01 May 1987

A Representation for Serial Robotic Tasks

Barry Ross Fox

Arlan R. Dekock
Missouri University of Science and Technology, adekock@mst.edu

A REPRESENTATION FOR SERIAL ROBOTIC TASKS

Barry Ross Fox* and Arlan R. DeKock

CSc-87-6

*This report is substantially the PhD dissertation of the first author, completed Spring, 1987.

ABSTRACT

The representation for serial robotic tasks proposed in this thesis is a language of temporal constraints derived directly from a model of the space of serial plans. It was specifically designed to encompass problems that include disjunctive ordering constraints. This guarantees that the proposed language can completely and, to a certain extent, compactly represent all possible serial robotic tasks. The generality of this language carries a penalty. The proposed language of temporal constraints is NP-Complete. Specific methods have been demonstrated for normalizing constraints posed in this language in order to make subsequent sequencing and analysis more tractable. Using this language, the planner can specify necessary and alternative orderings to control undesirable interactions between steps of a plan. For purposes of analysis, the planner can factor a plan into strategies, and decompose those strategies into essential components. Using properly normalized constraint expressions the sequencer can derive admissible sequences and admissible next operations. Using these facilities, a robot can be given the specification of a task and it can adapt its sequence of operations according to run-time events and the constraints on the operations to be performed.

ACKNOWLEDGEMENTS

The author would like to acknowledge the contributions of Arlan DeKock, George Zobrist, Stephen Cameron, Stan Green, and Karl Kempf. The research reported in this thesis was supported in part by the McDonnell Douglas Independent Research and Development Program. This thesis is dedicated to the memory of Phillip Arlo Fox (1912-1987).

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

I. INTRODUCTION

Automated manufacturing systems are quickly evolving from hard automation to programmable hardware and software. That evolution is made possible by a variety of technological advances. At the same time many of the technological advances are driven by the requirements of current and future applications. The availability of programmable robots has made it possible to replace complex, custom assembly hardware with robots which can adapt to new tasks simply by changing tools and software. The availability of programmable vision systems has made it possible to eliminate complex, custom parts presentation hardware and to make parts available on simple pallets or in bins. Advanced sensors and sophisticated robot programming languages have made it possible for robots to detect error conditions and to recover from those errors by corrective actions or by pursuing alternative courses of action. Systems have been proposed which require robots to adapt not just to error conditions but more generally to the dynamically changing state of the world. For example, it has been frequently proposed that robots, with suitable intelligence and autonomy, can be used for assembly and maintenance operations in remote or hazardous environments such as deep space[1] or within the containment vessel of a

[1]Schenker, P. S., NASA Telerobotics Research: program objectives and technology outreach, presentation at NASA/JPL Space Telerobotics Workshop, Pasedena, Calif, 1987.

damaged nuclear reactor[2]. In those situations it is impossible to prepare the environment in advance in order to structure and simplify the robot's task, nor is it possible to intervene when the robot encounters unexpected conditions. Such proposals necessitate the development of new robotic and artificial intelligence technologies.

[2] Rembold, U., Levi, P., Sensors and control for autonomous robots, in Proceedings Intelligent Autonomous Systems Amsterdam, The Netherlands, 1986, pp. 79-89.

II. STATEMENT OF THE PROBLEM

A. PLANNING, SEQUENCING, AND EXECUTION

An intelligent autonomous robot operating in a remote unstructured environment must have three capabilities. First, it must be able to create a plan or course of action according to the present state of the world, a goal state of the world, and some knowledge of its own abilities. Second, it must be able to determine a sequence of actions according to the constraints on the steps of the plan and the evolving state of the world. Finally, it must be able to produce the desired effect of those actions according to its abilities and the current state of the world. This three part delineation of capabilities follows the traditional boundaries between artificial intelligence, operations research, and robotics. The distinctions are conceptually useful but should not be construed as absolute.

The first capability, planning, has received considerable attention from the artificial intelligence community. In a general sense, planning is the process which derives a course of action which, when executed, will achieve a desired goal. More precisely, a planner is given a symbolic representation of an initial state of a system, a goal state of that system, and a set of operations which can be used to effect changes in that system. The plan which the planner produces consists of a subset of those operations plus some prescription for their order of execution.

Planning has always been recognized as a difficult problem and most of the literature is concerned with techniques which make planning more tractable. The earliest planning systems[3,4] searched, in chronological order, from an initial state of the world through the space of world states which were reachable by the application of the available operators. Later, non-linear planning methods[5] were developed because the linear representation of plans, used by the earlier systems, frequently imposed artificial constraints on the order of execution. Those artificial constraints often prevented the plan under construction from being successful. In contrast, the non-linear planners represented plans as a set of actions coupled with only the most necessary constraints on the ordering of those actions. When a non-linear plan was successfully produced a linear sequence was derived from those necessary ordering constraints. The earliest planning systems produced plans by searching for a sequence of operations which leads from the initial state to the goal state. Later hierarchical

---

[3]Fikes, R.E. and Nilsson, N.J., STRIPS: A new approach to the application of theorem proving to problem solving, Artificial Intelligence 2 (1971), 189-208.

[4]Sussman, G.J., A computational model of skill acquisition, Ph.D. dissertation, AI Technical Report 297, AI Laboratory, Massachusetts Institute of Technology, 1973.

[5]Sacerdoti, E.D., A structure for plans and behavior, Ph.D. dissertation, Technical Note 109, AI Center, SRI International, Inc., Menlo Park, Calif., 1975.

planners[6] produced plans by constructing a hierarchy of goals and sub-goals. Such planners decomposed the given problem into a set of sub-goals which must be achieved and those in turn were decomposed until the given problem had been reduced to a set of primitive operations to be performed. Other planning systems[7,8] have attempted to take advantage of specific knowledge of the problem domain by producing plans from a library of basic templates; these templates are then modified and refined according to the specific problem under consideration. Current research in planning is quite polarized. Some research is concerned with formal, domain independent methods for prescribing goals and actions and with formal methods for deriving plans[9]. Other research adopts the thesis that realistic planning is dominated by domain specific knowledge and heuristics[10]. The

---

[6]Tate, A., Project planning using a hierarchic non-linear planner, Report Number 25, AI Research Department, University of Edinburgh, 1976.

[7]Friedland, P.E., Knowledge-based experiment design in molecular genetics, Ph.D. dissertation, Report Number 79-771, Computer Science Department, Stanford University, 1979.

[8]Stefik, M.J., Planning with constraints. Ph.D. dissertation, Report Number 80-784, Computer Science Department, Stanford University, 1980.

[9]Chapman, D., Non-linear planning: a rigorous reconstruction, in Proceedings Ninth International Joint Conference on Artificial Intelligence, Los Angeles, Calif., 1985, pp. 1022-1024.

[10]Cheeseman, P., Overview of planning/scheduling problems and procedures, in Proceedings NASA/JPL Space Telerobotics Workshop, Pasedena, Calif., 1987 (to appear).

debate over the appropriate representation for plans continues with various formal languages[11,12] and models[13] in contention.

The second capability, sequencing, serves to reconcile the constraints imposed by the planner with the constraints imposed by the executor and the execution time environment. In a general sense, sequencing is the process which determines the order for executing the operations prescribed by a plan. That order must be consistent with the constraints imposed by the planner and it must be consistent with the availability of the resources which are required by the executor, i.e., an operation can be performed only after the necessary predecessor operations have been completed and only when the necessary resources are available.

The third capability required by an autonomous robot is the ability to produce the desired effect of the actions prescribed by the planner and ordered by the sequencer. Ultimately, the responsibility of the executor is to map abstract operations, such as **install-retaining-clip**, into a

---

[11]Cheeseman, P., A representation of time for automatic planning, in Proceedings Second IEEE International Conference on Robotics and Automation, Atlanta, Georgia, 1983, pp. 513-518.

[12]Allen, J.F. and Koomen, J.A., Planning using a temporal world model, in Proceedings Eighth International Joint Conference on Artificial Intelligence, Karlsruhe, West Germany, 1983, pp. 741-747.

[13]Drummond, M., Plan Nets: a formal representation of action and belief for automatic planning systems, Ph.D. Dissertation, Department of Artificial Intelligence, University of Edinburgh, 1986.

combination of real physical motions. These operations must be at an appropriate level of abstraction for the planner, which must reason about high-level goals and effects, but at the same time be at an appropriate level of abstraction for the executor, which must produce the desired effects. From this perspective, an abstract operation is simply a goal and execution may involve the recursive invocation of planning, sequencing, and execution over a domain of more primitive goals, operations, and constraints. For example the abstract operation, **install-retaining-clip**, involves the processes of locating the part, identifying the target location, acquiring the part, moving the part from the source to target locations, and successfully mating the part with the previously installed components. Hence an essential element of an autonomous intelligent system is a vocabulary of abstract operations which the planner and sequencer can treat as atomic even though the executor may decompose those operations into a vocabulary of more primitive operations. Current robotics research is involved in the definition and implementation of primitive operations, such as parts recognition[14], grasp planning[15], trajectory planning[16], and

[14]Bolles, R.C. and Cain, R.A., Recognizing and locating partially visible objects: the local-feature-focus method, International Journal of Robotics Research 1 (1982), 57-82.

[15]Salisbury, J.K. and Craig, J.J., Articulated hands: force control and kinematic issues, International Journal of Robotics Research 1 (1982), 4-17.

[16]Lozano-Perez, T., A simple motion planning algorithm for general robot manipulators, in Proceedings Fifth National

parts mating[17] which can be used to implement a vocabulary of abstract operations.

B. SEQUENCING

Sequencing and scheduling problems have traditionally fallen within the domain of operations research and because of extensive study in the field these problems have been divided into many specific categories. Among these are the problems of assembly line balancing, job shop scheduling, timetable scheduling, project scheduling, and routing. These categories are distinguished by the kinds of activities to be scheduled, the kinds of constraints on those activities, and the nature of the costs associated with them. Each category of scheduling problem is an abstraction of some frequently occurring real world problems. The expectation is that effective methods for solving these abstract problems will produce cost saving solutions to the real world problems. At the same time, each category of scheduling problem is associated with a fundamental combinatorial problem which best characterizes the essential nature of

Conference on Artificial Intelligence Philadelphia, Penn., 1986.

[17]Lozano-Perez, T., Mason, M.T., and Taylor, R.H., Automatic synthesis of fine-motion strategies for robots, in Robotics Research: The First International Symposium M. Brady and R. Paul (eds), MIT Press, Cambridge, Mass., 1984.

that problem, for instance bin packing, graph coloring, or the traveling salesman problem[18].

Often in the Operations Research literature the term scheduling is used interchangeably with the term sequencing. In this thesis every attempt will be made to use the two terms consistently according to the following conventions: sequencing is the process of determining the relative ordering of the operations under consideration and scheduling is the process of determining the absolute time of occurrence for those operations. Clearly, relative orderings can be directly derived from absolute times of occurrence. In many problems the reverse is also true, absolute times of occurrence can be derived from a combination of the relative orderings and the expected durations of a set of operations.

Discussion of sequencing and scheduling problems invariably involves some discussion of algorithms and efficiency. A few historical comments and some basic definitions will help give some perspective on the complexity of sequencing and scheduling.

Successful application of linear programming in planning the rate of production and distribution of materials quite naturally led to attempts to develop efficient, optimal algorithms for sequencing and scheduling the discrete activities of production and distribution. In

---

[18]Garey, M.R. and Johnson, D.S., Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, New York, 1979.

most cases these attempts were constantly frustrated. For example, the following comment is typical of early research in job-shop scheduling:

> "The general job-shop problem is a fascinating challenge. Although it is easy to state, and to visualize what is required, it is extremely difficult to make any progress whatever toward a solution. Many proficient people have considered the 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."[19]

Other authors, involved in assembly line balancing, recognized the essential difficulty of their problem but perceived that the failure to solve the problem was due to a failure to find just the right algorithm.

> "As noted by Kilbridge and Wester most, if not all, of the mathematically based methods proposed for line balancing problems seem to be impractical for realistically sized problems. This is due to the amount of enumeration that they all require. Our suggestion is likely to suffer from the same defect; the required enumeration of all feasible orderings will probably be impractical for 'large' problems."[20]

Later developments in the theory of computability and complexity made it possible to understand the essential difficulty of these problems. Most of the apparently intractable sequencing and scheduling problems are provably NP-Hard. Although it has not yet been proven, it is

---

[19]Conway, R. W., Maxwell, W. L., and Miller, L. W., Theory of Scheduling, Addison-Wesley, Reading, Mass., 1967, p. 103.

[20]Klein, M., On assembly line balancing, Operations Research 11 (1963), 281.

generally believed that any deterministic algorithm for the solution of an NP-Hard problem will require execution time which is exponentially related to the size of the given instance of the problem.

Technically, a problem is NP-Hard if it can be shown that a polynomial-time deterministic algorithm for that problem can be used to produce a polynomial-time deterministic algorithm for some NP-Complete problem. The NP-Complete problems are so related that a polynomial time deterministic algorithm for one can be used to construct a polynomial-time deterministic algorithm for any of them. The NP-Complete problems share the additional property that the best known deterministic algorithms for their solution require exponential execution time. It is generally believed, although it has not yet been proven, that no polynomial-time deterministic algorithm will ever be found for the NP-Complete problems. If this is true then no polynomial-time deterministic algorithm will ever be found for the NP-Hard problems either.

Current research in sequencing and scheduling specifically addresses the apparent intractability of these problems. For some researchers, the emphasis is on finding specific subclasses of these problems which do have efficient algorithms. For others, the emphasis is on finding methods which generally reduce the amount of time required to find the optimal solution, or which improve the quality of solution found in a fixed amount of time. Still others pursue higher-order methods which balance the increasing cost of increasingly accurate heuristics with the decreasing return which they yield.

1. Job Shop Scheduling. In one sense, the most general sequencing problem is the job shop scheduling problem which involves sequencing the execution of multiple activities making use of multiple resources. One model of the job shop scheduling problem[21] is defined by a set of machines and a set of jobs. Each job is composed of a set of steps and a precedence relation over those steps, and each step requires a specific processing time on a specific machine. A solution to a given job shop scheduling problem consists of an assignment of each job step to a machine and an interval of time such that the job steps obey the given precedence constraints and the machines are dedicated to only one job step at a time. The goal is to find a solution which minimizes the cost of production, as defined by some objective function, such as the time required to complete all of the jobs. This can be properly viewed as a sequencing problem since all of the absolute time intervals can be derived from the expected durations of the job steps and the relative ordering of jobs steps over the resources. Some variations of this model require that all machines are identical, or that all processing times are identical, or that the precedence constraints within a job are strictly linear. Other variations restrict jobs to single steps, but

[21]Conway, Maxwell, and Miller, Theory of Scheduling.

admit precedence constraints between jobs. The scope of the job shop scheduling problem is so broad that most texts on the matter introduce some classification scheme in the early chapters.[22,23]

Like most sequencing and scheduling problems, the job shop problem is NP-hard. This is demonstrated by proving that an arbitrary instance of the NP-Complete problem known as 3-Satisfiability can be encoded as an instance of job shop scheduling in polynomial time[24]. If the job shop scheduling problem could be solved deterministically in polynomial time then a given instance of the 3-Satisfiability problem could be encoded as a job shop scheduling problem and solved in polynomial time. (The total execution time would be the sum of the time required to encode the problem and the time required to solve the problem. The sum of two polynomial functions is itself a polynomial function.) Because of the apparent intractable nature of the problem, current job shop scheduling research has focused on heuristic methods or on special cases of the problem.

---

[22]Section 1-3: A Classification of Scheduling Problems, in Conway, Maxwell, and Miller, Theory of Scheduling.

[23]Section 2: A Classification of Scheduling Problems, in Scheduling and Sequencing, Michael S. Salvador, Handbook of Operations Research: models and applications, Moder, J.J. and Elmaghraby, S. E. (eds), Van Nostrand Reinhold, New York, 1978.

[24]Ullman, J. D., NP-complete scheduling problems, Journal of Computer and System Sciences 10 (1975), 384-393.

Two kinds of heuristics are frequently proposed in the literature: those which generally reduce the time required to find an optimal solution, and those which generally produce good solutions in a limited amount of time. Examples of the former are usually based upon a heuristic search strategy, such as branch-and-bound or A*[25]. Examples of the latter are usually based upon some dispatch rule which determines the priority of jobs waiting to be processed. Although branch-and-bound methods do significantly prune the search space, they are somewhat unpredictable. Moreover, the search space for job shop scheduling problems is so large that even branch-and-bound algorithms are effectively limited to small problems. For such reasons most job shop scheduling research has focused on the latter category of heuristics.

Scheduling algorithms based upon dispatching rules are very simple. They determine the assignment of job steps to time intervals by simulating the jobs, the machines, the advance of time, and the application of the dispatching rule. Each time a machine completes a job step the simulator identifies all job steps which are waiting for that machine and schedules the one that has the highest priority according to the given dispatching rule. Priority can be determined by any conceivable combination of factors: the amount of time for the current job step, the amount of time

[25]Pearl, J., Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley, Reading, Mass., 1984.

remaining in the job, the due date of the job, etc. Under this methodology, the time required to schedule n job steps is on the order of n^2: n scheduling decisions must be made, at each decision at most n job steps need be considered. If all of the jobs consist of just one step, there are no precedence requirements, and all of the machines are identical, then the time required for this process is on the order of n log m, where m is the number of machines. One survey of dispatching rules itemizes 113 different strategies and thirty-six articles which analyze their performance[26]. A more recent article[27] surveys thirty-four dispatching rules. According to that survey, the single most effective dispatching rule, when ranked according to cost-based criteria, is one which gives priority to the job step with the shortest imminent (SI) processing time. This is quite surprising! After thirty years of research the most effective dispatching rule is the simplest.

In spite of this volume of research scheduling by dispatching rules is only an approximate method which occasionally produces anomalous results. Simple examples can be constructed which yield worse solutions when the number of machines is increased, the precedence constraints are

---

[26]Panwalkar, S.S. and Iskander, W., A survey of scheduling rules, _Operations Research_ 25 (1977), 45-61.

[27]Blackstone Jr., J.H., Phillips, D.T., and Hogg, G.L., A state-of-the-art survey of dispatching rules for manufacturing job shop operations, _International Journal of Production Research_ 20 (1982), 27-45.

relaxed, or the processing times are reduced[28]. If schedulers are going to routinely rely upon such rules it is important to precisely quantify the bounds of their behavior. Most often the performance of a dispatching rule is expressed as a ratio. Some researchers characterize the performance as the ratio between the expected cost of a solution and the cost of the optimal solution. Others refer to the ratio between the worst case solution and the optimal solution. Ratios of the expected cost are hard to justify because so many simplifying assumptions must be made to make the analysis tractable. Moreover, the expected value of a solution gives no indication of the bounds on the solution produced by a given dispatching rule. On the other hand, if an instance of the job shop scheduling problem can be constructed which is guaranteed to yield the worst solution under a given dispatching rule then the performance of that rule can be bounded precisely[29]. Such worst cases instances can then be used to qualify suitable contexts for applying the given dispatching rule. Further analysis may even suggest modifications to the rule which correctly compensate for those worst case instances.

Perhaps the most interesting research involves what can best be characterized as higher-order heuristics. The

---

[28]Hu, T. C., Combinatorial Algorithms, Addison-Wesley, Reading, Mass., 1982, pp. 222-228.

[29]Garey, M. R., Graham, R. L., and Johnson, D. S., Performance guarantees for scheduling algorithms, Operations Research 26 (1978), 3-21.

general method is to choose some simple heuristic, such as lookahead, which can be applied to any desired depth, but to use it only to the depth where the value of the scheduling solution justifies the computational cost. Such methods are motivated by the law of diminishing return: exponentially more costly heuristics yield exponentially decreasing improvements in their solutions. Obviously there must be some saddle point where the increasing cost is unjustified. Refinements of this method result in algorithms which produce near-optimal solutions in polynomial time[30].

Other job shop scheduling research is not concerned with heuristic methods but with specific cases of the scheduling problem which can be solved in polynomial time. One case involves scheduling jobs that have tree structured precedence constraints and specifically limited execution times[31]. Other cases involve job shops of only two machines and specific limitations on the routing of jobs[32] or limitations on the execution times[33].

All of the research cited above falls clearly within the domain of Operations Research but a new development is the application of artificial intelligence to the same

---

[30]Graham, R. L., The combinatorial mathematics of scheduling, Scientific American 238 (1978), 124-132.

[31]Hu, Combinatorial Algorithms, 228-236.

[32]Gillett, B. E., Introduction to Operations Research, McGraw-Hill, New York, 1976, pp. 262-277.

[33]Coffman, E. G. and Graham, R.L., Optimal scheduling for two-processor systems, Acta Informatica 1 (1972), 200-213.

problems. One well established artificial intelligence scheduling system is ISIS[34], developed at the Robotics Institute of Carnegie-Mellon University, which is frequently described as a system for constraint directed reasoning. This system is effective primarily because it has facilities for itemizing and resolving all of the constraints that might possibly have an impact on the resulting schedules. These include the factors normally considered by OR systems, such as precedence constraints, capacity constraints, personnel costs, etc. The unique feature of ISIS is that it accepts any constraint that can be encoded as a LISP function or predicate. With this capability, a scheduler and a programmer can construct a model of any given job shop and include any desired elements of cost or constraint. The scheduling algorithm itself, however, is primitive. It operates, much like dispatch rule scheduling, by simulating the jobs, the machines, and the advance of time. Each time a machine completes a job step, the simulator identifies all job steps which are waiting for that machine. That set is then pruned by the given constraint functions, and those that remain are ranked by the given cost functions. That set is then reduced to the K best, and the scheduler systematically explores the schedules that emanate from the assignment of those K jobs to the idle machine. If this

---

[34]Fox, M. S., Constraint-Directed Search: A Case Study of Job-Shop Scheduling, Ph.D dissertation, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Penn.

process fails to produce an acceptable schedule, the scheduler attempts to construct the schedule in reverse, from the given due dates. If that fails, the scheduler systematically relaxes some of the constraints and tries again. Although the scheduling algorithm is primitive, the resulting schedules are considered to be quite good because the model builder can incorporate any factor which is considered significant.

2. Assembly Line Balancing. Focusing on situations that involve sequencing the execution of one job over multiple machines reduces the subject domain to the assembly line balancing problem. An instance of the assembly line balancing problem is defined by a set of tasks, the processing times for those tasks, a precedence relation over those tasks, and a desired cycle time for completing the entire set of tasks. A solution to a given assembly line balancing problem consists of an assignment of tasks to work stations such that the total amount of work assigned to any work station does not exceed the desired cycle time, and the order of the tasks is consistent with the given precedence relation. The goal is to find a solution which minimizes the total number of work stations.

Like most sequencing and scheduling problems, assembly line balancing is NP-hard. This is informally demonstrated by the following argument. Bin packing problems can be converted to assembly line balancing problems by a simple procedure. An instance of the bin packing problem consists of a set of items, the sizes of the items, and an unlimited number of bins of some fixed capacity. The problem is to determine the minimum number of bins required to contain the items. The conversion is accomplished by mapping items to tasks, the size of an item to the time required for the task, and the capacity of the bins to the cycle time. The remaining element of the assembly line balancing problem, the precedence relation is left empty. If a polynomial time algorithm is found for the assembly line balancing problem then a given instance of the bin packing problem can be converted to an assembly line balancing problem and solved in polynomial time. It has already been established that the bin packing problem is NP-hard[35], hence the same must be true of the assembly line balancing problem.

The earliest citation to assembly line balancing is Salveson[36] which serves to introduce the problem. The first practical algorithm[37] focused on the enumeration of feasible workstations. It is interesting to note that over twenty years later, this algorithm proved to be one of the most effective[38], and its influence is clearly seen in the state-

---

[35]Garey and Johnson, <u>Computers and Intractability</u>, pp. 124-128.

[36]Salveson, M. E., The assembly line balancing problem, <u>Journal of Industrial Engineering</u> 6 (1955), 18-25.

[36]Jackson, J. R., A computing procedure for a line
ckson, J. R., A computing procedure for a line balancing
problem, <u>Management Science</u> 2 (1956), 261-271.

[38]Johnson, R. V., Assembly line balancing algorithms: computational comparisons, <u>International Journal of Production Research</u> 19 (1981), 277-287.

of-the-art algorithm discussed below. A later algorithm[39] is based upon explicit enumeration of all possible orderings of the tasks and the partition of each ordering into workstations. Another algorithm[40] is based upon the explicit enumeration of all possible states in the execution of the tasks, which are less numerous than the possible orderings. Subsequent research introduced a wide variety of algorithms, including various dynamic programming, integer programming, and branch-and-bound formulations, which have been analyzed and compared in a number of papers.[41,42]. According to Mastor's study the dynamic programming algorithm of Held, Karp, and Sharesian was quite effective. The later study by Johnson indicates that Johnson's own branch-and-bound methodology, or Jackson's dynamic programming algorithm were generally faster and required less memory.

Recent work falls into two categories: refinements of established techniques, and variations on the problem formulation. Each of these will be discussed briefly.

---

[39]Klein, M. On assembly line balancing, Operations Research 11 (1963), 281.

[40]Held, M., Karp, R. M., and Shareshian, R., Assembly-line balancing - dynamic programming with precedence constraints, Operations Research 11 (1963), 442-459.

[41]Mastor, A. A., An experimental investigation and comparative evaluation of production line balancing techniques, Management Science 16 (1970), 728-746.

[42]Johnson, Assembly line balancing algorithms.

For instance, one algorithm[43] introduces a refinement of previous integer programming solutions and at the same time effectively applies the principles introduced by Jackson's dynamic programming solution. The general approach of that algorithm is to implicitly enumerate all possible task to workstation assignments under the guidance of a sophisticated set of heuristics. Unfortunately, the strict integer programming representation of assembly line balancing problems often requires an excessive number of variables and constraints. Hence, the first refinement introduced in that paper is a scheme which implicitly represents the integer programming problem without introducing all of the intermediate variables and constraints. The net effect is a much more compact representation with the possibility of solving much larger problems.

This compact representation is then coupled with a sophisticated set of problem specific heuristics. First, in order to prevent blind enumeration of task assignments, the algorithm precomputes the earliest and latest workstations into which a task can be assigned. At the same time it identifies all tasks, which because of their duration and the ordering constraints, must necessarily be assigned to a workstation alone. This information is then used to identify

[43]Talbot, F. B. and Patterson, J.H., An integer programming algorithm with network cuts for solving the assembly line balancing problem, Management Science, vol 30 (1984), 85-99.

distinguished workstations, called network cuts, at which special backtracking and fathoming heuristics can be applied. Additional heuristics involve comparison of the current partial solution with previous solutions. Those that show no significant differences are abandoned, as are those that show excessive idle time. Then to give the algorithm some initial momentum, a first solution is constructed using a simple heuristic, such as maximum task time first.

This work is significant in three ways. First, it carefully refines the algorithms and data structures of traditional integer programming. Second, it applies multiple heuristics which separately have been shown to be effective in previous systems. Third, it introduces heuristics which are selectively applied according to the structure of the problem and the state of the solution process. As a whole it represents an effective refinement and composition of established techniques and the introduction of unique context sensitive heuristics.

Another algorithm[44] relies upon some refinements of the dynamic programming solution first proposed by Held, Karp, and Shareshian. Although some dramatic improvements in the storage requirements of the algorithm are presented (reductions by at least half!) this method still requires the explicit enumeration of all possible states in the execution of the tasks. Because of this, the method is

[44]Kao, E. P. C. and Queyranne, M. On dynamic programming methods for assembly line balancing, Operations Research 30 (1982), 375-390.

limited to much smaller problems than the implicit enumeration method discussed above.

Other papers focus on variations on the assembly line balancing problem rather than refinements of the algorithms. These papers are motivated by the fact that real assembly lines are much more complex than the assembly line balancing model admits.

For instance, the basic model requires that the set of tasks to be performed is fixed, and the problem is simply to assign those tasks to workstations. However, the design of the assembly line, and the decomposition of the assembly into tasks are closely coupled. Hence, a fruitful area of research is to expand the scope of the problem and to produce algorithms which balance assembly lines which may have processing alternatives[45].

Other variations on the basic model involve the unique problems of robotic assembly lines. A robot cannot be programmed to perform an arbitrary combination of tasks. Each robot has specific limitations on its speed, strength, reach, accuracy, etc., and each capability has a clearly identifiable cost. All of the tasks assigned to a single robot must be within its capabilities but failure to use some capability is wasteful. Moreover, commercial robots are not yet equipped with general purpose hands; instead robots are usually equipped with interchangeable tools which are

[45]Pinto, P. A., Dannenbring, D. G., and Khumawala, B. M., Assembly line balancing with processing alternatives: an application, Management Science 29 (1983), 817-830.

specifically tailored to the operations that must be performed. The assignment of arbitrary combinations of tasks to a single workstation may require the robot to make frequent tool changes. Unlike humans, the task of changing to a new tool may be as difficult and time-consuming for a robot as performing its assembly tasks. Additionally, it is impossible to accurately predict the time required to perform a task without first designing the workstation and planning the robots operations[46]. Hence, balancing robotic assembly lines must take into consideration a large number of cost factors not normally associated with human assembly lines[47].

Other research has focused on less extravagant but still important, realistic, and frequently occurring variations on the assembly line balancing problem. For instance, one author presents nine variations of the problem and a branch and bound algorithm for their solution together in a single paper[48]. These variations include the assignment of tasks to particular types of stations, to specific stations, to a specific side of the assembly line, or even problems which prohibit the assignment of certain tasks to

---

[46]Kondoleon, A. S., Assessing cycle times for robot assembly systems, Robotics Today 3 (1981), 38-41.

[47]Graves, S. C. and Lamar, B. W., An integer programming procedure for assembly system design problems, Operations Research 31 (1983), 522-545.

[48]Johnson, R. V., A branch and bound algorithm for assembly line balancing problems with formulation irregularities, Management Science 29 (1983), 1309-1324.

the same station or problems which intentionally require an imbalance in the work assigned to the stations

3. Sequencing the Activities of a Robot. Focusing on situations that involve sequencing the execution of one job on one machine reduces the subject domain to the problem of sequencing the activities of a robot. An instance of this problem is defined by a single job, composed of a set of steps, where each step is subject to specific resource requirements and specific ordering constraints. If the resource requirements are guaranteed to be satisfied then a solution to a given instance of this problem is simply a linear ordering of the steps which is consistent with the ordering constraints. If the sequence of operations must be adapted to the availability of the resources then the problem becomes significantly more complicated. Unfortunately, even in the absence of resource constraints, the problem of sequencing the activities of a robot is very difficult. It will be shown in a later section that given a sufficiently general language for posing the ordering constraints on the steps of a job, the problem of finding a sequence of operations which is consistent with the given constraints is NP-hard.

This is a relatively new problem, motivated by the availability of general purpose robots, sophisticated programming languages, and the significant accomplishments in parts recognition, grasp planning, trajectory planning, and parts mating. For this reason, the papers devoted to this subject are relatively few and are found mostly in the robotics and artificial intelligence rather than operations research literature.

Three instances of this problem seem representative of the active research in the area. The earliest of these was an experimental assembly system developed at Edinburgh University[49,50]. That system consisted of a four degree of freedom arm and a vision system both mounted over a two degree of freedom cartesian table. Assembly tasks submitted to the robot consisted of a jumbled heap of parts on the table from which the robot was required to acquire the individual parts and perform the assembly. A more recent system was developed as a joint venture by SRI, Honeywell, and Adept Technology under contract from the Air Force[51]. That system consisted of a six degree of freedom robot and a vision system mounted over a fixed table. The assembly tasks submitted to the robot consisted of a jumbled bin of parts from which the robot was required to acquire the individual parts and perform the assembly. A system currently under

---

[49]Ambler, P., Barrow, H.G., Brown, C.M., Burstall, R.M., and Popplestone, R.J., A versatile computer-controlled assembly system. in Proceedings Third International Joint Conference on Artificial Intelligence, Stanford University, 1973, pp. 298-303.

[50]Ambler, P., Barrow, H.G., Brown, C.M., Burstall, R.M., and Popplestone, R.J., A versatile system for computer-controlled assembly, Artificial Intelligence 6 (1975), 129-156.

[51]Interim Report #3, Research for Intelligent Task Automation, Air Force Contract #F33615-82-C-5092, July 15, 1983.

development at the Robotics Institute of Carnegie Mellon University under a contract with Westinghouse consists of two robots, a vision system, and a conveyor[52]. The assembly tasks submitted to this robot consist of a series of parts delivered in random order by the conveyor from which the robot must acquire the parts and perform the assembly. All three problems share the same characteristics. The job to be performed consists of a well defined set of operations. Those operations have specific resource requirements, namely the parts, and specific ordering constraints. The problem is not simply to find one linear ordering of operations which satisfies the ordering constraints because there is no way to guarantee that the parts can be acquired or will be delivered in precisely that order. Instead the problem is to develop a method for sequencing the activities of the robot which will produce a sequence of operations consistent with the ordering constraints and the availability of parts.

The systems cited above developed three unique solutions to the problem of satisfying the ordering constraints simultaneously with satisfying the resource constraints. The approach used at Edinburgh was to pursue the task in two phases. In the first phase the robot separated the parts from the heap one by one, and according to their identity, moved them to fixed locations in the

---

[52]deMello, L.S.H. and Sanderson, A., And/Or graph representation of assembly plans, in Proceedings Fifth National Conference on Artificial Intelligence, Philadelpha, Penn., 1986, pp. 1113-1119.

workspace. In the second phase, with all of the parts identified and available, the robot performed the assembly according to a predetermined sequence of operations. The approach used at SRI was to build a better vision system. In fact the mandate of the research contract was to apply the best available vision hardware and software so that at each step of a predetermined assembly sequence the robot could locate and acquire the required part. Of course it is impossible to visually locate parts that are obscured and covered by other parts, so the assembly program included an escape mechanism: if the required part could not be located the robot would shake the bin or stir the parts with the expectation that the required part would be revealed. A third approach, advocated by this author and others[53,54,55,56,57,58,59] and adopted at Carnegie Mellon

---

[53]Fox, B.R. and Ho, C.Y., A relation control mechanism for flexible assembly, in Advanced Software in Robotics, A. Danthine and M. Geradin (eds), North-Holland, Amsterdam, 1984.

[54]Fox, B.R. and Kempf, K.G., Opportunistic scheduling for robotic assembly, in Robotics and Industrial Engineering, Selected Readings, E.L. Fisher and O.Z. Maimon (eds), Industrial Engineering and Management Press, Institute of Industrial Engineers, Atlanta, Georgia, 1986.

[55]Fox, B.R. and Kempf, K.G., Complexity, uncertainty, and opportunistic scheduling, in Artificial Intelligence Applications, The Engineering of Knowledge-Based Systems, C. Weisbin (ed), IEEE COmputer Society Press, Washington, D.C., 1986.

[56]Fox, B.R. and Kempf, K.G., A representation for opportunistic scheduling, in Third International Symposium on Robotics Research, O. Faugeras and G. Giralt (eds), MIT Press, Cambridge, Mass., 1986.

University[60], and at Edinburgh and Aberdeen Universities under a joint program of research[61], is to dynamically determine the sequence of operations according to the initial ordering constraints and the order of the availability of the parts.

On closer inspection, candidate solutions to this problem fall into one of the five categories: eliminate uncertainty, quantify uncertainty, restore certainty, increase the likelihood of progress, or increase the avenues for progress.

Eliminate Uncertainty: The most frequently proposed solution to this problem is to eliminate the uncertainty in the availability of the parts by re-engineering the task. Instead of delivering the parts in a bin or heap, deliver the parts affixed to a pallet with specific locations and orientations. Or instead of delivering the parts in random

---

[57]Fox, B.R. and Kempf, K.G., Planning, scheduling, and uncertainty in the sequence of future events, in Uncertainty in Artificial Intelligence, Vol. 2, J. Lemmer (ed), North-Holland, Amsterdam, 1987 (to appear).

[58]Fox, B.R., The implementation of opportunistic scheduling, in Proceedings Intelligent Autonomous Systems, Amsterdam, 1986, pp. 231-240.

[59]Fox, B.R. and Kempf, K.G., Reasoning about opportunistic schedules, in Proceedings IEEE International Conference on Robotics and Automation, Raleigh, North Carolina, 1987 (to appear).

[60]deMello and Sanderson, And/Or graph representation of assembly plans.

[61]Malcolm, C. and Fothergill, P., Some architectural implications of the use of sensors, in Proceedings Intelligent Autonomous Systems, Amsterdam, 1986, pp. 71-78.

order on the conveyor, introduce some system which guarantees the order of delivery. The robot can then be programmed to perform the task according to some predetermined sequence of operations according to the fixed locations of the parts or the expected order of arrival. Of course the cost of this solution is determined by the cost of engineering the pallets and transport. The cost may be justifiable in large volume applications but it is unacceptable in the aerospace industry, for instance, where millions of parts are produced in quantities of 100 or less per year. The inventory of pallets and fixtures would exceed the value of the parts to be produced. Moreover, in many circumstances human labor would be required to prepare the pallets. The motive for introducing vision and artificial intelligence in such applications is to automate the production without introducing the added expense of custom fixtures, pallets, and transport.

Quantify Uncertainty: Yet another solution might be to gather statistics over hundreds of trials and to program the robot to perform the assembly over the sequence of parts most likely to occur. Unfortunately, this only displaces the problem, it does not address the question of how to sequence the activities of the robot when the sequence of parts deviates from the programmed order. Hence, this is only a partial solution and it carries with it the added cost of performing the experiments and gathering the statistics.

Again, such costs are unjustifiable in small volume productions.

Restore Certainty: The solution implemented in the Edinburgh system cited above, is to restore certainty whenever it is lost. The parts were presented to the robot as an unordered heap which the robot immediately separated into individual parts which were moved to fixed layout locations. Although the parts would most certainly be acquired from the heap in a random order, the process of placing the parts in fixed locations restored the order necessary in order to perform the assembly according to a predetermined sequence of operations. The cost of this solution is determined by the added execution time required to lay out the parts. Although the number of motions required to perform the the assembly is fixed (one motion to acquire and buffer each part, and one motion to install each part) on the average, many of the intermediate motions are extraneous[62]. A more efficient solution would be to move a part to a buffer location only if it cannot be immediately installed in the assembly.

Increase the Likelihood of Progress: The system developed by SRI, Honeywell, and Adept Technology can be properly viewed as a Markov process. There is an initial state of the task, a final state of the task, and a fixed number of states in between. Each state is determined by the

[62]Fox and Kempf, Opportunistic scheduling for robotic assembly.

set of parts that have already been installed and each state transition is produced by the installation of a new part. Since the SRI robot was programmed to pursue a fixed assembly sequence, the corresponding Markov Process consists of a purely linear chain of states with no branching or alternative orderings. At each step of the process there is a certain probability of locating the next part and making progress and a complementary probability that the task will remain in the same state and that the robot will need to shake the bin or stir the parts. The goal of that research was to build a better vision system in order to increase the likelihood of finding the required part and thereby increase the likelihood of progress in each state of the process. The net effect was to reduce the expected duration of the assembly task. There is no way to completely remove the uncertainty from this system and this solution still carries with it a degree of uncertainty.

Increase the Avenues for Progress: There is a common weakness in the four prior categories of solutions. They focus exclusively on the process of acquiring the parts and they fail to apply knowledge of the assembly task itself. In those situations where there is no way to predict nor control the sequence of parts as they may be acquired or delivered, there may still be many feasible sequences for installing the parts in the assembly. The intelligent solution is to exploit this inherent flexibility and to opportunistically sequence the set of assembly operations at execution time, according to the availability of parts and according to the constraints on the assembly operations. This approach is also based upon a model of the assembly task as a Markov Process but with an important difference. The likelihood of transition from a particular state can be increased by increasing the number of possible successor states, that is, to enable alternative orderings for the installation of the parts.

C. RESTRICTION OF THE PROBLEM DOMAIN.

The implementation of an opportunistic scheduling strategy depends upon the consistent application of the principle of least commitment. It is not unusual for a planner to determine the steps of a task and also a single fixed sequence for their execution. However, most tasks can be executed according to many different sequences. It is very unlikely that one, chosen before execution time, will be the best. It is certain, however, that if only one sequence is passed to the robot it will have no flexibility in the execution of that task. In order to maximize flexibility, a plan should consist of a set of steps and a minimum set of ordering constraints thereby encompassing every feasible sequence of steps. Most extant planning systems, however, generate a plan as a linearly ordered set of actions or at best produce partially ordered sets of actions. Hence, a challenging area of research would be to develop a planner which could produce a plan with a minimally ordered set of actions.

This concept of plan leads to a unique problem of representation. The principle of least commitment dictates that the plan for a task encompass every feasible sequence of steps, and that in turn dictates a representation which completely and compactly encodes those sequences. Unfortunately, the most common forms for representing plans fail to satisfy one or both of these requirements. Hence, a second area of research would be to develop a representation for plans which would be complete and compact, and which would facilitate the processes of planning and sequencing. In fact, the consensus among several of the individuals actively engaged in this area of research is that the first issue that needs to be addressed is the matter of representation[63]. The significance of representation is emphasized in a note by Nilsson:

> "...artificial intelligence (AI) is primarily concerned with propositional languages for representing knowledge and with techniques for manipulating these representations. In this respect, AI is analogous to applied mathematics; its representations and techniques can be applied in a variety of other subject areas. Typically, AI research is (or should be) more concerned with the general form and properties of representational languages and methods than it is with the content being described by these languages."[64]

[63]Results of a personal poll conducted by the author among C. Malcolm and M. Drummond of Edinburgh, P. Fothergill of Aberdeen, A. Sanderson of Carnegie-Mellon, P. Cheeseman of NASA/ARC and K. Kempf of FMC Corp.

[64]Nilsson, N. J., from the abstract of Artificial Intelligence: Engineering, Science, or Slogan?, Technical Note 248, SRI International, Menlo Park, Calif, July 1981.

The remainder of this thesis will be concerned with the problem of representing the tasks that a robot is to perform and with methods for reasoning over those representations.

III. SURVEY OF THE CANDIDATE REPRESENTATIONS

The representation of plans plays a central role in the processes of planning and sequencing. In a general sense, a plan is a collection of information which has been produced by an analysis of the problem to be solved and has been encoded in some machine readable language from which the sequence of operations can be derived. The representation is the exchange point between the planner and the sequencer. It determines what information must be produced by the planner and it determines what information is available to the sequencer.

As noted previously, in early planning systems it was found that a linear representation for plans introduced some artificial ordering constraints which frequently caused the plan under construction to fail. Consequently, considerable effort was required to find a linear ordering which was successful. Similarly, a linear ordering of operations seriously limits the process of sequencing. If the plan delivered to the sequencer is a linear ordering of operations then the sequencer has only one option. The steps must be executed in exactly that order. If the resources required by the next step of the plan are not available then execution must be delayed until they are, even if the resources are available for some other steps. In later planners, use of partial orders facilitated the process of planning by accurately representing the necessary ordering constraints and by eliminating any which were unnecessary.

Likewise, partial orders have already been shown to increase the flexibility of the sequencer by allowing alternative orderings of the operations.[65] Frequently when one operation is delayed because of the unavailability of resources, there are other sibling operations that can be executed instead.

A. <u>THE CANONICAL REPRESENTATION OF PLANS</u>

Regardless of its form, a plan must precisely define the operations that are to be performed and it must precisely circumscribe the admissible orderings of those operations. Given that this work is concerned with plans that are to be executed by a single robot, the prescription for the admissible orderings carries with it an implicit constraint that the operations are to be performed one step at a time. Hence, the set of admissible orderings will in fact be a set of linear orderings over the given operations. This suggests a canonical representation for plans that can be used to compare the expressive power of candidate representations. For purposes of comparison, define the <u>canonical representation of a plan</u> to be a set of operations plus a set of linear orderings, or sequences, over that set. This definition of plan applies only to finite serial processes but it encompasses the domain of simple tasks to be performed by individual robots. As a matter of notation, a sequence over a set of symbols {Si | i=1,n} is an ordered list [Sp,Sq, ... Sr] in which each symbol occurs once and

[65]Fox and Ho, A relational control mechanism for flexible assembly.

only once. When the given symbols denote operations to be performed the ordinal position of a symbol in the list determines the temporal ordering of the corresponding operation. The leftmost operation should occur first, the following operation should occur second, and so on.

By defining a plan as a set of sequences, all of the possible plans over a set of K steps can be organized into a Boolean lattice. This gives a unique perspective on the space of possible plans and on the relationships between plans. For example, consider the set of all possible plans over a set of three operations labeled A, B, and C. There are six possible sequences over those operations: [A,B,C], [A,C,B], [B,A,C], [B,C,A], [C,A,B], and [C,B,A]. This in turn implies that there are 2^6 distinct plans over the three operations, one for each possible subset of the six sequences. If each of the six sequences is given a numeric label, as shown in Figure 1, and if each of the sixty-four possible plans is represented as a set of those numeric labels, as shown in Figure 2, then the sixty-four possible plans can be organized into the Boolean lattice shown in Figure 3. The corresponding lattice of plans over a set of four operations would entail 2^{24} nodes! The bottom node in that figure denotes the nil plan with no feasible sequences, the top node denotes the universal plan which admits any of the six sequences, and the intermediate nodes denote subsets of the six sequences. Arcs which meet below a node denote set intersection, and arcs which join above a node denote

1. [A,B,C]
2. [A,C,B]
3. [B,A,C]
4. [B,C,A]
5. [C,A,B]
6. [C,B,A]

Figure 1.

Six possible sequences over {A,B,C}.

{ }	{1}	{1,2}	{2,3}	{3,4}	{4,5}	{5,6}
	{2}	{1,3}	{2,4}	{3,5}	{4,6}	
	{3}	{1,4}	{2,5}	{3,6}		
	{4}	{1,5}	{2,6}			
	{5}	{1,6}				
	{6}					

{1,2,3}	{2,3,4}	{3,4,5}	{4,5,6}
{1,2,4}	{2,3,5}	{3,4,6}	
{1,2,5}	{2,3,6}	{3,5,6}	
{1,2,6}	{2,4,5}		
{1,3,4}	{2,4,6}		
{1,3,5}	{2,5,6}		
{1,3,6}			
{1,4,5}			
{1,4,6}			
{1,5,6}			

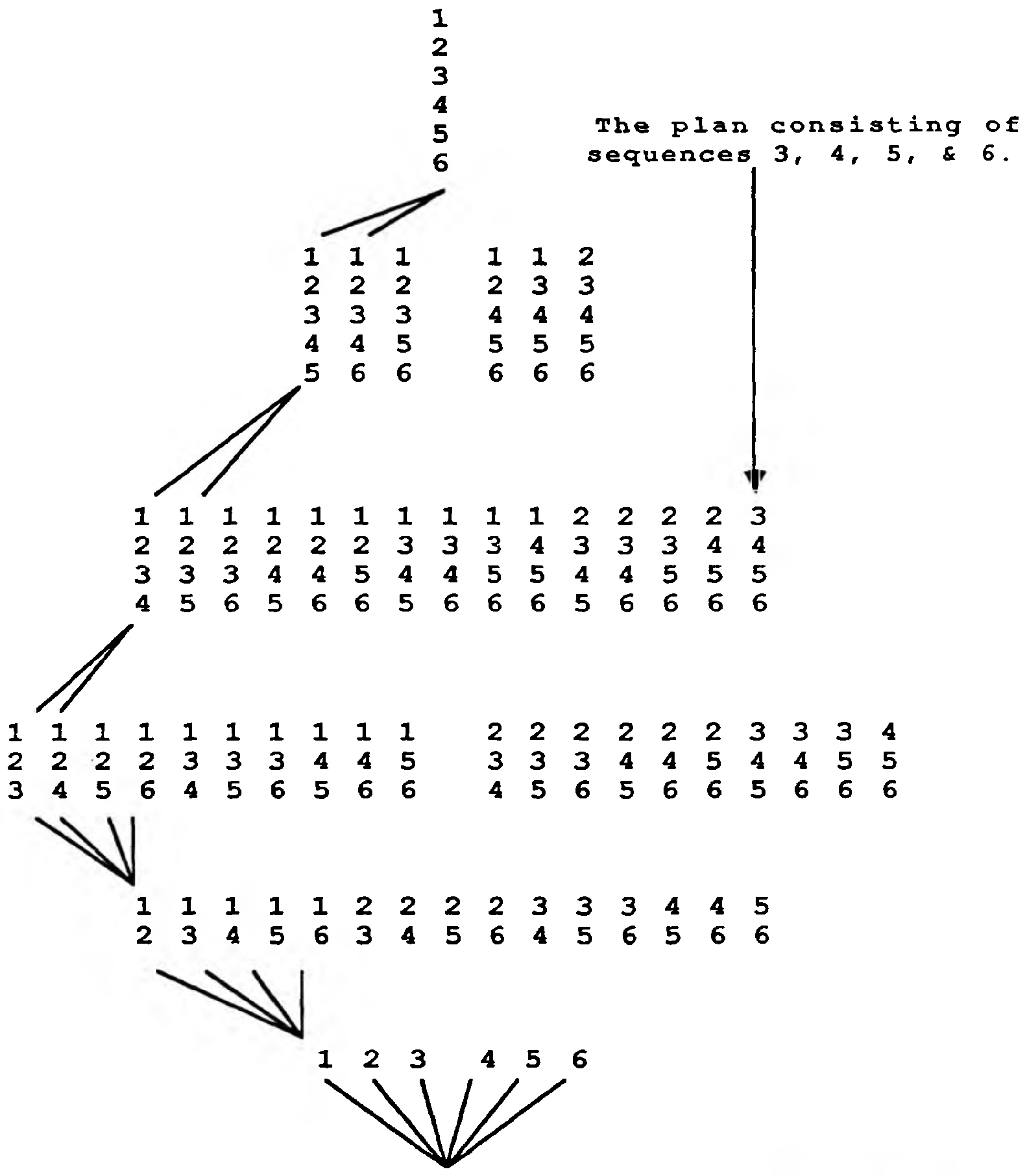
{1,2,3,4}	{2,3,4,5}	{3,4,5,6}
{1,2,3,4}	{2,3,4,6}	
{1,2,3,6}	{2,3,5,6}	
{1,2,4,5}	{2,4,5,6}	
{1,2,4,6}		
{1,2,5,6}		
{1,3,4,5}		
{1,3,4,6}		
{1,3,5,6}		
{1,4,5,6}		

{1,2,3,4,5}	{2,3,4,5,6}
{1,2,3,4,6}	

{1,2,3,4,5,6}

Figure 2.

Sixty-four possible plans over {A,B,C}.



(Sets and their elements are listed vertically.)

Figure 3.

The Boolean lattice of plans over {A,B,C}.

set union, but for simplicity, only a few arcs are shown in the figure. From this perspective, it appears that complex plans can be viewed as the intersection and union of more basic plans.

B. FOUR CRITERIA FOR COMPARISON

This map of the space of possible plans provides the first criterion for comparing candidate representations. A representation can be considered complete if and only if every possible plan over a fixed set of operations can be represented within that framework. If a representation fails to be complete then it may impede the process of planning and it will most certainly artificially constrain the options of the planner.

This map of the space of possible plans also suggests a second criterion for comparing candidate representations. The obvious disadvantage of the canonical representation is that the size of each plan is directly proportional to the number of sequences admissible under that plan. Consider a plan of **K** steps. The simplest plan, which imposes no constraints on the order of the operations, requires **K!** sequences in its canonical form, and plans that impose the simplest of ordering constraints require **K!/2**. A goal which will be very difficult to achieve, is that a representation should be reasonably compact. The size of the plan under some representation should generally be proportional to the complexity of the plan not the number of sequences which it encodes. Although it would be impractical, if not impossible, to determine the size of every plan over K steps when encoded in some representation, it is useful to consider the two extremes: the largest plan which has no constraints and K! sequences and the smallest plan, which admits only one sequence.

The third criterion for the comparison of candidate representations is that the representation should be tractable for the planner to produce. This means that its content should be primarily analytic information rather than synthetic. For example, the production of the canonical representation requires the creation of the individual sequences which are admissible. Obviously these cannot be produced by some cursory inspection of the task to be performed. Instead these can only be synthesized from some more primitive information which has been gathered by an analysis of the task. The representation should be based upon the most primitive collection of information that defines the set of operations to be performed and their admissible orderings. This is emphasized for two reasons. First, it encourages the consideration of more basic representations and second, it discourages suggestions of more complex representations. Either the planner must be able to directly produce the candidate representation or some other structure must be identified that the planner can produce which can be used to derive the candidate representation. The goal is simply to agree upon the appropriate exchange point between the planner and sequencer.

The fourth criterion for the comparison of candidate representations is that it should be tractable for the sequencer to use. The representation of a plan can be used in a variety of ways. Ultimately it will be used to determine the order of execution, but it can also be analyzed prior to execution in order to compare alternative plans or to predict expected behavior. The problem of greatest interest is the problem of constructing the set of admissible next operations given some representation of the plan and the current state of the task. The first standard of comparison is the traditional delineation between polynomial-time and exponential-time algorithms. Three cases will arise in the following discussions. For a given representation and the problem of constructing the set of admissible next operations, it may be possible to demonstrate a polynomial-time algorithm, or it may be possible to demonstrate an exponential-time algorithm but the actual complexity of the problem has not been determined, or finally it may be possible to demonstrate an exponential-time algorithm and it can be demonstrated that the problem is NP-hard.

Usually the terms polynomial-time and exponential-time qualify the expected execution time with respect to the size of the input. This can be misleading because the size of a plan under some representations can be exponentially related to the number of the operations but the algorithm for determining the set of admissible next operations can be polynomially related to the size of the plan. For instance, given the canonical representation of a plan, the time required to determine the set of admissible next operations is a simple polynomial function of the number of sequences. This can be accomplished by inspecting each of the S sequences in the plan. If the initial steps of a sequence match the sequence of steps already completed, then the next step in that sequence can admissibly be executed next. If each sequence consists of K operations then at most S*K comparisons need to be made in order identify the admissible next operations. Of course the complication is, that for some simple plans, the number of sequences will be on the order of K!. Similar situations will be pointed out when they occur.

In the distant future it is imaginable that truly autonomous systems will be developed which can plan, sequence, and execute their activities independent of any human contribution or supervision. In the immediate future, however, humans will be the most active intelligent element in robotic systems. Hence, a representation for plans must also satisfy two additional requirements. A candidate representation must be <u>tractable for a human to produce</u> and must be <u>tractable for a human to understand and analyze</u>.

C. A SIMPLE ASSEMBLY PROBLEM

The comparison of the candidate representations will be facilitated by the introduction of a simplified version of an assembly problem defined in a recent paper[66]. The product to be assembled consists of four parts labeled **cap, stick, receptacle,** and **handle** as shown in Figure 4. The **stick** is a solid cylinder which fits inside of the **receptacle,** and the receptacle is a hollow cylinder which is closed on one end by the **cap** and on the other end by the **handle.** The assembly is to be performed by a single robot forming a single product without use of sub-assemblies (the original formulation required two robots and allowed the use of subassemblies). At one level of resolution the task consists of four operations: **install-cap, install-stick, install-receptacle, install-handle.** For brevity, the part names will be frequently reduced to simply their first letters **C, S, R,** and **H,** and the operations will be reduced to **i-C, i-S, i-R,** and **i-H.** In its canonical form the plan for this task consists of the four operations plus the twelve sequences shown in Figure 5.

D. THE CANDIDATE REPRESENTATIONS

The representations that have been proposed in the literature have focused on different elements of the tasks to be performed. The basic elements, from which tasks are defined, are the objects to be manipulated and the

[66]de Mello and Sanderson, And/Or graph representation of assembly plans.

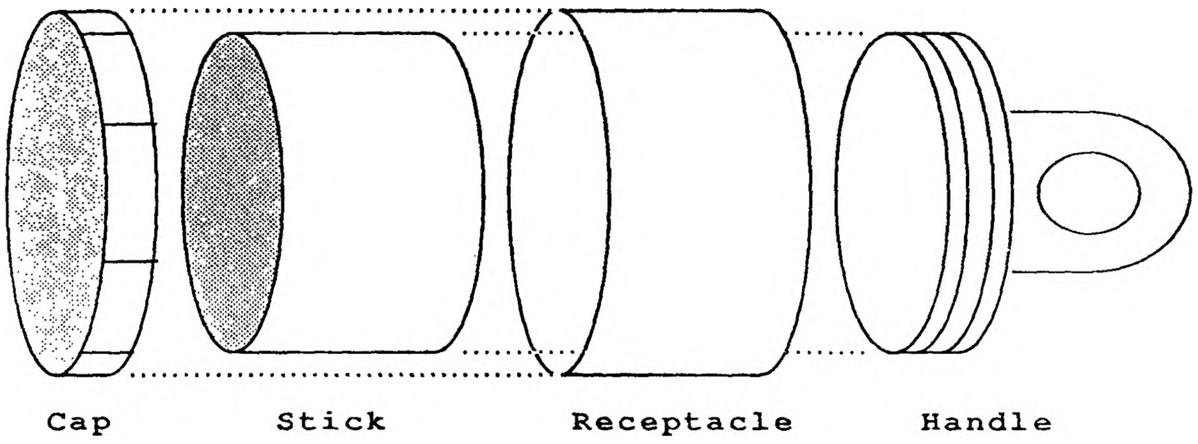


Figure 4.

A simple product of four parts.

1. [i-C,i-S,i-R,i-H]
2. [i-C,i-R,i-S,i-H]
3. [i-S,i-C,i-R,i-H]
4. [i-S,i-R,i-C,i-H]
5. [i-S,i-R,i-H,i-C]
6. [i-S,i-H,i-R,i-C]
7. [i-R,i-C,i-S,i-H]
8. [i-R,i-S,i-C,i-H]
9. [i-R,i-S,i-H,i-C]
10. [i-R,i-H,i-S,i-C]
11. [i-H,i-S,i-R,i-C]
12. [i-H,i-R,i-S,i-C]

Figure 5.

Twelve admissible sequences for the simple assembly problem.

operations to be applied to them. From one perspective, the state of all of the objects, when treated as an aggregate, defines the state of the task. In that context, the admissible orderings of the operations is implicitly defined by the set of admissible states and state transitions. From another perspective, the operations can only be applied to specific operands. Some of the operands may be initially available, some may be the result of other operations. The dependency of operations on the availability of operands, which in turn may need to be produced by other operations, implicitly defines the necessary ordering of operations. Alternatively, knowledge of the operations, their prerequisites, and effects can be used to formulate explicit constraints on the order of their execution. Each perspective leads to a distinct method of representation. Several proposed representations will be considered including state space representations, problem decomposition representations, and several representations based upon explicit temporal constraints. These representations will be compared using the criteria defined above, and whenever possible, using the simple assembly problem for examples.

1. State Transition Networks. One of the earliest candidate representations for tasks that robots are to perform is state transition networks[67]. In a system proposed

[67]Chapter II, Section B1: State-space representation, in The Handbook of Artificial Intelligence, vol 1, A. Barr and E.A. Feigenbaum (eds), William Kaufmann, 1981.

by Whitney[68] the robot plus its working environment was considered to be a single system which could be described by a state vector of discrete parameters. Coupled with this was a vocabulary of discrete operators which caused transitions between states. From this perspective, planning the activities of a robot was simply a matter of search through the space of all states reachable from an initial state by the application of the available operators. Cost functions could be applied to the operators and metric functions could be applied to the states in order to control the search process and improve the quality of the solutions.

The plans produced in this system were purely linear but this same combination of states and operators can be used to represent minimally ordered plans. Based upon an analysis of the physical process, the simple assembly problem can be encoded in a state transition network as shown in Figure 6. In this case, each state is determined by the set of parts that have been installed. The initial state is the empty set and the final state is denoted by the set of all four parts. Due to the natural constraints on the assembly task not every configuration of parts is admissible and the state space for this task is a subset of the 2^4 possible sets over four parts. A directed arc between two nodes denotes a state transition effected by the installation of a single part. Each path from the initial

[68]Whitney, D.E., State space models of remote manipulation tasks, IEEE Transactions on Automatic Control, vol AC-14, #6, Dec. 1969.

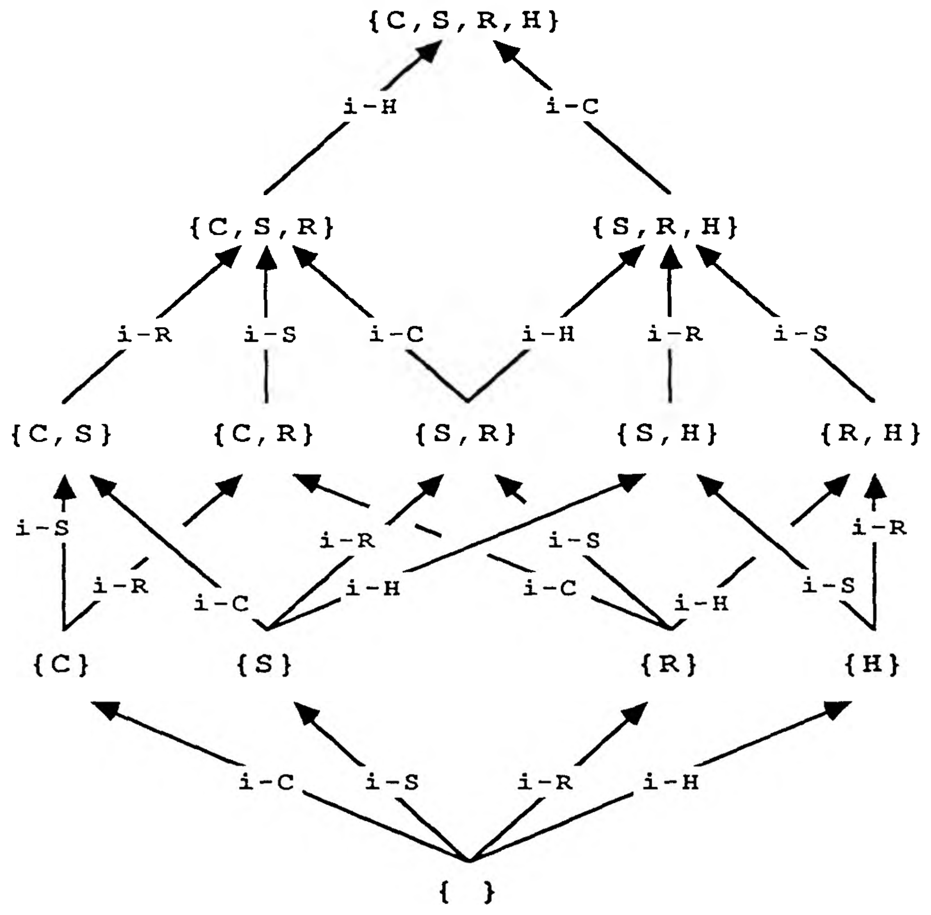


Figure 6.

State transition network for the simple assembly problem.

state to the final state requires the execution of all four operations, each exactly once. There are twelve paths through the network corresponding to the twelve sequences in the canonical representation.

When properly formed, the state transition network representing a plan contains no loops or cycles and only one terminal state. Given that arcs denote operations, the state transition network representing a plan is properly formed when every path from the initial state to the final state requires the execution of each operation exactly once and the set of all possible paths defines the set of admissible sequences.

More generally, it can be demonstrated that state transition networks provide a _complete_ representation for plans. The argument is based upon a fundamental theorem of formal language theory. Every finite language is accepted by some deterministic finite state automaton. If the set of sequences which define a plan are viewed as strings over the alphabet of operators, then the state transition network which represents that plan can be found by constructing a finite state machine which accepts that language.

The construction of a state transition network from the canonical representation must be done very carefully. In the case of the simple assembly problem, the state of the process is exactly determined by the set of parts that have been installed, and equivalently, by the set of operations that have been performed. However, this is an artificially simple case. Consider the plan over the four steps **A**, **B**, **C**, and **D**, which consists of only the two sequences **[A,B,C,D]** and **[B,A,D,C]**. The order of the last two steps, **C** and **D**, cannot simply be determined from the fact that **A** and **B** have been completed. The order of execution is significant. The execution of **A** then **B** leads to a different state than the execution of **B** then **A**.

The proper construction is based upon sets of sequences and subsequences. Each state of the network is distinguished by the set of sequences which lead from that state to the final state of the task. The initial state is identified by the set of all admissible sequences and the final state by the null set. The set of admissible operators in any given state is found by inspecting the leading operators of the sequences which define that state. Successor states are constructed in two steps. First, the sequences which define the current state are partitioned according to their first operator. Second, each partition is then used to construct a successor state which can be reached by the application of the leading operator. The sequences which define the successor state are constructed from the sequences in the partition simply by removing the leading operator. For instance, the sets of sequences construction of the state transition network for the simple assembly problem is shown in Figure 7. Although it will not always be true, in this case the network derived from an analysis of the sequences

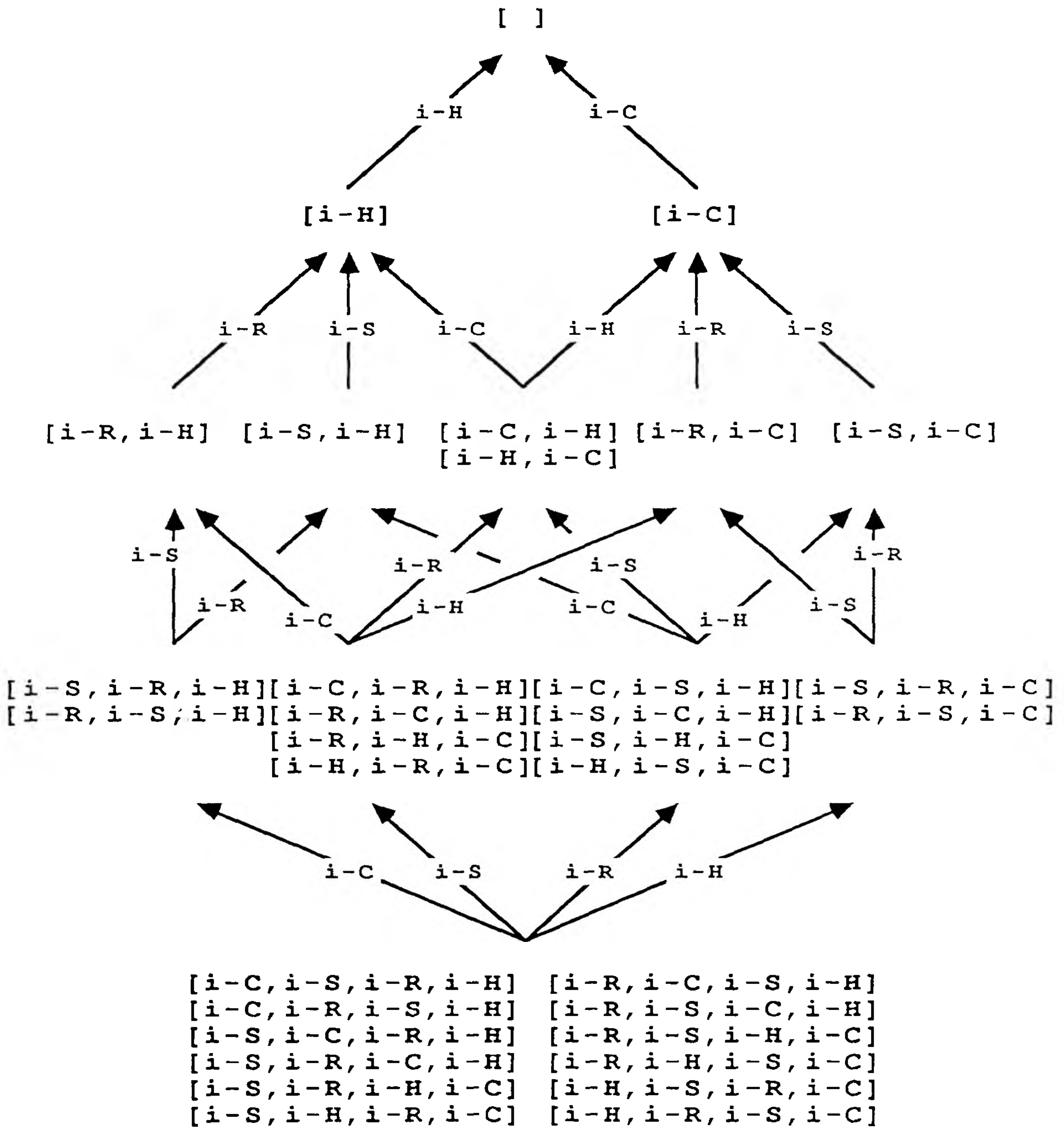


Figure 7.

Construction of the state transition network for the simple assembly problem.

is identical in structure to the network derived from an analysis of the physical process.

This construction is presented in order to emphasize that the states of a task may not be sufficiently distinguished by the set of operations that have been completed. Instead a more refined concept of state is needed for many practical applications. For instance, the state of a chemical process cannot be determined solely on the basis of the set of chemicals that have been added to a reactor. The order is significant. Some reactions produce heat, some require heat, some produce by-products that can contaminate other reactions, and some neutralize contaminants.

The compactness of the representation can be estimated by considering two cases: the smallest plan over K steps consisting of one sequence and the largest plan over K steps consisting of K! sequences. In the first case the state transition network consists of a linear chain of K+1 states with K transitions in between. In the second case the order of execution is unconstrained and the set of sequences for completing the task from some given state is independent of the sequence of operations which led to that state. Hence, the state transition network can safely be constructed from the 2^K possible subsets of the K operations. For an unconstrained task of twenty operations this implies a network of over a million states! While this may not be prohibitive in the context of modern memory prices this does raise serious questions about the methods for creating and analyzing plans using this representation.

From the perspective of the planner, state transition networks pose serious difficulties. Except for very small problems, the work required to build a state transition network exceeds human capabilities. Simple problems will have an enormous number of possible states, complex problems will require a tremendous analytic effort. In some problems the state is determined not only by the operations that have been completed but also by their ordering, which further compounds the analysis. It is arguable that a computer could be programmed to produce the state transition network, but this only differs the question of representation. The _input_ to such a program must be some representation of the task to be performed which accurately and completely delimits every possible sequence of operations. Suggestions that a computerized planning system could produce this representation are equally suspect. Planners, such as STRIPS, that produce linear plans by systematically searching the reachable states of a state transition network have proven to be slow and inefficient. The prospects of circumscribing the set of _every admissible sequence_ over a set of operations using such methods seem remote.

From the perspective of the sequencer, state transition networks are ideal. Every possible state of the task has been itemized, and for every state, every admissible operator and state transition has been itemized. The cost of determining the set of admissible next operations for a given state is negligible. Unfortunately, humans will find state transition networks as intractable to understand and analyze as they are to construct. While the elements of the representation are simple: states, operators, and successor states, the profusion of states, and state transitions can obscure even simple patterns in a plan or make it impossible to recognize common elements when comparing plans.

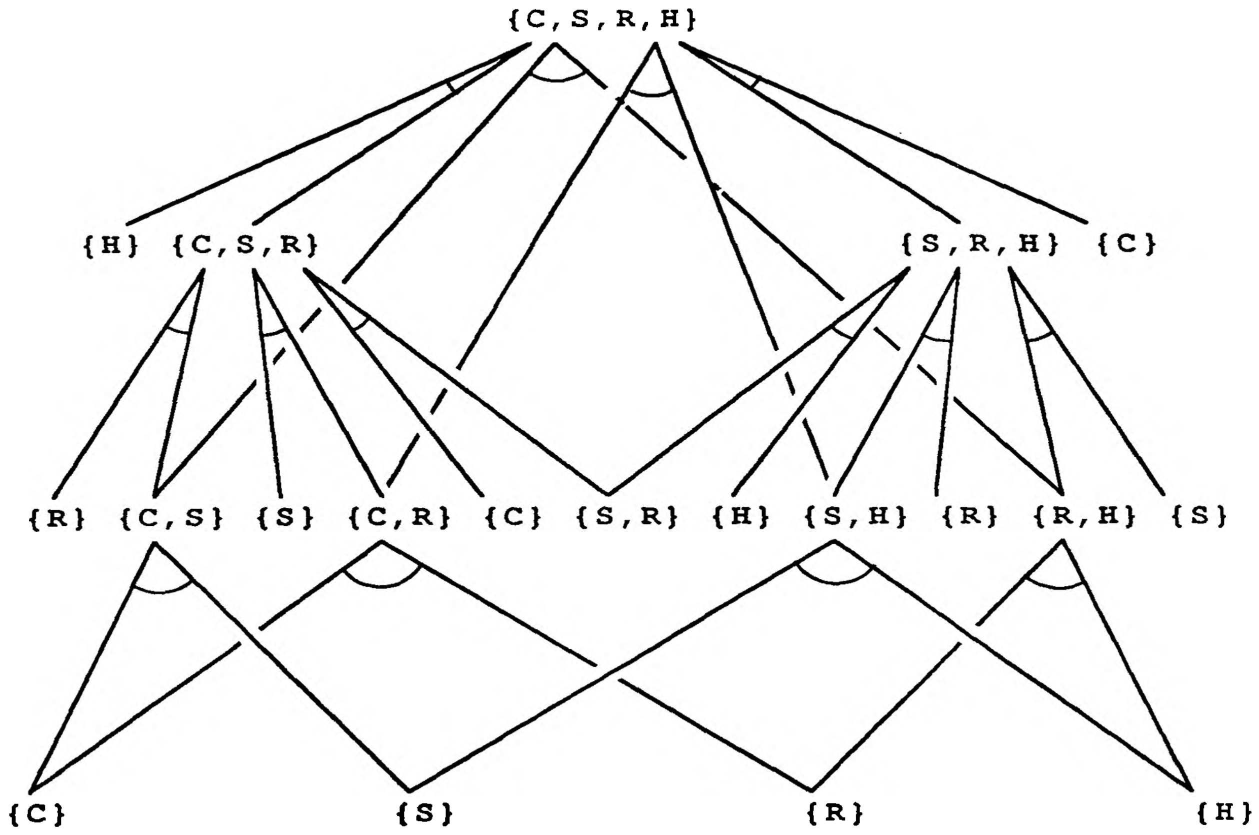
2. And/Or Graphs. A recently proposed representation for tasks that robots are to perform is based upon sub-assembly decomposition. In a system developed by deMello and Sanderson[69] an assembly problem was decomposed into a set of stable sub-assemblies and those in turn were decomposed until the product had been factored into its constituent parts. By exhaustively considering every stable decomposition of an assembly all of the feasible methods for producing it could be circumscribed.

The decomposition of assemblies into sub-assemblies can be very conveniently represented by And/Or graphs[70]. For instance, the simple assembly problem as originally posed by deMello and Sanderson is represented by the And/Or graph shown in Figure 8. The topmost node in that figure, labeled {C,S,R,H} denotes the completed assembly. The leaf nodes

---

[69]de Mello and Sanderson, AND/OR graph representation of assembly plans.

[70]Chapter II, Section B2: Problem Reduction representation, in The Handbook of Artificial Intelligence, vol 1, A. Barr and E.A. Feigenbaum (eds), William Kaufmann, 1981.



(for simplicity, nodes which represent individual parts have been freely replicated.)

Figure 8.

And/Or graph representation of the simple assembly problem.

denote the constituent parts. Nodes in between represent the possible stable sub-assemblies which can be used to form the product. Arcs descending from a node which are joined by a bar are referred to as hyperarcs and lead to a set of components that can be combined to form the sub-assembly in that node. As formulated by deMello and Sanderson, each hyperarc denotes an operation. The node from which it descends represents the result of the operation and the nodes to which it connects denote the operands. A singular arc can be used to represent operations with only one operand, such as inspecting or drilling. When more than one arc or hyperarc descends from a node, each denotes an alternative method for producing that sub-assembly. For instance, the topmost node has four descending hyperarcs which denote four alternative methods for constructing the completed product, the sub-assembly labeled **{C,S,R}**, in turn, can be produced from three different combinations of sub-assemblies but the leftmost node, labeled **{C,S}**, can only be produced by mating the two parts **{C}** and **{S}**.

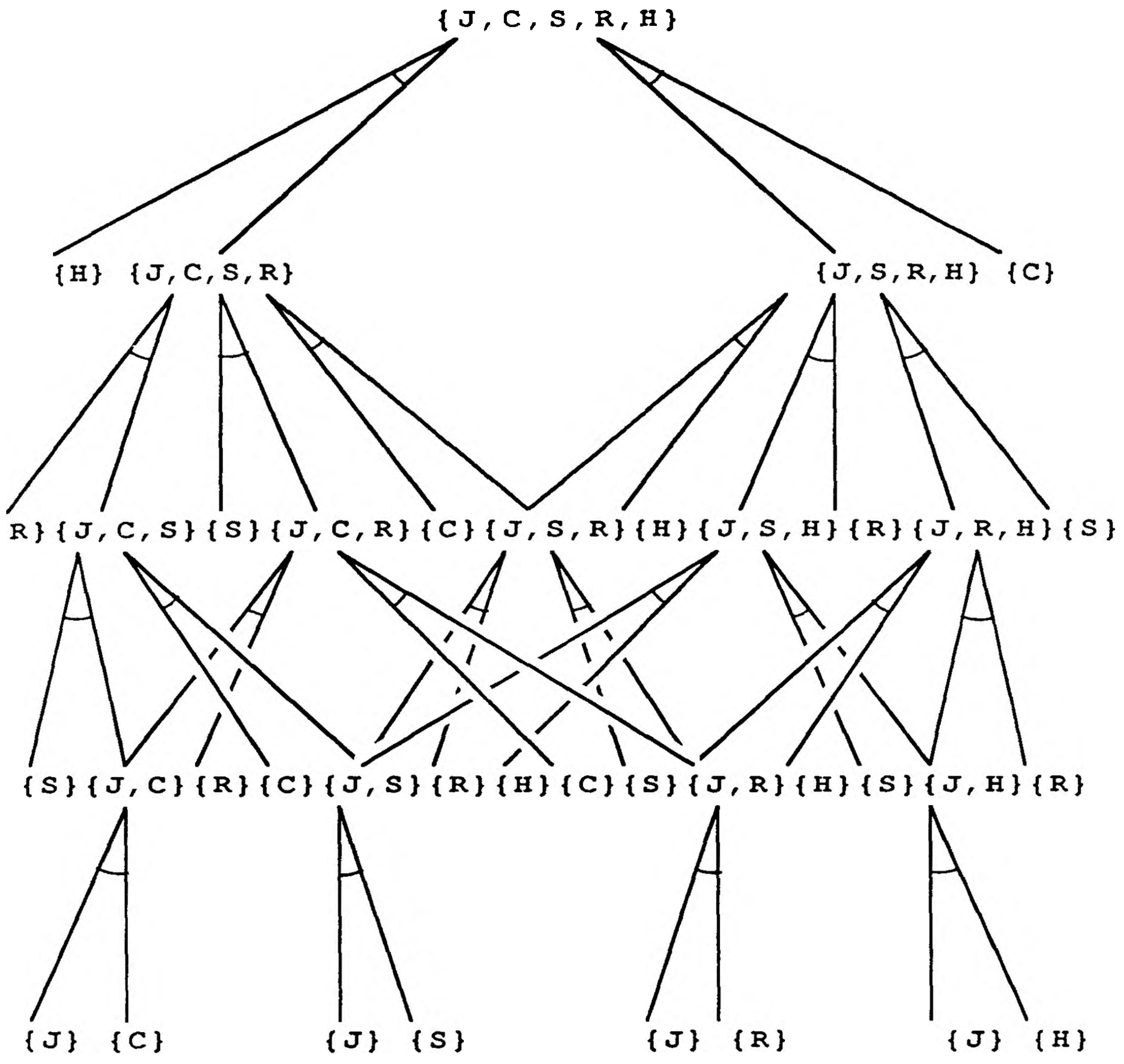
When properly formed, the And/Or graph representing a plan contains no loops or cycles and a single root node. A solution to an And/Or graph can be found by <u>marking</u> a sub-graph according to the following rules. First mark the root node. If some node has been marked, then select exactly one arc or hyperarc descending from that node and mark the descendant nodes. The process is complete when no additional nodes can be marked. Given that arcs and hyperarcs denote operations, the And/Or graph representing a plan is properly formed when every solution requires the execution of each operation exactly once and the set of all possible solutions defines the set of admissible sequences.

Figure 8 is an accurate representation of the simple assembly problem, as originally posed by deMello and Sanderson, but it contains elements not found in the problem defined in this paper. There are three important differences. Their version of the problem allowed the use of two robots, allowed the use of sub-assemblies, and each hyperarc in the And/Or graph represented the operation of mating two components using the two arms. If the problem is restricted to one robot, no sub-assemblies, and the vocabulary of four operations, **install-cap**, **install-stick**, **install-receptacle**, and **install-handle**, the And/Or graph contains some significant differences. Most important, with only one robot it becomes impossible to directly mate two parts. Instead one part must be first be mated with a jig or other stabilizing fixture in the workspace and then the second part can be mated with it.

With these restrictions the And/Or graph of Figure 8 can be revised to accurately represent the simple assembly problem. First, any hyperarcs which denote the combination of sub-assemblies must be removed. Second, a specific symbol, J, must be introduced to denote the jig, or fixture which stabilizes the assembly. Finally, any hyperarcs which denote the combination of two individual parts must be replaced with a sub-graph representing the process by which those parts can be mated using the jig. The result of these revisions is shown in Figure 9. As before, each node in the graph denotes a stable sub-assembly and hyperarcs which descend from a node lead to the components which can be combined to form that sub-assembly. Hyperarcs still denote the process of mating, but in this context, all matings must be between a single part and the partial product already completed.

Unfortunately, Figure 9 contains a profusion of arcs which can obscure the basic underlying structure of the assembly problem. A second set of revisions can simplify the graph and at the same time reveal its most important characteristics. Each hyperarc in the graph consists of two branches: one leading to an individual part and a second leading to a stable sub-assembly. Given this uniform pattern of usage, each hyperarc can be collapsed to a single arc labeled with the individual part and leading to the sub-assembly. The And/Or graph resulting from this revision is shown in Figure 10, and in this form, should be easily recognized. It is identical in structure to the state transition networks presented in the previous section.

This illustrates a very important point. Unless a problem can in fact be partitioned into independent sub-problems, an And/Or graph which represents that problem will be nothing more than a state transition network. By virtue of their ability to contain embedded state transition


{J,C,S,R,H}
{H} {J,C,S,R}
{J,S,R,H} {C}
R}{J,C,S}{S}{J,C,R}{C}{J,S,R}{H}{J,S,H}{R}{J,R,H}{S}
{S}{J,C}{R}{C}{J,S}{R}{H}{C}{S}{J,R}{H}{S}{J,H}{R}
{J} {C}
{J} {S}
{J} {R}
{J} {H}

(for simplicity, nodes which represent individual parts have been freely replicated.)

Figure 9.

Revised And/Or graph of the simple assembly problem.

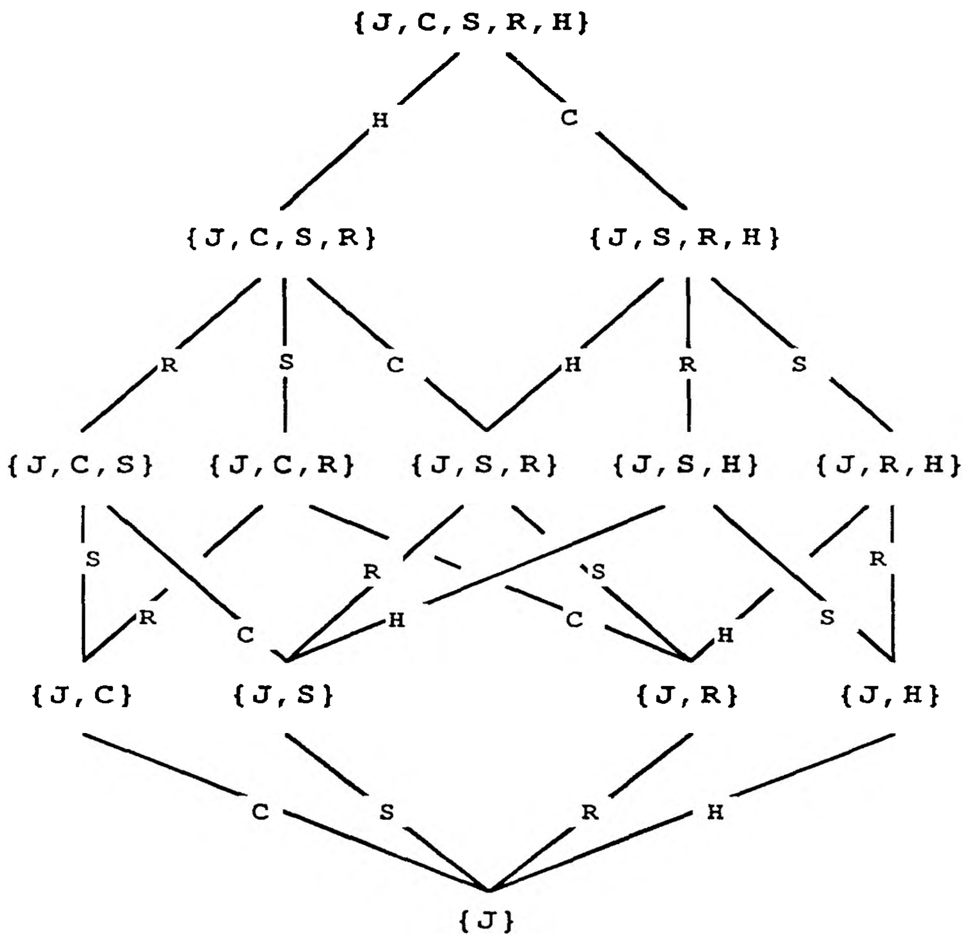


Figure 10.

Simplified And/Or graph representation of the simple assembly problem.

networks, And/Or graphs are complete, but they are conceptually and computationally useful only when a problem can be decomposed into two (or more) independent sub-tasks.

As formulated by deMello and Sanderson, nodes in the And/Or graph denote objects and the arcs and hyperarcs denote operations. Multiple arcs descending from a node denote the alternative methods for producing the object in that node. This is a very effective framework for representing the <u>alternative combinations</u> of sub-assemblies that can be used to produce a particular product. However, this usage of And/Or graphs does not facilitate the representation of <u>alternative orderings of operations</u>.

In order to illustrate this point, consider a set of K milling operations that are to be performed on a single piece of metal and which can be done in any order desired. The final product can be produced from any one of K alternative partial-products. It can be produced from a piece of metal which has gone through all but operation 1, or it can be produced from a piece of metal which has gone through all but operation 2, etc. In an And/Or graph the arcs descending from the root node can indicate the alternative objects that can be used to form the final product and can implicitly indicate those operations that can be performed last, but a complete state transition network of 2^K nodes must be constructed in order to indicate that K operations can be done in any order desired.

It is only when a problem can be decomposed into independent sub-assemblies that this representation becomes economical. Under a state transition representation, the number of states, and accordingly the number of nodes, in a problem would be equal to the <u>product</u> of the number of states in each independent sub-assembly. Under an And/Or graph representation, each independent sub-assembly can be represented by a separate sub-graph and the total number of nodes in the graph would be the <u>sum</u> of the number of nodes in each sub-graph. Hence, for problems which do not involve building independent sub-assemblies this representation is essentially identical to a state transition network but when sub-assemblies are used in the construction of the final product an And/Or graph representation is substantially more compact.

From the perspective of the planner, an And/Or graph representation of the decomposition of an assembly is subject to the same analysis. For problems which do not involve building independent sub-assemblies this representation is essentially identical to a state transition network and the same criticisms apply. When sub-assemblies are used in the construction of the final product an And/Or graph representation is both conceptually and computationally more tractable. If for no other reason, this is because the number of distinct states to be considered has been reduced from a product of states to a sum of states. More important, an And/Or graph accurately models one strategy that a planner might follow in planning. First decompose the problem into a set of independent sub-problems, and when that is accomplished recursively decompose each of those.

From the perspective of the sequencer And/Or graphs pose only modest complications. It is a trivial matter to determine the set of admissible next operations if the representation of a task is, in effect, a state transition network. All of the admissible states and state transitions have already been enumerated. For sake of efficiency, it may be desirable to collapse and label hyperarcs using the same methods demonstrated in Figure 10. If the task has been partitioned into sub-assemblies, then the sequencer must separately consider the state of each active sub-assembly and properly shift attention when sub-assemblies are mated. This does not appear to present any serious difficulties.

The usefulness of this representation is limited to the degree that a problem is decomposed into independent sub-problems. An And/Or graph representing any problem or sub-problem that cannot be decomposed is simply a state transition network. Hence to a large degree, this representation has the same merits and demerits as state transition networks, but for some specific problems, it can be quite economical.

3. Ordered Sets. Given that a plan consists of a set of operations plus a set of admissible sequences over those operations it is quite natural to consider methods for

defining the admissible sequences by some formula or prescription.

a. <u>Linearly Ordered Sets</u>. The simplest method for prescribing the ordering of a set of operations is to explicitly state a feasible execution sequence. This is the weakest representation possible. Of sixty-three feasible plans in Figure 11, only the six marked with asterisks, can be represented within this framework. The other fifty-seven can only be approximated by selecting one of their constituent linear sequences. Of course, this will eliminate any sequencing options that would otherwise be available to the sequencer. Although far from complete, it can be argued that this representation is compact. The size of a plan is proportional only to the number of operations. The price of this compactness, though, is a loss of information. It has already been argued in the discussion of planning and in the general discussion of representation, that this form of representation interferes with the process of planning and artificially constrains the process of sequencing.

For purposes of comparison, a linear plan for the simple assembly problem can be any one (but only one) of the sequences shown in Figure 5.

b. <u>Partially Ordered Sets</u>. The representation most frequently used to state the ordering constraints over a set of operations is the precedence diagram[71]. As used in the

[71]Prenting, T.O. and Battaglin, R.M., The precedence diagram: a tool for analysis in assembly line balancing.

Production Engineering and Operations Research literature, a precedence diagram consists of a set of nodes and a set of directed arcs. Each node represents an operation to be performed and each arc represents a constraint on the order of two operations. By convention, arcs are drawn as arrows from predecessor to successor operations. Operations that have no predecessors can be performed at any time, and those with predecessors can only be done after all of the predecessor operations have been completed. When properly formed, a precedence diagram contains one node for each operation to be performed, one arc for each necessary ordering constraint, and no loops or cycles.

Clearly, a precedence diagram is nothing more than a directed acyclic graph with a specific interpretation applied to the nodes and arcs. An equivalent mathematical structure is the strict partial order, consisting of a set of objects and a set of ordered pairs which define a transitive asymmetric relation over those objects. Of course the objects in question are the operations and the relationship of interest is precedence.

For example, a precedence diagram which represents a restricted form of the plan for the simple assembly problem is shown in Figure 12. The operations **i-C** and **i-R** can be performed at any time, the operation **i-S** can be done only after the operation **i-C**, and the operation **i-H** can be done only after **i-S** and **i-R**.

Journal of Industrial Engineering, vol 15, #4, pp. 208-211, July-Aug 1964.

```
                              1
                              2
                              3
                              4
                              5
                              6


                        1 1 1   1 1 2
                        2 2 2   2 3 3
                        3 3 3   4 4 4
                        4 4 5   5 5 5
                        5 6 6   6 6 6


                1 1 1 1 1 1 1 1 1 1 2 2 2 2 3
                2 2 2 2 2 2 3 3 3 4 3 3 3 4 4
                3 3 3 4 4 5 4 4 5 5 4 4 5 5 5
                4 5 6 5 6 6 5 6 6 6 5 6 6 6 6


          1 1 1 1 1 1 1 1 1 1   2 2 2 2 2 2 3 3 3 4
          2 2 2 2 3 3 3 4 4 5   3 3 3 4 4 5 4 4 5 5
          3 4 5 6 4 5 6 5 6 6   4 5 6 5 6 6 5 6 6 6


                1 1 1 1 1 2 2 2 2 3 3 3 4 4 5
                2 3 4 5 6 3 4 5 6 4 5 6 5 6 6


                         1 2 3   4 5 6
                         * * *   * * *
```

(Sets and their elements are listed vertically.)

Figure 11.

Six linear plans over {A,B,C}.

$$\begin{tikzcd}
 & \text{i-H} & \\
\text{i-S} \arrow[ur] & & \text{i-R} \arrow[ul] \\
\text{i-C} \arrow[u] & &
\end{tikzcd}$$

Figure 12.

A restricted plan for the simple assembly problem.

The execution of a task according to the constraints embodied in a precedence diagram is analogous to the process of _pebbling_ the nodes of that graph. That process is governed by one basic rule. A node can be pebbled only if all of its predecessor nodes have been previously pebbled. By default, an initial node, with no predecessors, can be pebbled at any time. The process is complete when all of the nodes are covered. Given a properly formed precedence diagram, the set of all possible pebbling sequences is identical to the set of all possible execution sequences.

Precedence diagrams are clearly compact. The representation of a plan over K operations will always contain K nodes and never more than K(K-1)/2 arcs. The smallest plan, a linear sequence over K steps, requires only (K-1) arcs, and the largest plan, which admits K! sequences, requires 0 arcs. Hence, it appears that this representation best approximates a previously stated ideal. The size of a plan under a given representation should be proportional to its complexity, not simply the number of sequences which it encodes.

Precedence diagrams are conceptually and computationally useful to the planner. During the course of planning, operations are added to the plan in order to produce specific desired effects. In order to guarantee those effects, an operation must necessarily follow the operations that enable its execution, it must necessarily precede the operations that require its effect, and the operations that require its effect must precede any operations that negate its effect. Hence, each time that an operation is added to the plan a number of ordering constraints must be added to the plan as well. It is not necessary to determine a linear ordering of the operations, it is only necessary to tabulate the constraints and to check for any inconsistencies among them. However, a collection of ordering constraints over a set of operations is nothing more than a partially ordered set or precedence diagram. Moreover, the satisfiability of a conjunction of ordering constraints over K steps can be determined by a simple algorithm whose execution time is on the order of K^3.

Precedence diagrams are conceptually and computationally useful to the sequencer. They contain only the most essential information, the operations and the necessary ordering constraints, in a form that can be easily understood by humans. In contrast to state transition networks and And/Or graphs, they are guaranteed to be tractably small in both the number of nodes and the number of arcs. Unlike state transition networks, the admissible states and state transitions have not been explicitly enumerated but it is a simple matter to determine the set of admissible next operations using the previously stated pebbling rule. A number of other simple algorithms will be presented in the next section of the paper which make it possible to compare and analyze plans and strategies that have been represented using this formalism.

Unfortunately, precedence diagrams are not complete. This can be easily demonstrated by considering a simple task over three steps. In its most abstract form, the problem is to perform three operations, A, B, and C, but step B cannot be done last. Of six possible sequences over three steps, exactly four satisfy this constraint as shown in Figure 13.

The ordering constraints that must necessarily be imposed on the three steps can be determined by a simple analysis of three cases. The ordering of operations A and B cannot be constrained because they occur in one order in sequence 1 and in the opposite order in sequence 2; the ordering of operations A and C cannot be constrained because they occur in one order in sequence 1 and in the opposite order in sequence 3; and finally, the ordering of operations B and C cannot be constrained because they occur in one order in sequence 1 and in the opposite order in sequence 4. Unfortunately, this eliminates every possible constraint. Hence, the only possible precedence diagram which is consistent with these four sequences is the one which represents six sequences. This is clearly unsatisfactory. Only four sequences are admissible under this plan. The other alternative, is to approximate this plan by one of the precedence diagrams shown in Figure 14, either one of which represents three of the four admissible sequences.

For a more global perspective on the matter, the set of all plans over three steps that can be represented by precedence diagrams are marked with asterisks in Figure 15.

1. [A,B,C]
2. [B,A,C]
3. [B,C,A]
4. [C,B,A]

Figure 13.

Canonical representation of the plan: "B Not Last."

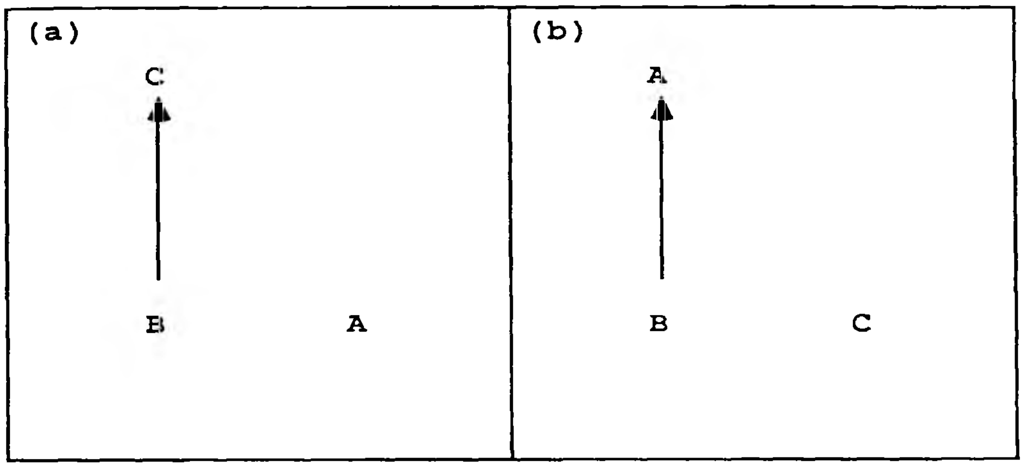


Figure 14.

Two precedence diagrams which approximate the plan: "B Not Last."

```
                    1
                    2
                    3
                    4
                    5
                    6
                    *

              1 1 1   1 1 2
              2 2 2   2 3 3
              3 3 3   4 4 4
              4 4 5   5 5 5
              5 6 6   6 6 6


      1 1 1 1 1 1 1 1 1 1 2 2 2 2 3
      2 2 2 2 2 2 3 3 3 4 3 3 3 4 4
      3 3 3 4 4 5 4 4 5 5 4 4 5 5 5
      4 5 6 5 6 6 5 6 6 6 5 6 6 6 6


1 1 1 1 1 1 1 1 1 1   2 2 2 2 2 2 3 3 3 4
2 2 2 2 3 3 3 4 4 5   3 3 3 4 4 5 4 4 5 5
3 4 5 6 4 5 6 5 6 6   4 5 6 5 6 6 5 6 6 6
* * *                                 * * *

      1 1 1 1 1 2 2 2 2 3 3 3 4 4 5
      2 3 4 5 6 3 4 5 6 4 5 6 5 6 6
      * *           *   *       * *

              1 2 3   4 5 6
              * * *   * * *
```

(Sets and their elements are listed vertically.)

Figure 15.

Nineteen partially ordered plans over {A,B,C}.

clearly, this representation only sparsely covers the space of possible plans. Notably, no plan over **K** steps that contains more than **K!/2** sequences but less than **K!** can be represented by a precedence diagram!

Instances of this kind of problem are common. The simple assembly problem is a good example. The **stick** is contained within the space formed by the other three parts, **cap**, **receptacle**, and **handle**, and obviously cannot be installed last. Likewise, if the **cap**, **stick**, and **handle** have been mated it is impossible to then install the **receptacle**. In fact, the set of admissible assembly sequences for this product is exactly circumscribed by those two constraints. As in the previous problem, there are several precedence diagrams, as shown in Figure 16, that can represent a subset of the admissible sequences, but there does not exist a single precedence diagram that encompasses all 12.

The weakness of precedence diagrams is very simple to identify. A precedence diagram is a <u>conjunction</u> of ordering constraints. There is no notational convention for including <u>disjunctive</u> constraints within that representation. Any problems which are subject to a disjunction of ordering constraints cannot be represented within this framework.

Disjunctive constraints occur naturally in a number of circumstances. A common situation is that a step is enabled by any one of several predecessors. For example, if step **Z** is enabled by either of the steps **X** or **Y**, then unless otherwise limited, **X** must precede **Z** <u>or</u> **Y** must precede **Z**.

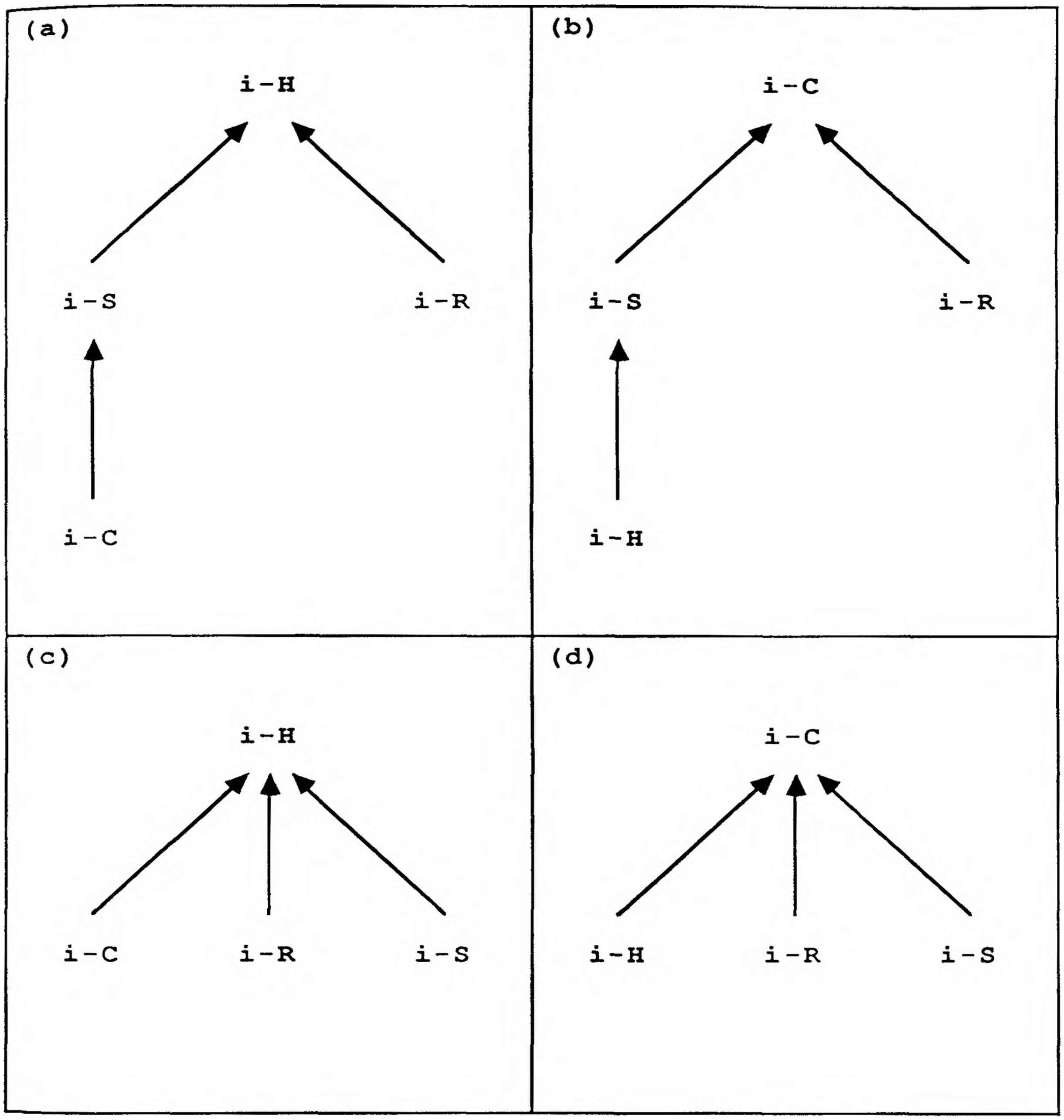


Figure 16.

Four precedence diagrams which approximate the plan for the simple assembly problem.

Another common situation is when a step is disabled upon the completion of every one of several steps. As in the simple assembly problem, if step **X** is disabled when both **Y** and **Z** are both completed, then **X** must precede **Y** <u>or</u> **X** must precede **Z**. A case which leads to a more complicated combination of constraints is when two sets of operations sufficiently interfere so that their execution cannot be interleaved. For example, if steps **X** and **Y** interfere with steps **Z** and **W** then **X** and **Y** must be performed before **Z** and **W** <u>or</u> **Z** and **W** must be performed before **X** and **Y**. The situations are common and the possibilities are endless.

c. <u>And/OR Graphs Reconsidered</u>. Is it possible that under a suitable encoding, And/Or graphs can be used as a general representation for the constraints over the steps of a plan? The most natural encoding would be to interpret nodes of the graph as steps of the plan, arcs of the graph as ordering constraints drawn as arrows from predecessor to successor steps, hyperarcs which descend from a node as conjunctions of ordering constraints, and unconnected arcs and hyperarcs which descend from a node as a disjunction of the constraints which they denote. Clearly, every precedence diagram can be represented in this fashion simply by joining the arcs which meet beneath a node with a bar to denote a conjunction of the predecessors. It is equally possible to represent constraints of the form **X** must precede **Z** <u>or</u> **Y** must precede **Z**, as a pair of independent arcs drawn from nodes **X** and **Y** to **Z**.

This usage of And/Or graphs poses two difficulties. First, under this interpretation of nodes and arcs it is not possible to represent state transition networks and problem decompositions as discussed earlier. Second, this alternative usage still does not cover the variety of constraints that may be presented by real problems. For instance, without some extended notation, it is impossible to represent the disjunction of constraints, X must precede Y _or_ X must precede Z, or the disjunction of constraints, X and Y must precede Z and W _or_ Z and W must precede X and Y. Attempts to extend And/Or graphs with sufficient notational conventions to encompass these and other more complicated problems will result in a system of nodes, arcs, and hyperarcs not suitable for graphical presentation.

d. _Interval Algebra_. Since the problem of sequencing the activities of a robot is in fact a problem of reasoning about time, it is natural to consider the languages that have been proposed for systems of temporal reasoning. The foremost among these is the language proposed by Allen[72]. The basic elements of this language are intervals of time and 13 primitive relationships that can hold between two intervals as shown in Figure 17. Each phrase in this language defines the relationship between two intervals and is formed as a disjunction of any of the thirteen primitive

[72]Allen, J., Maintaining knowledge about temporal intervals, _Communications of the ACM_ 26 (1983), 832-843.

relationships. The constraints on the order of a set of operations is formulated as a conjunction of such phrases.

This extensive vocabulary of interval relationships makes this language particularly useful when specifying the constraints over a set of concurrent activities. However, most of these relationships are superfluous in the context of a single robot performing a set of atomic operations.

More interesting, this language provides a limited facility for stating disjunctions of constraints. This facility produces an ironic effect. Although it is impossible to directly pose the constraint, X must precede Y _or_ Z must precede W[73], it has been shown that, under a clever encoding, this language can be used to represent any NP complete problem[74]. Because of this inherent complexity the problem of determining the satisfiability of a given set of interval constraints is NP-Complete. The constraint propagation algorithm that Allen proposes runs in polynomial time, but it is not guaranteed to derive all of the necessary relationships implied by the constraints, nor is it guaranteed to detect every possible inconsistency. Other complete constraint propagation algorithms have been considered but the expected execution time is prohibitive[75].

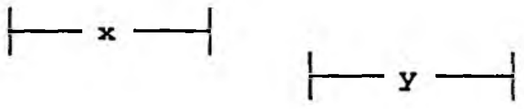
---

[73]Allen, J., Maintaining knowledge about temporal intervals.

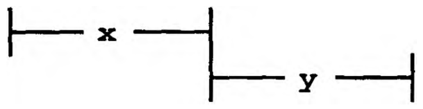
[74]Vilain, M. and Kautz, H., Constraint propagation algorithms for temporal reasoning, in _Proceedings Fifth National Conference on Artificial Intelligence_, Philadelphia, Penn., 1986, pp. 377-382.

[75]Allen, J. personal communication, February 1987..

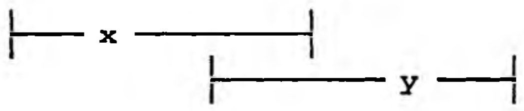
X before Y,
Y after X



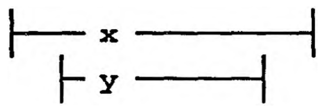
X meets Y,
Y met-by X



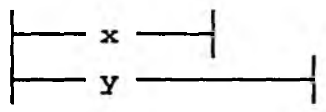
X overlaps Y,
Y overlapped-by X



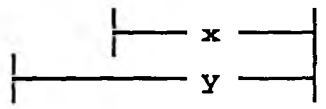
X contains Y,
Y contained-by X



X starts Y,
Y started-by X



X ends Y,
Y ended-by X



X equals Y

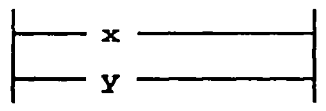


Figure 17.

Thirteen possible interval relationships.

4. Petri Nets. Other recent papers suggest that Petri nets[76] can be used to represent the tasks that robots are to perform. Not coincidentally two proposals have originated from Edinburgh. One is offered from the perspective of the planner[77,78,79] and the second from the perspective of the sequencer and executor[80,81].

Although quite a variety of Petri nets are found in the literature, the most common formulation consists of four elements: tokens, two kinds of nodes, and arcs which connect them. One kind of node, referred to as a place, is represented as a circle, the other kind, called a transition, is represented as a bar. All of the arcs are directed, drawn as arrows from places to transitions or from transitions to places but never directly from transition to transition or from place to place. Tokens reside in the circles which represent places and, according to some simple

---

[76]Petersen, J.L., Petri nets, Computing Surveys 9 (1977), 223-252.

[77]Drummond, M.E., Refining and extending the procedural net, in Proceedings Ninth International Joint Conference on Artificial Intelligence, Los Angeles, Calif, 1985, pp.1010-1012.

[78]Drummond, Plan Nets: a formal representation of action and belief for automatic planning systems.

[79]Drummond, M.E., Contingent plan structures for spacecraft, in Proceedings NASA/JPL Workshop on Space Telerobotics, Jet Propulsion Laboratory, Pasedena, Calif., 1987 (to appear).

[80]Malcolm, C., DAI Working Paper 187, Department of Artificial Intelligence, University of Edinburgh, 1986.

[81]Malcolm and Fothergill, Some architectural implications of the use of sensors.

rules and the structure of a net, are moved from place to place. The distribution of tokens in a Petri net determines its state and a set of future, reachable states.

The behavior of a Petri net is determined by a set of simple <u>firing</u> rules, not unlike the marking or pebbling rules associated with the other representations. A transition is enabled when all of its input places contain at least one token. At each step of execution, one of the enabled transitions is selected and fired. When fired, a transition removes one token from each of its input places and deposits one token in each of its output places. The number of input and output places need not be equal.

Using these elements it is a simple matter to construct a network which is identical to a state transition network in both structure and behavior. For example, the state transition network of Figure 6 can be translated into the Petri net shown in Figure 18. Each node of the state transition network is replaced by a place(circle) and each arc of the state transition network is replaced by an arc, transition(bar), arc combination. The labels attached to the nodes in the state transition network are attached to the corresponding places and the labels attached to the arcs of the state transition network are attached to the corresponding transitions. By virtue of this simple encoding of state transition networks, Petri nets must necessarily be a complete representation for plans. It is equally simple to encode And/Or graphs and precedence diagrams. For example,

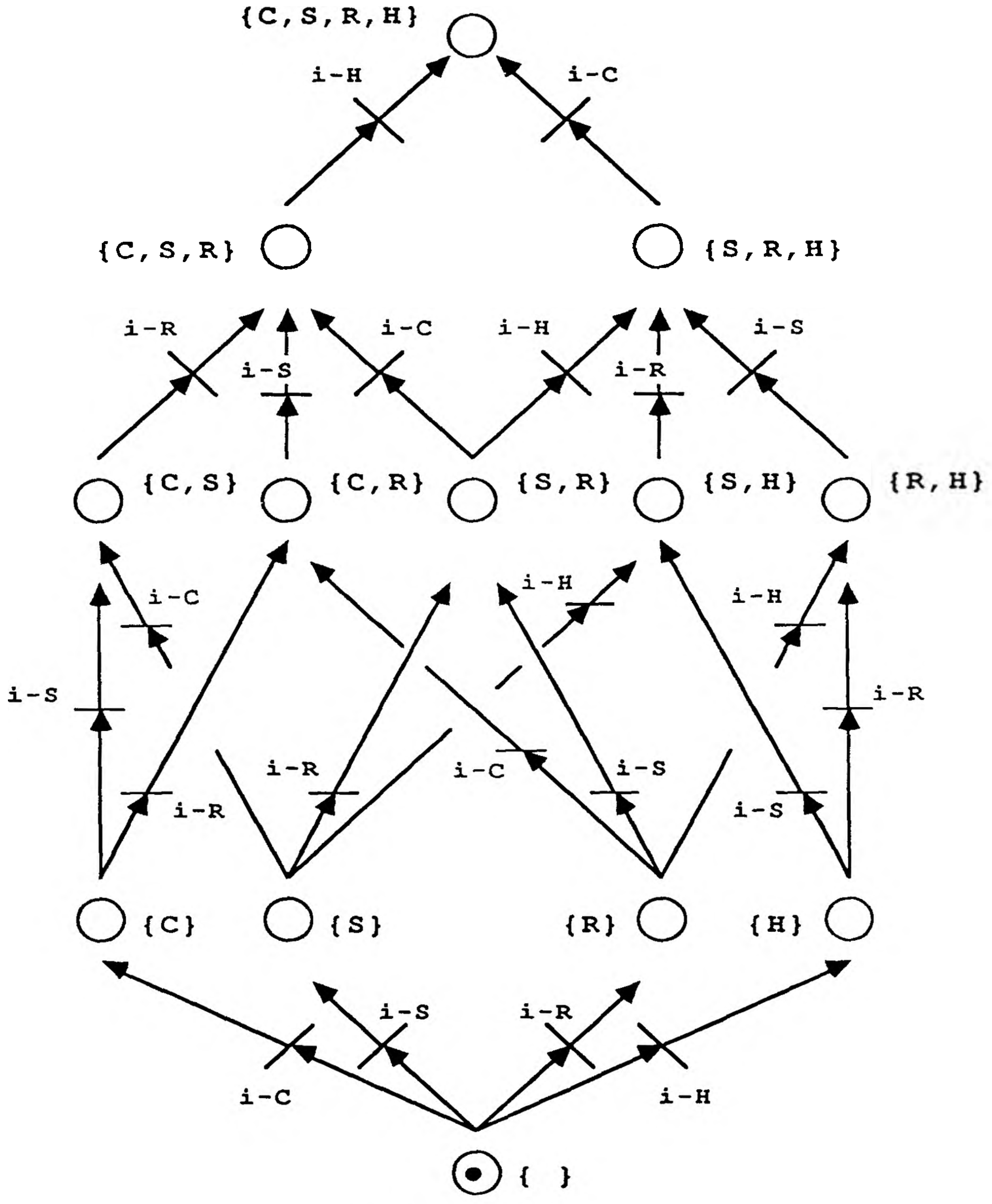


Figure 18.

Equivalent Petri net of the state transition network of Figure 6.

the precedence diagram of Figure 12 can be reformulated as the Petri net shown in Figure 19.

The unique aspect of Petri nets is found in a careful interpretation of the places and transitions. In a general sense, each place denotes a precondition for the transitions which follow. Each transition denotes an action or event and multiple input places form a conjunction of necessary enabling conditions. The places which follow a transition then denote the post-conditions or effects of the action. Given this interpretation, a Petri net for a task of **K** steps can be constructed from a set of **K** transitions which denote those steps, **K** places which denote the effects of those operations, **K** places which denote the necessary preconditions, and sufficient network circuitry to establish those preconditions. Petri nets constructed in this fashion are guaranteed to be compact. The size will be proportional to the number of operations in the task and the complexity of the enabling conditions. For example, the Petri net shown in Figure 20 represents the simple task discussed above, "B not last". The three topmost places denote the conditions that the actions have been completed, the transitions below those denote the respective operations, and the places below the transitions denote the necessary enabling conditions. The remaining network elements have been so constructed as to establish the enabling conditions. For example, the action **A** is enabled for execution if action **B** has been completed or if the action **enable-A** has been completed.

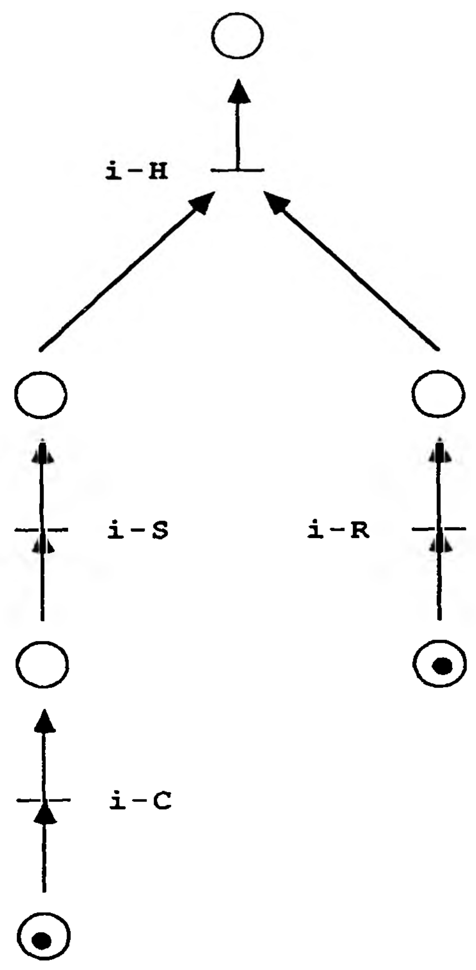


Figure 19.

Equivalent Petri net of the precedence diagram of Figure 12.

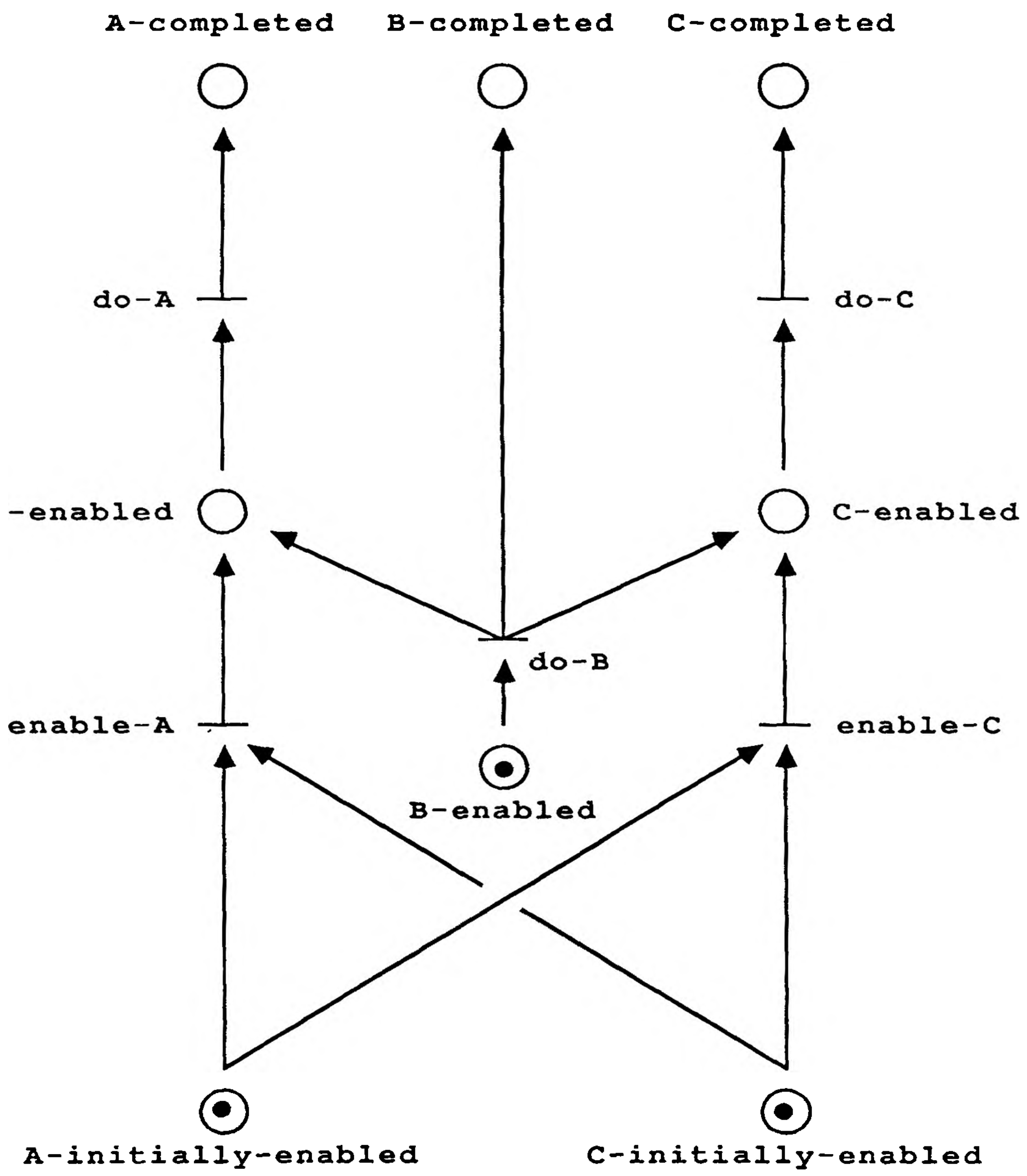


Figure 20.

Petri net representation of the plan defined in Figure 13.

However, the action **enable-A** can be executed only if both **A** and **C** are initially enabled. The Petri net for the simple assembly problem, which has similar enabling conditions, is shown in Figure 21.

The construction of a Petri net presents two difficulties. First, the enabling conditions for each action must be precisely circumscribed, and second, network structures must be fabricated which establish exactly those conditions. The former problem is the most critical. Although this may not involve the laborious process involved in building a state transition network, it does involve the enumeration and resolution of all of the factors that constrain a given operation. As in previously discussed representations, the analysis and resolution of the relevant constraints may exceed human capabilities and the implementation of computerized systems which can perform this same analysis requires some representation for the limiting constraints and some methods for analyzing them. Again, this only displaces the question of representation. What are the underlying constraints, how can they be represented, and what methods can be used to resolve them? Drummond suggests, that instead of completely resolving the necessary preconditions and encoding them in the Petri net, that a companion data structure be constructed which circumscribes the admissible orderings of operations. He further proposes that this information be derived from a state transition network constructed by simulating the

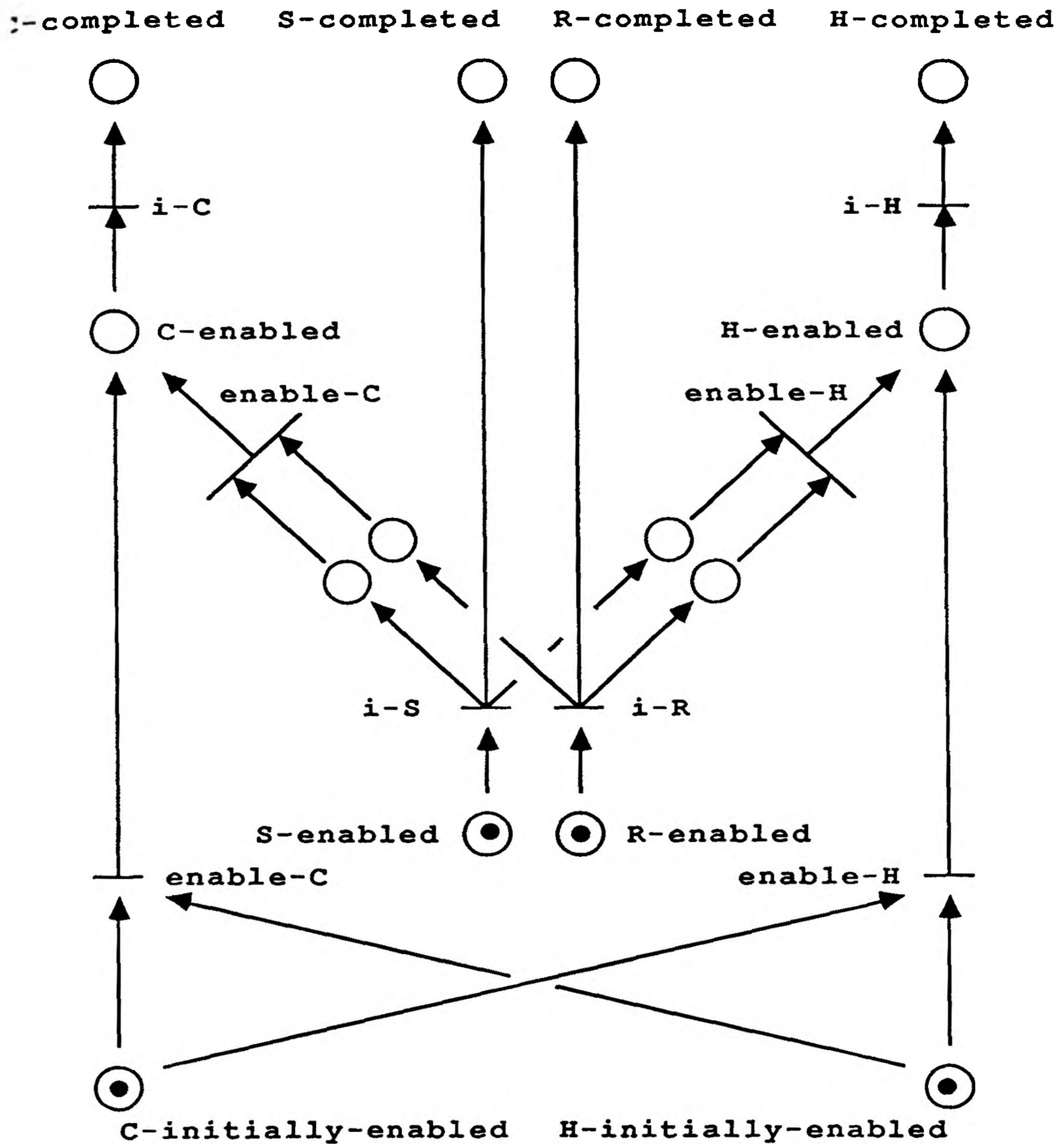


Figure 21.

Petri net representation of the plan defined in Figure 5.

execution of the task in question[82]. At present, Malcolm is currently more concerned with exploring the potential applications of Petri nets and will later consider methods for their production[83].

The analysis of a Petri net poses another difficulty. The predominant tool for the analysis of the behavior of a Petri net is the reachability graph[84]. Essentially, this is a state transition network constructed by simulating the execution of the Petri net. As in other representations, the least constrained problems will be the most difficult to analyze because of the exponential explosion in the space of reachable states. The most discouraging aspect of Petri nets is that without specific constraints on their structure, they are equivalent in computational power to Turing Machines. Hence, basic questions about behavior and termination are undecidable.

E. DISCUSSION

Three basic categories of representation have been proposed in the literature. The first category includes those that are based primarily upon the admissible states of the task. Of course, the primary representative of this category is the state transition network, but the problem decomposition representations rely upon this mode of

---

[82]Drummond, M.E., personal communication, January 1987.

[83]Malcolm, C. personal communication, December 1986.

[84]Petersen, Petri nets.

representation for those parts of problem which cannot be decomposed.

The second category includes those that are based primarily upon some scheme of precedence among the operations to be performed. The strictest representation in this category is the linear order. Precedence diagrams and partial orders impose the least constraint within the limits of purely conjunctive languages. The language of temporal intervals includes a restricted form of disjunction but the expressions which can be formed with this facility are superfluous in the context of single robots performing sets of atomic operations. Under an appropriate interpretation, And/Or graphs encompass a larger domain of problems but still fail to cover every possible plan.

The third category includes those that establish precisely the enabling conditions for each of the operations to be performed. The singular representative of this category is the Petri net. Since all of these representations are concerned with a common domain and since, to a large extent, they can be translated, one to another, it should not be surprising that each representation contains some elements of state, precedence, and precondition. The delineation of these categories is only intended to identify the most prominent distinguishing characteristics of the candidate representations.

With the exception of the interval algebra, all of these representations have one characteristic in common.

They are primarily graphical languages intended for two dimensional presentation. They are most effective for small or simple problems. When applied to larger or more complex problems these representations lose their elegance. Some forms of representation, like the precedence diagram, are structurally very simple and can only be used to encode simple problems. Some forms of representation, like the state transition network, are structurally very simple but the encoding of simple problems results in an excessively large structure. Other forms of representation may be both complete and compact, such as the Petri net, but may be prohibitively complex to produce and intractable to analyze.

A fundamental problem with graphical languages is that some of the relationships between the objects in question cannot be reduced to simple adjacency and connectivity. In some circumstances the relationships themselves are constrained or related. Consider, for example, the problem of extending the language of precedence diagrams to encompass general disjunction. Whatever graphical element is introduced to denote disjunction must apply to any imaginable constraints including the five cases shown in Figure 22: (a) X must precede Z <u>or</u> Y must precede Z; (b) X must precede Y <u>or</u> X must precede Z; (c) X must precede Y <u>or</u> Y must precede Z; (d) X must precede Y <u>or</u> Z must precede W; (e) (X must precede Y <u>and</u> Z must precede W) <u>or</u> (Y must precede X <u>and</u> W must precede Z). Clearly meta-arcs can be introduced to disjunctively connect arcs which denote the

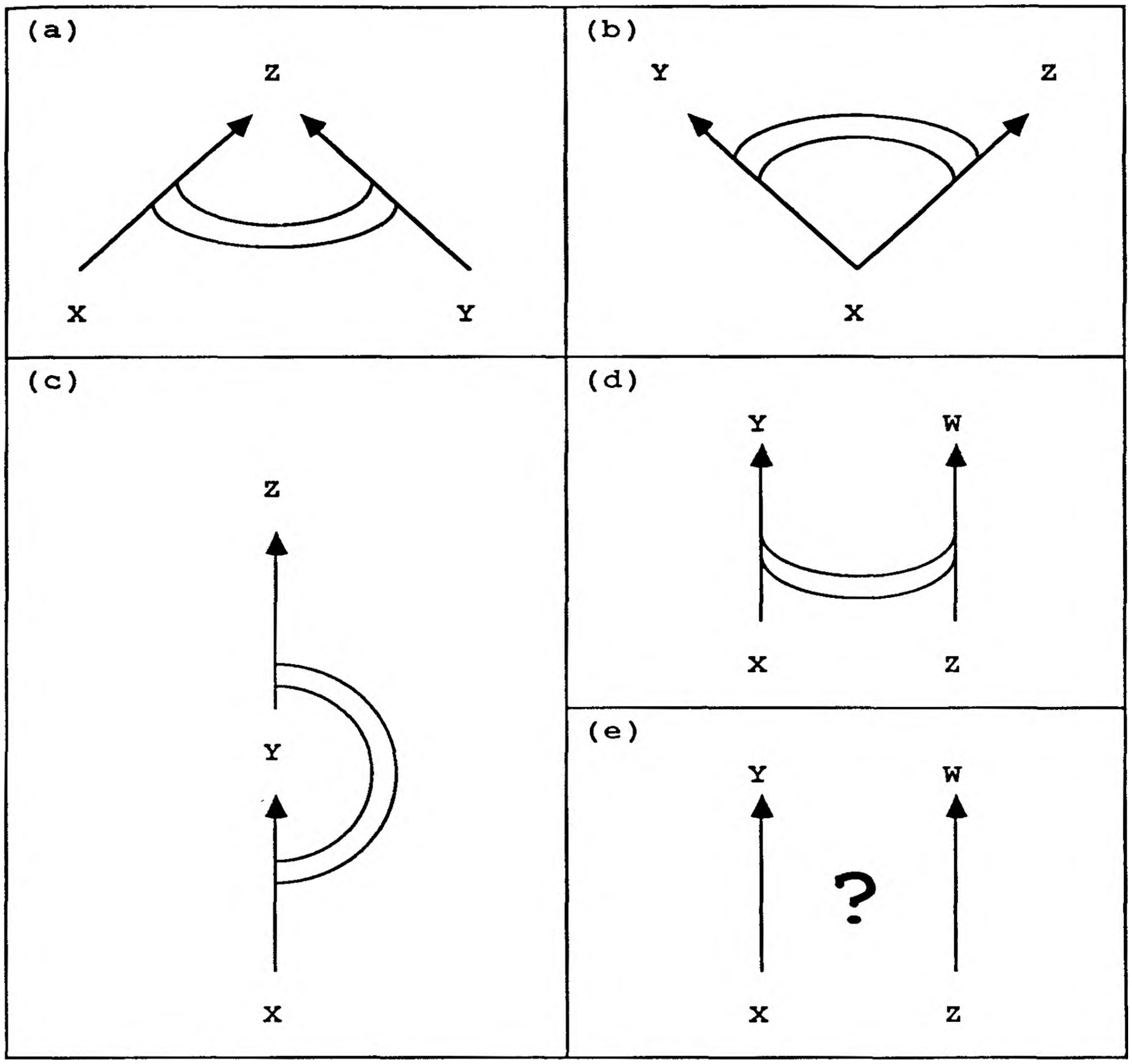


Figure 22.

Extended precedence diagram notation for five disjunctive constraints.

primitive ordering constraints in cases (a), (b), (c), and even (d). In case (e), however, the objects to be connected are logical entities that do not have a graphical analog. If sufficient graphical elements are introduced to model all of the relevant connections and combinations, the resulting representation will be so littered with arcs, and meta-arcs that it would be unsuitable for visual presentation.

The alternative is to consider purely textual forms of representation. One system, which is a logical extension of precedence diagrams and And/Or graphs, is presented next.

IV. THE PROPOSED REPRESENTATION

A. GENESIS

The genesis of the proposed representation took place over many months and involved the resolution of many different constraints and influences. It was recognized early in the course of the research that precedence diagrams were particularly easy to understand, produce, and utilize, and that form of representation was used in the author's first implementations of opportunistic scheduling[85,86]. Later, it was discovered that some problems simply could not be represented using that formalism. This discovery motivated the consideration of alternative representations and the definition of a canonical representation that could be used for purposes of comparison. With a clearer understanding of the problem space to be represented and with a clearer understanding of the properties of the candidate representations it was possible to create a more general, but still conceptually tractable representation.

The canonical representation provided the necessary insight. If each plan over a fixed set of operations is interpreted as a set of admissible sequences then the set of all possible plans over a fixed set of operations can be organized into a Boolean lattice such as that shown in

[85]Fox and Ho, A relational control mechanism for flexible assembly.

[86]Fox and Kempf, Opportunistic scheduling for robotic assembly.

Figure 3. Given that organization, each plan in the lattice can be constructed from the intersection and union of other plans. This suggested that each plan could be defined by the intersection and union of certain basic plans. Under the canonical representation, each plan is defined as the union of its constituent linear sequences. However, each linear sequence can be formed as the intersection of the plans defined by its constituent primitive ordering constraints. Hence, the primitive ordering constraints can be used to define the requisite basic plans and each plan in the lattice can be defined by the intersection and union of those elements.

B. SYNTAX

From this perspective, precedence diagrams are precisely those plans that can be formed using only intersections of the basic plans. The proposed representation is simply a textual version of precedence diagrams with additional notation for union (disjunction) and complement (negation). Syntactically it is composed of

(1) **symbols** which denote the activities to be sequenced,

(2) **primitive constraints** of the form **(X < Y)**, where **X** and **Y** denote activities to be sequenced under the constraint that **X** must precede **Y**, and

(3) **logical combinations** of primitive constraints constructed using the operators **and**, **or**, and **not**.

The formal syntax of the representation recursively defines the structure of a constraint expression **C**, as a composition of subexpressions C1 and C2, atomic symbols S1 and S2, and the literals <, (,), and, or, and not:

```
C --> (C1 and C2)
C --> (C1 or C2)
C --> (not C1)
C --> (S1 < S2)
```

In addition to this basic syntax, a number of macro expressions have been defined in order to abbreviate some frequently occurring conjunctive constraints. For instance, if several operations X, Y, and Z must all precede W then the corresponding constraint expression:

```
(((X < W) and (Y < W)) and (Z < W))
```

can be condensed to simply the phrase:

```
((X Y Z) < W).
```

Likewise, multiple successors can be combined within a single constraint as in:

```
(X < (Y Z W))
```

and by natural extension, the phrase

```
((X Y) < (Z W))
```

denotes the constraint that both X and Y must precede both Z and W. (For sake of clarity, infix notation will be used throughout this presentation although the Lisp programs which have been implemented expect constraint expressions in an equivalent prefix notation.)

C. SEMANTICS

Coupled with this syntax are two axioms which define the semantics of the representation. The first defines the transitive nature of order:

> Transitive Axiom:
> for all X, Y, and Z, (X < Y) and (Y < Z) implies that (X < Z).

The second defines the nature of serial processes:

> Serial Axiom:
> (a) it is never true that (X < X),
> (b) it is never true that ((X < Y) and (Y < X)).
> (c) for all distinct X and Y, ((X < Y) or (Y < X))

The second axiom, in a sense, tailors the representation to sets of activities which must be performed one step at a time such as the steps of an assembly performed by a single robot.

The second axiom also defines the proper interpretation of negation. Negation applied to a constraint expression can be distributed over conjunction and disjunction according to the rules of Boolean algebra:

(not (C1 and C2)) --> ((not C1) or (not C2)),

(not (C1 or C2)) --> ((not C1) and (not C2)),

double negations can eliminated:

(not (not C)) --> C,

and negation applied to a primitive constraint can be removed according to a rule derived from the Serial Axiom:

(not (S1 < S2)) --> (S2 < S1).

Hence, negation can be used freely to state constraints in their most natural form but it can be easily eliminated from constraint expressions in order to facilitate the analysis of those constraints.

D. COMPARISON UNDER THE FOUR CRITERIA

1. The Simple Assembly Problem. Using this notation the problems discussed in the previous chapter have a very succinct representation. For instance, the problem of three steps A, B, and C in which B should not be performed last, can be represented by the constraint expression

(not ((A C) < B))

or by the equivalent constraint expression

((B < A) or (B < C)).

Similarly, the simple assembly problem can be represented by the constraint expression:

(((i-C i-S i-R) < i-H) or ((i-H i-S i-R) < i-C)).

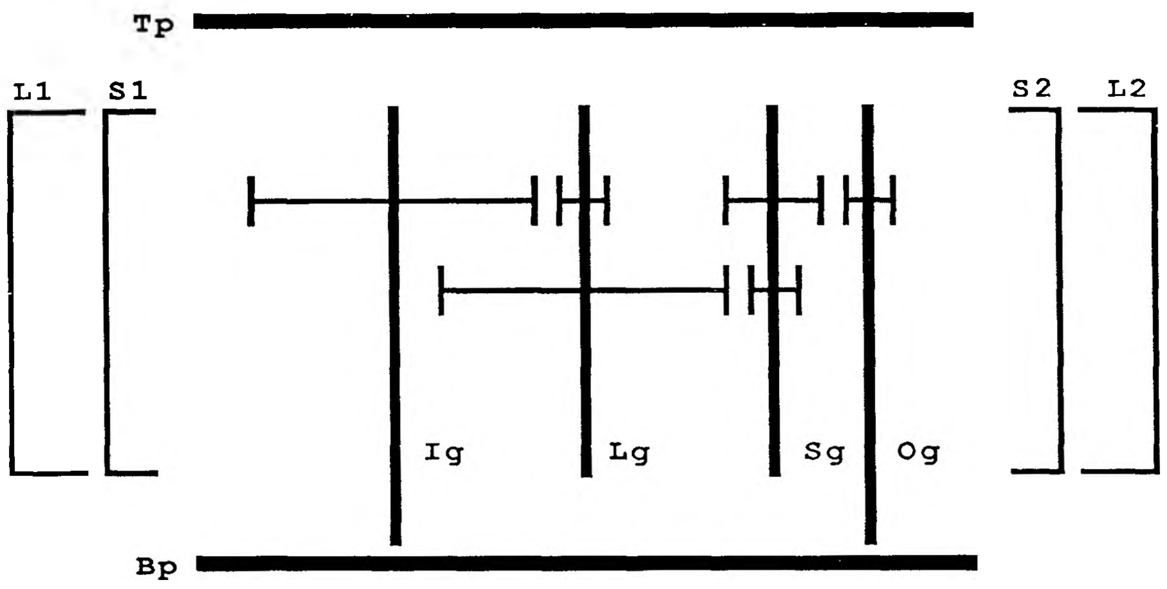
2. Completeness and Compactness. Because this representation is derived directly from the lattice structure of the space of possible plans it is guaranteed to be complete. However, it is not guaranteed to be compact. This is because of the restricted set of basic plans which can be used to construct other plans within the lattice. An arbitrary element of the lattice is not guaranteed to be composed of sequences which are dominated by simple ordering constraints. For instance, consider a plan for a set of activities A, B, C, D, and E consisting of the five sequences [A,B,C,D,E], [B,C,D,E,A], [C,D,E,A,B], [D,E,A,B,C], [E,A,B,C,D]. Obviously these sequences cannot be defined by a precedence diagram since every step occurs

both before and after every other step. In fact the most compact representation of this plan, when using the proposed representation and the primitive ordering constraints as the set of basic plans, is simply a disjunction of the five admissible sequences. However, ordering constraints derived from the most common physical constraints appear to be dominated by simple conjunctions and disjunctions of primitive ordering constraints.

3. Utility to the Planner. This representation can be particularly useful to a planner. The process of constructing a state transition network requires the enumeration of every feasible state of a task; the process of constructing an And/Or graph requires the enumeration of every possible decomposition of a task and the enumeration of every feasible state within the sub-problems that cannot be decomposed; the construction of a Petri net requires the enumeration and resolution of every constraint that defines the admissible sequences; the construction of a precedence diagram requires the enumeration of a necessarily satisfiable conjunction of constraints. Using the proposed representation, it is only necessary to enumerate the limiting constraints. Any further analysis or resolution of the constraints can be performed by machine using the algorithms discussed below.

Most frequently occurring physical constraints can be mapped directly into very simple temporal constraints. For instance, consider the gearbox shown in a schematic sideview in Figure 23. It is composed of a top plate Tp, bottom plate Bp, input gear Ig, large transfer gear Lg, small transfer gear Sg, output gear Og, and four distinct retaining clips L1, L2, S1, and S2 which secure the assembly after both the top and bottom plates are in place. The operations of installing those parts are represented by the symbols i-Tp, i-Bp, i-Ig, i-Lg, i-Sg, i-Og, i-L1, i-L2, i-S1, and i-S2. A normal assembly sequence would be to choose one of the plates, insert the four gears into that plate, mate the other plate with the gears, and secure the top and bottom plates using the four retaining clips.

The minimal constraints on the assembly are derived from the fundamental spatial and functional relationships among the parts. For instance, a recurring spatial relationship involves parts that are contained between other parts. All of the internal gears are contained between the top and bottom plates. It would be impossible to install those parts after installing the two plates. Similarly, the large transfer gear is contained between the input gear and the bottom plate and it would be impossible to install that gear after both the input gear and the bottom plate were installed. Other instances of between-ness and containment could be itemized as well. The functional relationships among the parts imply some additional ordering constraints. The top and bottom plates have the very useful property that either of them can serve as a jig for installation of the internal gears. Hence, it is quite logical to require that



(schematic side-view)

Figure 23.

The gearbox assembly problem.

one of the plates be installed before any attempt is made to install the gears. Similarly the retaining clips secure the assembly after both the top and bottom plates have been installed and it is quite logical to impose the constraint that the clips can only be installed after the plates. The complete itemized list of temporal constraints which are implied by these spatial and functional relationships are shown in Figure 24.

The explicit enumeration of these constraints may, at first, appear to be a complicated affair. In reality, any system which plans and executes a set of activities must identify and resolve every relevant sequencing constraint. This appears complicated because humans perform this analysis naturally, almost without conscious effort.

4. Utility to the Sequencer.

a. Complexity. By simplifying the task of the planner the task of the sequencer becomes substantially more difficult. It is simple to demonstrate that the problem of determining the satisfiability of an arbitrary temporal constraint expression is NP-Complete and the process of determining an admissible first step is NP-Hard. A sketch of the proof follows. First, it must be demonstrated that a proposed solution to a given sequencing problem can be verified in polynomial time. Second, it must be demonstrated that any method which can determine the satisfiability of a temporal constraint expression in polynomial time can be

Both the top and bottom plates must be installed before the four retaining clips can be installed:

```
(i-Bp < (i-L1 i-L2 i-S1 i-S2)) and
(i-Tp < (i-L1 i-L2 i-S1 i-S2)) and
```

The input gear, large transfer gear, small transfer gear,and output gear are all bound between the top and bottom plates:

```
(not ((i-Tp i-Bp) < i-Ig)) and
(not ((i-Tp i-Bp) < i-Lg)) and
(not ((i-Tp i-Bp) < i-Sg)) and
(not ((i-Tp i-Bp) < i-Og)) and
```

The input gear is bound between the top plate and the large transfer gear:

```
(not ((i-TP i-Lg) < i-Ig)) and
```

The large transfer gear is bound between the input gear and the bottom plate:

```
(not ((i-Ig i-Bp) < i-Lg)) and
```

The large transfer gear is bound between the small transfer gear and the bottom plate:

```
(not ((i-Sg i-Bp) < i-Lg)) and
```

The small transfer gear is bound between the top plate and the large transfer gear:

```
(not ((i-Tp i-Lg) < i-Sg)) and
```

All of the gears are self-jigging in either the top or bottom plate so one of those two parts must precede the installation of the internal gears:

```
((i-Bp < (i-Ig i-Lg i-Sg i-Og) or
 (i-Tp < (i-Ig i-Lg i-Sg i-Og))).
```

Figure 24.

Temporal constraints on the gearbox assembly.

solution to a given constraint expression can be verified in polynomial time.

The prototype NP-complete problem is the problem of Boolean satisfiability. Given a formula over a set of K Boolean variables v_1, v_2, $\ldots v_k$, does there exist an assignment of true and false values to the K variables which causes the formula to be true? It is a trivial matter to translate an arbitrary Boolean formula, in polynomial time, into a temporal constraint expression which is satisfiable if and only if the given Boolean formula is satisfiable. If some algorithm could be found which determines the satisfiability of a temporal constraint expression in polynomial time then it would be possible determine the satisfiability of a Boolean formula in polynomial time. The translation proceeds in two parts. First, for each variable in the Boolean formula, v_i, create two symbols ${}_0v_i$ and ${}_1v_i$. Second, create an exact copy of the given Boolean formula but for each occurrence of a variable v_i, substitute the temporal constraint $({}_0v_i < {}_1v_i)$. Ultimately, if a given Boolean formula fails to be satisfiable then it requires some variable v_i, to be both true and false. This fault translates into the temporal constraint

$$(({}_0v_i < {}_1v_i) \text{ and } ({}_1v_i < {}_0v_i))$$

which according to the Serial Axiom cannot be satisfied. Hence, the constructed temporal constraint expression is satisfiable if and only if the Boolean formula is satisfiable.

Although only a sketch of the proof, both elements have been demonstrated: a proposed solution to a temporal constraint expression can be verified in polynomial time and an established NP-complete problem can be translated into a temporal constraint expression in polynomial time. Hence, the problem of temporal satisfiability is NP-complete.

An immediate consequence is that the problem of constructing an admissible sequence of operations, or even the problem of selecting an admissible first step is NP-Hard! This is demonstrated by constructing a sequencing problem which can be used surreptitiously to establish the satisfiability of some Boolean formula. Given an arbitrary Boolean formula, construct a temporal constraint expression C, using the methods outlined above. Then construct the temporal constraint expression:

(((A < B) and C) or ((B < A) and (not C)))

where **A** and **B** are unique symbols not used in the construction of **C**. Obviously, **A** can be sequenced first only if **C** is satisfiable and **B** can be sequenced first only if **not-C** is satisfiable. Both are candidate first steps only if both **C** and and its inverse are satisfiable. If some polynomial time algorithm could be found for selecting an admissible first step, then that algorithm can be used to determine the satisfiability of a Boolean formula in polynomial time. Hence, the problem of sequencirg a set of activities according to a temporal constraint expression is NP-Hard.

b. Normalization. The inherent complexity of this language of temporal constraints should not be surprising. Artificial Intelligence is dominated by NP-Hard problems. The challenge is to discover some methods which make the problems more tractable. The key observation in this case is that some temporal constraint expressions are very easy to analyze. Any problems which can be represented by a conjunction of primitive constraints (i.e., a precedence diagram) can be analyzed and sequenced by simple polynomial time algorithms. Although not every problem can be reduced to such a simple representation, every problem can be represented as the union of a set of conjunctive plans (i.e., a disjunction of conjunctions of primitive constraints) which together cover every admissible sequence of operations. Most algorithms which can be applied to conjunctive constraint expressions can be adapted to the more general disjunctive normal form.

This is not to suggest that the planner should bear the responsibility for producing a plan in this form. Given an arbitrary constraint expression, an equivalent disjunctive normal form constraint expression can be produced by purely algebraic means. Since this is an NP-complete language there are two hidden costs associated with this process. If simple, direct methods are used to create the disjunctive normal form then the size of the resulting expression can be prohibitively large and if more sophisticated methods are used to produce the smallest possible disjunctive normal form the time required may be prohibitive. However, an investment of time can be justified since it enables simple polynomial time algorithms to be used in subsequent stages of analysis. For many problems, careful use of heuristic methods yields a reasonably small disjunctive normal form constraint expression in a reasonable amount of time.

Consider again, the gearbox assembly problem shown in Figure 23. Negation can safely be removed from the constraints shown in Figure 24, resulting in a positive constraint expression with nine embedded disjunctions as shown in Figure 25. Production of the disjunctive normal form of that constraint expression using the distributive law of Boolean algebra,

(X and (Y or Z)) --> ((X and Y) or (X and Z))

results in a set of 512 conjunctive clauses. (In general, the size of the disjunctive normal form grows exponentially with the number of applications of the distributive law.) However, all of the admissible sequences for performing this assembly are covered by two conjunctive clauses shown as precedence diagrams in Figure 26. Of the other 510 clauses, some are inconsistent some are inconsistent and can be covered by the clauses shown in Figure 26 and can be safely be removed.

Other systematic methods for explicitly generating the disjunctive normal form are possible. For instance, given a positive constraint expression (a constraint expression with no occurrence of negation) every relevant conjunctive clause

Both the top and bottom plates must be installed before the four retaining clips can be installed:

```
(i-Bp < (i-L1 i-L2 i-S1 i-S2)) and
(i-Tp < (i-L1 i-L2 i-S1 i-S2)) and
```

The input gear, large transfer gear, small transfer gear,and output gear are all bound between the top and bottom plates:

```
((i-Ig < i-Tp) or (i-Ig < i-Bp)) and
((i-Lg < i-Tp) or (i-Lg < i-Bp)) and
((i-Sg < i-Tp) or (i-Sg < i-Bp)) and
((i-Og < i-Tp) or (i-Og < i-Bp)) and
```

The input gear is bound between the top plate and the large transfer gear:

```
((i-Ig < i-Tp) or (i-Ig < i-Lg)) and
```

The large transfer gear is bound between the input gear and the bottom plate:

```
((i-Lg < i-Ig) or (i-Lg < i-Bp)) and
```

The large transfer gear is bound between the small transfer gear and the bottom plate:

```
((i-Lg < i-Sg) or (i-Lg < i-Bp)) and
```

The small transfer gear is bound between the top plate and the large transfer gear:

```
((i-Sg < i-Tp) or (i-Sg < i-Lg)) and
```

All of the gears are self-jigging in either the top or bottom plate so one of those two parts must precede the installation of the internal gears:

```
((i-Bp < (i-Ig i-Lg i-Sg i-Og) or
 (i-Tp < (i-Ig i-Lg i-Sg i-Og))).
```

Figure 25.

Positive temporal constraints on the gearbox assembly.

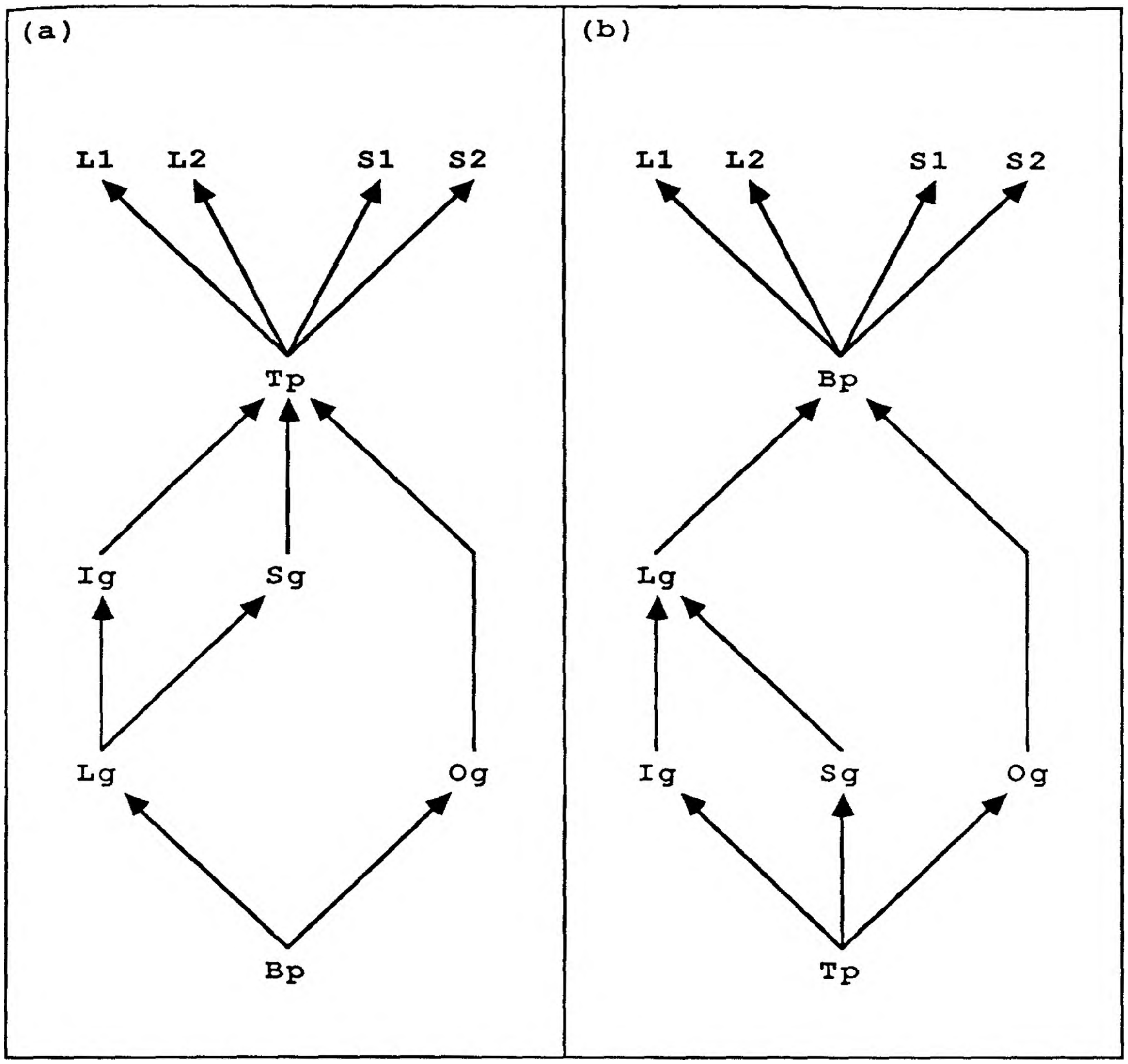


Figure 26.

Two conjunctive clauses which cover the admissible sequences for the gearbox assembly.

can be formed from subsets of the primitive constraints contained in that expression. Of course, the number of possible subsets grows exponentially with the number of constituent primitive constraints. Alternatively, given a constraint expression in conjunctive normal form (a conjunction of disjunctive clauses) every relevant conjunctive clause can be formed by constructing every possible combination of one primitive constraint from each disjunctive clause. Again the number of possible conjunctions grows exponentially with the number of clauses.

Given that the number of possible conjunctions grows exponentially with the size of the given constraint expression, the only feasible means of constructing the requisite disjunction of conjunctive clauses is to use some strategy of implicit enumeration. An efficient method for deriving a small number of covering conjunctive clauses is based upon a very careful analysis of the given constraints and a systematic expansion of a tree structured search space. Each node in the tree consists of two parts: a **partial solution** consisting of a conjunction of primitive constraints and a **partial problem** consisting of a constraint expression in conjunctive normal form. (Conjunctive normal form is necessary to the analytic and algebraic processes involved in the expansion of nodes into successor nodes.) Each node implicitly represents all of the possible ways that the partial solution can be completed and at the same time the partial problem satisfied. In the root node, the partial solution is nil and the partial problem consists of the initial constraint expression. In the target leaf nodes, the partial solution consists of a conjunction which satisfies the original constraint expression and the partial problem is nil.

The process of expanding a node into its two successor nodes begins by selecting one of the primitive constraints within the partial problem. The left successor is constructed under the assumption that this constraint will be realized in the solution. This implies that the selected constraint and any additional constraints implied by transitive closure necessarily become part of the partial solution. This also implies two reductions of the partial problem. First, any clause which contains a constraint which has been added to the partial solution should be removed since it is guaranteed to be satisfied. Second, since a constraint and its inverse will never be realized simultaneously, any clause which contains the inverse of a constraint which has been added to the partial solution should be reduced by the removal of that inverse constraint. The right successor node is constructed under the assumption that the selected constraint will not be realized in the solution. This implies that the partial solution should remain unchanged in the successor node and the partial problem should be reduced by the removal of the selected constraint from all of the clauses which contain it.

The partial solutions produced by this process are inconsistent if they require some cyclic ordering constraints such as ((X < Y) and (Y < X)) in violation of the serial axiom. The partial problems produced by this process are guaranteed to be non-satisfiable if some clause is reduced to length zero leaving no possibility for the clause to be satisfied.

The expansion of the search space proceeds in a leftmost, depth first fashion. If the partial solution in some node is inconsistent then the partial solution of every descendant node will be inconsistent. If the partial problem in some node is non-satisfiable then the partial problem of every descendant node is non-satisfiable. Hence, the expansion of a subtree terminates when a partial solution is produced which is inconsistent or when the partial problem has been reduced to a formula which is clearly non-satisfiable. Expansion also terminates when the partial problem has been completely reduced leaving a (partial) solution which satisfies the initial constraint expression. Each solution produced in this fashion is guaranteed to satisfy the given constraint expression and all of the solutions together are guaranteed to cover all of the admissible sequences.

The main factor which affects the performance of this algorithm is the method used for selecting the primitive constraint which is used to expand a given node. First choice is always given to constraints that obviously must be true and then preference is given to the constraints which most reduce the partial problem. This approach will either lead quickly to a contradiction and the elimination of a large subtree of the search space, or it will lead quickly to a solution which covers a large number of the admissible sequences.

A constraint obviously must be true if it is the only primitive constraint within a clause. One measure of the size of a partial problem is the size of disjunctive normal form which can be generated from its constituent constraints. As noted above, this can be related to the number of disjunctions, the number primitive constraints, or the number of clauses, depending upon the form of the constraint expression. The number of conjunctive clauses which can be generated from a conjunctive normal form constraint expression will always be less than or equal to the product of the number of primitive constraints in each of the clauses.

As implemented, the process of constructing the disjunctive normal form of a given constraint expression begins with the construction of the conjunctive normal form. This removes all tautologies, all negation and leaves the constraints in a regular structure which facilitates subsequent analytic and algebraic processes. Although conversion to clausal form can potentially result in a prohibitively large constraint expression, most of the problems that have been examined are dominated by conjunction and show only a modest increase in size. Occasionally, a constraint expression will be encountered which is a disjunction of two or more subexpressions. Such formulae are guaranteed to be excessively large when converted to conjunctive normal form. However, conjunctive normal form is only an intermediate form and not the ultimate goal. Whenever a disjunction of two or more subexpressions is given, the logical approach is to separately convert each subexpression to disjunctive normal form and then to disjunctively combine the results.

One common situation can easily lead to an unnecessarily large disjunctive normal form constraint expression. Suppose that a set of activities can be divided into two independent sets of activities S1 and S2 such that there are no sequencing constraints between the two sets. Further suppose that the constraints over S1 can be converted to a disjunctive normal form expression

(X1 or X2 or X3 or ...)

and the constraints over S2 can be converted to a disjunctive normal form expression

(Y1 or Y2 or Y3 or ...).

Since the constraints over S1 must be satisfied simultaneously with the constraints over S2, the disjunctive normal form constraint expression over the combined sets of activities will necessarily be the Cartesian product of the two constraint expressions

((X1 or X2 or X3 ...) and (Y1 or Y2 or Y3 ...)) =

((X1 and Y1) or (X1 and Y2) or (X1 and Y3) ...).

Given a set of activities and a temporal constraint expression which governs their execution, it is quite simple to partition the activities into independent sets. First, extract all of the the primitive constraints that occur in the given constraint expression (regardless of the logical structure of the formula). Then, for each primitive constraint (X < Y) construct the constraint (X Rel Y) which denotes the fact that X is related to Y. The relation Rel satisfies the definition of an equivalence relation. It is reflexive (X Rel X), symmetric (X Rel Y) implies (Y Rel X), and transitive (X Rel Y) and (Y Rel Z) implies (X Rel Z). Next, form the closure of the equivalence relation Rel. Each of the resulting equivalence classes contains only activities that are related by some combination of ordering constraints. Hence, the given set of activities can be partitioned into independent sets of activities along the boundaries defined by the equivalence relation. The execution of those activities should be treated as the interleaved execution of multiple independent tasks defined by the equivalence classes.

Some of these processes are illustrated in a small example shown in Figure 27. The initial conjunctive normal form constraint expression consists of one unary clause and

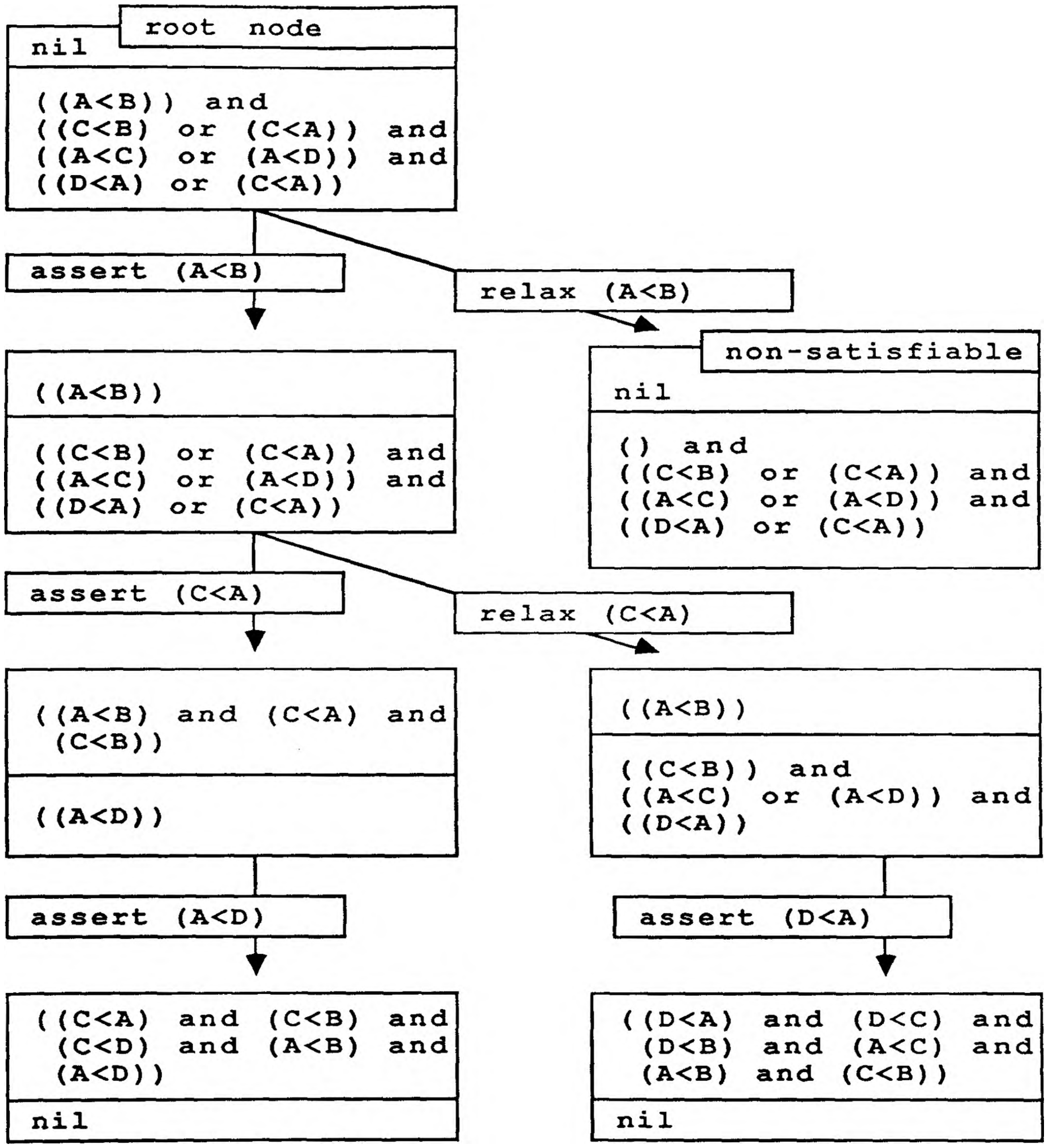


Figure 27.

Derivation of the disjunctive normal form
of a simple constraint expression.

three binary clauses:

((A < B)) and

((C < B) or (C < A)) and

((A < C) or (A < D)) and

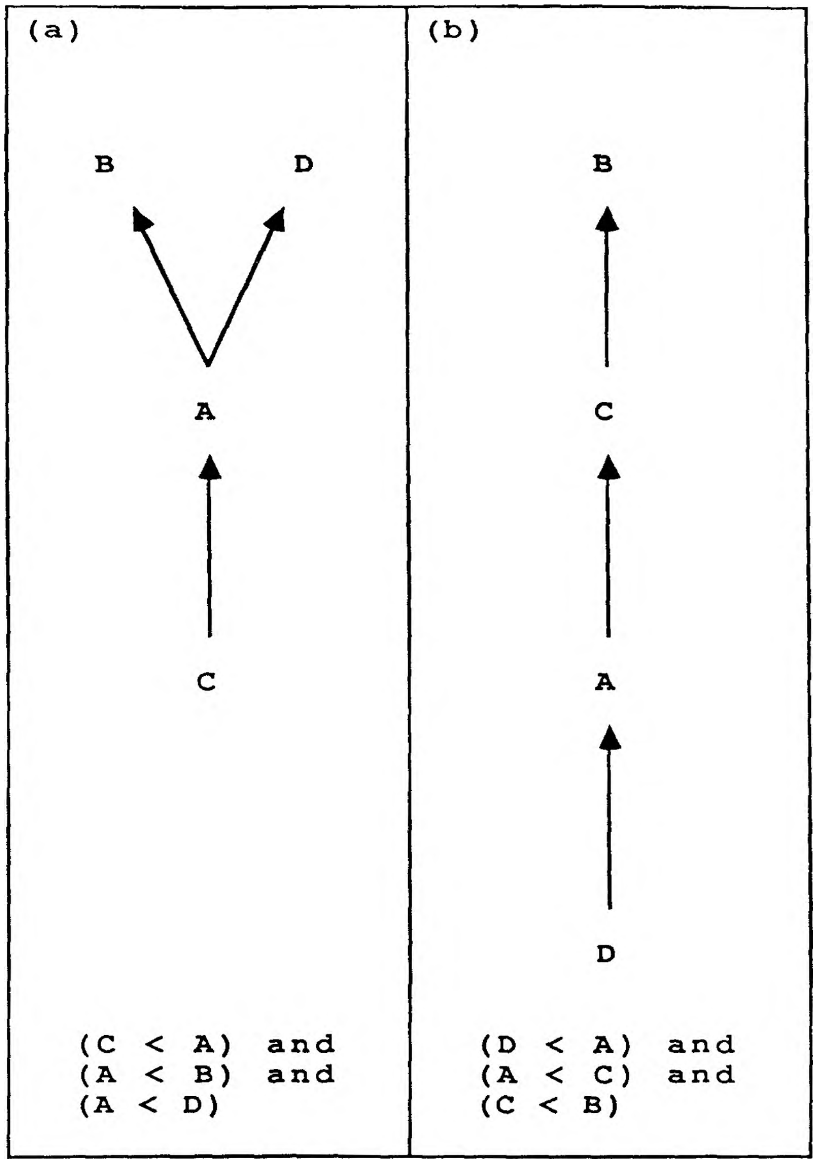
((D < A) or (C < A)).

The initial node is expanded by the constraint (A < B) since this constraint must necessarily be satisfied. The left successor node moves (A < B) into the partial solution and removes the clause ((A < B)) from the partial problem. Of course the right successor node is non-satisfiable since it is impossible to construct a solution without this constraint. (In actual practice, if a node is expanded by a necessary constraint then only the left successor node is constructed.)

The leftmost successor of the root node can be expanded by any of the remaining constraints but the constraint (C < A) is estimated to most reduce the partial problem. If that constraint is realized in the solution then the clause ((C < B) or (C < A)) is guaranteed to be satisfied, the clause ((A < C) or (A < D)) can only be satisfied by the constraint (A < D), and the clause ((D < A) or (C < A)) is guaranteed to be satisfied. Hence, if that constraint is realized in the solution the partial problem which remains to be satisfied is the unit clause ((A < D)). If the constraint (C < A) is not realized in the solution then the constraint (C < B) must be satisfied in the clause ((C < B) or (C < A)), the second clause of the partial problem is unaffected by the assumption, and (D < A) must be satisfied in the clause ((D < A) or (C < B)). Hence, if that constraint is not realized in the solution the partial problem which remains to be satisfied is a conjunction of the clauses ((C < B)), ((A < C) or (A < D)), and ((D < A)).

The remainder of the tree expansion is controlled by a series of constraints that must necessarily be satisfied. The two conjunctive clauses that satisfy the initial constraints and cover all of the admissible sequences are shown as precedence diagrams in Figure 28.

In small problems it is quite feasible to enumerate every possible conjunctive clause and then prune the redundant and non-satisfiable clauses. In problems of only modest size this is impossible. Using the methods described above the derivation of the disjunctive normal form of the gearbox assembly problem produced exactly 2 conjunctive clauses. This is considerably more efficient than producing 512 clauses only to eliminate 510. Consider the constraints over a second gearbox assembly problem shown in Figure 29. Any explicit methods for producing the disjunctive normal form will produce 1024 clauses, some of which are inconsistent and some of which are redundant. However, all of the admissible sequences for performing the task defined by these constraints are embodied in only twenty-two conjunctive clauses. Using the methods described above, forty consistent conjunctive clauses were produced. During that process six solutions were eliminated because they were



(C < A) and
(A < B) and
(A < D)

(D < A) and
(A < C) and
(C < B)

Figure 28.

Two conjunctive clauses which satisfy the simple constraint expression.

```
((co < cl))                    and
((ba < cl))                    and
((co < st) or (co < dr)) and
((dr < co) or (dr < ba)) and
((ba < dr) or (ba < ca)) and
((ra < co) or (ra < ba)) and
((mi < co) or (mi < ba)) and
((sm < co) or (sm < ba)) and
((ri < co) or (ri < ba)) and
((sm < ba) or (sm < mi)) and
((mi < sm) or (mi < ra)) and
((ra < mi) or (ra < co))
```

Figure 29.

The constraint expression for the second gearbox assembly problem.

subsumed by later solutions, twelve were rejected because they were subsumed by earlier solutions and twenty-two were retained in the final solution set.

c. Interpretation. The disjunctive normal form of a given constraint expression has several important interpretations. Like the original constraint expression, it still represents the task to be accomplished; it still represents the constraints over the individual operations to be performed; and it still represents the set of admissible sequences for performing those operations. The added interpretation that can be applied to the disjunctive normal form is that each conjunctive clause can be interpreted as a strategy for performing the given task. This can be particularly useful to engineers responsible for designing and managing the execution of complex tasks. They can first focus on the constraints over the constituent operations. Then they can verify and refine the specification of the tasks by analyzing the admissible strategies which are produced by this normalization.

The clearest way of presenting these strategies to a human analyst is in the form of precedence diagrams. As described earlier, a precedence diagram consists of a set of nodes which denote the operations to be performed and a set of arcs which denote the ordering constraints among the operations. If all of the necessary and implied constraints are mapped into the arcs of a precedence diagram the resulting figure would be a confusion of crossing lines. The most effective figure displays the transitive cover of a strategy which includes only the necessary constraints and eliminates any which are implied. The easiest way of distinguishing the necessary and implied constraints is to execute the transitive closure algorithm and each time a pair of constraints match the pattern ((X < Y) and (Y < Z)) mark the implied constraint (X < Z) as unnecessary. A simple method for producing the pictorial representation of a precedence diagram is to make a crude assignment of nodes to points on a grid, to draw the resulting figure on a high resolution monitor, and to give the human analyst interactive tools to edit and modify the placement of nodes.

d. Analysis. Given two or more strategies for performing a set of operations, it is natural to inquire whether they hold any execution sequences in common and whether there exists a single strategy which represents their intersection. Such questions can be answered in a simple and direct fashion. The set of sequences held in common between two strategies must simultaneously satisfy the constraints of both. Hence, the intersection of two or more given strategies is defined by their conjunction. If that conjunction is satisfiable the intersection is non-empty and every sequence which is consistent with that conjunction is consistent with every one of the given strategies.

Given two or more strategies for performing a set of operations, it is equally natural to inquire whether there exists a single strategy which exactly represents their union. Two cases are particularly easy to detect and resolve. The first case is when two strategies exactly partition the strategy which is their union. For example, if two strategies hold a conjunction of constraints C in common but are distinguished by a pair of complementary constraints, (A < B) in one case and (B < A) in the other, then all of the sequences defined by the two strategies are exactly covered by the conjunction C. The second case is when one strategy completely covers all of the sequences defined by another. For example, suppose that two strategies hold a conjunction of constraints C in common but one of them imposes some additional conjunctive constraints. The additional constraints only reduce the set of sequences admissible under the conjunction C. Again, all of the sequences defined by the two strategies together are exactly covered by the conjunction, C. This situation occurs frequently, and this sort of simplification is performed continuously during the process of normalization.

It is not always easy to identify strategies which can be combined under a more general strategy. For example, suppose that three operations, A, B, and C, are to be sequenced according to the disjunctive normal form constraint expression:

(((A < B)) or ((B < C)) or ((C < A))).

The first strategy, ((A < B)) defines the three admissible sequences [A,B,C], [A,C,B], and [C,A,B], the second strategy, ((B < C)), defines the three admissible sequences [A,B,C], [B,A,C], and [B,C,A], and the third strategy, ((C < A)) defines the three admissible sequences [B,C,A], [C,A,B], and [C,B,A]. Together, the three strategies cover every possible sequence over the three operations. The union of these strategies is the single strategy which imposes no constraint. Unfortunately, it is only an analysis of the constituent sequences or a series of algebraic manipulations which reveals this fact.

Under most strategies the combined ordering constraints limit the admissible sequences of operations but do not remove every sequencing option. In many cases, the number of admissible sequences provides a useful estimate of the inherent flexibility that can be exploited in sequencing those operations. For instance, consider a task of six operations. Ignoring differences produced by different labelings of the tasks, there are exactly 318 distinct strategies for executing the six operations. One strategy imposes no constraints and admits 720 sequences of operations and one strategy imposes a linear set of constraints and admits exactly one sequence of operations. The number of sequences admitted by the other strategies are bound strictly between 1 and 720. A variant of the SRI vision directed assembly problem, discussed above, was constructed and 100 assemblies were simulated for each of the 318 strategies. The average execution time as a function of the number of admissible sequences is shown in Figure 30.

The linear strategy required, on the average, 22.1 steps and the flat strategy required exactly 6.

In the absence of other considerations, the flexibility inherent in a strategy is directly related to the number of admissible sequences.

The naive approach to computing the number of admissible sequences would be to explicitly enumerate all of the feasible sequences by simulated pebbling or other more sophisticated algorithms[87]. Unfortunately, the least constrained problems will prove the most intractable. Consider a serial task of 15 steps with no sequencing constraints. There exists 15! = 1,307,674,368,000 admissible sequences. General methods are available which can determine the number of feasible sequences without explicit enumeration. These methods were first reported in a mathematics textbook by Wells[88] on computational methods for combinatorial problems . Generally this computation can be accomplished by recursive application of the function S defined by the following three rules:

---

[87]Kalvin, A.D. and Varol, Y.L., On the generation of all topological sortings, Journal of Algorithms 4 (1983), 150-162.

[88]Wells, M.G., Elements of Combinatorial Computing, Pergamon Press, Elmsford, New York, 1971.

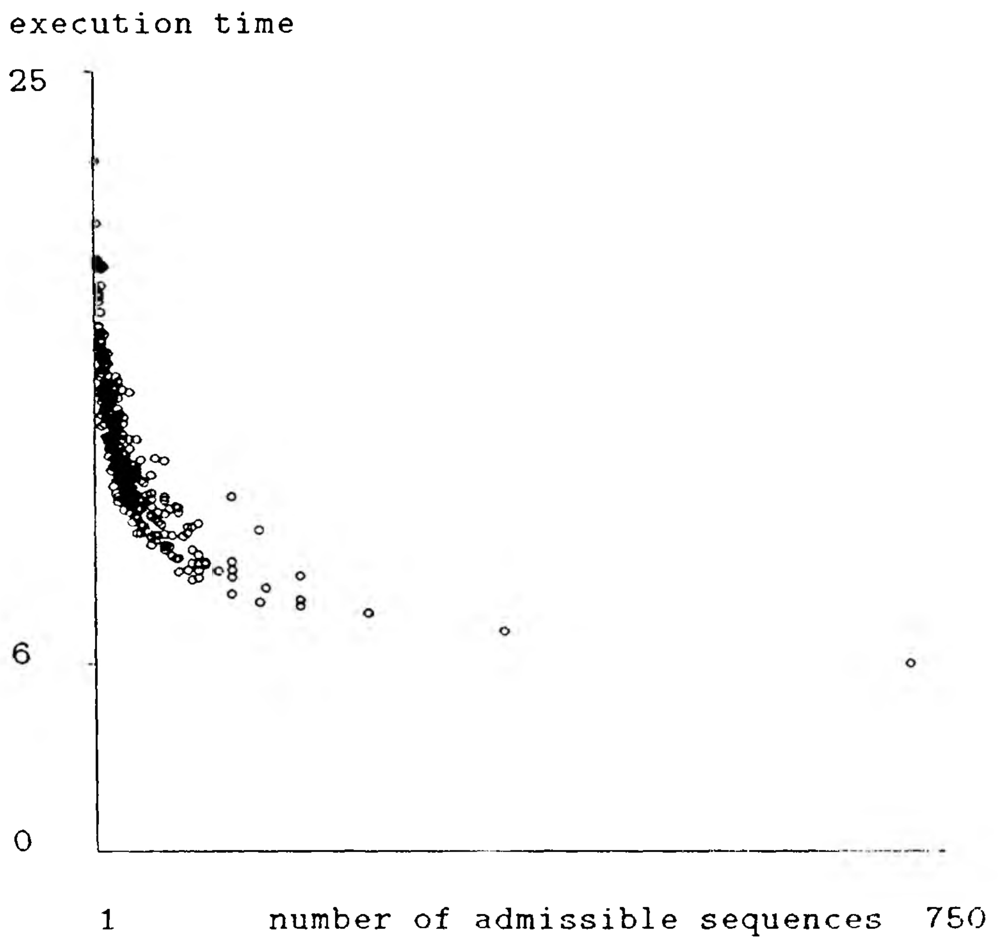


Figure 30.

Comparison of 318 strategies over six operations.

(1) If a set of operations P, under a set of constraints C, can be partitioned into two subsets, P_1 and P_2, such that all of the operations in P_1 must precede all of the operations in P_2, then

$$S(P,C) = S(P_1,C) * S(P_2,C).$$

(2) If a set of operations P, under a set of constraints C, can be partitioned into two subsets, P_1 and P_2, such that all of the operations in P_1 are independent of all of the operations in P_2, then

$$S(P,C) = S(P_1,C) * S(P_2,C) * ((|P_1| + |P_2|)!) / (|P_1|! * |P_2|!),$$

where the later part of the equation determines the number of ways that one sequence from the first partition can be interleaved with one sequence from the second partition.

(3) If a set of operations P, under a set of constraints C, cannot be divided into two subsets according to rules (1) or (2) then

$$S(P,C) = S(P,C1) + S(P,C2),$$

where

$$C_1 = (C \text{ and } (X < Y)),$$
$$C_2 = (C \text{ and } (Y < X)),$$

and the operations X and Y are elements of P but are unconstrained in C. In effect this third rule partitions the set of operations not into subsets but into substrategies, C_1 and C_2, which have no sequences in common.

Repeated application of these three rules is guaranteed to work regardless of the operations **X** and **Y** chosen when using rule 3, but the number of partitions generated is significantly affected by the choice. Wells offered no clear rule for the selection of those operations but an analysis of the situations when rule 3 must be invoked reveals the proper choice. If four points, **X**, **Y** , **Z**, and **W**, are subject to the <u>zig-zag</u> of constraints shown in Figure 31, rules 1 and 2 are guaranteed to fail and it will be necessary to invoke rule 3. The only way to successfully partition this set of four points into dependent and independent sets is to first produce two strategies for the execution of those operations. The first requires that **X** precede **Y**, **(X < Y)**, and the second requires that **Y** precede **X**, **(Y < X)**.

Even with this refinement the number of necessary partitions grows exponentially with the number of zig-zag patterns contained in the given strategy. It would be interesting to explore additional refinements which might control this explosion. Although the performance of this algorithm is known to be exponential, the author has not yet established the essential complexity of implicitly enumerating all of the admissible sequences.

Using this method the number of admissible sequences for the strategy shown in Figure 32 can be determined without explicit enumeration. The strategy can be

$$\begin{tikzcd}
Z & Y \\
X \arrow[u] & W \arrow[u] \arrow[ul]
\end{tikzcd}$$

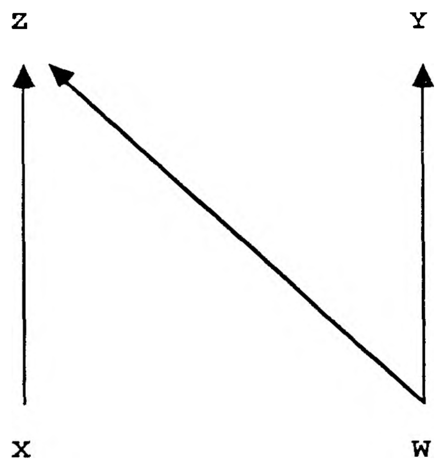


Figure 31.

The zig-zag relationship which requires the application of the decomposition rule 3.

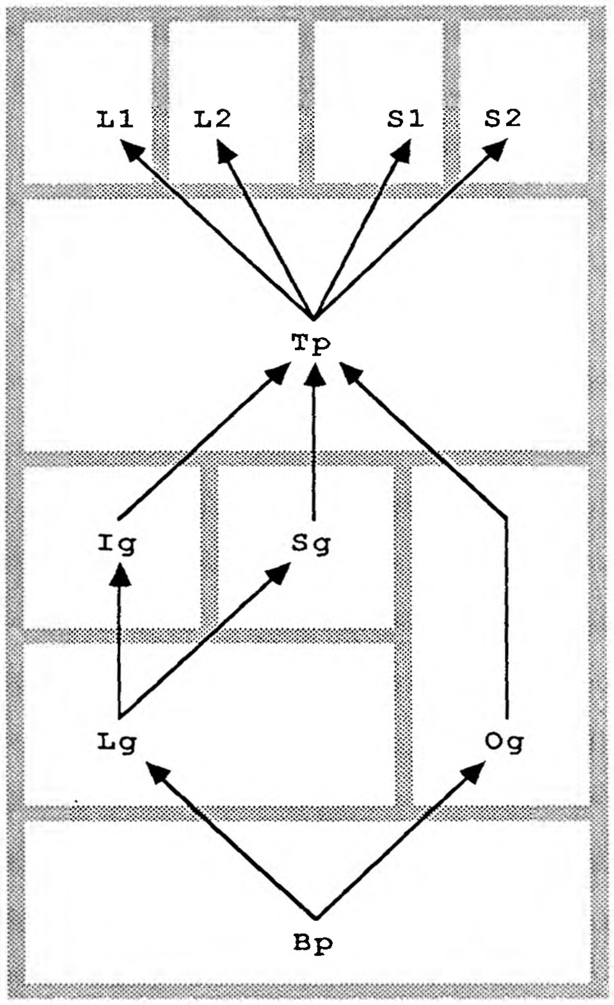


Figure 32.

The decomposition of one of the strategies for the gearbox assembly problem.

partitioned into four dependent sets of operations,

{i-L1, i-L2, i-S1, i-S2},

{i-Tp},

{i-Ig, i-Lg, i-Sg, i-Og}, and

{i-Bp},

and the total number of admissible sequences is the product of the number of admissible sequences for each of these subsets.

The set {i-L1, i-L2, i-S1, i-S2} admits twenty-four sequences, the set {i-Tp} admits one sequence, the set {i-Ig, i-Lg, i-Sg, i-Og} admits eight sequences, and the set {i-Bp} admits one sequence for a total of 192 sequences.

The eight admissible sequences over the set {i-Ig, i-Lg, i-Sg, i-Og} results from a partition into the two independent sets {i-Ig, i-Lg, i-Sg} and {i-Og}. There are two admissible sequences over {i-Ig, i-Lg, i-Sg}, one admissible sequence over {i-Og}, and four admissible ways that the operation i-Og can interleaved with the operations i-Ig, i-Lg, and, i-Sg. The other partitions are generated in a similar fashion.

Given two or more strategies for performing a set of operations it is a simple matter to determine the number of admissible sequences under each strategy, but a bit more complicated to determine the total number of admissible sequences. This is because the given strategies are not guaranteed to be disjoint. They may in fact hold some sequences in common. At present, the best solution is to compute the number of sequences under each strategy and to systematically subtract the number of sequences found in their intersections. Given the tools discussed above, this is straightforward. First, impose an index scheme on the given strategies. Then consider each of the strategies one at a time and form the sum of the following results. For strategy i, determine the number of admissible sequences under that strategy, but subtract the number of sequences found in the intersection between that strategy and each strategy with index greater than i. Admittedly this is a direct solution, but using the methods described above, the intersections and their decompositions can all be constructed automatically. Moreover, given J admissible strategies, at most J(J-1)/2 intersections need be constructed. For problems which have been investigated by the author, most of the intersections are nil.

The computations described above have an important side-effect. Application of the three decomposition rules results in the partition of a set of operations into dependent sets, independent sets , and disjoint sub-strategies. Just like the decomposition of a plan into strategies can be useful to human analysts and planners, this decomposition of strategies into their component sets can be used by human analysts to better understand the structure of the tasks that they must plan and coordinate.

e. Sequencing. The normalization and analysis described above serve only one purpose: the production of a set of constraints which precisely circumscribe the admissible sequences over a set of operations and the transformation of those constraints into a form suitable for real-time sequencing. The methods for sequencing a set of operations, given a disjunctive normal form constraint expression can be derived directly from methods for sequencing a set of operations given a simple precedence diagram.

The execution of a task according to the constraints embodied in a precedence diagram is analogous to the process of pebbling the nodes of that graph. That process is governed by one basic rule. A node can be pebbled only if all of its predecessor nodes have been previously pebbled. An initial node, with no predecessors, can be pebbled at any time. By analogy, the set of steps that can admissibly be executed next correspond exactly to the set of nodes that can admissibly be pebbled next. The process is complete when all of the nodes are covered. Given a properly formed precedence diagram, the set of all possible pebbling sequences is identical to the set of all possible execution sequences.

With three modifications, this same execution analogy can be extended to a set of precedence diagrams defined by a disjunctive normal form constraint expression. First, an operation is admissible if the corresponding node can be pebbled in at least one of the given precedence diagrams. Hence, the set of admissible operations is simply the union of the admissible operations derived from each of the given strategies. This is a natural interpretation of disjunction. Second, when an operation is selected, the corresponding node must be pebbled simultaneously in every precedence diagram which admits this as the next operation. Of course, the selected operation need not be admissible under every strategy but subsequent sequencing decisions must be based only on those strategies which are consistent with the selected operation. Hence the third modification, when an operation is selected, eliminate from further consideration every precedence diagram which does not admit this as the next operation.

This physical analogy suggests a straightforward algorithm for the construction of the set of admissible next operations. For each precedence diagram under consideration, initially let the set of admissible next operations be nil. Then consider each of the operations to be sequenced. If an operation has not been performed but every predecessor of that operation has been completed then add that operation to the set of admissible next operations. Hence, for each precedence diagram this set can be constructed by 2*K set operations to identify the admissible steps plus at most K set operations to build the resulting set. The final result is simply the union of the admissible operations derived from each of the given precedence diagrams. When an operation is selected, all of the precedence diagrams which are inconsistent with this choice must be excluded from further consideration.

By focusing on the constraints rather than the operations it is possible to determine the set of admissible next operations without the added process of explicitly eliminating the inconsistent strategies. For each constraint $(X < Y)$ within a particular strategy there are three possibilities. If neither X nor Y have been executed then it must be guaranteed that X precedes Y. This can be accomplished by removing the operation Y from the set of admissible next operations. If X has been executed before Y then the constraint has been satisfied and no special action needs to be taken. If Y has been executed before X then the constraint has been violated and it must be guaranteed that this strategy does not contribute any operations to the final result. Based upon these observations given a constraint expression C, in disjunctive normal form, and an execution history H the admissible next operations can be determined by the recursive function N defined by the following equations:

$N[C_1 \text{ or } C_2,H] = N[C_1,H] + N[C_2,H]$,

$N[C_1 \text{ and } C_2,H] = N[C_1,H] * N[C_2,H]$,

if $(X < Y)$ is satisfied in H

then $N[(X < Y),H] = \text{REMAINING}[H]$,

if $(X < Y)$ is violated in H

then $N[(X < Y),H] = \text{nil}$,

if neither X nor Y have been executed

then $N[(X < Y),H] = \text{REMAINING}[H] - \{Y\}$

where the operators *, +, and - denote the set operations intersection, union, and difference, and the function **REMAINING** returns those operations remaining to be executed.

E. <u>SUMMARY</u>

The proposed representation can be viewed from several perspectives. First, it is a symbolic language defined by a simple syntax and a small set of axioms. All of the methods for normalizing and analyzing temporal constraint expressions posed in this language must be consistent with this algebraic structure. Second, it is a temporal language which can be used to state the ordering constraints over a set of operations which must be executed serially. Third, it is a planning language. It has been derived from a specific model of the space of possible plans. A given constraint expression denotes a single element of that space by prescribing the set of admissible sequences over a given set of operations. Normalization and analysis of a given constraint expression can reveal the fundamental logical and chronological structures of a task, i.e., the constituent strategies of a plan and the decomposition of those strategies. Fourth, it is a sequencing language. Simple algorithms can be used to determine the properties of the constituent strategies of a plan and the relationships among them. Other algorithms can be used to determine the set of admissible next operations given a properly normalized constraint expression and an execution history. Finally, it is an NP-Complete language. The penalty for its generality is its inherent complexity. It has been shown that by investing the time required to normalize a given constraint expression and by investing the space required to store the resulting disjunctive normal form expression, subsequent analysis is much more tractable. Although defined by a simple syntax and semantics, it is a useful and interesting language which presents many challenging problems and areas of research.

V. CONCLUSIONS

The problem of sequencing the activities of a robot is a relatively new problem which has attracted investigators from both artificial intelligence and robotics. Research in this area is motivated by rapid advances in effectors, sensors, and computers and by the ambitious applications of autonomous robots proposed by NASA and the various defense agencies. Much of the research is concerned specifically with the issue of representation. What structures can be used to represent the tasks that a robot is to perform? What methods can be used to translate the specification of a task into that representation? What methods can be used to derive an admissible course of action from that representation?

Several representations have been proposed in the literature including state transition networks, And/Or graphs, precedence diagrams, and Petri nets. These have been discussed in great detail and compared with respect to four criteria: completeness, compactness, utility to the planner, and utility to the sequencer.

The representation proposed in this thesis is a language of temporal constraints derived directly from a model of the space of serial plans. It was specifically designed to encompass problems that include some form of disjunctive ordering constraints. This guarantees that the proposed language can completely and, to a certain extent, compactly represent all possible serial robotic tasks. The generality of this language carries a penalty. The proposed language of temporal constraints is NP-Complete. Specific methods have been demonstrated for normalizing constraints posed in this language in order to make subsequent sequencing and analysis more tractable. Using this language, the planner can specify necessary and alternative orderings to control undesirable interactions between steps of a plan. For purposes of analysis, the planner can factor a plan into strategies, and decompose those strategies into essential components. Using properly normalized constraint expressions the sequencer can derive admissible sequences and admissible next operations. Using these facilities, a robot can be given the specification of a task and it can adapt its sequence of operations according to run-time events and the constraints on the operations to be performed.

Continuing research in this area can pursue several questions. At a purely abstract level, what are the algebraic properties of this language of temporal constraints? At a theoretical level, is it possible to derive the temporal constraints over a set of operations from some specification of a task and its operations? Can this facility for stating and analyzing temporal constraints be applied in some established problem domains such as graph coloring or job-shop scheduling? At a more practical level, is it possible to apply some metrics to the set of admissible next operations in order to determine which of those operations might be best, or is it possible to apply some metrics to the constituent strategies of a plan in order to determine which of those might be best? Answers to such questions require the definition of some model of the executional environment and some mapping from real-world applications into that model. Perhaps the most challenging avenue of research would be to consider extensions to the model of the space of possible plans and to the language of temporal constraints in order to represent concurrency.

BIBLIOGRAPHY

Interim Report #3, Research for Intelligent Task Automation, Air Force Contract #F33615-82-C-5092, July 15, 1983.

Allen, J., Maintaining knowledge about temporal intervals, _Communications of the ACM_ 26 (1983), 832-843.

Allen, J.F. and Koomen, J.A., Planning using a temporal world model, in _Proceedings Eighth International Joint Conference on Artificial Intelligence_, Karlsruhe, West Germany, 1983, pp. 741-747.

Ambler, P., Barrow, H.G., Brown, C.M., Burstall, R.M., and Popplestone, R.J., A versatile computer-controlled assembly system. _in Proceedings Third International Joint Conference on Artificial Intelligence_, Stanford University, 1973, pp. 298-303.

Ambler, P., Barrow, H.G., Brown, C.M., Burstall, R.M., and Popplestone, R.J., A versatile system for computer-controlled assembly, _Artificial Intelligence_ 6 (1975), 129-156.

Barr, A. and Feigenbaum, E.A. (eds), _The Handbook of Artificial Intelligence_, Vol 1, Chapter II, Section B1: State-space representation, William Kaufmann, 1981.

Barr, A. and Feigenbaum, E.A. (eds), _The Handbook of Artificial Intelligence_, Vol 1, Chapter II, Section B2: Problem Reduction representation, William Kaufmann, 1981.

Blackstone Jr., J.H., Phillips, D.T., and Hogg, G.L., A state-of-the-art survey of dispatching rules for manufacturing job shop operations, International Journal of Production Research 20 (1982), 27-45.

Bolles, R.C. and Cain, R.A., Recognizing and locating partially visible objects: the local-feature-focus method, International Journal of Robotics Research 1 (1982), 57-82.

Chapman, D., Non-linear planning: a rigorous reconstruction, in Proceedings Ninth International Joint Conference on Artificial Intelligence, Los Angeles, Calif., 1985, pp. 1022-1024.

Cheeseman, P., A representation of time for automatic planning, in Proceedings Second IEEE International Conference on Robotics and Automation, Atlanta, Georgia, 1983, pp. 513-518.

Cheeseman, P., Overview of planning/scheduling problems and procedures, in Proceedings NASA/JPL Space Telerobotics Workshop, Pasedena, Calif., 1987 (to appear).

Coffman, E. G. and Graham, R.L., Optimal scheduling for two-processor systems, Acta Informatica 1 (1972), 200-213.

Conway, R. W., Maxwell, W. L., and Miller, L. W., Theory of Scheduling, Addison-Wesley, Reading, Mass., 1967, p. 103.

deMello, L.S.H. and Sanderson, A., And/Or graph representation of assembly plans, in Proceedings Fifth National Conference on Artificial Intelligence, Philadelpha, Penn., 1986, pp. 1113-1119.

Drummond, M., Plan Nets: a formal representation of action and belief for automatic planning systems, Ph.D. Dissertation, Department of Artificial Intelligence, University of Edinburgh, 1986.

Drummond, M.E., Contingent plan structures for spacecraft, in Proceedings NASA/JPL Workshop on Space Telerobotics, Jet Propulsion Laboratory, Pasedena, Calif., 1987 (to appear).

Drummond, M.E., Refining and extending the procedural net, in Proceedings Ninth International Joint Conference on Artificial Intelligence, Los Angeles, Calif, 1985, pp.1010-1012.

Fikes, R.E. and Nilsson, N.J., STRIPS: A new approach to the application of theorem proving to problem solving, Artificial Intelligence 2 (1971), 189-208.

Fox, B.R., The implementation of opportunistic scheduling, in Proceedings Intelligent Autonomous Systems, Amsterdam, 1986, pp. 231-240.

Fox, B.R. and Ho, C.Y., A relation control mechanism for flexible assembly, in Advanced Software in Robotics, A. Danthine and M. Geradin (eds), North-Holland, Amsterdam, 1984.

Fox, B.R. and Kempf, K.G., A representation for opportunistic scheduling, in Third International Symposium on Robotics Research, O. Faugeras and G. Giralt (eds), MIT Press, Cambridge, Mass., 1986.

Fox, B.R. and Kempf, K.G., Complexity, uncertainty, and opportunistic scheduling, in Artificial Intelligence Applications, The Engineering of Knowledge-Based Systems, C. Weisbin (ed), IEEE COmputer Society Press, Washington, D.C., 1986.

Fox, B.R. and Kempf, K.G., Opportunistic scheduling for robotic assembly, in Robotics and Industrial Engineering, Selected Readings, E.L. Fisher and O.Z. Maimon (eds), Industrial Engineering and Management Press, Institute of Industrial Engineers, Atlanta, Georgia, 1986.

Fox, B.R. and Kempf, K.G., Planning, scheduling, and uncertainty in the sequence of future events, in Uncertainty in Artificial Intelligence, Vol. 2, J. Lemmer (ed), North-Holland, Amsterdam, 1987 (to appear).

Fox, B.R. and Kempf, K.G., Reasoning about opportunistic schedules, in Proceedings IEEE International Conference on Robotics and Automation, Raleigh, North Carolina, 1987. (to appear)

Fox, M. S., Constraint-Directed Search: A Case Study of Job-Shop Scheduling, Ph.D dissertation, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Penn.

Friedland, P.E., Knowledge-based experiment design in molecular genetics, Ph.D. dissertation, Report Number 79-771, Computer Science Department, Stanford University, 1979.

Garey, M. R., Graham, R. L., and Johnson, D. S., Performance guarantees for scheduling algorithms, Operations Research 26 (1978), 3-21.

Garey, M.R. and Johnson, D.S., Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, New York, 1979.

Gillett, B. E., Introduction to Operations Research, McGraw-Hill, New York, 1976, pp. 262-277.

Graham, R. L., The combinatorial mathematics of scheduling, Scientific American 238 (1978), 124-132.

Graves, S. C. and Lamar, B. W., An integer programming procedure for assembly system design problems, Operations Research 31 (1983), 522-545.

Held, M., Karp, R. M., and Shareshian, R., Assembly-line balancing - dynamic programming with precedence constraints, Operations Research 11 (1963), 442-459.

Hu, T. C., Combinatorial Algorithms, Addison-Wesley, Reading, Mass., 1982, pp. 222-228.

Jackson, J. R., A computing procedure for a line balancing problem, *Management Science* 2 (1956), 261-271.

Johnson, R. V., Assembly line balancing algorithms: computational comparisons, International Journal of Production Research 19 (1981), 277-287.

Johnson, R. V., A branch and bound algorithm for assembly line balancing problems with formulation irregularities, Management Science 29 (1983), 1309-1324.

Kalvin, A.D. and Varol, Y.L., On the generation of all topological sortings, Journal of Algorithms 4 (1983), 150-162.

Kao, E. P. C. and Queyranne, M. On dynamic programming methods for assembly line balancing, Operations Research 30 (1982), 375-390.

Klein, M., On assembly line balancing, Operations Research 11 (1963), 281.

Kondoleon, A. S., Assessing cycle times for robot assembly systems, Robotics Today 3 (1981), 38-41.

Lozano-Perez, T., A simple motion planning algorithm for general robot manipulators, in Proceedings Fifth National Conference on Artificial Intelligence, Philadelphia, Penn., 1986.

Lozano-Perez, T., Mason, M.T., and Taylor, R.H., Automatic synthesis of fine-motion strategies for robots, in Robotics Research: The First International Symposium M. Brady and R. Paul (eds), MIT Press, Cambridge, Mass., 1984.

Malcolm, C. and Fothergill, P., Some architectural implications of the use of sensors, in Proceedings Intelligent Autonomous Systems, Amsterdam, 1986, pp. 71-78.

Malcolm, C., DAI Working Paper 187, Department of Artificial Intelligence, University of Edinburgh, 1986.

Mastor, A. A., An experimental investigation and comparative evaluation of production line balancing techniques, Management Science 16 (1970), 728-746.

Nilsson, N. J., from the abstract of Artificial Intelligence: Engineering, Science, or Slogan?, Technical Note 248, SRI International, Menlo Park, Calif, July 1981.

Panwalkar, S.S. and Iskander, W., A survey of scheduling rules, Operations Research 25 (1977), 45-61.

Pearl, J., Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley, Reading, Mass., 1984.

Petersen, J.L., Petri nets, Computing Surveys 9 (1977), 223-252.

Pinto, P. A., Dannenbring, D. G., and Khumawala, B. M., Assembly line balancing with processing alternatives: an application, Management Science 29 (1983), 817-830.

Prenting, T.O. and Battaglin, R.M., The precedence diagram: a tool for analysis in assembly line balancing. Journal of Industrial Engineering, vol 15, #4, pp. 208-211, July-Aug 1964.

Rembold, U., Levi, P., Sensors and control for autonomous robots, in Proceedings Intelligent Autonomous Systems Amsterdam, The Netherlands, 1986, pp. 79-89.

Sacerdoti, E.D., A structure for plans and behavior, Ph.D. dissertation, Technical Note 109, AI Center, SRI International, Inc., Menlo Park, Calif., 1975.

Salisbury, J.K. and Craig, J.J., Articulated hands: force control and kinematic issues, International Journal of Robotics Research 1 (1982), 4-17.

Salveson, M. E., The assembly line balancing problem, Journal of Industrial Engineering 6 (1955), 18-25.

Schenker, P. S., NASA Telerobotics Research: program objectives and technology outreach, presentation at NASA/JPL Space Telerobotics Workshop, Pasedena, Calif, 1987.

Salvador, M.S., Scheduling and Sequencing, Section 2: A Classification of Scheduling Problems,in Handbook of Operations Research: models and applications, Moder, J.J. and Elmaghraby, S. E. (eds), Van Nostrand Reinhold, New York, 1978.

Stefik, M.J., Planning with constraints. Ph.D. dissertation, Report Number 80-784, Computer Science Department, Stanford University, 1980.

Sussman, G.J., A computational model of skill acquisition, Ph.D. dissertation, AI Technical Report 297, AI Laboratory, Massachusetts Institute of Technology, 1973.

Talbot, F. B. and Patterson, J.H., An integer programming algorithm with network cuts for solving the assembly line balancing problem, Management Science, v30 (1984), 85-99.

Tate, A., Project planning using a hierarchic non-linear planner, Report Number 25, AI Research Department, University of Edinburgh, 1976.

Ullman, J. D., NP-complete scheduling problems, Journal of Computer and System Sciences 10 (1975), 384-393.

Vilain, M. and Kautz, H., Constraint propagation algorithms for temporal reasoning, in Proceedings Fifth National Conference on Artificial Intelligence, Philadelphia, Penn., 1986, pp. 377-382.

Wells, M.G., Elements of Combinatorial Computing, Pergamon Press, Elmsford, New York, 1971.

Whitney, D.E., State space models of remote manipulation tasks, IEEE Transactions on Automatic Control, vol AC-14, #6, Dec. 1969.

VITA

Barry Ross Fox was born February 11, 1952 in Espanola, New Mexico. He received his primary and secondary education in Overland Park, Kansas. He received a B.S. in Mathematics from the University of Missouri-Kansas City and a M.S. in Computer Science from the University of Kansas. He has been enrolled in the Graduate School of the University of Missouri-Rolla since August, 1981.