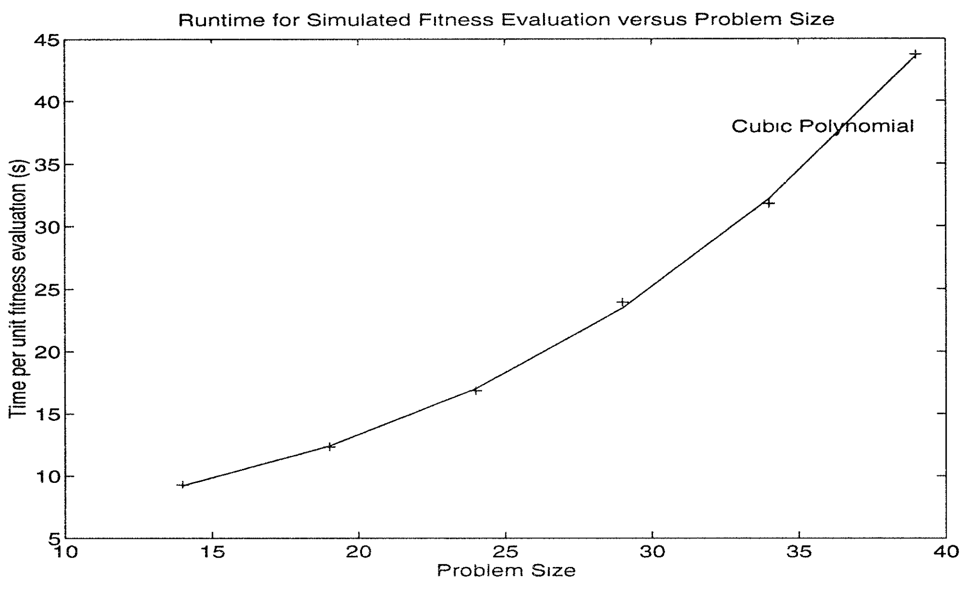


Figure 7-31: *The time required per unit fitness evaluation with simulation for PTSP-E versus problem size*

While the simulation runs yield results of comparable quality to the deterministic exact algorithm, they require considerably less computation. The average time required for evaluating the expected cost of an individual through simulation is in order $0(kn^3)$, as shown in Figure 7-31 (where k denotes the number of measurements k made and is generally a constant). Through sampling of fitness values, the runtime is reduced from exponential time to polynomial time *on average.* In short, evaluating the fitness of a solution through simulation requires an expected runtime which is polynomial in problem size and therefore simulation is a reasonable alternative to the exact fitness evaluation.

7.7.2 Terminating Constraints

Another approach, similar in spirit to simulation, relies on inexact evaluation of the fitness values. Evaluation of fitness through the exact algorithm can be made more efficient by adjusting the terminating constraints, such as those limits on the maximum cost, the minimum probability or the maximum depth a solution instance is allowed to reach.

Let us denote these limits as MDL for the maximum depth, MCL for the maximum cost and MPL for the minimum probability. *Evaluating a dynamic reliability value by using $MDL = 0$ is equivalent to a static evaluation in which no failures are taken into account.* At the other extreme, the exact evaluation of robustness measures for a path with n edges requires that $MDL = n$ since up to n failures may coexist. Using values between 0 and n will underestimate fitness values (because the reliability values may only increase when further alternatives are considered). Because the evaluation of these measures takes time in order $0(2^{MDL})$, what is lost in accuracy will have been gained from an

Table 7.5: Results of 10 GA runs for a random PTSP-R-12 under various depth limits

Solutions	$MDL = 12$	$MDL = 7$	$MDL = 5$	$MDL = 3$
Best1 ($F = 0.776$)	5	2	0	0
Best2 ($F = 0.755$)	3	5	7	4
Best3 ($F = 0.745$)	2	2	3	4
Best4 ($F = 0.723$)	0	1	0	2
Time per fitness evaluation (sec)	6.41	2.78	2.33	1.02

exponentially reduced runtime. This method can also be implemented by varying the maximum cost limit or the minimum probability limit. We call this method of using those cutoff values that result in early termination of the fitness evaluation process as **low cutoff heuristic.**

We observed that for most randomly constructed examples, using consistently low cutoff limits produces often results optimal even under high cutoff limits. This is not surprising because a solution optimum for the case with at most k failures is likely to be optimal (or only slightly suboptimal) when a larger number of failures ($k' > k$) are allowed. Table 7.5 indicates the distribution of the best designated solutions of final population in 10 different GA runs for a random PTSP-R-12 problem (in which the edge probability values are set randomly). As can be seen from the table, the GA runs with lower MDLs have been able to obtain the near optimal solutions. The table also shows the average runtime per unit fitness evaluation for these runs. Even though the runtime savings are not exponential with the depth limit, (because these have been performed with the constraint $MPL = 10^{-9}$, most realizations with low probabilities will have already been eliminated before the depth limit is reached), they are significant in comparison to the marginally increased risk of obtaining bad solutions.

The major concern with the low cutoff heuristic is the fact that using underestimated values might cause convergence to incorrect solutions. A solution optimal under certain limits is not necessarily optimal and might even be considerably suboptimal when these limits are increased. **Adjusted cutofff heuristic** tries to alleviate this problem by starting with smaller limits and increasing them gradually.

A particular implementation of this method is as follows: The exact fitnes evaluation algorithm is implemented with a constraint on the initial probability of any given branch. If the initial probability of a branch is lower than the minimum probability limit MPL, it is not further expanded. The GA starts evaluating and ranking the fitnesses of solutions under a relatively large limit on MPL. With each new generation, this limit is gradually reduced (Its final value is set according to the required precision). This method might be compared to the gradual temperature reduction in simulated annealing. The GA starts with a relatively coarse fitness evaluation but uses an increasingly more accurate fitness evaluation with each next generation. In this method, the initial coarse evaluations direct the algorithm toward the good schemes that are likely to be common in solutions optimal under either low and high cutoff

Table 7.6: Results of 10 GA runs for a random PTSP-R-13 under adjusted cutoff heuristic

Solutions	$MPL = 10^{-7}$	$MPL = 10^{-3-0.08t}$
Best1 ($F = 0.846$)	4	4
Best2 ($F = 0.838$)	2	3
Best3 ($F = 0.835$)	4	3
Time per fitness evaluation (sec)	6.95	2.90

values. Later more sensitive evaluations help clear the good solutions optimal under high cutoff values.

In most randomly constructed examples, this heuristic produces results that are almost indistinguishable from those of the GA with the exact fitness evaluation. A particular example is described below: The GA is run 10 times with 50 individuals for 50 generations for the solution of the PTSP-R with 13 nodes. In case 1, the GA uses the exact fitness evaluation algorithm with $MPL = 10^{-7}$. In case 2, the GA uses this algorithm with the MPL being initially equal to 10^{-3} and decreasing toward 10^{-7} through the adjustment schedule $MPL(t) = 10^{-3-.08t}$. Table 7.6 compares the final distribution of the best convergence ratios for both of these cases. As can be seen from this table, the results are almost the same. Yet, the time saved in these runs is about 50% for the problems considered, as indicated in Figure 7-32. For a large problem size, we expect the speedup to reach the following asymptotic ratio:

$$\frac{\sum_{t=0}^{50} 2^{10^{3+0.08t}}}{\sum_{t=0}^{50} 2^{10^{7}}} \approx \frac{4}{50} \tag{7.28}$$

Note that the adjusted cutoff heuristic does not result in an exponential speedup.

The above algorithm can be further optimized by fine tuning the adjustment schedule of the cutoff limits as well as by adjusting the selection probability distribution (one can use an initially less sensitive selection). More sophisticated versions of the adjusted cutoff heuristic can be easily developed. (An example is to vary the MPL with each generation and to employ a best fit function for the estimation of fitness values for higher threshold values).

7.7.3 Plan Repair

A plan may recover from failures as long as it can be converted into another plan with relatively little cost. The fitness of a plan under dynamic optimization criteria therefore depends on its relative distance to other plans. In plan repair heuristic, one simply tries to repair a plan or a path that has a failure by finding a link to its closest alternative. This link may simply be an edge or a subpath connecting from the last failed node to a neighbor path. It is necessary to have a database of many plans to be searched for links as most of them can be eliminated from consideration due to the constraints imposed at the

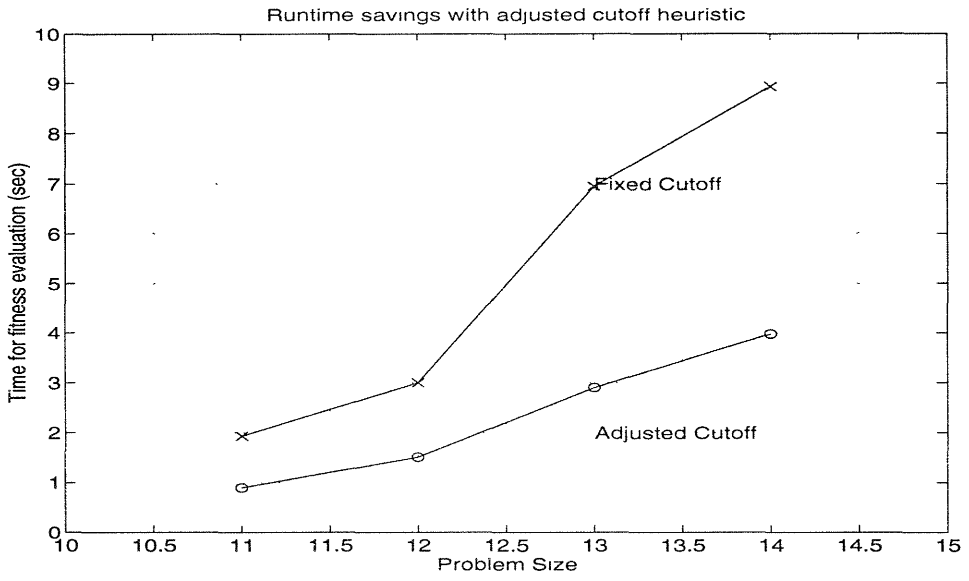


Figure 7-32: *Runtime savings with adjusted cutoff heuristic (MPL is varied)*

Table 7.7: The time required for GA runs with 50 generations with repair heuristics

Problem Size	Time (sec)
12	6460
14	9843
16	14515
18	20036

time of failure.

The following measure fits naturally within the framework of the exact evaluation algorithm: Let us assume that the optimization criteria used is the success probability. Then, the fitness of a path is evaluated by adding up the following terms: 1) the static path probability and 2) the path probability for those realizations with failures. The second term can be calculated by generating path variations with the following scheme: **a)** assign a failure for each edge, **b)** find the alternative path that starts from the departure node of the failed edge, omit its arrival node and continue along the given path. In this form, this heuristic produces the same results with low-cutoff heuristic with $MDL = n$ where n is the the number of edges that can fail at a time. Yet, it can also be easily modified to cover more complex cases by allowing to use transition paths that can connect any two paths in the database.

The results of the GA runs with the plan repair heuristics are shown in the following figures: Figure 7-33, Figure 7-34, Figure 7-35 and Figure 7-36. Table 7.7 shows the time required for these runs.

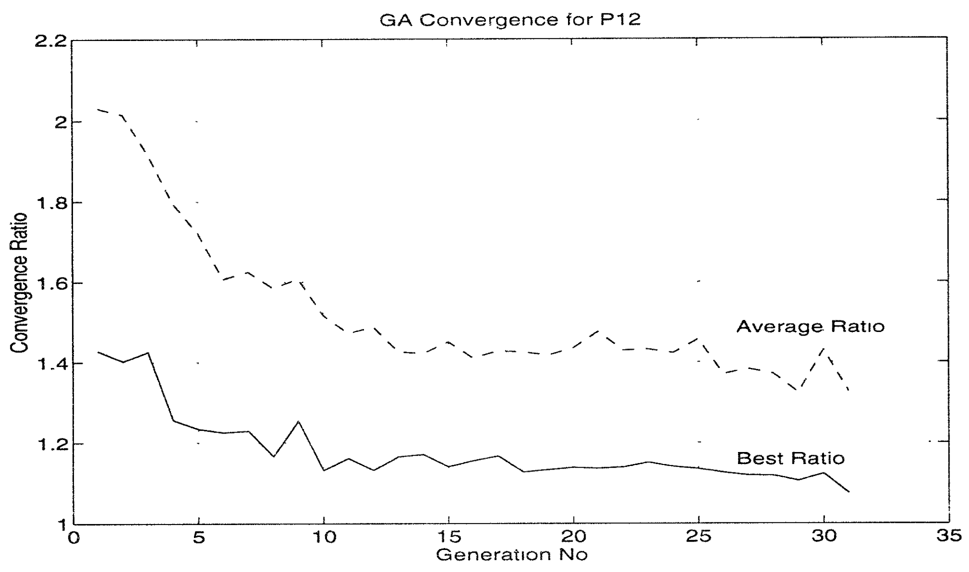


Figure 7-33: *Convergence for the GA run with path repair heuristic for minimizing the expected cost for the regular graph with 12 nodes*

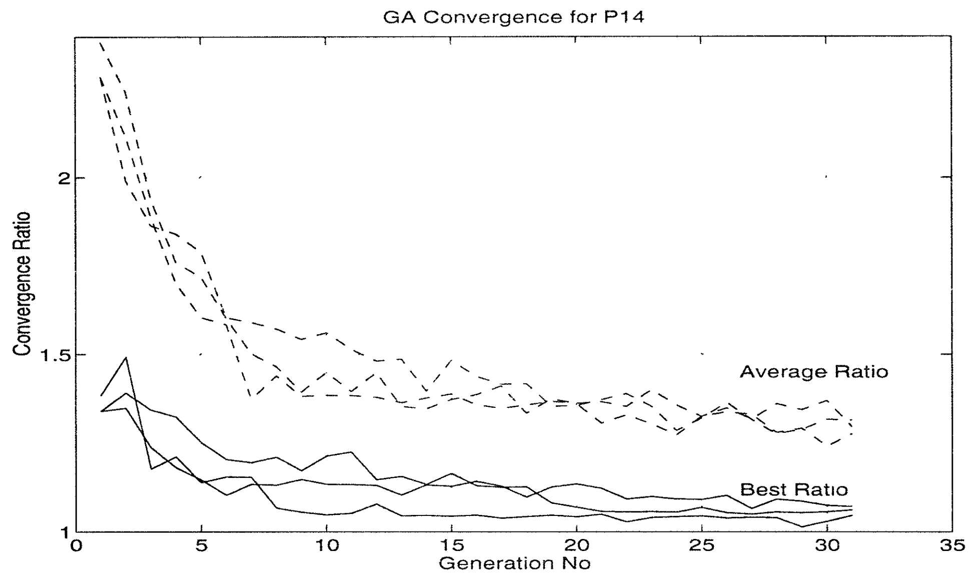


Figure 7-34: *Convergence for the GA run with path repair heuristic for minimizing the expected cost for the regular graph with 14 nodes*

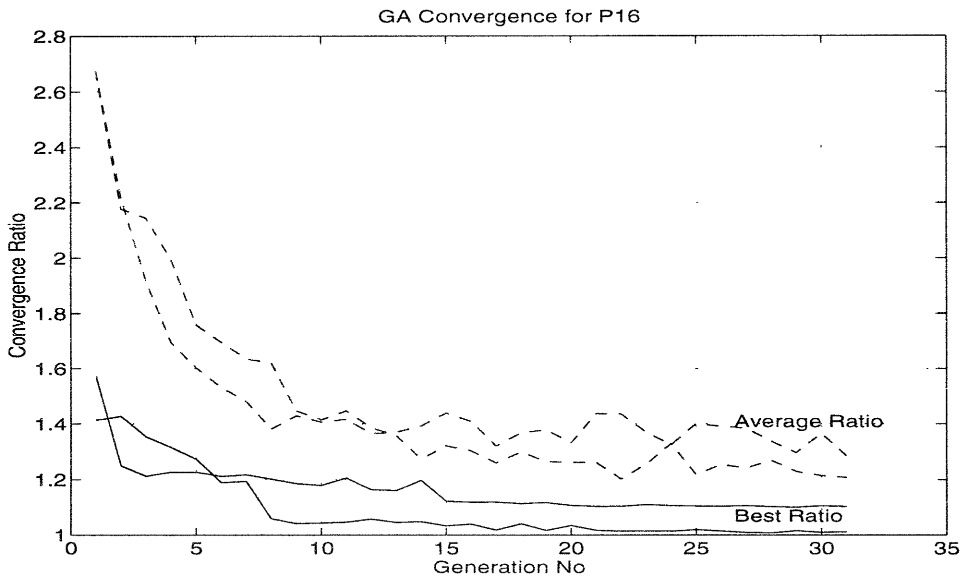


Figure 7-35: *Convergence for the GA run with path repair heuristic for minimizing the expected cost for the regular graph with 16 nodes*

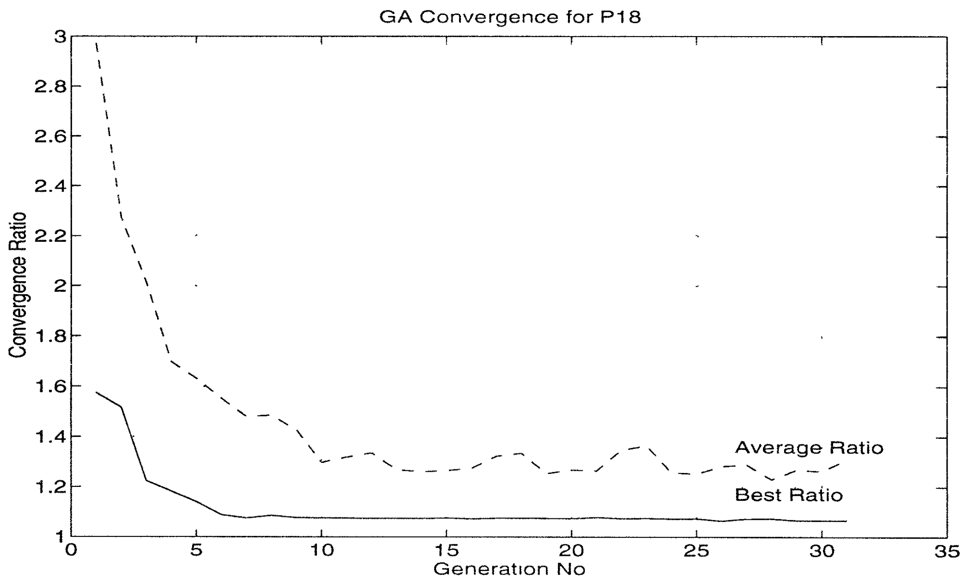


Figure 7-36: *Convergence for the GA run with path repair heuristic for minimizing the expected cost for the regular graph with 18 nodes*

Table 7.8: Comparison of the convergence ratios for table methods for PTSP-E

Convergence Ratio Achieved after 50 generations			
Problem Size	Table-1	Table-2	No Table
12	1.11	1.06	1.03
15	1.26	1.23	1.05
20	1.95	1.38	1.13
25	2.34	1.66	1.28

7.7.4 Memorization

The exact evaluation algorithm for the problems under reoptimization strategy determines a new option after each failure and then restores the original plan with this new option. Each of these restorations is distinct because they start with different constraints. A significant amount of computation can be saved by producing a database of these alternatives and by employing the plan restoration algorithm only when there can be found no information on this table. The benefits of memorization are proportional with the table size. Larger tables contain more options that can be employed but are more expensive to produce. The most simple table will contain a single path, preferably optimal, for any given pair of nodes. For a graph with n nodes, this requires $O(n^2)$ paths to be generated. Nevertheless, these paths are often useless as they may not satisfy most constraints (e.g. they may not fit within the constraints of a fixed budget). On the other extreme, we can generate all the possible paths between any given pair of nodes and pick among only those that can satisfy the specified constraints. Unfortunately, this is an intractable task for most graphs as the tables would require an exponentially increasing number of entries.

We have applied the table generation method to two particular cases distinguished by the size of the tables. In the first case, we set up a hashtable with m paths for any given pair of nodes by excluding each edge individually (where m is the number of edges). In the second case, we produce a hashtable by generating $\frac{m!}{(m-2)!2!} = \frac{m(m-1)}{2}$ paths between any given pair of nodes by excluding pair of edges individually. While the first table can accommodate a particular constraint on any given edge, the second table can accommodate constraints on any given pair of edges. The GA has been run with 50 individuals for 50 generations for both of these cases for the PTSP-E with a circular graph for problem instances with 12, 15, 20 and 25 nodes. The GA has also been run for a reference case without table generation. Table 7.8 compares the convergence ratios achieved after 50 generations for these 3 cases with different problem instances. Table 7.9 compares the time used by each of these methods. These information indicate that there should be an optimum value for the table size after which the savings from computation are offset with the required computation for generating such tables.

The idea of table generation can be further refined by memorizing the entries as they are calculated.

Table 7.9: Comparison of the time required for table methods for PTSP-E

Total Time after 50 Generations (sec)

Problem Size	Table-1	Table-2	No Table
12	1525	2755	7708
15	2872	4811	17693
20	6240	9603	55957
25	16629	26055	163900

Memorization allows us to save the data produced during the runtime for later use. This, with memorization, it may be unnecessary to fill out a large table at the very beginning of the computation for individual entries can be inserted into the table after they have been required and computed during a run. In other words, when a particular information is neeed, the algorithm first looks at the table and uses its entry if there exists one. If there is no information, it is generated *ad hoc* and is subsequently entered into the table for later use.

7.7.5 Edge Vitality

This heuristic is based on the idea that edges (actions) that are frequently encountered in most plans for a given graph problem should be important or vital. Recall that a rigorous version of the problem of finding the most vital edges or nodes has been defined for deterministic problems. The most vital edges in a path are those that their elimination from the path changes the overall fitness measure of the path in a most drastic way. The most vital edges in a graph are those that their elimination from the graph changes the overall fitness measure for the optimum path in a most drastic way. These concepts can be generalized to nondeterministic problems as well.

A particular implementation method we developed is based on the idea of weighting each edge by its relative importance in the paths or plans that it takes place. This relative importance can be calculated for each edge from the fraction of the path cost that is incurred by that edge. Given these information, a synthetic cost value can be attributed to each edge. This synthetic cost value is substituted for the normal cost values in calculating the overall cost of plans. This heuristic is implemented in the following fashion:

1. Construct randomly a large set P of paths with modelled failures. In particular, determine the appearence frequency of edge failures (modelled with cycles) in proportion with the success probability values. (If it is equal to p_{ij}, a returning edge (and therefore a cycle) is constructed with the probability $1 - p_{ij}$). The set P constitutes a sample of possible realizations.

2. Determine for every edge e_{ij} the fraction of the cost of the edge to the total cost for each path in

Table 7.10: Edge Vitality: GA results after 50 generations

Problem Size	Convergence Ratio	Time (sec)
12	1.28	5163
15	1.85	6976
20	2.49	7678
25	4.37	11121

the set P. Let this set be f. (For instance, $f_{ij} = \{.123, 0, .534, .643, .435, .384\}$)

3. Update the edge costs such that

$$d_{ij} \longleftarrow d_{ij}\left(\overline{f_{ij}} + (1 - \overline{f_{ji}})\right)$$

where $\overline{f_{ij}}$ is the average cost of the edge e_{ij} between the node i and the node j.

4. Determine the fitness of any path statically from new syntactic edge costs.

Note that in this scheme an edge that frequently fails is likely to appear in cycles. An edge with relatively little cost is easier to recover from and therefore the contribution of a resultant cycle to an average path is also likely to be smaller. Edge costs are weighted down as long as the edges are likely to contribute to the cycles and in proportion to the contribution of these cycles to the total path cost. Thus, this scheme captures the fact that failures and backtracks can increase the cost of a path. The results of the GA runs with 50 generations and 50 people per generation for the PTSP-E are shown in Table 7.10. (The size of the set of paths P were fixed at 10000). Although the results are not of high quality, the runtime characteristics are excellent. (After the initial processing, one needs only to sum up the syntatic edge costs in order to arrive at a fitness measure, therefore the fitness is determined in time in order $0(n)$.) Note that this heuristic is likelier to give more accurate results if the set of paths P were larger.

7.7.6 Comparisons of Heuristics

Figure 7-37 compares the time required for these heuristics. It is observed that the exact evaluation method takes exponential time and quickly becomes infeasible as the problem size increases. The simulation method has a time behavior as $O(kn^3)$ where n is the number of nodes in the graph. Although this method was even more expensive than the exact fitness evaluation for small problem sizes, its advantages show up as the problem size increases. Further note that its cost can easily be reduced by reducing the number of sampling measurements. The time behavior of the plan repair heuristic can be given with an exponential function but the growth rate of this function is slower compared to that of the exact evaluation method. The table-generation based methods have even smaller growth coefficient

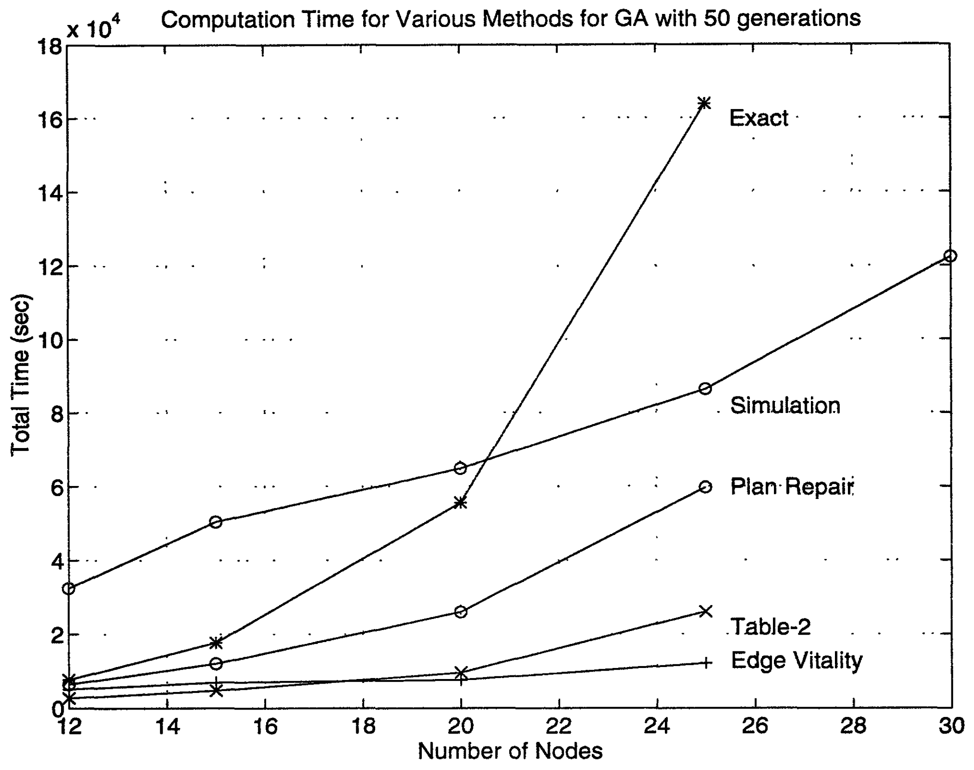


Figure 7-37: *Computation time for various heuristics used for solving a PSPACE-hard PTSP by GA with 50 generations and 50 people*

for their exponential behavior. We further observed that the time required for the edge vitality heuristic increases almost linearly with the problem size, making this method a particularly affordable one.

On the other hand, we observed that the edge vitality heuristic does not generally yield results of good quality. In contrast, the simulation results can be quite accurate; the large number of sample size we used has helped in obtaining solutions quite comparable to those of the exact evaluation method. The other heuristics (memorization and plan repair) had results in between those of the simulation and the edge vitality heuristics. These results indicate that one can obtain results with heuristics in shorter time compared to the exact method; however, the tradeoff is the decreased quality of the converged solutions. Simulation seems to yield the best tradeoff characteristics among the suggested heuristics.

7.8 Summary and Conclusions

In this chapter, we defined some path planning problems that are distinguished by their dynamic optimization criteria. We examined the complexity of these problems and showed that unless simplifying assumptions are made, these problems are computationally intractable. We showed that the objective (fitness) function used for determining robustness in the PTSP-E and the PSPP-E can be evaluated under special conditions in polynomial time. This property allows us to solve these problems with the NP-hard problem solution methods that arrive at a near-optimal answer by evaluating the complete solution instances. In particular, we showed that the genetic algorithms can solve these NP-hard dynamic optimization problems in polynomial time with the problem size. On the other hand, the evaluation of the fitness function for these robust planning problems under general conditions takes exponential time and polynomial space in computer resources. These problems, being PSPACE-hard, are not suitable for a direct application of GAs. We applied the GAs to these problems with the results found on the GA convergence similar to other problems. Unfortunately, the fact that the determination of fitness evaluation has exponential behavior makes the direct application of GAs intractable for large problem sizes. In order to resolve this issue, we developed some heuristics that are not more efficient than the GA with the exact fitness evaluation. These heuristics achieve efficiency by sacrificing from accuracy, i.e. they use fitness information that could be incomplete or inaccurate. We showed that such methods can help find good solutions in reasonable computer time. The suggested heuristics, such as Monte-Carlo simulation or plan repair, are independent of the problem properties and can be implemented for other problems of similar complexity without difficulty in adapting.

Chapter 8

Summary and Final Remarks

8.1 Summary

In this thesis, we were concerned with establishing a methodology for solving those planning problems that are in nature stochastic and dynamic. The examination of various planning paradigms has urged us to approach the problem as a problem of optimization presented in the context of graph models.

Because the deterministic models constitute the backbone of the probabilistic models, we first addressed the problem of optimal plan generation in deterministic settings. The review of literature, presented in **Chapter 2**, has shown that a large yet important set of planning problems belongs to the complexity class NP (Nondeterministic Polynomial). The practical versions of these problems are NP-hard, meaning that they cannot be solved in time polynomial with the problem size. NP-hard problems have been attacked with both problem-specific heuristics and problem-independent search and optimization methods (**Chapter 3**). Among these methods, recently developed semi-probabilistic optimization methods have shown more promise. These methods (such as *simulated annealing* or *genetic algorithms*) draw on the strengths of both random and systematic search techniques and have proven more efficient for search in moderately complex and moderately nonlinear problem spaces. We suggested the genetic algorithms as a general heuristic for the solution of distinct planning problems. Being adaptable to distinct problems, the genetic algorithm is a particularly attractive weak search method that simulates evolution. It starts with an initial set of solutions and proceeds by evolving these solutions through recombination, mutation and fitness (a surrogate for optimization measure) based selection process.

We developed an object-oriented software based on the technique of genetic algorithms that can generate near-optimal solutions to various path planning problems in polynomial time with the given problem size. In **Chapter 4**, we demonstrated the effectiveness of this solution technique by testing it on some benchmark instances of a well-known NP-complete problem, namely the traveling salesman problem. The application of this software on the TSP benchmarks has provided us with a confidence in the accuracy of our software as well as a semi-empirical model of the convergence characteristics of genetic algorithms.

We showed that the GAs are expected to converge to the vicinity of optimal solutions in time exponential with the desired accuracy, yet polynomial with the problem size for NP-hard problems. In particular, we found that the expected value of the runtime (T_{exp}) of the genetic algorithm for achieving a particular convergence ratio (y^*) for a problem of size S can be given by:

$$T_{\text{exp}}(y^*, n, S) \quad = \quad cnh(S)\left[\frac{\beta S + \gamma}{\log n} \log \frac{y_0 - 1}{y^* - 1}\right]^{1/a}$$

where y_0 is the convergence ratio of the initial population, n is the population size and a, c, β and γ are problem-specific constants. $h(S)$ is the time required for the determination of the fitness a individual solution. If this function is a polynomial, a fixed convergence ratio can be achieved in polynomial time, yielding an efficient solution method. These results are in line with the GA model we developed based on an extension of Schema Theorem, which states that the proportion of good concepts among the solution set increases exponentially with each new generation. We also found that the time it takes for a genetic algorithm (GA) to find the globally optimal solution is exponential with the problem size. Because the most NP-hard problems are also NP-hard to approximate, we concluded that these results are the best that can be hoped for.

We compared the technique of genetic algorithms to other search methods, such as A*-search, Monte-Carlo or simulated annealing, and found that only simulated annealing had a better performance than the GAs in the solution of the deterministic TSP. This comparison shows that near-optimal solutions can be obtained more efficiently with the semi-probabilistic search methods in relative to the systematic search techniques.

In the second part of the thesis, we extended the GA method to the probabilistic domains (**Chapter 5**). We showed that GAs can converge to near-optimal solutions by sampling the fitness of each solution instance as opposed to a deterministic evaluation. The statistical selection rules employed by the GA is expected to compensate for the noise inherent in sampling. Our experiments indicated that these GA models with noisy fitness information might even converge faster to the optimal solutions than the GA with the deterministic fitness evaluation. For optimally tuned GAs, however, noisy fitness data may cause a delay in the convergence time.

In the last part of the thesis, we focused on the responsiveness of plans to possible contingencies (**Chapter 6**). We stressed the importance of minimizing the cost of adapting to random events whose outcome become clear only after embarking on the plan and the importance of modelling the control actions of the plan executor. We stressed that the best laid plans are those that provide one with options and can be modified with relatively little cost. Although similar approaches exist in various paradigms, we felt a need to formalize the concept of robust plans. We developed a specific set of measures in order to measure the *"robustness"* of a given solution to changes that could take place during plan execution. These robustness measures, such as the dynamic reliability or the dynamic expected cost, expand on the classical objective functions by taking into evaluation the contributions of the options inherent in a given solution. These options consist of the alternative plan subsequences or modifications that can be resorted to when discrepancies arise between the plan and the actual work accomplished.

The examples of robust plans were examined in **Chapter 7** through path planning problems under dynamic optimization criteria. These problems can be characterized by unreliable edges and the possibility of backtracking and an increase in the plan cost upon failures. We developed formulas for the reliability and the expected cost of paths under dynamic optimization criteria. After finding some complexity results,we examined the reasons why the robust solutions could be different that the solutions that would be optimum under the conventional optimization criteria. We applied the genetic algorithms on selected instances of these problems. For those problem instances that are NP-hard, GAs converged to the solutions that were known to be optimum with convergence characteristics similar to those of deterministic NP-hard problems. For general dynamic optimization problems, the evaluation of the robustness measures takes exponential time and therefore a direct application of GAs to these problems is intractable.

In order to reduce the computation time, we developed problem independent heuristics that replaced the exact fitness evaluation. The superimposition of such heuristics in the general scheme of GAs proved successful as they helped produce acceptable solutions in reasonable times. In particular, we implemented a set of bounding techniques that terminate the fitness evaluation process after having reached to pre-set thresholds. These techniques save time by only slightly underestimating the exact robustness measures. We proved that the solutions found by GAs under these cut-off methods are still near-optimal for most problems. We also developed heuristic methods with polynomial computation time and showed that these heuristics as well can find near-optimal solutions. These techniques include the simulation of plan execution, giving high priority to certain plan tasks that are easy to recover from and the repair of failures in a plan with alternatives selected from a pool of solutions. Comparison of these techniques with the GA with the exact fitness evaluation verified their efficiency. These experiments showed that GAs are a good general heuristic for complex planning problems and can perform well for even harder problems in PSPACE by combining with other heuristics.

8.2 Open Problems

Many open problems and further research avenues exist. One of the most important issues concern the modelling of the GAs. The model we developed in section 5.3 explains the reasons for the GA's success in finding the optimal results. Further, the empirical equations we found express the convergence ratio expected to be achieved by the GA in terms of the generation number and the population size. Yet, there is a need for those expressions that yield the probability distributions of the convergence ratios in terms of other parameters, such as the generation number and population size. These probabilistic expressions will serve many purposes such as finding out when in a GA run a good solution is expected to appear with a specified probability.

Further improvements in the performance of genetic algorithms are possible. For example, for all the problems considered, the experimentation with different recombination procedures may help in achieving a more efficient process. Genetic algorithms can also be implemented in parallel computers taking advantage of the simultaneous evolution of a set of individuals. Genetic algorithms, being a general, weak heuristic allows to superimpose distinct heuristics. Using co-evolving parasites, an interesting heuristic suggested by some researchers, might prove useful. A related idea may be the co-evolution of the fitness methods at the same time with the evolution of solution instances. The use of increasingly selective objective functions might help achieve a faster convergence rate. Finally, the method of genetic programming, which extends the genetic algorithms to automatic programming, might prove more efficient in solving complex models as they might alleviate the need for finding efficient recombination procedures.

Further ideas on the problem independent heuristics can be developed. For example, **Graph Topology** is a rich source of heuristics. An idea that is closely related to the edge vitality heuristic utilize the articulation nodes (those that disconnect a given graph) of a given model. For nondeterministic problems, it may be possible to produce a probability value for the disconnection of a given graph. Such information can be used for attaching an importance (or an avoidance coefficient) to each node to be used during plan generation. Further, these probability values can be employed as additional constraints in the plan restoration. Another idea consists of solving a problem that is opposite to the problem of finding good structures. An opposite problem will ask for a specific set of edges or nodes that can make the maximum adverse impact on a plan. If the solutions to these problems are established beforehand, it might be easier to solve the problem of finding the most robust plans.

Further points that are not examined in this thesis but warrant close attention are the establishment of bounds for the robustness measures and the effect of different modification strategies.

Further, the genetic algorithms method must be compared to different solution methods, such as simulated annealing, in order to observe their relative performance. The application of search methods that incrementally construct a solution is especially interesting as these methods may have advantages in regard to computational resources. Even though such solution methods could be pathological (because a piece that is not yet considered might turn out to have a significant global impact on the solution), they may prove satisfactory for general purposes.

A insight obtained from this work is that the employment of the dynamic optimization criteria changes the flavor of the problem from one of the straightforward optimization to one of a game with nature. As a result, one might find the applications of game theory techniques particularly useful for those problems in which dynamic failures should be avoided.

Appendix A

Beta distribution

The beta distribution is a two parameter distribution and is defined for a random variable, x, in the range $[0, 1]$, where the **p.d.f.** $f_{m,n}(x)$ is given by [119]:

$$f_{m,n}(x) = \frac{x^{m-1}(1-x)^{n-1}}{\beta(m,n)} \tag{8.1}$$

where $\beta(m, n)$ is

$$\beta(m,n) = \int_0^1 z^{m-1}(1-z)^{n-1}dz \tag{8.2}$$

The function $\beta(m, n)$ can also be expressed in terms of gamma function (Γ) [114] as:

$$\beta(m,n) = \frac{\Gamma(m)\Gamma(n)}{\Gamma(m+n)} \tag{8.3}$$

Given an optimistic duration value t_o, a pessimistic duration value t_p and a most likely duration value t_m (such that $t_o < t_m < t_p$) for an activity, the random variable duration T is equal to :

$$T = t_o + (t_p - t_o)x \tag{8.4}$$

where x is a random variable whose p.d.f. is given by Equation 8.1. In order to make the most likely value equal to t_m m is given in terms of n as

$$\frac{m-1}{m+n-2} = \frac{t_m - t_o}{t_p - t_o} \tag{8.5}$$

(This can be confirmed by differentiation). It follows from the above equation

$$m = \frac{(n-2)(t_m - t_o) + (t_p - t_o)}{t_p - t_m} \tag{8.6}$$

Equation 8.5 does not describe a unique distribution for an infinite number of (m, n) pairs satisfy it. To fix the distribution, a further assumption is made in PERT, namely that the standard error is $\frac{1}{6}$th of the range $(t_p - t_o)$. Since the variance of beta distribution is given as [143] :

$$\sigma^2 = \frac{m(m+1)}{n(m+n+1)} \tag{8.7}$$

m and n values are fixed through Equation 8.6 and Equation 8.7.

Bibliography

[1] **Aha A V, Hopcroft J J, Ullman D J** *(74) - The Design and Analysis of Computer Algorithms, (Addison-Wesley), 1974*

[2] **Allen D, et. al.** *(91) - The Portable AI-Lab, Documentation accompanying the PAIL system, 1991*

[3] **Andreatta G, Romeo L** *(88) - Stochastic Shortest Paths with Recourse, Networks, v 18. pp 193-204, 1988*

[4] **Ankenbrandt C E** *(91) - An Extension to the Theory of Convergence and a Proof of the Time Complexity of GA, In Foundations of Genetic Algorithms, (Morgan Kaufmann), 1991*

[5] **Arora S, Lund C, Motwani R, et. al.** *(93) - Proof Verification and Hardness of Approximation Problems, Proc. 33rd Annual IEEE Symp. on Foundations of Computer Science, pp 14-23, Pittsburg 1993*

[6] **Arora S, Safra S** *(92) - Approximating Clique is NP-complete, Proc. 32nd Annual IEEE Symp. on Foundations of Computer Science, pp 2-13, 1992*

[7] **Baker K R** *(74) - Introduction to Sequencing and Scheduling, (John Wiley & Sons), 1974*

[8] **Ball M O, Golden B L, Vohra R V** *(89) - Finding the Most Vital Arcs In a Network, O.R. v 8 no 2, April 1989*

[9] **Bar-Noy A, Schieber B** *(91) - The Canadian Traveller Problem, Proc. of 2nd Annual ACM-SIAM Symposium on Discrete Algorithms, San Fransisco, 1991*

[10] **Bard J E, Bennett J E** *(91) - Arc Reduction and Path Preference in Stochastic Acyclic Networks, Management Science, 37, no 2 pp 198-216, 1991*

[11] **Barton E, Berwick R, Ristad E** *(87)- Computational Complexity and Natural Language, pp 27-9, (MIT Press), 1987*

[12] **Bellman R, Esogbue A, Nabeshima I** *(82) - Mathematical Aspects of Scheduling & Applications, (Pergamon Press), 1982*

[13] **Bernard D, Borillo M, Gaume B** *(90) - A Cognitive Temporal Model For Planning in Aircraft Maintenance, The Third International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems v 1, pp 318-323, 1990*

[14] **Berry D A, Fristtedt B** *(85) - Bandit Problems: Sequential Allocation of Experiments, (Chapman & Hall), 1985*

[15] **Bertsekas D** *(87) - Dynamic Programming, Deterministic and Stochastic Models, (Prentice-Hall), 1987*

[16] **Bertsekas D, Tsitsiklis J** *(90) - An Analysis of Stochastic Shortest Path Problems, Technical Report LIDS-P-1815, Laboratory for Information and Decision Systems, MIT, 1990*

[17] **Bremermann, H J** *(62) - Optimization through evolution and recombination, In Self-Organizing Systems ed. M C Yovits, G D Goldstein, G T Jacobi, pp 93-106, (Spartan), 1962*

[18] **Bui N T, Moon B R** *(94) - A New Genetic Approach for the Traveling Salesman Problem, To appear in IEEE Conference on Evolutionary Computation, 94, (Personal Communication), 1994*

[19] **Burke P, Prusser P** *(91) - A Distributed Asynchronous System for Predictive and Reactive Scheduling, AI in Engineering Journal, v 61, no 3, 1991*

[20] **Carley H W, Sha D** *(82) - Most Vital Links and Nodes in Weighted Networks, OR Letters, Sept 1982*

[21] **Chapman D** *(87) - Planning for Conjunctive Goals, in Readings in Planning, (Morgan Kaufmann), 1990*

[22] **Chapman D, Agre E** *(89) PENGI, AI Magazine, Spring 1989*

[23] **Choi W** *(93) - Contingency-Tolerated Robot Motion Planning and Control, PhD Thesis, Stanford University, STAN-CS-93-1477, 1993*

[24] **Christofides N** *(79) - The Traveling Salesmen Problem In Combinatorial Optimization, ed. N Christofides, A Mingozzi, S Toth, C Sandi, (John Wiley & Sons), 1979*

[25] **Coffman E G** *(76) - Computer and Job-Shop Scheduling Theory, (John-Wiley & Sons), 1976*

[26] **Colbourn C J** *(87a) - The combinatorics of network reliability (Oxford University Press), 1987*

[27] **Colbourn C J** *(87b) - Edge-packings of graphs and network reliability, Discrete Matematics, 72 pp 49-61, 1987*

[28] **Conway R, Miller R** *(67) - Theory of Scheduling, (Addison-Wesley), 1967*

[29] **Corea A, Kulkarni V** *(93) - Shortest Paths in Stochastic Networks with Arc Lengths Having Discrete Distributions, Networks, v 23 pp 175-183, 1993*

[30] **Cormen T, Leiserson C, Rivest R** *(93) - Introduction to Algorithms, (MIT Press), 1993*

[31] **Croucher J S** *(78) - A Note On the Stochastic Shortest Route Problem, Nav. Res. Logist. Quart. v 25, pp 729-732, 1978*

[32] **Dantzig G B** *(87) - Planning Under Uncertainty Using Parallel Computing, Operations Research Center, Stanford University, SOL-87-1, 1987*

[33] **Davidor Y** *(91) - Genetic Algorithms and Robotics - A Heuristic Strategy for Optimization, (World Scientific), 1991*

[34] **Dawkins R** *(89) - The Selfish Gene, 2nd ed. (Oxford University Press), 1989*

[35] **De Jong K** *(92) - Learning with Genetic Algorithms: An Overview In Genetic Algorithms, ed. B Buckles, F Petry, (IEEE Computer Society Press), 1992*

[36] **De La Vega W F, Lueker G S** *(81) - Bin Packing can be solved within 1+ϵ in Linear Time, Combinatorica, v 1 pp 349-355, 1981*

[37] **Dean T, Kaelbling L P, et. al.** *(94)- Planning Under Time Constraints in Stochastic Domains, Submitted to Artificial Intelligence Special Issue on Planning and Scheduling, 1994*

[38] **Deo N, Krishnamoorty M S** *(79) - Node deletion NP-complete problems, Networks, v 9, pp 189-94, 1979*

[39] **Elleby P, Grant T** *(86) - Knowledge-Based Scheduling, Appearing in Computer Assisted Decison Making, edited by G. Mitra, (Elsevier), 1986*

[40] **Feldman R C** *(66) - A Heuristic Approach to Using Linear Non-Convex Activity Time-Cost Curves In Scheduling Project Networks, MIT, Dept. of Civil Engr., Research-Report R66-19, 1966*

[41] **Fikes R, Nilson N, et. al.** *(72), Learning and Executing Generalized Robot Plans, Artificial Intelligence 3:251-88, 1972*

[42] **Findler N V** *(91) - Contributions to A Computer-Based Theory of Strategies, (Springer-Verlag), 1991*

[43] **Fitzpatrick J M, Grefenstette J J** *(88) - Genetic Algorithms in Noisy Environments, Machine Learning, 3 (2/3), pp 101-20, 1988*

[44] **Fortune S, Hopcraft J, Wyllie J** *(80) - The directed homeomorphism problem, Theoretical Computer Science, v 10, pp 111-21, 1980*

[45] **Fox R B** *(90) - Non-Chronological Scheduling, AI, Simulation and Planning in High Autonomy Systems, Proceedings, U. of Arizona 1990*

[46] **Frank H** *(69) - Shortest Paths in Probabilistic Graphs, Operations Research, 17, pp 583-99, 1969*

[47] **Frankel, E G** *(90) - Project Management in Engineering Services and Development, (Butterworths), 1990*

[48] **Fredman M, Tarjan R E** *(85) - Fibonacci Heaps and Their Uses in Improvement of Network Optimization Algorithms, Proc. 25th FOCS Conference, 1985*

[49] **French S** *(82) - Sequencing and Scheduling, (Ellis Horwood Series), 1982*

[50] **Futo I, et. al.** *(91) - AI and Simulation, (McMilland), 1991*

[51] **Gallant S I** *(93) - Neural Networks in Learning and Expert Systems, (MIT Press), 1993*

[52] **Garey M R, Johnson D S** *(79) - Computers and Intractability - A Guide to the Theory of NP-Completeness, (Freeman), 1979*

[53] **Garey M R, Johnson D S** *(77) - Two processor scheduling with start-times and deadlines, SIAM J. Comput. v 6, pp 416-26, 1977*

[54] **Gates K H, Lenart M** *(90) - PROJCON: An Expert System for Project Controls in Construction Management, The Third International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems, v 1 pp 300-307, 1990*

[55] **Georgeff M** *(90) - Planning, in Readings in Planning, (Morgan Kaufmann), 1990*

[56] **Ginsberg E** *(89) - Universal Planning: An Almost Universally Bad Idea, AI Magazine, Spring 1989*

[57] **Glover F** *(90) - Tabu Search : A Tutorial, Interfaces, v 20, no 4, pp 74-94, 1990*

[58] **Goldberg D E** *(89a) - Genetic Algorithms in search, optimization and machine learning, (Addison-Wesley), 1989*

[59] **Goldberg D E, et. al.** *(89b) - Messy Genetic Algorithms: Motivation, Analysis and First Results, TCGA Report No: 89003, (University of Alabama), 1989*

[60] **Goldberg D E** *(92) - Sizing Populations for Serial and Parallel Genetic Algorithms In Genetic Algorithms ed. B Buckles, F Petry, (IEEE Computer Society Press), 1992*

[61] **Goldberg D E, Lingle R** *(85) - Alleles, loci, and the travelling salesman problem, In Proceedings of the First International Conference on Genetic Algorithms and Their Applications, (Lawrence Erlbaum), 1985*

[62] **Graves S** *(81) - A Review of Production Scheduling, Operations Research, v 29 no 4, 1981*

[63] **Green F, Zigli R** *(84) - Pert Probability: Network Paths and Completion Time, Project Management Journal, pp 54-58, August 1984*

[64] **Grefenstette J J, Gopal R, et. al.** *(85) - Genetic Algorithms for the travelling salesman problem, In Proceedings of the First International Conference on Genetic Algorithms and Their Applications, (Lawrence Erlbaum), 1985*

[65] **Grefenstette J J** *(87) - Incorporating Problem Specific Knowledge into Genetic Algorithms, In Genetic Algorithms and Simulated Annealing, ed. L Davis, (Morgan Kaufmann), 1987*

[66] **Grefenstette J J, Baker J** *(92) - How Genetic Algorithms Work: A Critical Look at Implicit Parallelism, In Genetic Algorithms ed. B Buckles, F Petry, (IEEE Computer Society Press), 1992*

[67] **Gruber T, Fikes R, Feigenbaum E** *(94) - How Things Work Project, Knowledge Systems Laboratory, Stanford University, http:www-ksl.stanford.edu, 1994*

[68] **Gudes E, Kuflik T, Meisels A** *(90) - An Expert Systems Based Methodology for Solving Resource Allocation Problems, The Third International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems, v 1 pp 309-317, 1990*

[69] **Hadawi K, Hsu W L, et. al.** *(92) - An Architecture for Real-Time Distributed Scheduling, AI Magazine, Fall 1992*

[70] **Hartley R, Coombs M** *(91) - Reasoning with graph operators, In Principles of Semantic Networks: Explorations on the Representation of Knowledge, (Morgan Kaufmann), 1991*

[71] **Hayhurst K J, Shier R** *(90) - A factoring approach for the stochastic shortest path problem, Technical Report No, 90-07, Department of Mathematics, College of William and Mary, Williamsburg, VA, 1990*

[72] **Hendler J, Tate A, Drummond M** *(90) - AI Planning: Systems and Techniques, AI Magazine, Summer 1990*

[73] **Hillis D** *(90) - Co-evolving parasites improve simulated evolution as an optimization procedure, Physica D 42 pp 228-34, 1990*

[74] **Holland J H** *(75) - Adaptation in Natural and Artificial Systems, (University of Michigan Press), 1975*

[75] **Hopfield J J, Tank D W** *(85) - Neural Computation of Decisions in Optimization Problems, (Biological Cybernetics), v 52, pp 141-52, 1985*

[76] **Hosley W** *(87) - The Application of Artificial Intelligence Software to Project Management, Project Management Journal, August 1987*

[77] **Jaillet P** *(85) - Probabilistic Traveling Salesman Problem, MIT PhD Thesis, Civil Engineering, 1985*

[78] **Jaillet P** *(92) - Shortest Path problems with Node Failures,* Networks, *v 22 pp 589-605, 1992*

[79] **Jog P, Suh J Y, Gucht D V** *(89) - The Effects of Population Size, Heuristic Crossover and Local Improvement in a Genetic Algorithm for the Traveling Salesman Problem,* In Proceedings *of the Third International Conference on Genetic Algorithms (Morgan-Kaufmann), 1989*

[80] **Kangari R, Boyer L** *(87) - Risk Management in Expert Systems, Project Management Journal, August 1987*

[81] **Kao E P C** *(78) - A Preference Order Dynamic Programming for a Stochastic Traveling Salesman Problem, Operations Research, v 26, no 6, pp 1033-1045, 1978*

[82] **Karger D, Motwani R, Ramkumar G D S** *(93) - On Approximating the Longest Path in a Graph Submitted to SIAM J. Computing 1993*

[83] **Kasahara T et. al.** *(89) - Automated Scheduling of Maintenance in Nuclear Power Plants, ANS Topical Meeting on Computer Applications for Nuclear Power Plant Operation and Control, 1989*

[84] **Keler M, et. al.** *(91) - On Comparison of Search Methods, Transactions on Eighth International Conference in AI Applications, 1991*

[85] **Kirkpatrick S, Gelatt C D, Vecchi M P** *(83) - Optimizaton by Simulated Annealing, Science, 220, pp 671-80, 1983*

[86] **Koza J R** *(92) - Genetic Programming, (MIT Press), 1992*

[87] **Kúçera L** *(90) - Combinatorial Algorithms, (Adam Helger), pp 280-3, 1990*

[88] **Langlotz C P, Shortliffe E H** *(89) - Logical and Decision Theoretic Methods for Planning Under Uncertainty, AI Magazine, Spring 1989*

[89] **Laurière, J L** *(90) - Problem-Solving and Artificial Intelligence, (Prentice Hall), 1990*

[90] **Lee R** *(85) - A Logic Programming Approach to Building Planning and Simulation Models, Appearing in Expert Systems and Artificial Intelligence in Decision Support Systems, Proceedings of the Second Mini EuroConference, Lunteren, The Netherlands, 17-20 November 1985*

[91] **Lewis E E** *(87) - Introduction to Reliability Engineering, (Wiley), 1987*

[92] **Levitt R E, Kunz J** *(85) - Using Knowledge of Construction and Project Management for Automated Schedule Updating, Project Management Journal, December 1985*

[93] **Lozana-Perez E, et. al.** *(87) - Compliant Motion, in Artificial Intelligence by D. Knight, 1987*

[94] **Lund C, Yannakakis M** *(93) - On the Hardness of Approximating Minimization Problems, Proc. 25th ACM Symp. on Theory of Computing, 1993*

[95] **Malik K, Mittal A K, Gupta S K** *(89) - The k Most Vital Arcs in the Shortest Path Poblem", OR Letters, v 8, pp 223-7, 1989*

[96] **Martin J J** *(65) - Distribution of time through a directed acyclic network, Operations Res., v 13, pp 46-66, 1965*

[97] **McAllester D** *(93) - Class Notes for 6.824 : Artificial Intelligence, MIT, 1993*

[98] **Minsky M** *(91) - The Society of Mind, (Simon and Schuster), 1991*

[99] **Mirchandani P B, Soroush H** *(85) - Optimal Paths in Probabilistic Networks: A Case with Temporary Preferences, Comput & Oper. Res. v 12, no 4, pp 365-85, 1985*

[100] **Mühlenbein H** *(92) - How Genetic Algorithms Really Work: I. Mutation and Hillclimbing, Parallel Problem Solving From Nature -2-, ed R Männer, B Manderick (North Holland), 1992*

[101] **Navinchandra D** *(85) - Intelligent Use of Constraints for Activity Scheduling, MIT, S.M. Thesis, Civil. Engr 1985*

[102] **Neumann K** *(90) - Stochastic Project Networks, (Springer Verlag), 1990*

[103] **Norvig P** *(92) - Paradigms of Artificial Intelligence Programming, (Morgan Kaufmann), 1992*

[104] **O'Connor R** *(78) - Planning Under Uncertainty : Multiple Scenarios and Contingency Planning, Conference Board Report, no 741, 1978*

[105] **Palay A J** *(85) - Searching with Probabilities (Pitman), 1985*

[106] **Papadimitriou C H** *(85) - Games Against Nature, Journal of Computer and System Sciences, vol 3, pp 288-301, 1985*

[107] **Papadimitriou C, Steiglitz K** *(82) - Combinatorial Optimization: Algorithms and Complexity, (Prentice & Hall), 1982*

[108] **Papadimitriou C H, Yannakakis M** *(88) - Optimization, Approximation, and Complexity Classes, Proc. 20th ACM Symposium on Theory of Computing, 1988, pp 229-234*

[109] **Papadimitriou C, Yannakakis M** *(89) - Shortest Paths Without A Map, Proc. ICALP pp 611-620, 1989*

[110] **Phiphobmongkol S, Chang K** *(90) - A Multiagent Planner using Meta-Knowledge and Agent Constraints, The Third International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems, v 1, pp 258-267, 1990*

[111] **Pierce M** *(89) - Heuristics for Job-shop Scheduling, MIT-AI-TR 89-142, 1989*

[112] **Polychronopoulos G** *(92) - Stochastic and Dynamic Shortest Distance Problems, PhD Thesis, CS, MIT, 1992*

[113] **Pratt V** *(75) - Every Prime Has a Succinct Certificate, SIAM J Comp v 4 pp 214-20, 1975*

[114] **Press W, Teukolsky S, Flannery B, Vetterling W** *(92) - Numerical Recipes in C: The Art of Scientific Computing, 2nd Edition, (Cambridge University Press), 1992*

[115] **Pritsker A B** *(68) - GERT Networks, The Production Engineer pp 499-506, October 1968*

[116] **Ramakumar R J** *(93) - Engineering Reliability: Fundamentals and Applications, (Prentice-Hall), 1993*

[117] **Rich E, Knight K** *(91) - Artificial Intelligence (McGrawHill), 1991*

[118] **Rippon S** *(91) - International outage management practices Nuclear News, October 1991*

[119] **Ritchie E** *(85), Network Based Planning Techniques: A Critical Review of Published Developments, Appearing in Further Developments in Operational Research, ed. Rand G K, Eglese R W, (Pergamon), 1985*

[120] **Robertazzi T G** *(90) - Computer Networks and Systems: Queueing Theory and Performance Evaluation, (Springer-Verlag), 1990*

[121] **Rabin M O** *(76) - Probabilistic algorithms, In Algorithms and Complexity ed. J F Traub, (Academic) pp 21-39, 1976*

[122] **Reeves C** *(93) - Using GAs with Small Populations, In Proceedings of the Fifth International Conference on Genetic Algorithms, ed. S Forrest, (Morgan-Kaufmann), 1993*

[123] **Reinelt G, Bixby R** *(90) - TSPLIB 1.2, Universitaet Augsburg, (anonymous ftp: softlib.cs.rice.edu:pub), 1990*

[124] **Sanders N** *(91) - Stop Wasting Time : Computer-Aided Planning and Control, (Prentice Hall), 1991*

[125] **Schaffer J D** *(87) - Some effects of selection procedure on hyperplane sampling by genetic algorithms, In Genetic Algorithms and Simulated Annealing ed. L Davis, (Pittman), 1987*

[126] **Schoppers M J** *(89) - Universal Plans for Reactive Robots in Unpredictable Environments, In Proceedings IJCAI 10, pp 1039-46, 1989*

[127] **Shaw M, Whinston A** *(85) - Automatic Planning and Flexible Scheduling: A Knowledge-Based Approach, IEEE Transactions of the Fourth Conference on AI Applications, 1985*

[128] **Shor P W** *(94) - A Polynomial Time Algorithm for Factoring Integers on a Quantum Computer, Personal Communacation, 1994*

[129] **Stark H M** *(82) - An Introduction to Number Theory, (MIT Press), 1982*

[130] **Strang G** *(86) - Introduction to Applied Mathematics, (Wellesley-Cambridge), 1986*

[131] **Talluri K** *(91) - Issues in the design and analysis of survivable networks PhD Thesis in Operations Research, MIT, 1991*

[132] **Tan, J S** *(93) - Optimizing Preventive Maintenance: An Aging Reliability, Monte-Carlo, and Neural Network Approach, Personal Communication, Paper in Progress, 1993*

[133] **Tate A, Curroe K** *(90) - O-PLAN: Control in the Open Planning Architecture, Appearing in Readings in Planning, (Morgan Kaufmann), 1990*

[134] **Taylor G M** *(91) - Nuclear News survey: risk, duration, scope are top outage concerns, Nuclear News, October 1991*

[135] **Therrien W C** *(89) - Decision Estimation and Classification, (John Wiley & Sons), 1989*

[136] **Tidor B, de la Maza M** *(91) - Boltzmann Weighted Selection Improves Performance of Genetic Algorithms, AI Memo No 1345, (MIT AI Lab.), 1991*

[137] **Ullman J D** *(75) - NP-complete scheduling problems, J.C.S.S., v 10, pp 384-93, 1975*

[138] **Valiant L K** *(79) - The complexity of enumeration and reliability problems", SIAM Journal of Computing v 8, pp 410-21, 1979*

[139] **Van Hentenryck P** *(89) - Constraint Satisfaction in Logic Programming, (MIT Press), 1989*

[140] **Varela F, Thompson E, Rosch E** *(91) - The Embodied Mind, pp 185-201, (MIT Press), 1991*

[141] **Vose M** *(93) - Modeling Simple Genetic Algorithms, In Foundations of Genetic Algorithms, ed. D Whitley, (Morgan Kaufmann), 1993*

[142] **Walmsley S J** *(89) - Artificial Intelligence Methodologies Applied to Construction Management Decision Support Systems, ANS Topical Meeting on Computer Applications for Nuclear Power Plant Operation and Control, 1989*

[143] **Whitehouse E** *(73) - System Analysis and Design Using Network Techniques, (Prentice-Hall), 1973*

[144] **Whitley D** *(93a) - An Executable Model of a Simple Genetic Algorithm, In Foundations of Genetic Algorithms -2- ed. D Whitley, (Morgan Kaufmann), 1993*

[145] **Whitley D** *(93b) - A Genetic Algorithm Tutorial, Technical Report CS-93-103 (Colorado State University), 1993*

[146] **Wilkins D** *(86) - Practical Planning: Extending the Classical AI Paradigm, (Morgan Kaufmann), 1986*

[147] **Williamson D** *(93) - On the Design of Approximation Algorithms for a Class of Graph Problems, CS PhD Thesis, MIT, 1993*

[148] **World Bank** *(85) - Incorporating Uncertainty into Planning of Industralization Strategies for Developing Countries, World Bank Working Paper, no 503, 1985*