Heuristics for Job-Shop Scheduling

by

Kenneth Alan Pasch

B.S.M.E. Massachusetts Institute of Technology
(1981)

M.S.M.E. Massachusetts Institute of Technology
(1984)

Submitted to the
Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of

Doctor Of Science

at the
Massachusetts Institute Of Technology
January 1988

Signature of Author_

Kenneth Alan Pasch
Department of Mechanical Engineering
January 23, 1988

Certified by____

_____

Professor Warren P. Seering
Committee Chairman

Accepted by____

_____

Ain A. Sonin
Chairman, Departmental Graduate Committee

Heuristics for Job-Shop Scheduling

by

Kenneth A. Pasch

Submitted to the Department of Mechanical Engineering on January 10, 1988 in partial fulfillment of the requirements for the degree of Doctor of Science in Mechanical Engineering.

Abstract

Two methods of obtaining approximate solutions to the classic *General Job-Shop Scheduling Problem* are investigated. The first method is iterative. A sampling of the solution space is used to decide which of a collection of space pruning constraints are consistent with "good" schedules. The selected space pruning constraints are then used to reduce the search space and the sampling is repeated. This approach can be used either to verify whether some set of space pruning constraints can prune with discrimination or to generate solutions directly.

Schedules are represented as trajectories through a cartesian space. Under the objective criteria of *Minimum Maximum Lateness* a family of "good" schedules (trajectories) are geometric neighbors (reside within some "tube") in this space. The second method of generating solutions takes advantage of this adjacency by pruning the space from the outside in thus converging gradually upon this "tube." On the average this method significantly outperforms an array of the *Priority Dispatch Rules* when the objective criteria is that of *Minimum Maximum Lateness* . It also compares favorably with a recent iterative relaxation procedure.

Thesis Committee:
Prof. Warren Seering, Chairperson
Prof. Stephen Graves
Prof. Tomas Lozano-Perez

Acknowledgments

First of all I would like to thank my thesis advisor Professor Warren P. Seering for his input, encouragement, and support throughout this research. This thesis is made possible through his guidance and that of my other committee members Professor Tomas Lozano-Perez and Professor Stephen C. Graves.

I would also like to thank the members of Professor Seering's Design Group who helped throughout my research; especially Michael Caine, Steven Eppinger, Steve Gordon, Neil Singer, Karl Ulrich, Erik Vaaler, and Al Ward. All of them played important roles in my research. Thanks are also due to Phillippe Brou and Sundar Narasimhan for their roles as computational gurus.

The IBM corporation is greatly appreciated for their support along with that of the Office of Naval Research, the Systems Development Foundation, the Department of Defense and the numerous other supporters of the AI lab. Thanks to the AI lab's generous support staff and facilities.

A very special thanks goes to my parents for their love and support throughout my graduate and undergraduate studies. And last but not least, thanks to my wife Jeanne who has only known me as a student and who deserves to receive an honorary degree for tolerating the PhD process.

To my parents:

Charles Joseph and Christina Marie Pasch

Contents

List of Figures

Introduction - Review

Chapter 1

Two methods of obtaining approximate solutions to the classic *General Job-Shop Scheduling Problem* are investigated. The first method is iterative. A sampling of the solution space is used to decide which of a collection of space pruning constraints are consistent with "good" schedules. The selected space pruning constraints are then used to reduce the search space and the sampling is repeated. This approach can be used either to verify whether some set of space pruning constraints can prune with discrimination or to generate solutions directly.

Schedules can be represented as trajectories through a cartesian space. Under the objective criteria of *Minimum Maximum Lateness* a family of "good" schedules (trajectories) are geometric neighbors (reside within some "tube") in this space. This second method of generating solutions takes advantage of this adjacency by pruning the space from the outside in thus converging gradually upon this "tube." On the average this method significantly outperforms an array of the *Priority Dispatch Rules* when the objective criteria is that of *Minimum Maximum Lateness* . It also compares favorably with a recent iterative relaxation procedure.

1.1 General Problem Description

One way of representing a superset of the scheduling problems considered in this thesis is with a *PERT/CPM* network. This representation will be introduced and will then be specialized down to the *Job-Shop Scheduling* problem. Typical applications are the scheduling of construction projects and research programs. The term *PERT/CPM* is an abbreviation for **P**rogram **E**valuation and **R**eview **T**echnique/**C**ritical **P**ath **M**ethod. This representation was devised by the US Navy to help plan and expedite the development of the Polaris Missile. A PERT network is a graphical representation of how the activities of a project are related. Activities require resources for their processing and can represent either a single event (pour concrete) or a given operation repeated on a batch of parts (paint these widgets). In one incarnation, the **A**ctivity **O**n **N**ode (AON) variety, the nodes of the graph correspond to activities while the directed edges correspond to precedence relations. The usual interpretation of one of these precedence relations is that the activity at the tail of the directed edge must be completely finished being processed before the activity at the head can be started. The edge lengths in an AON *PERT* network have no significance. Shown in Figure 1.1 is a AON *PERT* network relating activities A_1 through A_7.

The nodes of the AON *PERT* network represent activities to be processed. These are usually labeled with information specified from the project data and with information computed from the network. Project data includes, the nature of the activity, the processing time of the activity (one exact number or some distributional information usually no more complex than optimistic and pessimistic estimates of processing time), and the amount, type and cost of (possibly multiple) resources

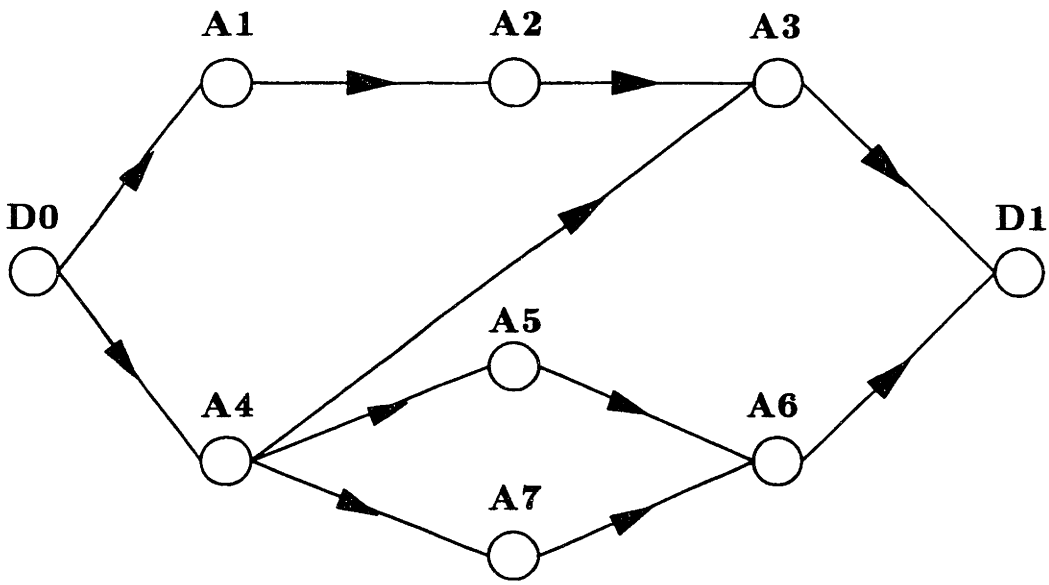


Figure 1.1: Activity On Node (AON) *PERT* network

required to process this activity. Note that dummy nodes D_0 and D_1 have been added to the start and to the end of the network. *Early Start Times* (EST), *Late Start Times* (LST) and Slacks are defined and can be computed from the network as follows. To compute the EST of an activity (the earliest time at which an activity can be started without violating precedence constraints) assume the dummy start node is finished at time 0. Then for nodes who's predecessors have all been assigned *Early Start Times* set the EST of the node to the max of the EST+(Process Time)'s of all the predecessor nodes. This will result in assignment of EST to all nodes including the dummy finish node. To compute the LST's (the latest time at which an activity can be started without increasing the length of the overall project) set LST=EST of the finish node, then for nodes who's successors have all been assigned LST's, set the LST of the node to the minimum of the LST-(Process Time)'s of all the successor

nodes. Note that the nodes are visited in Topological order. An activity's slack is then defined as the difference LST-EST. This slack indicates the amount of freedom allowed in scheduling this activity without extending the overall length of the project or violating precedence constraints. Note that some slacks are dependent.

Note also, if the EST's, LST's and slacks are computed as described above, then there will be a chain (of width 1 or more) of activities from start to finish which have slack values equal to 0. This chain or path is the so called *Critical Path.* Assuming that infinite resources are available, this *Critical Path* determines the overall duration of the project. In order to shorten the project duration one or more activities along the *Critical Path* must be shortened. The **C**ritical **P**ath **M**ethod (CPM) is a tool used to find the most cost effective way to shorten ("Crash") the *Critical Path* assuming that there is some time/cost tradeoff for each activity.

Instead of being explicitly represented as activities and precedence relations one of these networks can be thought to be composed recursively in terms of series, parallel and N subgraphs [14]. These subgraphs are shown if Figure 1.2. The overall structure of a network can then be encoded as a tree with Series (S), Parallel (P) and N (N) nodes with activities as leaves. The tree encoding of the network of Figure 1.1 is shown in Figure 1.3. Note that there is an implied ordering in the branches of the S and N nodes.

In the previous discussion it was assumed that infinite amounts of the resources required to process the activities were available at some cost. Specializing this assumption down to finite amounts of each resource gives rise to the class of resource constrained network problems. In these problems the *Critical Path* may or may not determine the overall project duration. This is because some activities may have to wait for resources to become available. Therefore, instead of allocating cash in a

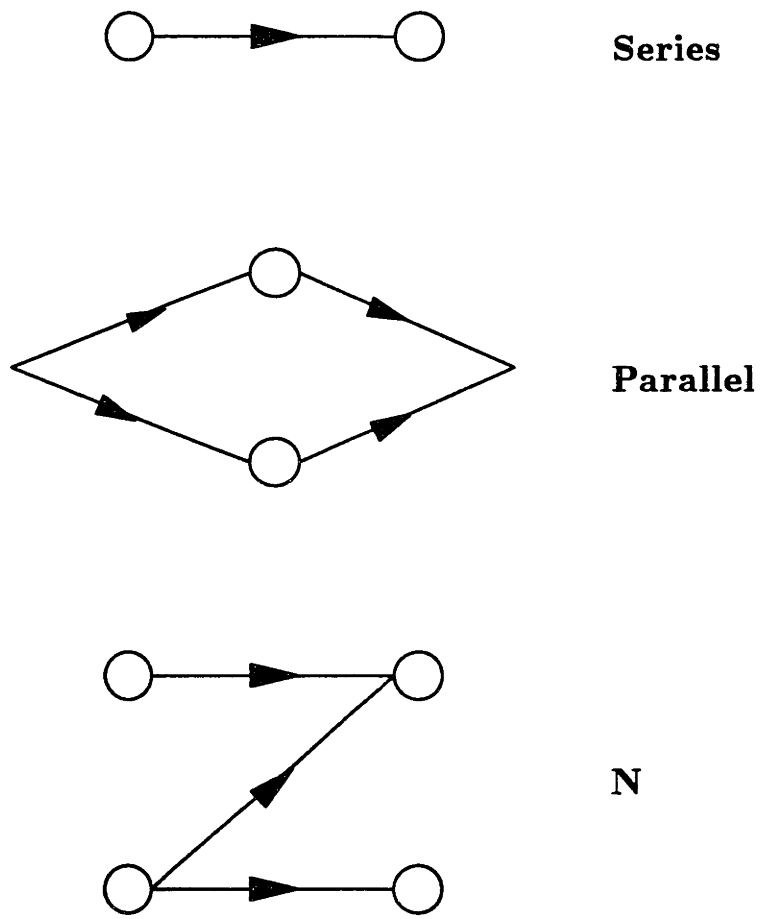


Figure 1.2: AON Series, Parallel and N Subgraphs

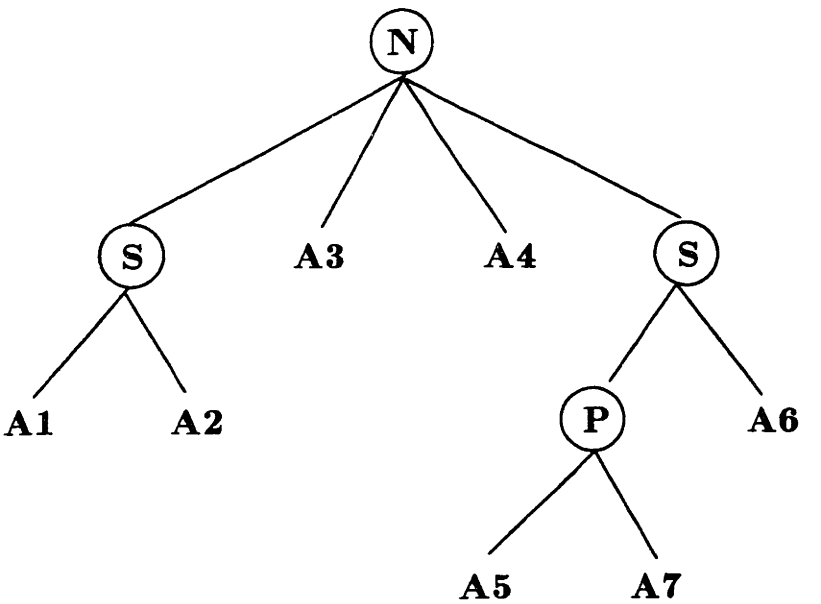


Figure 1.3: Tree Representation of an AON *PERT* Network

effort to reduce the *Critical Path* the objective is to find an allocation of the available resources to activities such that some criteria is optimized (e.g. overall project duration).

Specialization of the resource constrained network problem to the case where each activity requires only a unit quantity of one resource gives rise to the *Job-Shop* type problem with arbitrary precedence relations (Also, in a *Job-Shop Scheduling* problem there is only one unit of each resource available.). Specialization of the problem further to require the network to consist of Parallel linear chains of activities results in the classic *Job-Shop Scheduling* problem. Each of the chains defines a "job." In this case the resources are the machines in a "shop." If each of the jobs visits the machines in the same order then the shop is termed a *Flow Shop*; if the jobs are free to visit the machines in any order then the shop is termed a *General Shop.* In this

thesis only problems of the *General Job-Shop* variety will be considered. Note that specifying a linear order of the activities on each machine fully specifies a solution to the scheduling problem.

The precedence relations used in the *PERT* network representation have the property of transitivity. If activity A_i precedes A_j and A_j precedes A_k then A_i precedes A_k. Explicitly computing the precedences among all nodes in the network is called the transitive closure. In contrast, reducing the set of precedence relations to a minimum by eliminating redundant precedence relations is called the transitive reduction.

Closely related to the *PERT* network representation for a *Job-Shop Scheduling* problem is the *Disjunctive Graph* representation (Figure 1.4). The directed edges of the disjunctive graph correspond exactly to the precedence relations of the *PERT* network of Figure 1.1. The *Disjunctions* are unresolved precedence relations, their orientations are yet to be determined. The disjunctions are typically shown as dotted lines or as a pair of parallel directed edges each with an opposite orientation. When taken together, the precedence relations and disjunctions among a set of activities which all require the same resource form a complete graph. There is either a precedence relation or a disjunction between each pair of activities in this set. For the network shown, grouping the activities according to resource required results in the following sets: $\{A_1, A_4\}$, $\{A_2, A_5, A_7\}$, and $\{A_3, A_6\}$.

When the scheduling algorithm which processes this graph is finished, each of the disjunctions will have been "settled" - one of the two possible precedence relations will have been chosen. The precedence relations will then form complete graphs on each set of activities requiring the same resource. Performing a transitive reduction on the final graph results in a linear chain of precedence relations through each of

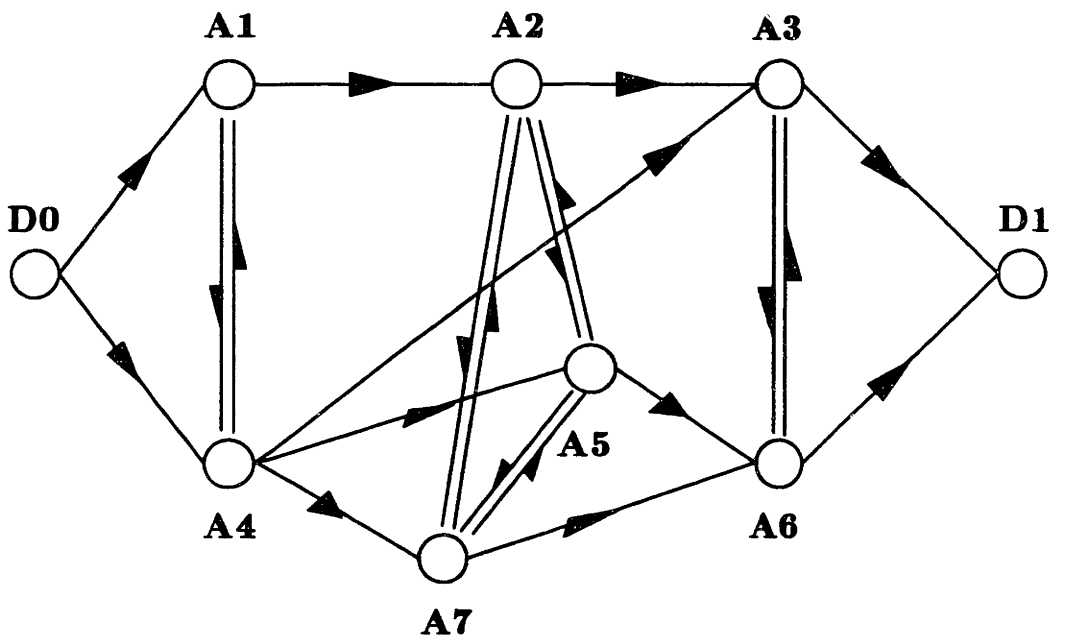


Figure 1.4: Disjunctive Graph

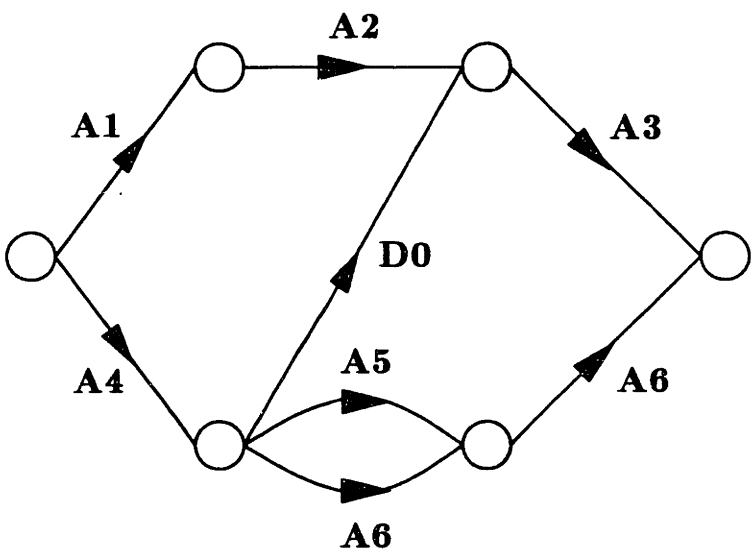


Figure 1.5: AOE Pert Network

these sets. Thus a linear ordering of activities is specified at each machine. Note that as the algorithm proceeds, the graph becomes a progressively more ordered partially ordered set.

There exists a dual formulation of the *PERT* network termed the AOE (**A**ctivity **O**n **E**dge) *PERT* network (Figure 1.5). The directed edges in these networks correspond to activities instead of precedence relations. The nodes correspond to "project milestones" or the event that the preceding activities are finished being processed. More care must be taken in formulating the AOE style network than a AON networks because in some cases dummy activities must be introduced to achieve the desired precedence relations. In particular, the AOE equivalent of the AON N subgraph requires such a dummy activity (see Figure 1.6).

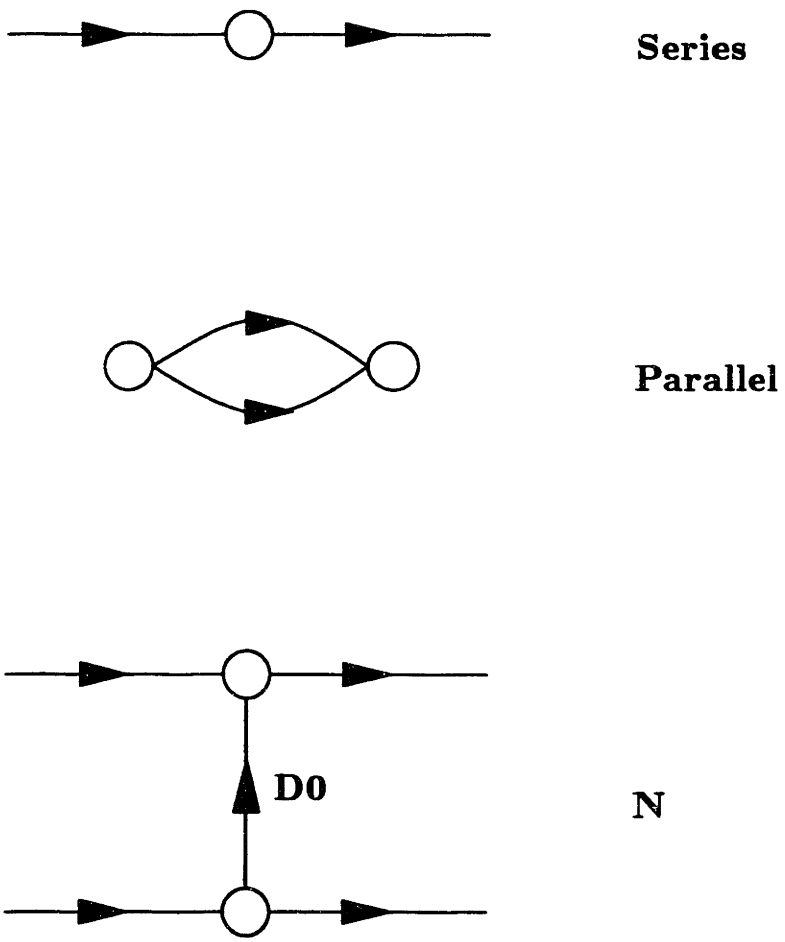


Figure 1.6: AOE Series, Parallel and N Subgraphs

1.2 Specific Problem Description

The *General Job-Shop* scheduling problem is an abstraction of a discrete-parts, batch or lot production process. Typically, orders exist for a number n of jobs (commodities) to be produced, each of which is to be processed on all of m available machines (processing facilities). In general, each job may take a different path through the shop. The goal is to specify an ordering for the jobs at each machine such that both the technological constraints of the jobs are satisfied and a measure of schedule goodness is maximized. An instance of one of these problems can be specified using the notation of Rinnooy Kan [73] as the four-tuple $\{n, m, \text{Special-Constraints}, \text{Objective-Criteria}\}$. In this notation, n is the number of jobs and m is the number of machines. Special-Constraints such as G (general flow of jobs through the shop) and d (each job has an arrival date and a due date) define the problem's structure while Objective-Criteria is the measure of optimality (e.g. $L_{max} \equiv$ *Minimum Maximum Lateness*) used to compare schedules.

It is assumed that only short term detailed scheduling is to be done. The manufacturing facility has a fixed amount of equipment as determined by some method of long range planning. And that short range aggregate planning has been done by a system such as *MRP* resulting in a product mix specification consistent with the goals and limitations of the facility. *MRP* (Material Requirement Planning) is an inventory control system which starts by forecasting end-item demand, then develops a master production schedule, and then explodes this schedule using a bill of materials. These gross requirements are then compared to existing inventory levels to determine net requirements. This static specification is to be contrasted with the dynamic case where jobs arrive unpredictably.

In addition to this basic scenario, the following limiting restrictions are assumed to apply. Each job is composed of a sequence of individual activities the partial ordering of which is specified by precedence relations. Each activity requires a certain fixed amount of time for its processing and these processing times are sequence independent. There is no pre-emption of activities from machines (i.e. once an activity begins being processed its processing continues until completion). Each job is assumed to have an arrival date and a due date. All machines are independent and can process only one activity at any given time.

This family of problems has been studied extensively and a variety of types of algorithms have been developed for it. Many forms of the problem are known to be NP-hard [73]. In other words, it is widely believed that no algorithm exists (although one could theoretically exist) which is capable of solving these or any other NP-hard problems in an amount of time which is some polynomial function of the size of the input data. Consequently, exact methods were developed for relatively small problems, while heuristic methods were developed for large problems. One thread which is common to most of these algorithms is that they use the addition of a time ordering or precedence constraint as a method of pruning the solution space. Using such constraints allows a partial (or complete) solution (schedule) to be represented as a partially ordered set of activities.

For example, this time ordering is explicit in the case of algorithms which branch and bound on disjunctions or perform relaxation by interchanging the order of activities along a *Critical Path.* In the case of algorithms which run in a simulation type environment the time orderings of the activities are fixed in a topological order.

An assumption underlying the use of these precedence constraints is that the sets of sequences representable as partially ordered sets is rich in the sense that local rules

can make progress toward a solution without the need for excessive backtracking. It seems that under certain circumstances this is the case while in others it is not, as evidenced by the narrow success of most algorithms.

Fox and Kempf [34], [35] propose a provably complete language of sequences in which arbitrary sequencing constraints can be expressed succinctly using the precedence relation $\succ$, the negation operator *not* combined with the logical connectives *and* and *or*. Actually, a *nand* or *nor* connective along with the precedence relation $\succ$ would be sufficient. Fox and Kempf used the language in connection with the offline generation of alternative "opportunistic" schedules for later use on line. Their objective was to generate offline sets of schedules which will offer maximum opportunity to accommodate for uncertainty in the order of parts arrival at an assembly station. Such a language allows more freedom in the selection and utilization of constraints when designing algorithms for sequencing problems.

A common assumption is that the goal is to find a provably "optimal" sequence or a quantifiably close approximation. This is a narrow view as it focuses only upon some cost and neglects the other aspects such as robustness; perhaps a family of good schedules could be found such that small variations in shop operation or problem specification could be accommodated. The constraints that are typically used to build and describe solutions usually preclude the specification of such families.

In this thesis two heuristics will be developed. The first one (H1/CT) is iterative and based on Monte Carlo Sampling. Information gained from the generation of sample schedules is used to prune the search space via some given set of pruning constraints. Under certain objective criteria, H1/CT is observed to converge upon a family of *good* schedules. Choosing a schedule from the "middle" of this family should result in a schedule which is insensitive to small variations in shop conditions

(other reasonably good alternative schedules can be easily reached from this one). Another use for heuristic H1/CT is to test whether some set of pruning constraints is capable of selectively eliminating undesirable schedules from the search space.

Heuristic H2 is a one pass procedure which is tuned to converge upon a region of the search space containing a high density of low *Minimum Maximum Lateness* schedules. In the chosen representation this region is a "tube" connecting the origin and the far corner of a cartesian space.

1.3 Literature Review

In order to compare previous work with the two heuristics developed in this thesis and to describe the various algorithms uniformly, it is now necessary to introduce the concept of a cartesian completion space. Related to a job shop scheduling problem is a cartesian space (first introduced by Akers [5] and later used by Hardgrave and Nemhauser [52]) in which position along an axis corresponds to the degree of completion of a job. Shown in Figure 1.7 is a two dimensional job space representation of a two job problem. Each of the coordinate axes corresponds to a "job." Each job consists of a linear chain of activities and is drawn in the AOE (Activity On Edge) style. The length of each edge corresponds to the processing time of the associated activity. Any point along the length of a job defines a state of completion of the job. A trajectory through this space corresponds to a schedule, which starts from a state of zero completion at the origin and goes to a state with all jobs completed at the far corner.

Thus, any point in the space corresponds to each of the jobs being in a partially completed state. A trajectory through this space can be mapped directly to a sched-

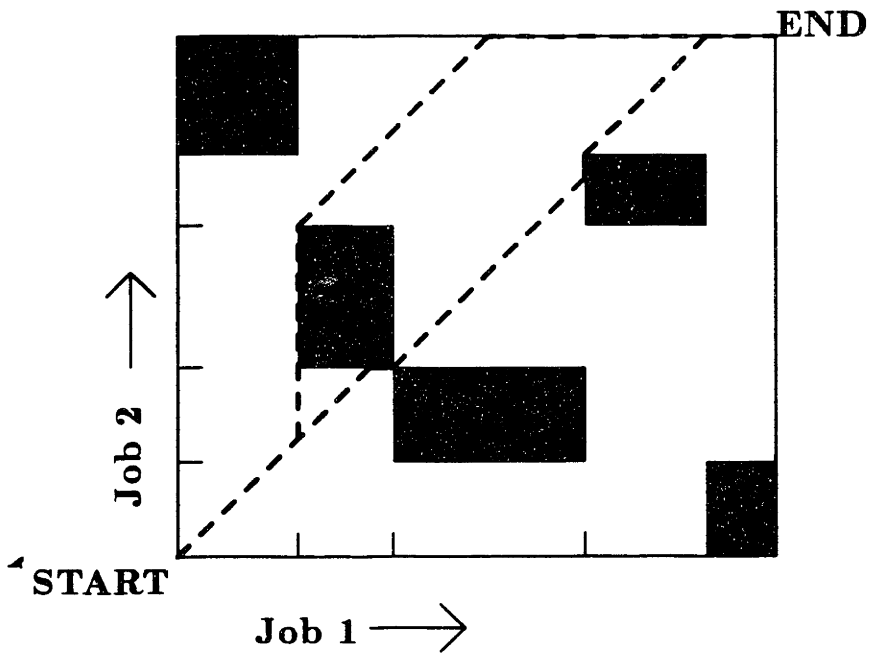


Figure 1.7: Example space associated with a 2 job problem

ule. When a trajectory traverses only one coordinate direction only one job is being processed. Similarly, when a trajectory is advancing in multiple dimensions the jobs are being co-processed. The processing time for each piecewise linear segment of the trajectory can be found by finding the maximum of the projections of it on to the coordinate axes. Total schedule completion time is then the summation of these maxima over the piecewise linear segments.

The shaded regions are infeasible states of completion due to resource constraints. Obstacles include the area up to but not including the lines (2D case) which form their boundaries. For example, the first activity of Job 2 and the last activity of Job 1 require the same resource which is available in quantity 1.

A survey of some of the established methods used to generate solutions for the

problem of *Job-Shop Scheduling* follow. When appropriate, a brief description of the algorithm in the context of a completion space will be given.

Since the problem is NP-hard [73], research has focused in two general areas, exact (and usually computationally intense) methods to find "optimal" solutions to small problems and heuristic (and usually computationally simple) methods to find approximately optimal solutions to large problems. The heuristics developed in this thesis are of intermediate complexity and thus fall between the two areas. Branch and bound methods have traditionally been used to find exact solutions. The performance of the branch and bound method depends heavily upon the strength of the lower bound used to estimate the cost of completing a partial solution. If the bound is exact (not an estimate) then an optimal solution can be found directly. If the lower bound is inexact then some barren branches will be explored even though they do not lead to an optimal solution. If the bound used is not a lower bound but an approximation accurate to some known accuracy, then an approximately optimal solution (within this known accuracy) can be found. Surveys of categorization and complexity of algorithms and problems over various objective criteria can be found in [9], [24], [26], [49], [67], [73] and [77].

Some of the steps of an example branch and bound algorithm can be visualized in a 2D completion space as follows. Assume the algorithm is to branch and bound on disjunctions along the current critical path. The first step is to find the critical path. Start a trajectory at the origin of the space and proceed diagonally (along a 45 degree line) (ignoring obstacles along the way) or as near to this direction as possible until the state of total completion is reached. This trajectory corresponds to the schedule (possibly infeasible) which would result by assuming that there are no resource limitations and that each activity is scheduled to start as soon as possible

consistent with the given precedence constraints. The second step is to locate the disjunctions along this critical path. Each intersection of the trajectory with the previously ignored obstacles corresponds to a disjunction along the critical path. Next, tentatively "settle" one of the disjunctions (each disjunction corresponds to an obstacle). This "settling" of a disjunction (addition of a precedence relation) is equivalent to requiring all subsequent trajectories to detour around the obstacle in a given direction. A new critical path (trajectory) is then found and the process repeated. It is usually necessary at some point to backtrack and try some of the alternative detours.

One of the simplest and still widely used tools to aid a human scheduler is the Gantt Chart. This chart is a graphical display of how resources are allocated to activities over time. This technique is puzzle-like because the scheduler has to rearrange the boxes representing the processing of various activities by the resources until a good enough schedule is obtained. Schedule quality reflects the skill and experience of the scheduler.

Time is represented explicitly along the x-axis and either precedence constraints within a job or resource limitations are enforced by the non-overlapping of the boxes depending whether the y-axis corresponds to resources or jobs respectively. The human scheduler must take care of the other constraint.

For an example of such a chart where the resource limitations are enforced graphically see Figure 1.8. In this chart each row of boxes (activities) corresponds to operations scheduled to be done on an individual machine. The numbers in the boxes are an arbitrary numbering of the activities. Job 1 is composed of activities 1 through 10; job 2 is composed of activities 11 through 20 and so on. The empty space between boxes is idle time inserted between activities so that the precedence

relations will not be violated. The schedule shown is one feasible solution for the notorious {10, 10, G, Makespan} problem found in [47]. The makespan for this example solution is 985.

Very small problems can be solved by using brute force to enumerate all the possible sequences. Slightly larger problems can be solved by enumerating the dominant members of equivalence classes of sequences. Erschler *et al* [32] take a "pyramidal" approach specific to single machine problems. Akers and Friedman [5] discuss how to reduce the set of schedules using only non-numerical means. Giffler and Thompson [46] exploit dominance by enumerating all of the members of the set of so called *Active Schedules.* The defining characteristic of an *Active Schedule* is that no activity can be re-scheduled to start at an earlier time without forcing some other activity to start at a later time. The set of all *Active Schedules* is smaller than the related set of *Non-Delay Schedules.* Although the size of the set of schedules is reduced, it is still not small enough to allow practical enumeration of a moderate {6, 6, , } size problem (6 jobs each 6 activities long). Hardgrave and Nemhauser [52] show how trajectories corresponding to the Active Schedules map into the completion state space.

Much effort has been put into the generation and evaluation of priority dispatching rules [24]. These rules are used to rank the jobs waiting in a queue for processing. This type of scheme is attractive because only a small amount of locally available information is needed to make the queueing decisions and changes in the shop or problem specification have no effect on the partial solution generated so far. These rules are usually based on some job or queue attribute believed to be relevant to the generation of good schedules. Some example rules are the Shortest Processing Time (SPT) rule, the First Come First Served rule, the Least Work in Next Queue rule, and the Random rule (see Section 5.4 for definitions of these and other rules).

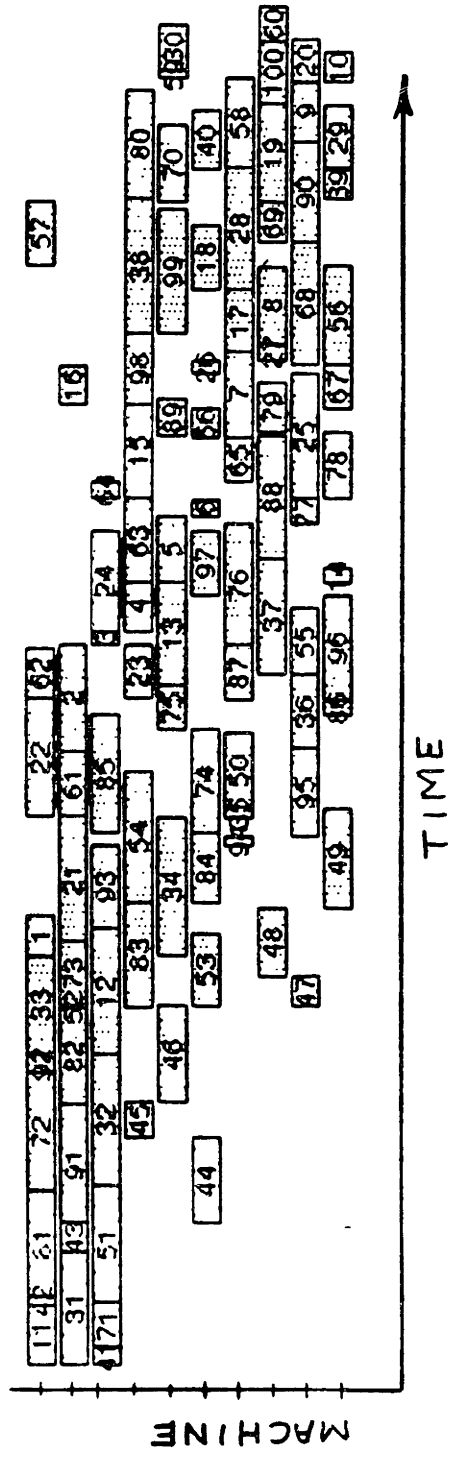


Figure 1.8: Solution shown in Gantt Chart form for a 10 job 10 machine problem.

The average performance of some of the rules, most notably SPT, is better than the Random rule; however the performance of a particular rule on a given problem instance is unpredictable.

In addition to being used directly in a running shop, the priority dispatch rules have been used as the basis of simulation studies. Moore and Wilson [65] and Weeks and Fryer [85] have evaluated the relative performance of sets of rules. Nugent [66] evaluated the merits of adding some randomness to the dispatch rules and Bunnag [15] used computer search to determine what weighted combination of rules gives good performance for a typical problem. Additionally, much work has been done to provide simulation environments [20], [31], [71] and [83], and modeling systems [12], [37] and [38].

Priority dispatching can be mapped into the completion space as follows. Start a trajectory at the origin and proceed diagonally until the corner of an obstacle is encountered. Use a dispatch rule to decide which way to detour around the obstacle. Continue is this fashion until the state of total completion is reached. Following this procedure generates one member of the class of the so called *Non-Delay Schedules.* In other words, if some machine is idle and its queue is not empty, then one activity will be selected via the dispatch rule and started immediately. This is to be contrasted with the class of Active Schedules in which it is possible to have a schedule in which a machine remains idle even though its queue is not empty. The supposition is that waiting a little time now might enable one to avoid excessive idle time later.

Various randomized searching strategies have been explored. The strategy is simple, sample schedules are generated and then compared with the best schedule generated so far. Sampling with equal probability from the set of all schedules does not give good results, consequently efforts in this area have concentrated on biasing

the sampling procedure so as to increase the probability of generating better schedules (hopefully without excluding the best schedules). One way to bias the procedure is to sample from the class of non-delay schedules. This is the same type of schedule as generated by using the priority dispatch rules, and could in fact be generated by using the Random dispatch rule. Here, as many activities are processed in parallel as possible until some conflict arises (a queue with more than one element in it). This conflict is then resolved by choosing at random from among the elements of the queue. This sampling scheme increases the probability of generating schedules which have fewer conflicts to resolve.

Another alternative, taken by Giffler and Thompson [47], is to sample from the set of Active Schedules. In an Active Schedule, an activity's processing may be delayed even though it and the necessary resources are immediately available. The activity is delayed in favor of some other job which will become available later. Another characteristic of Active Schedules is that no activity can be rescheduled to start at an earlier time without forcing some other activity to start at a later time. Choosing at random when there is more than one choice again increases the probability of generating schedules with fewer conflicts to resolve, unfortunately in numerical problems (as opposed to those with unity processing times) this biasing does not necessarily work out to advantage. And although given enough samples this method is guaranteed to come up with the optimal solution, the computational cost is prohibitive as problem size increases. Also, in most cases, the odds of producing a superior schedule are greater when sampling from the set of non-delay schedules than when sampling from the set of Active Schedules.

Yet another alternative is to bias the sampling based on some quality or measure thought to be relevant to a good schedule. Toward this end Nugent [66] prioritized

the activities in a queue using a dispatch rule and then applied a geometrically decreasing probability to choosing successive elements in the queue. It was found that the distributions of schedules obtained via these randomized dispatch rules were improved from those of purely random selection and a significant fraction of the samples were better than those for the unrandomized dispatch rule. The conclusion was that a fixed amount of randomization helps, but that the optimal amount of randomization varies among problem instances.

A Bayesian approach has been used to estimate the likelihood of obtaining a better schedule with the next sample. In this approach, some *a priori* distribution of schedule attributes is chosen. Then information gained by sampling is used to update the assumed distribution. If the assumed distribution is general enough it will asymptotically converge to the actual distribution. This updated distribution can be used to estimate the probability of obtaining a better sample than the best found so far. One of the problems with this type of approach is finding a computationally tractable *a priori* distribution. According to Rinnooy Kan [73] this method is only of academic interest because it depends on asymptotic results and is only applicable to structured situations.

Some less structured approaches to solution generation (modification) come under the headings of Relaxation [1], Interchange, Neighborhood Search [56] and Annealing. In these methods, some (possibly random) change is made in an existing solution. This produces a new schedule which is in some sense close to (a neighbor of the original schedule in some space) the initial solution. Perhaps this change is made by interchanging the order of two activities which lie along the *Critical Path* of a PERT network. The objective function of interest is then evaluated on this new schedule; if there is an improvement, then the modified schedule is accepted (Relaxation) and

the process is repeated. If the value of the objective function is degraded then the schedule is accepted (Annealing [55],[64]) with some probability depending upon the annealing *schedule*. If the *annealing schedule* is selected correctly, then the algorithm will not get stuck in a local minimum.

The relaxation process can be visualized in the completion space by starting with a trajectory corresponding to any complete feasible schedule. Then, using a modification rule, jump a portion of the trajectory over one or more obstacles. Interchanging the order of two activities along a critical path [63] corresponds to jumping over a single obstacle.

A variety of search techniques have evolved over the years. These methods can be either exact or heuristic depending the strictness of the bounds used in the various branching decisions and whether or not one is content with the best schedule found so far. See [11] for an overview of search strategies. In the Operations Research literature, search algorithms can be found under the headings of Branch and Bound and Implicit Enumeration. These searches are usually carried out in either a depth first mode [10] or beam search mode [39] as complexity of the problem usually precludes using breadth first mode. These algorithms usually based either on settling disjunctions along a critical path or on resolving which activity to schedule next during Active Schedule generation. Some examples of application of the branch and bound technique can be found in [17], [19], [48].

The main difficulty with these approaches is that it is hard to find a lower bound which on one hand is strict or tight enough to prune the search space effectively at an early stage and on the other hand is not too computationally expensive. Consequently, the majority of work in this area has been in the development of stricter and more efficient bounds. In the extreme case the computation of the bounds themselves

is NP-hard [58]. Baker [8] studied the tradeoffs of various bounds used in flow-shop branch-and-bound and elimination algorithms. Brooks [13] used a lower bound as a decision rule for developing a single solution. Work has also been done using a relaxed lagrangian transformation of the problem to generate stricter lower bounds [33]. Picard [70] used a related time dependent traveling salesman problem to compute bounds. The difficulty in obtaining tight lower bounds may in part be due to choice of type of constraint to add. In general, the overall run time of this type of algorithm is not predictable.

Many operations researchers have approached the problem from a more theoretical point of view. Usually an optimal or near optimal solution is sought using a simplified model of actual shop conditions. One technique, dynamic programming used by Schrage and Baker [76] on a one machine problem, recursively decomposes the problem into subproblems, solves each unique subproblem once, then selectively recombines the solutions. This technique is limited to small problems. A dynamic programming approach suitable for two job problems presented in the completion space can be found in [80]. Here the subproblems correspond to trajectory segments between the outer corners of the obstacles which can be connected with a straight line. Lawler [59] used a series parallel decomposition on a single processor problem. Another type of decomposition is hierarchical. Problems are decomposed into different levels of some hierarchy (e.g. capacity planning, long range, short range, detailed scheduling [4] and Gershwin *et al* [42], [43], [44], [45], have worked in making an intermediate level of a decomposition dynamically adapt to changing (breakdown/repair) shop conditions. Lipton [60], [61] used a hierarchical approach with rescheduling while Dempster [28] used a two stage approach.

The dynamic programming approach can be interpreted as a search through a

graph superposed on the completion space. Consider a scheduling problem in which all the activities have integer processing times. Then the start and finish times of each activity must occur at integer times regardless of the schedule. At all other times a number of the jobs are being co-processed. Then the set of all possible combinations of start and stop states of activities is included in a regular lattice on the space at unit spacing. The possible transitions from lattice point to lattice point are defined by allowing an increment of unity in one or multiple coordinate directions (no lattice points can exist within an obstacle). This corresponds to processing one or multiple jobs for one time unit. The dynamic programming problem is to find the least cost path to each lattice point. This is accomplished by formulating the cost of the path to a given lattice point in terms of the costs of the paths to other immediately reachable lattice points which are closer to the origin. Davis [25], [27] used this approach to transform a PERT [86] network problem into a shortest route problem (with a combinatorial number of cities).

Relatively recently Petri Nets have started being used to model and analyze scheduling systems [75], [18], [74]. These Nets were originally developed [69], [72] with the intent of modeling interlocking concurrent computation systems in a time independent manner. In the Petri Net representation both resource limitations and sequencing constraints can be treated uniformly, although some modifications need to be made to represent time considerations. These nets or their associated matrices can be used to derive certain invariants and characteristics of the system. These nets are equivalent to Vector Addition Systems. In a Petri Net representation of a *Job-Shop* scheduling problem, the vectors correspond to the possible trajectory segments in the completion space representation. If the scheduling problem being represented has only integer processing times, then the vectors being added combine to point to

the lattice points defined in the previous paragraph.

An exact polynomial-time (in certain problem characteristics) algorithm has recently been improved upon by Sidney and Steiner [78]. This algorithm is applicable to the total weighted completion time problem, the total discounted cost problem, the least-cost fault detection problem, and the jump number problem. In this algorithm, dynamic programming along with a partial-order decomposition allows exact solution of sequencing problems with worst case complexity $n^{(w+1)}$. Where n is the number of activities to be scheduled and w is the Dilworth number associated with the Graph P formed by the n activities (nodes) and precedence relations (edges). The Dilworth number is defined to be equal to the minimum number of chains needed to partition the vertices of P. This algorithm increases the number of sequencing problems (those with a reasonable value of w) that can be solved in practical time; however this class does not include those of the *Job-Shop* variety. For example, a $\{10, 10, , \}$ problem (10 linear chains of activities each 10 activities long) would be of complexity $100^{(10+1)}$. The Dilworth number w is also equal to the dimensionality of the completion space associated with the problem.

Cartesian Representation

Chapter 2

In Chapter 2 I will extend the representation introduced in [6] to provide a framework on which to construct subsequent results. Then, in Chapter 3, I will develop within this framework an iterative heuristic (H1/CT) based on Monte Carlo Sampling. Results obtained using heuristic H1/CT serve to justify the formulation of heuristic H2. This second heuristic (H2) is presented in detail in Chapter 4. In Chapter 5 the performance of heuristic H2 will be documented. It's performance will be tested under various objective criteria in relation to the *Priority Dispatch Rules* and in relation to some biased search techniques. In Chapter 6 there will be conclusions about heuristics H1/CT and H2, and about the ideas of adjacency of solutions and the types of constraints used to prune the search spaces.

2.1 Cartesian Completion Space

The idea of using a cartesian completion space to represent both the resource limitations and the precedence constraints of scheduling problems was introduced by Akers [6]. Subsequently, an algorithm was developed for two job (2D space) problems by

Szwarc [80] and an attempt was made to extend the results to the 3D case. The idea was to first optimally solve each of the individual 2D problems defined on three orthogonal faces of the 3D space. Then, the individual solutions were to be combined to yield an optimal trajectory through the 3 space. The problem with this approach is that only 2 projections are necessary to uniquely describe a trajectory in 3 space. There is nothing in the formulation which guarantees the consistency of the third subproblem solution with the other two.

Hardgrave and Nemhauser [52] suggested using a set of extremal trajectories to define a region of the space within which the optimal solution lies. In the 2D case, these trajectories are found by starting a pair of trajectories from the origin and proceeding "diagonally" through the space. When trajectory 1 encounters an obstacle it always branches in coordinate direction 1. When trajectory 2 encounters an obstacle it always branches in coordinate direction 2. These two trajectories define a "cone" shaped region with the apex at the origin. This procedure can then be repeated starting from the state of total completion and working backwards. The optimal solution then lies within the intersection of these two regions.

This procedure is well defined for the 2D case, but is unmanageable for higher dimensional problems. In the 2D case, the boundaries of the regions are defined by easily generated trajectories while in the higher dimensional cases the regions are defined by $n-1$ dimensional hyper-surfaces. In Appendix 1 is the derivation of the volume fraction of the space enclosed by such a generalized "cone." The bottom line is that even though the amount of space which remains to be searched is greatly reduced, the complexity of the remaining space is still formidable.

2.2 Capabilities and Limitations of the Representation

The completion state space incorporates both the resource constraints and the precedence constraints of the *Job-Shop Scheduling* problem uniformly. And although efficient results can be obtained in 2D problems, its usefulness in more complex problems is to aid intuition and qualitative reasoning.

One of the limitations of this state space is that time is not explicitly represented. This makes it difficult to merge into the space time based constraints such as arrival dates, due dates and time specific availability of equipment. In order to utilize this type of constraint, some processing of the space needs to be done such as finding the minimum cost path to each interesting point in the space thereby establishing a unique time value there. More generally, such a value can be established for any path dependent function such as one which includes the effect of sequence dependent set-up times.

For the constraints which are represented (precedence and resource limitations) the space is a true state space. Each point in the space corresponds uniquely to the degrees of completion of the various jobs. Therefore, all of the techniques applicable to such a formulation can be used. For example, any function with a unique value for some state of completion can be immediately defined at each point in the space. An example of this would be in-process inventory costs.

When an arbitrary configuration of precedence relations is allowed, some corners of the space are truncated. This is because the partial ordering of activities defined by the precedence relations excludes certain states of completion from being feasible.

The number of lines emanating from the leading corner of an obstacle corresponds

to the number of alternative activities which could be scheduled at this point. For example, consider a trajectory encountering the leading corner of an obstacle in a two dimensional problem. At this point, the trajectory can branch in one of the two possible directions around the obstacle. Each of these directions corresponds to choosing one of the two activities in a queue.

2.3 Obstacles in N-Space

In the classic *Job-Shop Scheduling Problem*, it is assumed that resource availability is limited to one machine of each type. A resource which is available in quantity j gives rise to a basic obstacle of dimensionality $j + 1$ in the space. In a two dimensional (two job) problem with unit resource availability the resulting obstacles are two dimensional rectangles (See the top left box of Figure 2.1). In a three dimensional (three job) problem with unit resource availability the obstacles are defined as the union of individual 2D obstacles (found on the 2D faces of the 3 space) which have been projected through the remaining third dimension thus forming rectangular bars (See the top center box of Figure 2.1). In general, basic $j + 1$ dimensional obstacles defined in the unique $j+1$ dimensional subspaces are projected through the remaining $n - (j + 1)$ dimensions to form the obstacles.

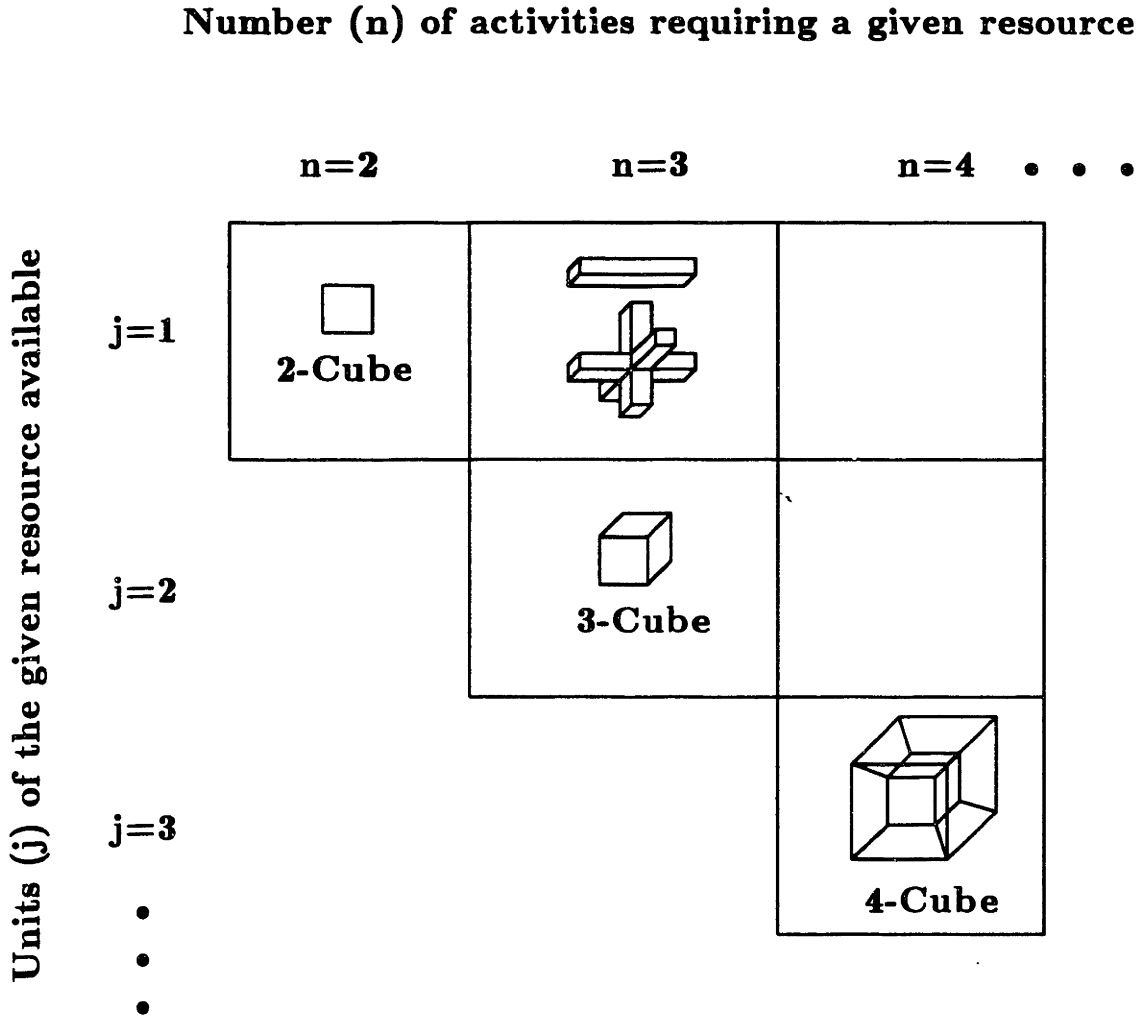


Figure 2.1: Types of obstacles resulting from different resource constraints

Heuristic H1/CT

Chapter 3

In this Chapter I will present heuristic H1/CT. This heuristic is built upon Monte Carlo sampling (see Appendix C). Based on a set of samples taken, the search space is pruned such that undesirable schedules are eliminated thereby yielding a more favorable distribution for subsequent sampling. This heuristic has three uses. The first use is to generate a single solution. This is accomplished by keeping the best schedule found among all the samples taken. The second use is to prune down the search space thus leaving a reduced problem with more desirable properties. If the size of the resultant space is sufficiently small, then it may be possible to employ an exact method to search it. Alternatively, choosing some schedule from the *middle* of the remaining space should yield a *robust* schedule. Presumably, other schedules of *good* quality are *near* the chosen one. Therefore, if the chosen schedule gets slightly off track, then it is likely that it's quality will not degrade drastically. And the third use is to test whether some set of constraints is capable of pruning the space effectively (hence the /CT in the name for Cconstraint Tester). If the set of constraints used is appropriate for the problem, then the distributions of schedule quality seen during

successive iterations should improve. If the distributions do not improve, then it is probably not worthwhile pursuing the use of the given set of constraints. Thus one could avoid, for example, the frustration of developing a branching indicator and a lower bound for a branch and bound algorithm based on this set of constraints which is unlikely to perform well.

3.1 Definition of Heuristic H1/CT

This heuristic is defined as follows:

Step 1. Generate a collection of pruning constraints.

Step 2. Generate k new sample schedules.

Step 3. Pick the l best samples from all samples generated.

Step 4. Apply as many of the pruning constraints as possible without removing space containing the l best samples.

Step 5. Repeat Steps 2,3,4 for a fixed number of iterations.

3.2 Explanation and Interpretation of Steps

Steps 1 through 4 will now explained in more detail.

Step 1. The constraints used here are in one to one correspondence with the set of 2D obstacles residing on the 2D faces of the cartesian space. Every trajectory (schedule) in this space detours one of two ways around each obstacle. The constraints to be imposed are of the following form: all subsequent trajectories (schedules) shall detour this way around a certain obstacle. This is equivalent to *settling* a disjunction.

Of course it is possible to use other constraints, but they must meet certain requirements. First of all, it must be possible to determine whether a given constraint is consistent with some set of sample schedules. Otherwise, Step 4 can not be executed. Second, the technique used to generate sample schedules must be able to function when an arbitrary (but technologically consistent) subset of the constraints is imposed (Technologically consistent meaning that some feasible schedule exists within the pruned space). This is necessary to insure that the sample schedule generator will not get stuck in a dead end or *trap* (petri-net terminology).

Step 2. Here the schedules will be generated using a random queueing heuristic in the context of a simulation of a *Job-Shop*. Equal priorities are assigned to all activities queued at a machine and one is chosen at random. In the completion space this is equivalent to starting a trajectory at the origin, traversing diagonally until some obstacle is encountered, then choosing at random which way to detour around it. Here, the number of schedules generated k is typically 100.

One might possibly consider other sampling schemes such as biasing the random priority dispatch rule towards one of the other dispatch rules. Alternatively, one might sample from the set of *Active Schedules* or some biased version thereof.

Step 3. One must decide upon a measure of schedule optimality. Here, the two measures considered are makespan and flowtime. Choosing the l (typically 10) best schedules is simply a matter of selecting the schedules with the best values of the optimality criterion. As it turns out, the choice of optimality criterion has a profound effect on the convergence of this heuristic.

Step 4. For each of the l best schedules found, a binary string was generated with one digit for each of the obstacles. A "0" digit implies detouring one way around the associated obstacle, and a "1" digit implies detouring the other way. These binary

strings were then compared. If the *ith* digits were the same across the ten binary strings, then all ten of the l best schedules detoured the same way around the given obstacle, and subsequent schedules were required to detour around the given obstacle the same way.

Technically, it is necessary to insist upon total agreement among the *ith* digits of the strings to guarantee that the pruned space contains feasible schedules. However, many experiments were performed in which only $9/10ths$ agreement was required, all of which retained feasible schedules.

Some results obtained using heuristic H1/CT serve as a motivation for the way heuristic H2 is implemented. The following experiments were designed to show that a family of good schedules are geometric neighbors in the completion space representation. For each schedule, there is a corresponding trajectory through the space, and there exists a "tube" connecting the origin to the far corner within which the density of good schedules is relatively high. Consequently, heuristic H2 has been designed to converge upon such a "tube" full of *good* schedules.

The addition of a precedence relation to a problem specification corresponds to the removal of a "corner" of the completion space as shown in Figure 3.1. The removed corner contains states of completion which become technologically infeasible when the precedence constraint is added.

A set of precedence relations added to a problem can be interpreted as the specification of an n-dimensional "tube" as follows. In each of the $\binom{n}{2}$ 2D subspaces of the n job problem (form a unique 2D space from each unique pair of jobs) delete the corners corresponding to the relevant precedence relations added. The unpruned region in each of the $\binom{n}{2}$ 2D spaces is the projection of the tube (nD) on to the 2D space.

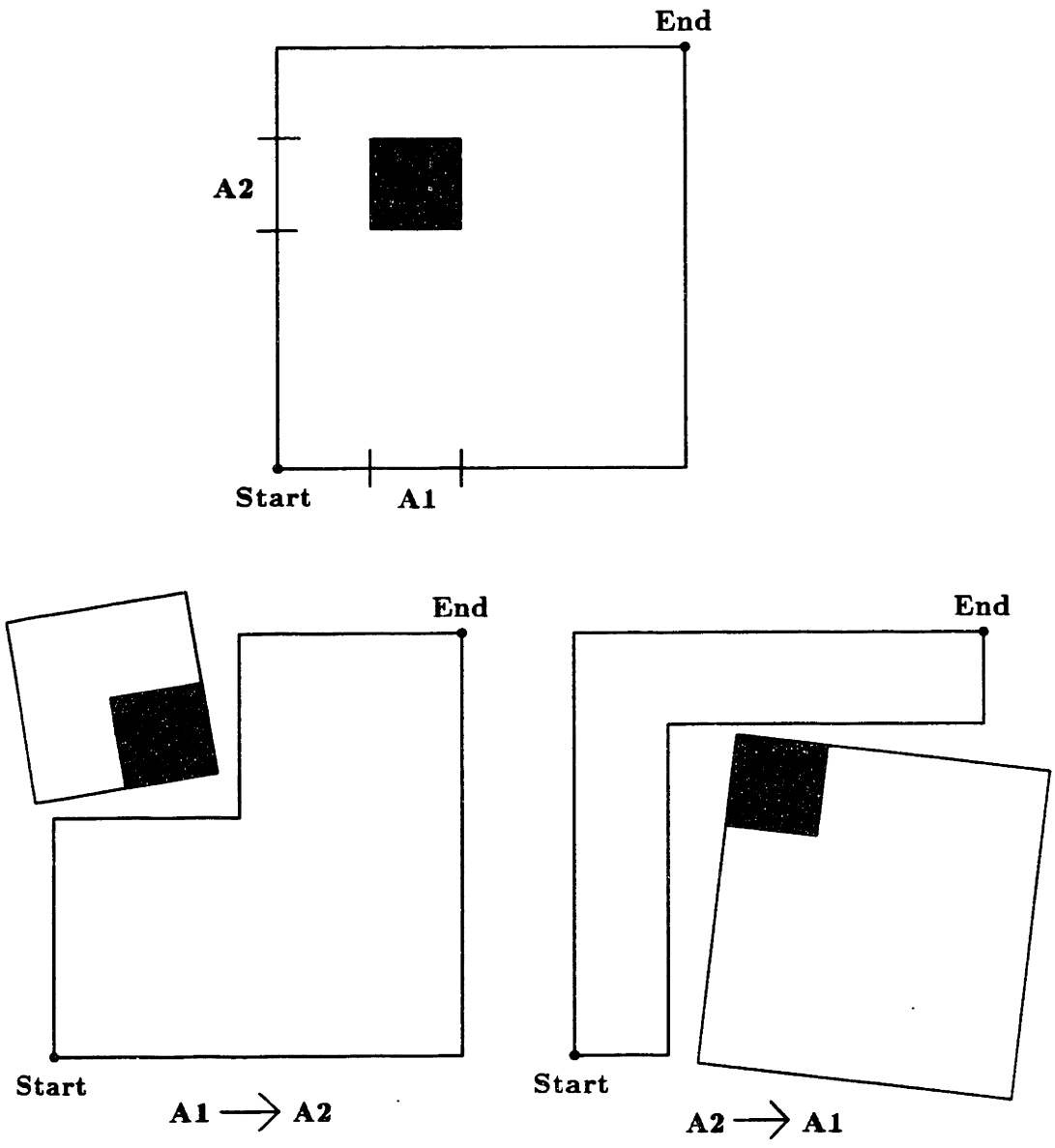


Figure 3.1: Two spaces which could result when the precedence is set

3.3 Experiments

A number of experiments were performed using heuristic H1/CT to determine whether or not the addition of precedence constraints would prune the space effectively under the optimality criteria of makespan and flowtime. There is one precedence constraint in the constraint set for each pair of activities which require the same resource for processing.

The experiment is set up as follows: First a set of schedules is generated using the *Random* priority dispatch rule in an event based *Job Shop* simulation. For these experiments the number of sample schedules generated per iteration was 100. Next, the makespan is computed for each of these 100 sample schedules. They are then sorted by increasing value of makespan (for plotting purposes). The best 10 schedules (those with minimal values of makespan) were then selected. Precedence relations which were common to all 10 schedules were then used to prune the space. The sampling scheme was then repeated within the pruned space lumping the 10 best samples in with the new samples.

The distributions of schedule makespans (obtained using *Random* priority dispatching) for the original problem and for the pruned versions of it are displayed using quantile plots. In these plots a separate curve is formed for each set of 100 samples taken. The value of the objective function for each member of the set is plotted versus its position in the sorted data set. The quantile is closely related to the percentile. If the value of the objective function is x at the $30th$ quantile then 30% of the samples have objective function values $\leq x$. Results were obtained for a $\{6, 6, G, \}$ problem [47] using the objective criteria of makespan and flowtime. Three curves are plotted in Figure 3.2.

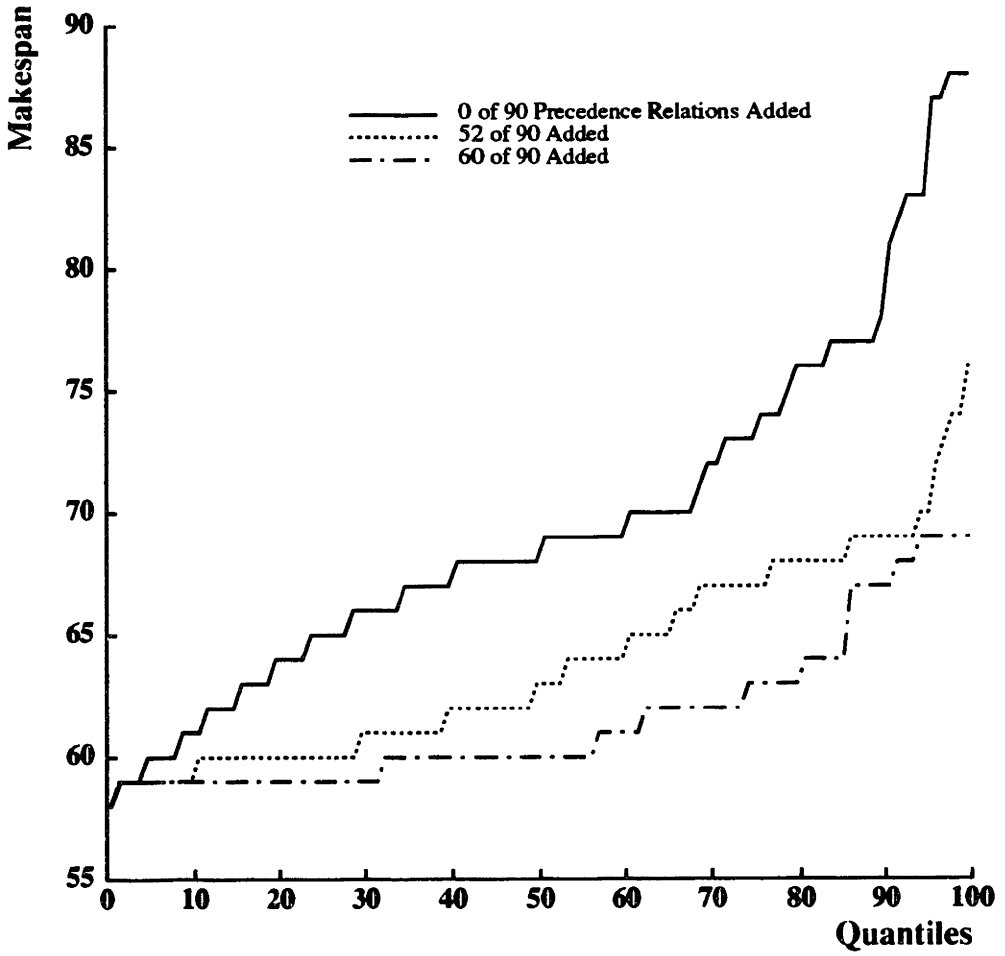


Figure 3.2: H1/CT Applied to a 6 job 6 machine problem

The first (top) curve represents a random sampling before any additional precedence constraints have been added to the problem. The second curve represents a random sampling after the precedence constraints common to the best 10 of the first 100 samples have been used to prune the space. And the third curve represents a random sampling after the precedence constraints common to the best 10 schedules of the second sample set have also been added.

The distributions indicate that a class of *good* schedules have certain precedence relations in common for this $\{6, 6, G, \text{Makespan}\}$ problem. This suggests that an algorithm which successively reduced the search space by adding selected precedence relations could successfully prune regions of the search space characteristic to undesirable schedules while keeping regions favorable to *good* schedules.

This shows that trajectories corresponding to a group of *good* schedules under the makespan criterion lie near one another in the associated nD completion space. This can be seen as follows. Consider any 2D subspace of the space: some of the corners of this 2D space will have been deleted as a result of the precedence relations added leaving a relatively narrow connected region from origin to end. Intersecting the projections of these regions would result in a nD tube which connects the origin to the state of total completion of all jobs (end). All of the schedules in the pruned problem lie within this tube. Note that there will be obstacles adjacent to and intersecting portions of the tube. Shown in Figure 3.4 are the 2D spaces associated with the $\{6, 6, G, \text{Makespan}\}$ problem. The borders of the non-deleted space are outlined with dotted lines. The origin of each subspace is in the lower left, and the projections of the 10 best trajectories upon which the the last pruning operation was based are shown as solid lines.

Heuristic H1/CT was applied to 50 $\{10, 10, G, \text{Makespan}\}$ problems in order to

test it's performance relative to Monte-Carlo sampling. The best solution obtained by H1/CT throughout all of the iterations was compared to the best solution obtained by Monte-Carlo sampling. There were 100 samples taken during each of 5 iterations of H1/CT for a total of 500 samples, and there were 500 samples taken in the Monte-Carlo approach. The average improvement over the Monte-Carlo approach for the 50 problems was 1.0%. If this experiment was repeated on a similar 50 problems, then the average improvement would fall with 95% confidence into the interval 0.3% to 1.7%.

Figure 3.3 shows the results of a repeat of the experiment on the $\{6, 6, G, \}$ problem but under the flowtime criterion. Results were quite different for this case. The precedence relations common to the 10 best sample schedules pruned the space rather indiscriminately. This situation is improved only slightly with the next iteration and the distribution would have been more favorable if no pruning had been done. This suggests that the type of constraint used (adding precedence relations) to prune the search space is not appropriate for this case. In other words, the orientation of a group of precedence relations is insufficient to characterize a set of lesser flowtime schedules (Even though the precedence relations added were consistent with the 10 best flowtime schedules found). That is, the pruning that was done during the first iteration decreased the probability of generating *good* schedules during the subsequent iteration. This suggests that both good and bad schedules were eliminated as a result of the pruning. Therefore, an algorithm which successively reduces the search space by adding selected precedence relations would be unlikely to succeed as the reduced space would include only a small fraction of *good* schedules.

Whether or not heuristic H1/CT converges depends heavily upon the spatial distribution of *good* schedule trajectories through the space. As was seen under the

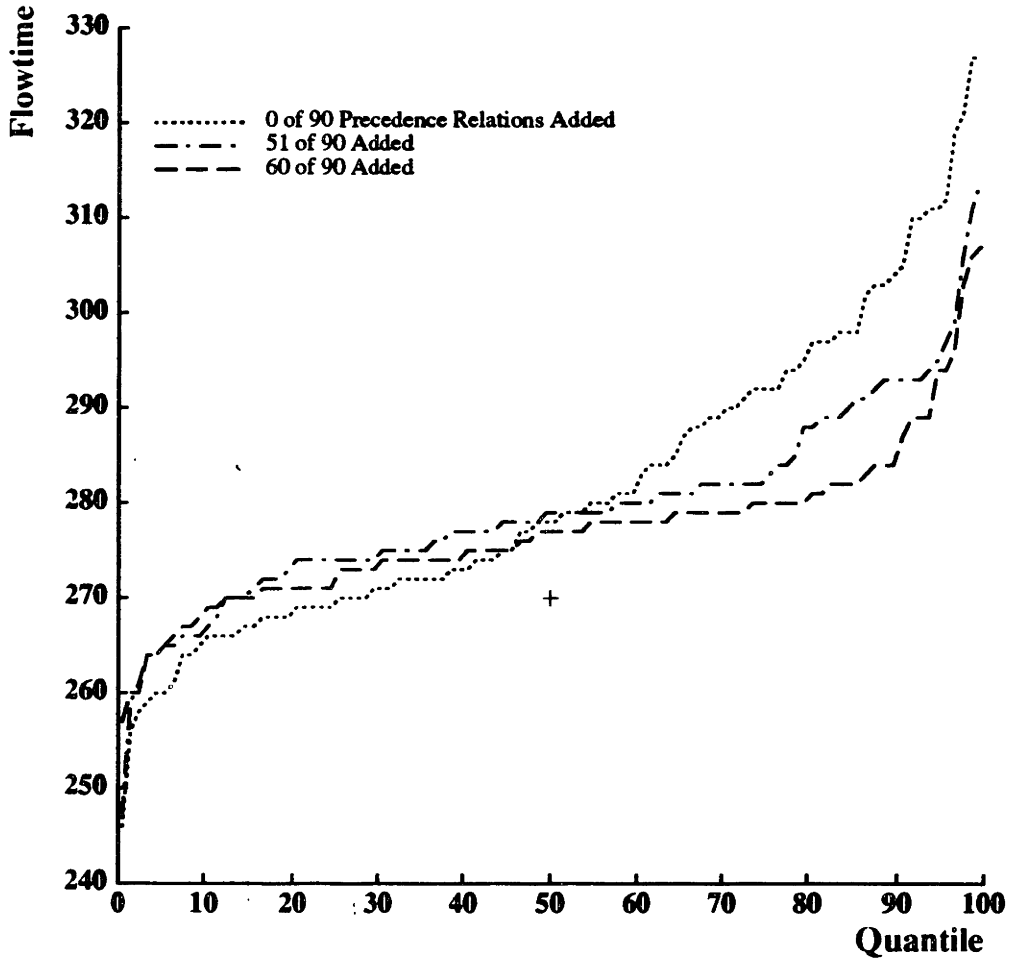


Figure 3.3: H1/CT Applied to a 6 job 6 machine problem

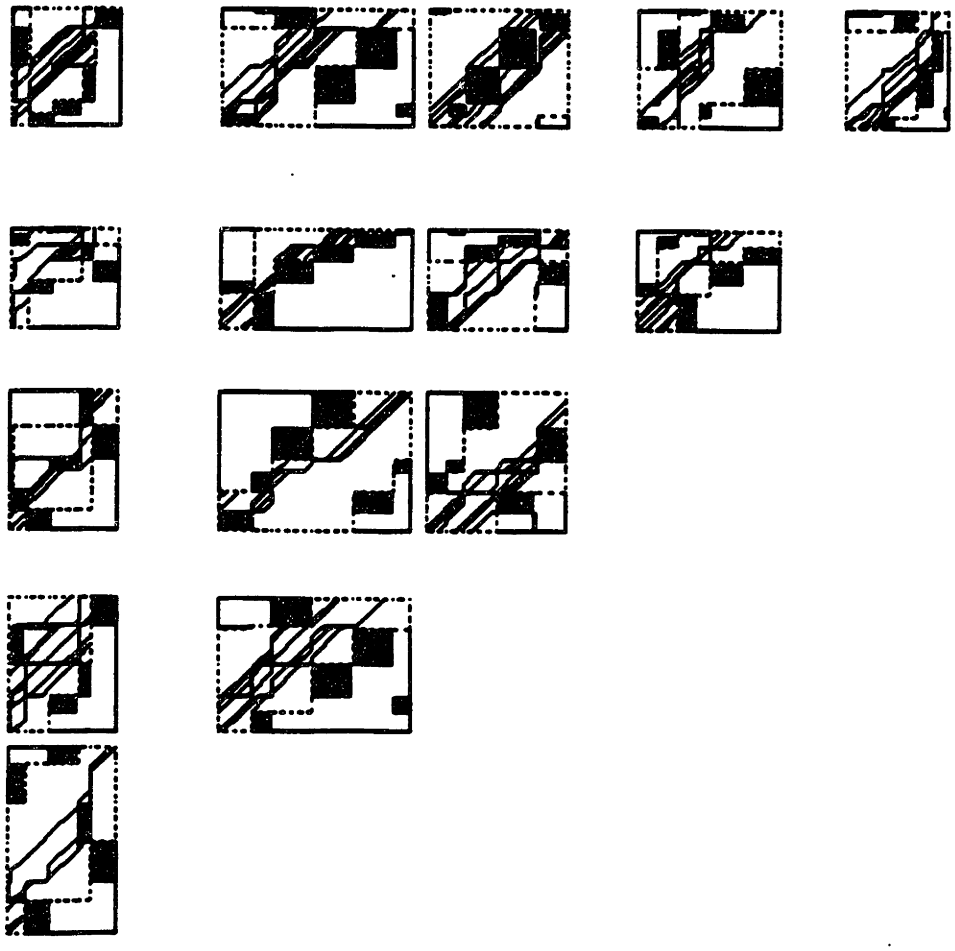


Figure 3.4: Pruned 2D Subspaces

makespan criterion, a high density of *good* schedules was found to lie within a tube connecting the origin to the state of total completion in the cartesian space. Such a tube can be specified by a set of precedence relations. The spatial distribution of *good* flowtime schedules could not be captured (characterized) by a set of precedence relations. Certain problem structures may influence the spatial distribution of schedule trajectories a predictable manner. See Appendix D for an example where the effects of and strategies to compensate for symmetry introduced into the problem structure are shown.

Heuristic H2

Chapter 4

Based on the results of Chapter 3 a heuristic (H2) was developed for specifying precedence constraints so as to converge upon a region of the completion space containing a high density of "good" schedules under the objective criterion of *Minimum Maximum Lateness*. When each job has the same arrival date and the same due date this criterion is equivalent to the makespan criterion.

4.1 Definition of Heuristic H2

The heuristic is defined as follows:

Step 1. Generate a collection of the obstacles (disjunctions).

Step 2. Update the *Early Start Times* and the *Late Start Times* of each activity.

Step 3. Select an obstacle for deletion.

Step 4. Delete the selected obstacle (resolve the selected disjunction).

Step 5. Repeat Steps 2,3,4 until no obstacles remain.

4.2 Explanation and Interpretation of Steps

Steps 1 through 4 will now be explained both in terms of disjunctions and in terms of obstacles.

Step 1. Each obstacle in the space corresponds to a possible conflict over a scarce resource. In the classic *Job-Shop Scheduling* problem each activity requires a single type of machine for it's processing and there is only one of each type of machine available. In order to generate a collection of the obstacles, find all pairs of activities which require the same machine yet are in different jobs. Each pair also corresponds to a disjunction (an unresolved precedence relation between the elements of the pair).

Step 2. An activity's *Early Start Time* is the earliest possible time at which the processing of an activity can be started without violating job arrival times or the precedence relations between activities. These times are computed assuming that unlimited resources are available. An activity's *Early Start Time* is the maximum of the *Early Start Time* plus processing time of the activity's immediate predecessors (i.e. the *Early Finish Times* of the immediate predecessors) and the activity's arrival date (if it has one). To find an activity's *Late Start Time* first find the minimum *Late Start Time* of the given activity's immediate successors. Then subtract the given activity's processing time from the minimum of the previous result and the given activity's due date.

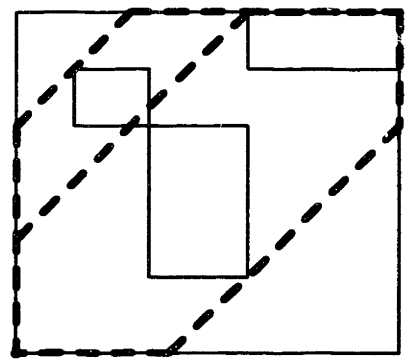
Step 3. For each obstacle compute the following two quantities from the pair of activities (A_i, A_j) which gave rise to the obstacle. The two quantities are the *slacks* between the two activities for their two possible orderings. If activity A_i precedes A_j then the slack is the *Late Start Time* of A_j minus the *Early Finish Time* of A_i. If activity A_j precedes A_i then the slack is the *Late Start Time* of A_i minus the *Early*

Finish Time of A_j. The maximum of these two slacks is the Max-Slack associated with the obstacle while the minimum is the Min-Slack. Select the obstacle with the smallest value of Min-Slack. If there is a tie in Min-Slack value then break the tie on the basis of largest Max-Slack.

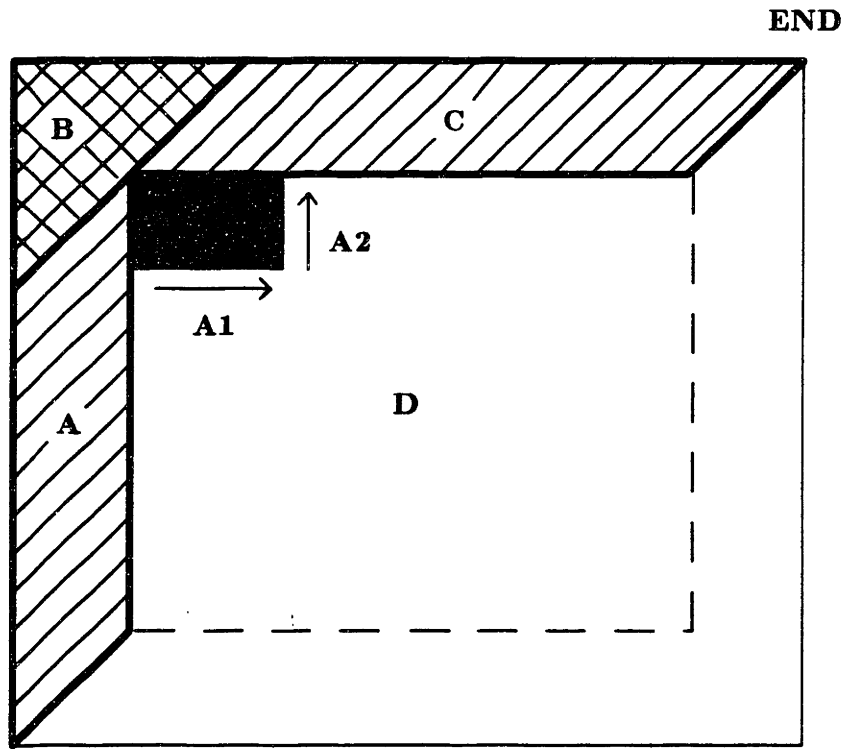
Figure 4.1a shows what this selection process looks like in a two dimensional example space. Figure 4.1a shows a number of trajectories used in the obstacle selection process. These trajectories were obtained by ignoring all but one obstacle and finding the *longer* of two possible trajectories around this remaining obstacle. This process was then repeated on each of the obstacles to yield the trajectories shown. (The length of a trajectory is determined by summing the maxima of the projections of it's piecewise linear segments on to the axes.) Then the obstacle associated with the longest of these paths is selected for deletion. The longest path is easily determined by visual inspection to be the one with the minimum amount of co-processing (diagonal segments).

Step 4. The "longest" path selected in Step 3 is actually a lower bound on the schedule length (makespan) for trajectories (schedules) which detour around the same side of the obstacle as the longest path. In all of these schedules activity A_2 is processed before activity A_1. The space is pruned such that these "longer" schedules can no longer be generated. This pruning is accomplished by requiring that activity A_1 always be processed before activity A_2. The pruned region of the space shown in the lower left of Figure 4.2 corresponds to states which become infeasible as a result of requiring that A_1 precede A_2. The lower right of Figure 4.2 shows how the space would have been pruned if we had required that A_2 precede A_1. These two possible pruning operations correspond to the two possible ways of "settling" a disjunction.

Figure 4.1b shows the space divided into regions associated with the selected



(a)



(b)

Figure 4.1: Obstacle Selection

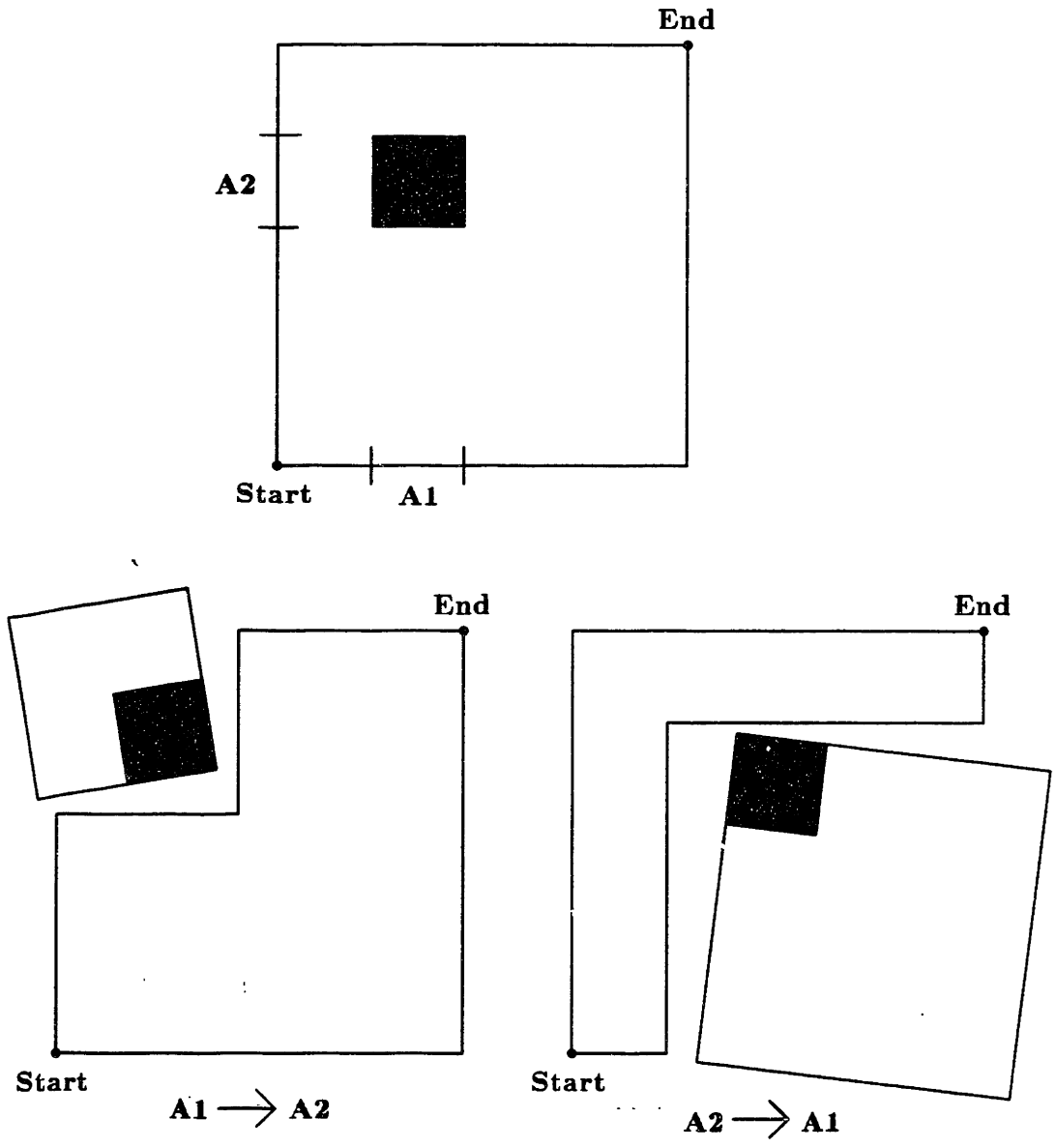


Figure 4.2: Two spaces which could result when the precedence is set

obstacle. Any trajectory which passes through regions A and then C must be at least as long as the longest path computed above. Those trajectories which also pass through region B are even longer. Note that there is also a simple upper bound on "length" for paths which traverse through the union of regions A, B, and C. This is simply the path along the outer boundary of the space. The closer the obstacle under consideration is to the outer limits of the space, the less difference there is between these simple upper and lower bounds on the path length. In the limiting case of an obstacle wedged into the corner the lower bound is equal to the upper bound.

Consider a trajectory which starts out in region A, then enters region D and finally enters region C. Regardless of what obstacles were previously ignored in D, the path segment through D can be no longer than that of a vertical segment along the inner boundary of A from the crossover point to the corner of the obstacle added to a horizontal segment along the lower boundary of C from the corner of the obstacle to the crossover point in C. Therefore, it is likely that deleting the corner of the space including the obstacle and a portion of region B, which consequently limits trajectories to those passing through D, will result in the pruning of an undesirable set of schedules.

The above mentioned bounds and approximate results get weaker when applied to obstacles residing in the inner regions of the space. Therefore in order to maximize the effectiveness of the bounds used, obstacles at the outskirts are deleted first then the space is gradually pruned from the outside inward. This way, the obstacles under consideration will lie near the outskirts of the space. Figure 4.3 shows the obstacle-less space resulting from repeated application of the selection and deletion process. Also shown is one of a set of possible trajectories through the space.

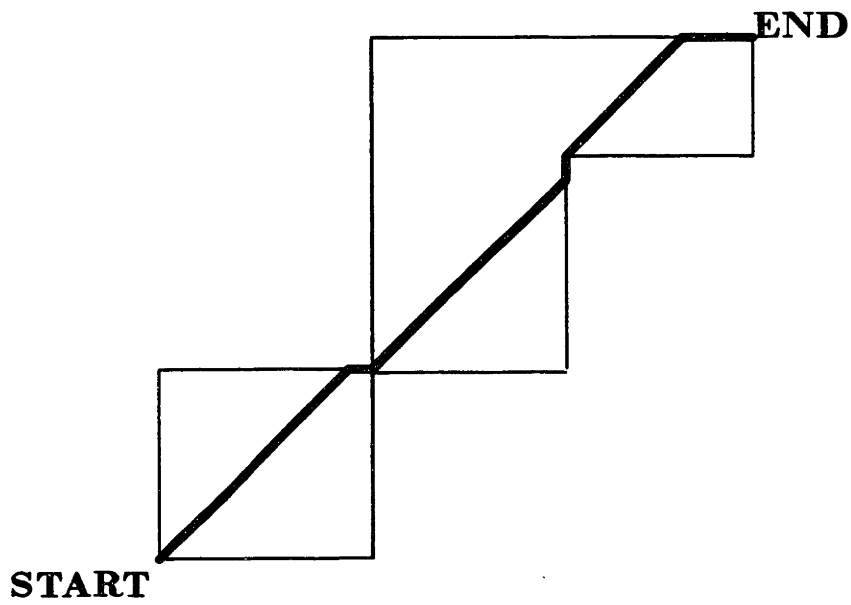


Figure 4.3: Trajectory Through Pruned Space

This bound or branching indicator has been used by other researchers (e.g. P^* [73] pg. 123) as a node selection criterion for a branch and bound algorithm. In this application, the obstacle which both lies on the critical path and has the minimum value of P^* is chosen to branch upon (In H2 an obstacle is selected from the set of all undeleted obstacles, not from only those on the critical path.). It (P^*) has been found to be unsatisfactory for this purpose because it is a relatively weak branching indicator and because the resultant search tree can be very deep ($m\binom{n}{2}$ See section 4.7). I believe this weakness is caused by limiting the selection process to obstacles along the critical path. Because the critical path (trajectory) lies near the body diagonal of the space and not near the outer edges, this weakness is not surprising. Instead of using this as a branching indicator this bound is used by H2 as a selection rule. It has the effect of choosing obstacles which lie near the outer regions of the

space. These obstacles are deleted first, then those close to the outer edges are deleted. The overall effect is to slowly converge about a tube shaped region of the space.

4.3 Intuitive Explanation of Heuristic H2

Heuristic H2 looks for what seems to be the most detrimental decision possible - (this decision would either lengthen the critical path or maximally reduce slack somewhere in the network of activities). It then makes the decision complimentary to this most detrimental one. In other words, it avoids the worst possible decision at any given point.

Making such a decision leaves open more options for future decisions. This happens two ways. First, in general there is more slack left in the network than would be if the worst possible decision was made. Second, in general this decision does not decrease the number of subsequent decisions to be made (transitive closure of the decisions [precedence relations added] so far usually does not restrict any of the remaining decisions). So, in some sense, small decisions are made. The graphical interpretation is this: Usually when a 2D obstacle is deleted along with its appropriate corner of the space, only that one particular obstacle is deleted. Note, that when the obstacle and corner are deleted only a relatively small portion of the total space is removed.

By always working near the outer fringes of the space, the relatively simple bound (which the decisions are based upon) are improved. The outer edges of the space are involved in subsequent calculations which would not be the case, if for example, the first obstacle considered for deletion was near the center of the space.

4.4 Extension to Higher Dimensions

The above heuristic, although presented in two dimensions, can easily be applied to higher dimensionality problems. Figure 4.4 shows an example three dimensional, three job space. This corresponds to a "3x3 Job Shop" problem. Only two obstacles are shown for clarity. The bar shaped obstacle results from two activities (the first activity of Job 2 and the first activity of Job 3) which require the same resource. The cross shaped one results from three activities in different jobs each of which requires the same resource. This obstacle is formed by the intersection of three rectangular bar segments thereby forming a three dimensional cross. In the classic *Job-Shop Scheduling* problem (each job visits each machine once) there are three of these cross shaped obstacles nested together with the protruding ends trimmed to form the space. These obstacles occupy a large fraction of the volume of the space. This density issue is dealt with in detail in Appendix A. An unobstructed region of the three dimensional space corresponds to a situation in which all three jobs can be co-processed. A unobstructed plane formed between the bar shaped extensions of neighboring obstacles corresponds to a situation in which two jobs can be co-processed. And a line corresponds corresponds to a situation in which only one job can be processed. These unobstructed regions, planes and lines are available for traversal by trajectories.

Instead of working directly in this three dimensional space, the heuristic is applied to the $\binom{n}{2} = 3$ unique two dimensional subspaces (non-redundant faces of the three cube each of which corresponds to a unique pair of jobs). These are shown in Figure 4.5. Each of the 2D obstacles is considered separately as in the previous 2D case, but when the bounds are computed the entire 3D space is used with the

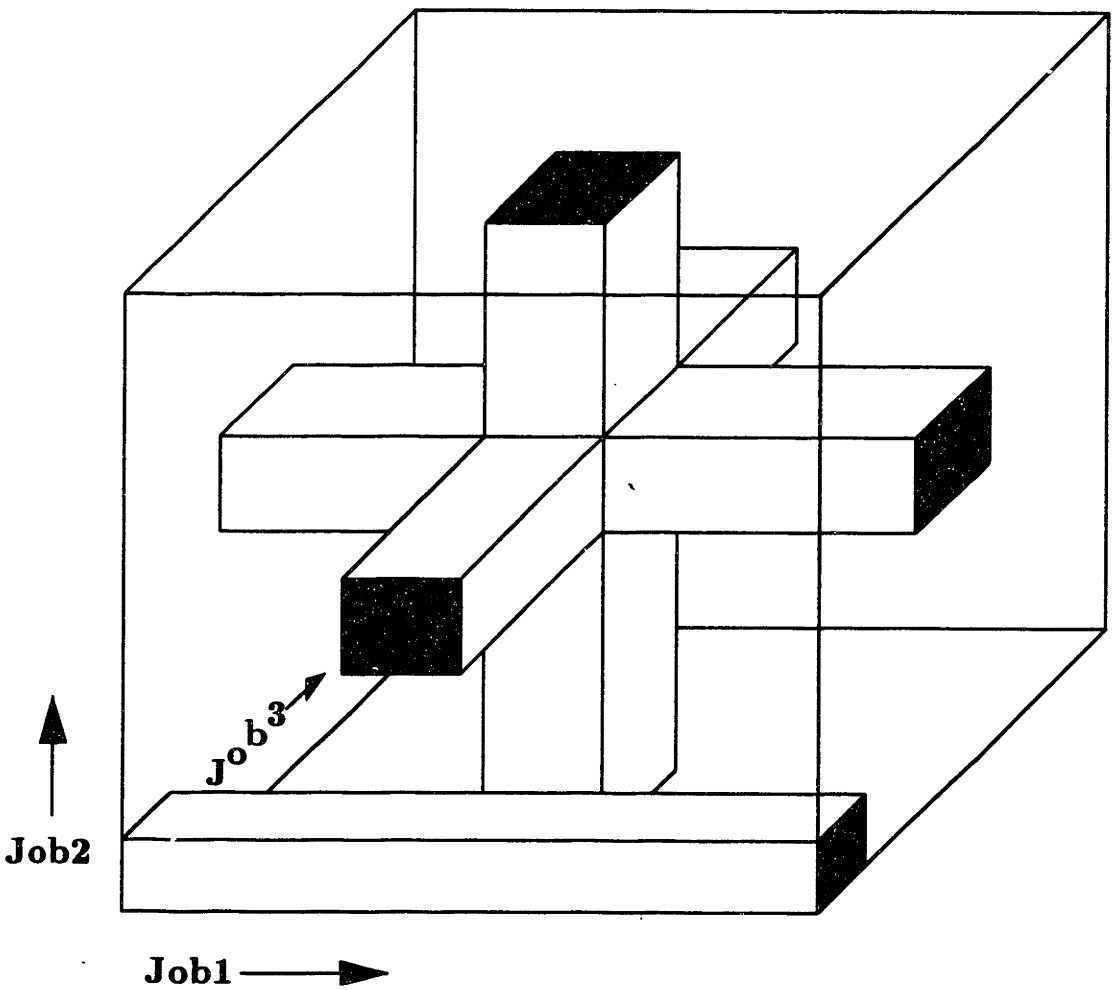


Figure 4.4: Example Space Associated With A Three Job Problem

capacity constraints relaxed on *all* machines. The only relevant constraints at this point are those from the single obstacle component in the 2 space and the outer edges of the 3D space.

In the three dimensional example, it was assumed that only one of each machine type was available. This gave rise to the 3D cross type of obstacle. If one assumes that two of each machine type is available then the obstacle consists of only the center of the cross (see Figure 2.1). In this case, the heuristic as described above could not be applied and another similar heuristic which uses a three dimensional base case would be required.

4.5 Correctness Sketch

Does following the steps of this heuristic guarantee that some technologically feasible schedule will be generated? The answer is yes, and the question can be rephrased negatively in two different ways. One, when the space is pruned can the pruning occur such that the resulting space is composed of disjoint regions? Two, (equivalently) is it possible to generate a directed cycle of precedence relations?

Shown in Figure 4.6 is how successive pruning operations could result in the separation of the space into disjoint regions. First the upper left is deleted along with the lower right hand obstacle. Then the lower right is deleted along with the upper left hand obstacle. This resultant space is composed of the small upper right and lower left regions with no possible path between them and consequently no possible feasible schedule.

To show that this cannot happen, we resort to a Gantt chart representation of the problem. Here the task is to show that a directed cycle is not generated as a

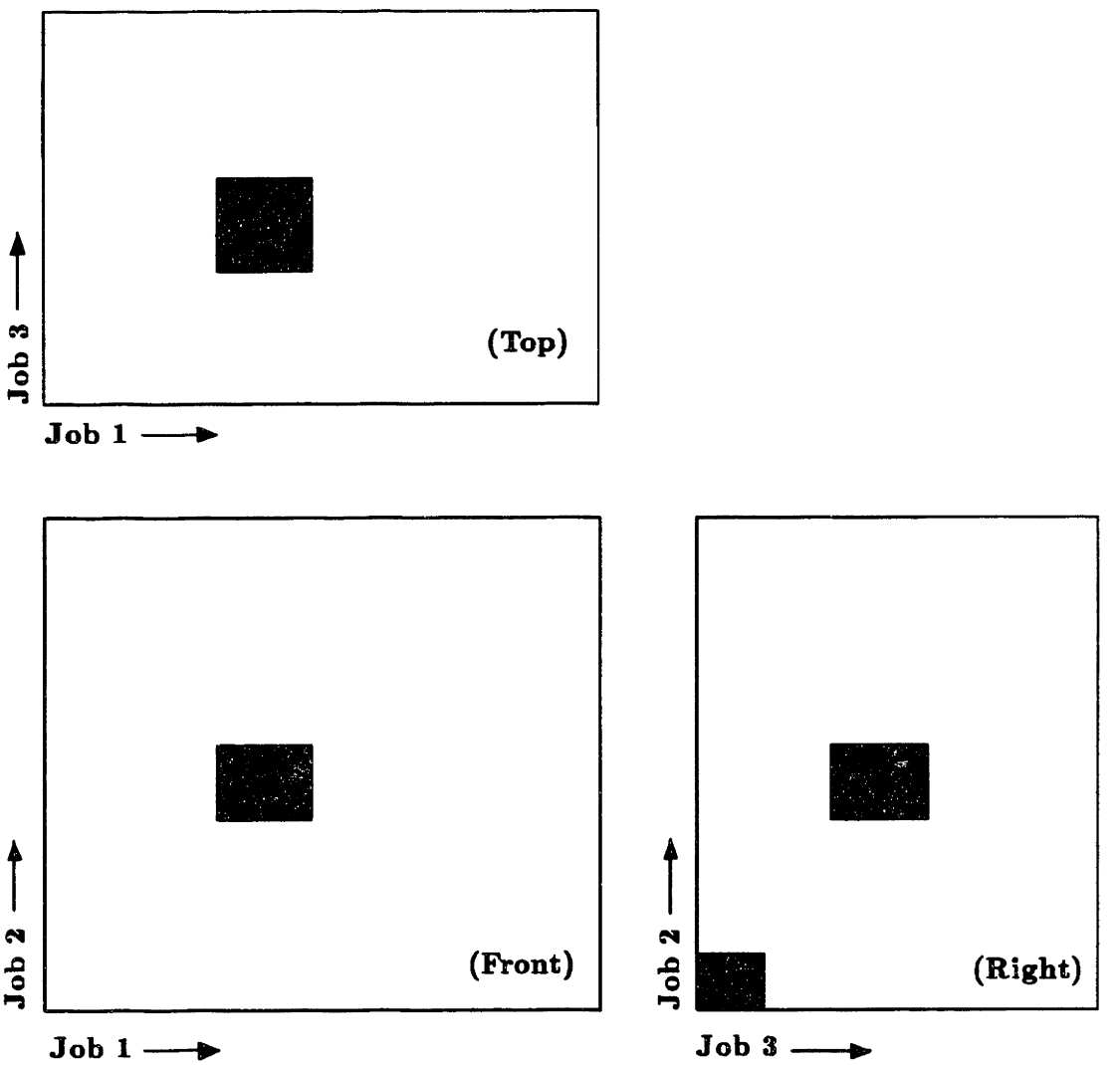


Figure 4.5: Two Dimensional Subspaces Associated With A Three Job Problem

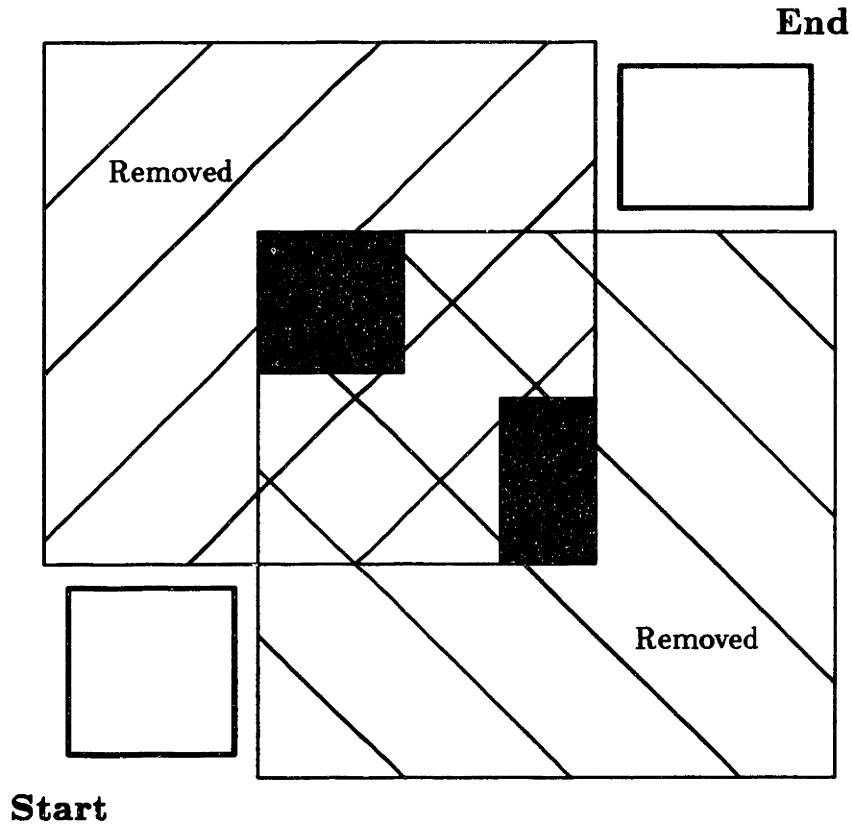


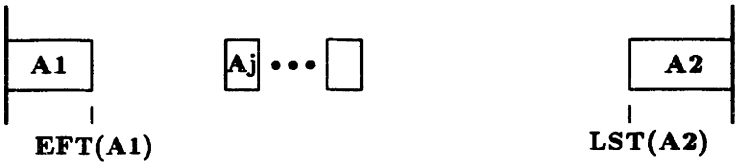
Figure 4.6: Pruning which results in disjoint regions

result of the addition of some precedence relation. In order to generate a directed cycle by the addition of a single precedence relation, there must already exist a linear sequence of two or more activities (ordering of some activity with respect to itself is not considered). Such a sequence is shown in Figure 4.7. Activity A_1 is constrained to precede the (possibly zero-length) sequence of activities A_j the last of which precedes activity A_2.

It is assumed that activities A_1 and A_2 are members of different jobs and that we are seeking to order these two activities by the addition of a precedence relation. The current relationship of the two activities ($A_1 \succ A_2$) was assumed to be established implicitly (via transitivity) at some previous step of the algorithm. In order to decide whether A_1 should precede A_2 or vice versa the "slack" between these two operations is computed for the two possible configurations (See Section 4.1). The precedence relation is then added in the orientation corresponding to the larger slack. The top of Figure 4.7 indicates the slack calculation for the case of activity A_1 preceding activity A_2. This is simply the *Late Start Time* of activity A_2 minus the *Early Finish Time* of activity A_1. These times are annotated in the figure.

The *slack* for the alternative configuration ($A_2 \succ A_1$) can be computed as *Late Start Time* of A_1 minus the *Early Finish Time* of A_2. This can be expressed in terms of the precious calculation as shown in Figure 4.7b. The *Early Finish Time* of activity A_2 is equal to the *Early Finish Time* of activity A_1 plus the sum of the processing times of the intervening activities A_j plus the processing time of activity A_2. Similarly, the *Late Start Time* of activity A_1 is equal to the *Late Start Time* of activity A_2 minus the sum of the processing times of the intervening activities A_j minus the processing time of activity A_1. Subtracting the *Early Finish Time* of activity A_2 from the *Late Start Time* of activity A_1 is strictly less than

SLACK-IF(A1,A2) = LST(A2) - EFT(A1)



(a)

LST(A1)

SLACK-IF(A2,A1) = [LST(A2) - SUM(P(Aj)) - P(A1)] -

[EFT(A1) + SUM(P(Aj)) + P(A2)]

EFT(A2)

= LST(A2) - EFT(A1) - [P(A1) + P(A2) + SUM(P(Aj))]

= SLACK-IF(A1,A2) - [P(A1) + P(A2) + SUM(P(Aj))]

(b)

Figure 4.7: Gantt Chart for Correctness Proof

the *slack* computed for the acyclic orientation of the precedence.

4.6 What Is Different About This Algorithm?

Heuristic H2 is most closely related to the algorithms which can be interpreted in the completion space. These algorithms can be grouped into two classes. The first class includes those which start a trajectory at the origin and subsequently delete the space adjacent to the evolving trajectory (e.g. priority dispatching). The second class includes those which generate the *tube* by pruning out large regions of the space adjacent to the current critical trajectory (as found by the *Critical Path Method*)(e.g. branching and bounding on disjunctions along the critical path). In these two classes, the early pruning operations are major ones. In other words, relatively large numbers of alternatives are precluded early on. If these decisions were not correct, then the algorithm must either backtrack or fail to find an optimum.

The algorithm developed in this thesis falls into a third class. In this class the *tube* is gradually converged upon by pruning away small pieces from the outer edges of the space. With this approach the first decisions are minor ones precluding only relatively small numbers of well bounded alternatives. Thus it is likely that the pruning decisions made early on by heuristic H2 are correct ones. The outside in approach just described is not attractive for use in branch and bound algorithms because the large number of decisions made corresponds to a relatively deep search tree.

4.7 Complexity

One drawback is that heuristic H2 has to explicitly deal with every 2D obstacle in every subspace. This results in the majority of cpu time being expended iterating through the collection of obstacles.

The steps of heuristic H2 as defined in Section 4.1 follow:

Step 1. Generate a collection of the obstacles (disjunctions).

Step 2. Update the *Early Start Times* and the *Late Start Times* of each activity.

Step 3. Select an obstacle for deletion.

Step 4. Delete the selected obstacle (resolve the selected disjunction).

Step 5. Repeat Steps 2,3,4 until no obstacles remain.

For a complexity analysis assume that we have an $\{n, m, \mathrm{G}, L_{max}\}$ problem. Here there are n jobs each of which is a linear chain of m activities. Each job must visit each machine exactly once. The nm activities can be collected into groups of size n (one group for each resource) in $O(nm)$ time. This $O(nm)$ time is an estimate (in terms of input problem size) of how many cpu cycles (or memory cells) are required to compute the desired result. Typically, constant factors and terms which grow relatively slowly with input problem size are omitted from such estimates. Then $\binom{n}{2}$ obstacles are generated from each of these groups. Step 1 is only performed once and takes $O(nm + m\binom{n}{2}) = O(n^2m)$ time.

Step 2 involves updating the slacks in a PERT network graph. This can be done in $O(Vertices + Edges)$ [2] time. There are $O(nm)Vertices$ and $O(nm)Edges$ to start and $O(n^2m)Edges$ added as the algorithm proceeds (one for each deleted obstacle.)

Therefore step 2 takes $O(2nm + n^2m) = O(n^2m)$ time. Step 3 is merely adding a new edge to the PERT network graph which takes constant time.

Steps 2 and 3 are repeated once for each obstacle resulting in $O(n^4m^2)$ for an overall estimate of run time. If one assumes that the problem is "square" (i.e. $n = m$) then the time complexity is $O(a^3)$ where there are a activities to be scheduled. From the point of view of the number of obstacles $o = m\binom{n}{2}$ the complexity is $O(o^2)$.

Preliminary experiments indicated that much cpu time was being expended trying to update sections of the PERT network graph that were not affected by the latest edge added. To alleviate this, a quick slack updating procedure was developed. In this procedure, the effects of adding a new edge were propagated only as far as necessary. This procedure could take longer in the worst case but on average takes less time than the procedure which always processes the entire graph. Experience shows that problems involving a couple of hundred activities can be processed in a few minutes on a *Symbolics 3650 Lisp Machine* equipped with 2 Megabytes of core memory.

The quick slack update procedure is as follows. Starting from the tip of the precedence relation just added propagate *Early Start Times* forward by giving preference to the activities which will be most affected. This effects an implicit topological sorting because no activity can be affected more than the largest change in *Early Start Times* and guarantees that the updating is performed in a correct order. Propagating large effects first supersedes smaller propagated effects, therefore each node will be visited a maximum of one time.

Then repeat the procedure propagating *Late Start Times* starting from the tail of the precedence relation added in a backwards direction toward the beginning of the graph. The activities being processed are inserted into a "heap" [2] using the change

in the activity's slack as a "key." Using a heap allows an element to be inserted and allows the element with the maximum "key" to be removed in $O(log(n))$ time if there are n elements in the "heap."

To put the complexity of H2 into perspective, consider the complexity of some of the existing algorithms. First consider the prospect of complete enumeration. Assuming that we are trying to solve a $\{n, m, \mathrm{G},\}$ problem, there is a simple upper bound on the number of distinct possible sequences. This is given by $(n!)^m$. This bound could be attained by structuring the problem such that all the jobs visit the machines in the same order. Then the n jobs could be ordered independently at each of the m machines.

Second consider a Branch and Bound algorithm which branches on disjunctions. In an $\{n, m, \mathrm{G},\}$ problem, each of the m activities in a job requires a unique machine for processing. There are $\binom{n}{2}$ disjunctions associated with each of the m machines for a total of $m\binom{n}{2}$ disjunctions. This is an upper bound on the maximum depth of the Branch and Bound search tree. The search tree is binary because any given disjunction can be resolved in two ways. The computation of the lower bound used in branching decisions range in complexity from trivial to NP-hard.

Third, consider a Branch and Bound algorithm based on the generation of *Active Schedules*. In this case, the depth of the search tree is bounded by the maximum number of conflicts which can occur on all machines. This maximum depth is nm and reflects the possibility of all the n jobs conflicting at each of the m machines. Again computation of the lower the bounds used branching decisions range in complexity from trivial to NP-hard.

Forth, consider a Dynamic Programming approach. For a single machine problem $(\{n, 1, \mathrm{G},\})$ a convenient representation to use is the n dimensional completion space.

One dimension represents each job with the lengths of the axes corresponding to the processing times of the various jobs. Since there is only one machine available for processing, trajectories are constrained to follow the edges of the space. The Dynamic Programming formulation is to find the optimal cost trajectory to each of the vertices in terms of the immediate predecessor vertices. In this case there are 2^n possible vertices to visit.

When there is more than one type of machine available, the situation is more complex. We no longer have a lattice of vertices defining the endpoints of the trajectory segments. This is because the trajectories are no longer constrained to the edges of but may traverse diagonally through regions of the space. There is a way around this though. If we are willing to constrain the problem to have only integer processing times, then each of the n coordinate axes can be divided at unit intervals. This divides the space up into unit n *cubes.* All trajectory segments are guaranteed to have as endpoints the vertices of these n *cubes.* There are approximately $mp + 1$ vertices along the length of any given axis; p vertices for each activity of approximate processing time p and one point at the end. The total number of vertices is approximated by $(mp + 1)^n$.

Fifth, consider a *Priority Dispatch Rule* being used in a simulation environment. Here, similar to the Branching and Bounding on *Active Schedules*, there is an upper bound on the maximum number of conflicts which can occur. This bound is mn. Associated with each of the conflict resolutions is the computation of the *Dispatch Rule.* This ranges from a trivial constant (e.g. a due date) to looking ahead to the next queue to possibly some function of all of the activities. In general relatively simple indicators are used. *Priority Dispatching* could be thought of as a depth first search of the search tree associated with Branching and Bounding based on *Non*

Delay Schedules. Perhaps the Lower Bounds developed for the Branch and Bound algorithms could be adapted to form more sophisticated *Dispatching Rules.*

Similar to *Priority Dispatching*, Heuristic H2 can be thought of as a depth first search of the search tree associated Branching and Bounding on disjunctions. In this case the maximum depth is attained and is given by $m\binom{n}{2}$.

4.8 Local Rule

Heuristic H2 is a local rule in the sense that only a relatively small amount of information (slack) is used in deciding where and how the space will be pruned. In some cases, the use of a local rule can be proved to lead to an optimal solution; but in the case of heuristic H2 some simple counterexamples show that this is not the case (see Section 4.11). This is no surprise considering the problem is NP-Hard. What is surprising is how well it works given the nature of the "bounds" (local rule) used to decide where and how to prune the space. There exist a spectrum of lower bounds developed over the years ranging from ones as simple as the rule used in H2 to ones which are NP-hard.

Adams *et al* [1] used bounds (which were NP-hard) in an iterative procedure involving some redoing of earlier decisions. First, all machines but one were ignored and the activities scheduled on the remaining machine. Then, leaving the ordering of these scheduled activities intact, activities requiring the next machine were scheduled and so on. Periodically, activities scheduled on one machine were freed up and then rescheduled as a method of relaxation to accommodate for factors which were not accounted for earlier. Heuristic H2 produced a superior schedule on the $\{10, 10, G, \text{Makespan}\}$ problem found in [47] even though the local rule used by H2

was trivial in comparison to the lower bound used by Adams.

4.9 Rate of Pruning

In Figure 4.8 is shown an estimate of the number of feasible states of completion within the cartesian space as heuristic H2 proceeds. This particular graph was from the $\{10, 10, G, \text{Makespan}\}$ problem found in [47]. For this problem there are 450 disjunctions to be settled. If one were to define 11 points along each of the coordinate axes (one at the beginning and end of each activity), then a lattice of points (states of completion of activities) would be defined by choosing one coordinate (point) along each axis. The graph is an estimate of the number of such lattice points left in the cartesian space as the 450 disjunctions are successively settled. This estimate was obtained using the numbering scheme in [76]. Notice that the number of remaining lattice points is almost a constant fraction of the number from the previous iteration.

4.10 Attempts at Increasing Performance

A number of alternative obstacle selection rules for heuristic H2 were evaluated before arriving at the final selection rule. Presently, each obstacle O_i has two values associated with it (see Section 4.2 **Step 3**). These values correspond to what slack would exist between the two activities for the two possible orientations of the precedence relation between them. The larger of these two slacks is defined as Max-Slack while the smaller one is Min-Slack. The obstacle with the smallest associated Min-Slack is chosen for deletion first. If there is a tie then the obstacle with the largest value of Max-Slack is chosen. Using the smaller of Max-Slack seems to make only a minor

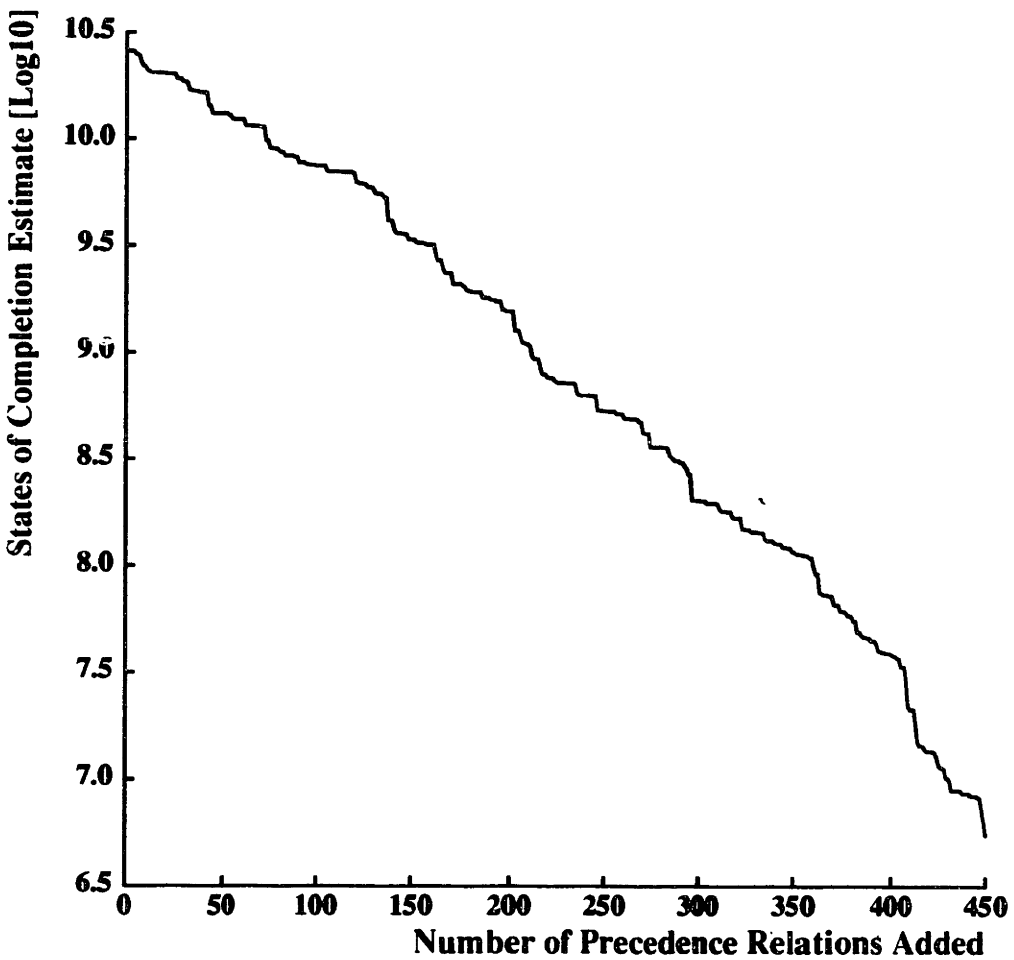


Figure 4.8: Estimate of number of states of completion remaining as heuristic H2 proceeds on a 10 job 10 machine problem

difference, possibly improving the solution obtained in certain problem instances.

Some of the other (fruitless) alternative obstacle selection schemes tried are:

- Instead of breaking ties with the largest value of Max-Slack, ties were temporarily ignored. This delaying of resolution of the most critical obstacles was found to be detrimental to performance.
- Choose the obstacle with the smallest Max-Slack.
- Choose the obstacle with the largest Max-Slack.
- Choose the obstacle with the largest Min-Slack.

One question which arises in problems of this sort is; is the solution locally optimal? Even though the global optimum may not have been obtained it is desirable that the solution be at least locally optimal, otherwise some simple modification should be used to improve it.

In order to test the local optimality of the solutions obtained, a downhill only relaxation routine was applied to some sample solutions. The relaxation is effected by interchanging two adjacent activities along the critical path. In the test, the interchange is accepted if the reordering results in a lower value of the objective function, otherwise a reordering of another pair activities is tried. Applying this procedure to some trial solutions resulted in little or no improvement indicating that the initial solutions found by heuristic H2 were very near to or at a local optimum.

4.11 2D Counterexamples

When solving $\{2, m, \mathrm{G}, \mathrm{Makespan}\}$ problems, heuristic H2 almost always finds an optimal solution. Several hundred randomly generated $\{2, 5, \mathrm{G}, \mathrm{Makespan}\}$ problems

were tested before one was found on which heuristic H2 failed to find the optimal solution. The optimal solution which was used for comparison was found via exhaustive enumeration. Figure 4.9 shows two problem instances in which the heuristic failed to find an optimal trajectory. The numbers correspond to the order in which the obstacles were selected for deletion. Obstacles number 1, 2, and 3 have already been deleted and which way to delete number 4 is under consideration. The two alternative trajectories shown in each case are rated with equal value by the decision procedure and the suboptimal one was chosen by chance. In both cases the suboptimal performance was due to ignoring the presence of the last obstacle when deciding how to delete the next to last one.

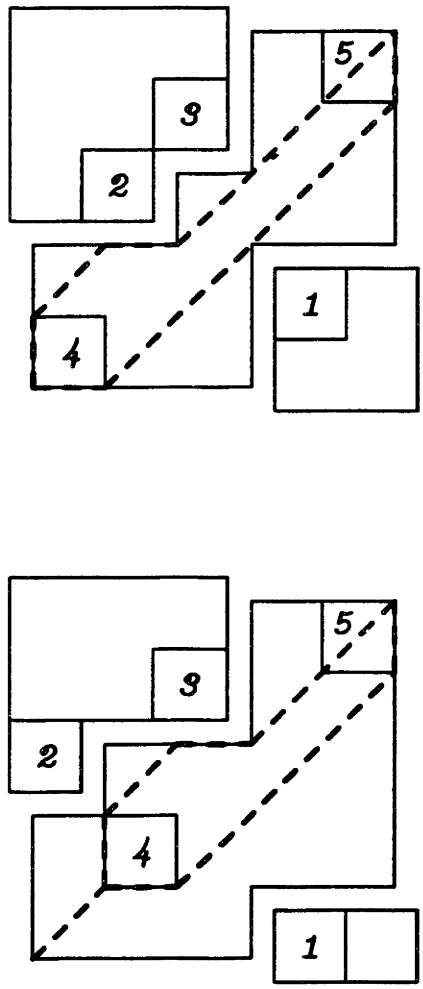


Figure 4.9: Suboptimal Trajectories in 2D Problems

Experiments and Results

Chapter 5

5.1 Experimental Objectives

The objective of this chapter is to show how heuristic H2 performs as certain properties of the test problems are varied. Heuristic H2 will be benchmarked against an array of the *priority dispatch* rules and some biased search methods. These experiments are designed to show:

- Performance relative to the priority dispatch rules,
- Performance as problem dimensionality increases, and
- Performance over various objective criteria.

5.2 Measures of Performance

From the work of A. H. G. Rinnooy Kan the following collection of criteria form a reasonably rich set of *regular measures* which can be used (possibly in combination) as measures of schedule goodness.

A regular measure f has the following property expressed in terms of C_i the completion time of Job_i.

If

$$f(C_1, \ldots, C_n) < f(C'_1, \ldots, C'_n) \tag{5.1}$$

then

$$C_i < C'_i \tag{5.2}$$

for at least one i.

These criteria are based on arrival times, completion times and due dates. They can be combined to reflect a variety of shop operating costs [15], [73]. These six are:

- C_{max} $\equiv$ Max Completion Time or Makespan
- $\sum C_i$ $\equiv$ Sum Of Completion Times or Flowtime
- $\sum w_i C_i$ $\equiv$ Weighted Sum Of Completion Times
- L_{max} $\equiv$ Max Lateness
- $\sum T_i$ $\equiv$ Sum Of Tardiness
- $\sum w_i T_i$ $\equiv$ Weighted Sum Of Tardiness

Lateness is the difference between a job's due date and it's completion time. The tardiness is defined as the maximum of 0 and the lateness. These criteria do not include costs incurred for the early completion of jobs.

An additional criterion which would be useful would be some measure of robustness. This would measure cost sensitivity of the schedule either to minor variations in problem data (some activity takes longer to process, etc.) or to major changes such as shop reconfiguration (some machine breaks down). This would probably lead to

schedule specifications which included alternative contingency sequences. This has yet to be quantified.

5.3 Problem Structure/Sources

The test problems used in these benchmarking experiments were either obtained from the literature [47], derived from such a problem by interchanging some of the activities, or by random means. The problems were either of the flow type or the general type. In flow type problems, each job visits each machine once and all jobs visit the machines in the same order. In the cartesian space representation, all the obstacles lie along the diagonals. In the general problem, again each job visits each machine once, but any job can visit the machines in any order. In the cartesian space representation, the obstacles are scattered about - one in each row and one in each column.

A $\{10, 10, \mathrm{G}, \mathrm{Makespan}\}$ problem was chosen from the literature as a basic test case [47]. This problem involves 10 jobs each 10 activities long and 10 different machines. It is an instance of the *General Job Shop Scheduling Problem* and a value exists for the optimal makespan solution for it. This optimal makespan is 930 time units. Additional problems were derived from this one by interchanging the order of some activities within a job. From the same source came a $\{6, 6, \mathrm{G}, \mathrm{Makespan}\}$ problem with similar attributes. The optimal makespan for this problem is 55 time units.

Arrival dates and due dates were added to the problems to generate additional problems. The arrival date for the first job was set at 0, for the second at 10% of the average job duration, the third at 20% and so on. The due dates were similarly

skewed, and the average due date was set such that all the solution techniques had difficulty avoiding tardiness.

For problems that were generated totally randomly, either the flow or general constraints were imposed, a machine ordering was randomly chosen, and processing times were independently generated from a flat probability distribution. In [47] is specified a problem parameter R, the ratio of the longest to shortest (integer) processing time. They used values of 2, 3, 5, 8, 10 and 15. When the ratio is 1 (equal processing times for all activities), the distribution of objective function value over the schedules is approximately Gaussian. As the parameter R is increased, additional peaks appear in the distribution which makes the problem more unpredictable. In the problems generated here, relatively large values of R (e.g. 10) were used. Processing times were assumed to be flatly distributed within the range of 100 to 1000 time units.

5.4 H2 vs. Priority Dispatching

The heuristic was benchmarked against an array of the traditional priority dispatch rules. Each rule was used to generate a single solution for comparison against the solution obtained by heuristic H2. Fifty $\{10, 10, \text{Gen}, \}$ problems were generated as described above with all jobs arriving simultaneously. The priority rules used for comparison were:

- **Earliest Due Date (DDATE)** - The job with the earliest due date is chosen.
- **Expected Work in Next Queue (XWINQ)** - Look ahead to the next queue that a job will visit. Add the job's processing time to the sum of the processing times of the jobs in the next queue. Choose the job with the smallest value.

- **Fewest Jobs in Next Queue (NINQ)** - Look ahead to the next queue that a job will visit. Choose the job headed to the shortest next queue.

- **Fewest Operations Remaining (FOPNR)** - Choose the job with the smallest number of operations remaining until completion.

- **First Come First Served (FCFS)** - Choose the job which arrived in the queue earliest.

- **Least Slack First (SLACK)** - For each job subtract from the job's due date the current time and the processing time of the rest of the job. This is the slack. Choose the job with the least slack.

- **Least Slack Per Operation (SOPN)** - For each job, divide the slack by the number of operations remaining. Choose the job with the least of these values.

- **Least Work in Next Queue (WINQ)** - For each job, sum up the processing times of the jobs in the next queue which the job is going to visit. Choose the job with the minimum value.

- **Least Work Remaining (LWRK)** - Choose the job with the least total amount of processing left to be done.

- **Modified Due Date (MDD)** - For each job, compute max(1,(due date - current time)/total processing time of the rest of the job). Choose the job with the smallest such value.

- **Process Time + (Slack / Operations Remaining) (PSOPN)**

- **Process Time + Expected Work in Next Queue (PWQP)**

- **Process Time + Work Remaining (PWRK)**
- **Process Time / Operations Remaining (POPNR)**
- **Process Time Difference J, J+1 (PSP)**
- **Shortest Processing Time (SPT)**
- **(Slack * Process Time) / Work Remaining (MSPON)**
- **Work in Next Queue / Next Operation Process Time (PWQP)**

Detailed descriptions of these rules and their relative merits can be found in [24]. Ties were broken using the *First Come First Served* rule.

Shown in Figures 5.1 and 5.2 are plots of the relative performance of the heuristic and dispatch rules under the makespan optimality criterion. Shown in Figures 5.3 and 5.4 is relative performance under the flowtime criterion.

Each individual plot shows the performance of the rule listed in the title position relative to the best solutions found by all of the rules taken together. Problem number increases along the x axis. For each of the fifty problems, the relative difference between the solution found by the rule under consideration and the best solution found for the problem was plotted along the y axis. If a given rule always found the best solution then its curve would lie along the x axis. The plots for each rule all use the same y scale to facilitate visual comparison. Next to each title are a mean and a standard deviation summarizing the performance of that rule.

As can be seen from the plots, heuristic H2 outperforms the array of priority dispatch rules under the criterion of makespan. Under the criterion of flowtime the least work remaining rule is consistently best. Heuristic H2 is "tuned" to converge upon a tube shaped region in the cartesian space. It does this by gradually pruning

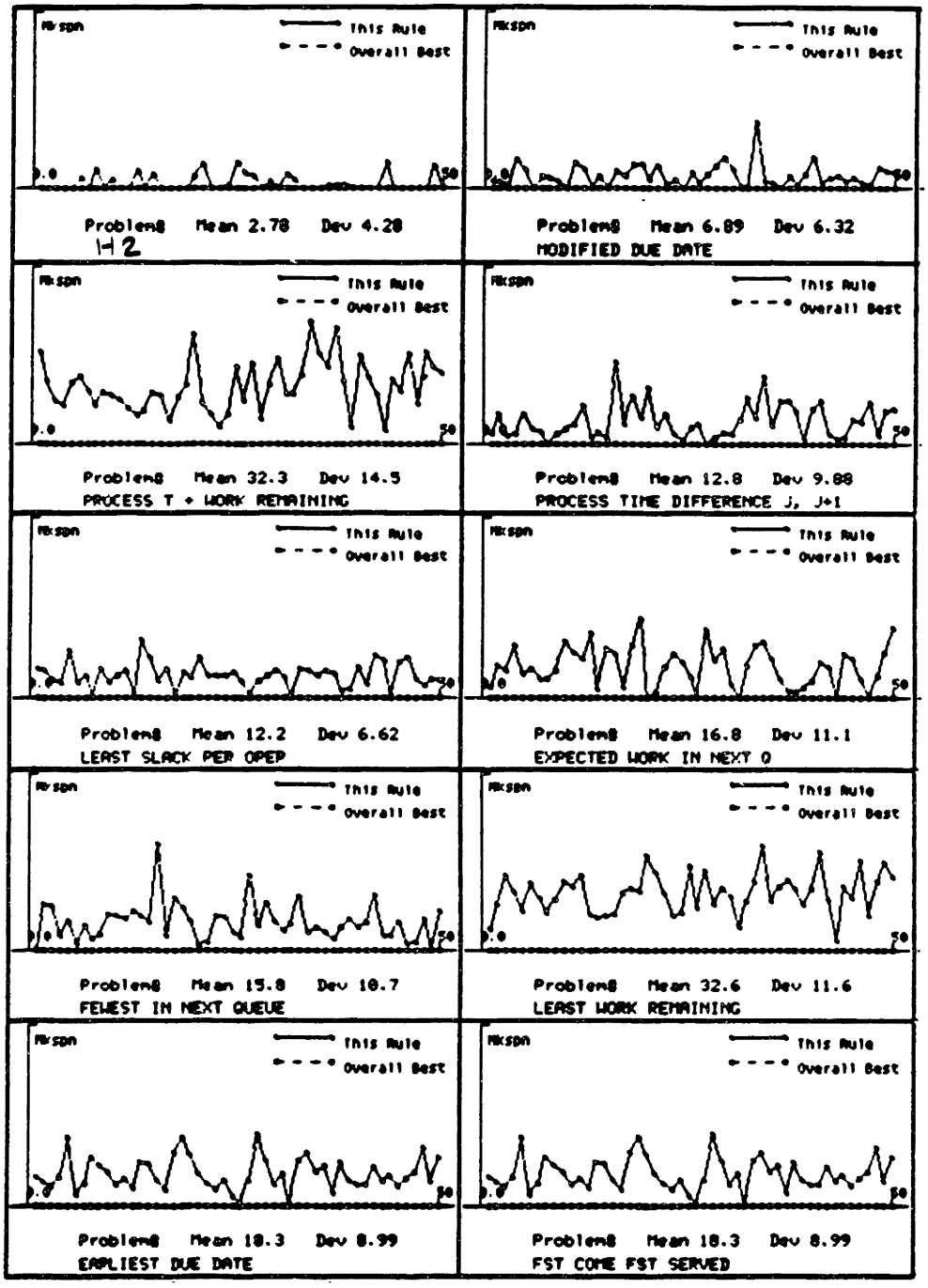


Figure 5.1: Relative performance under the makespan criterion (page 1 of 2)

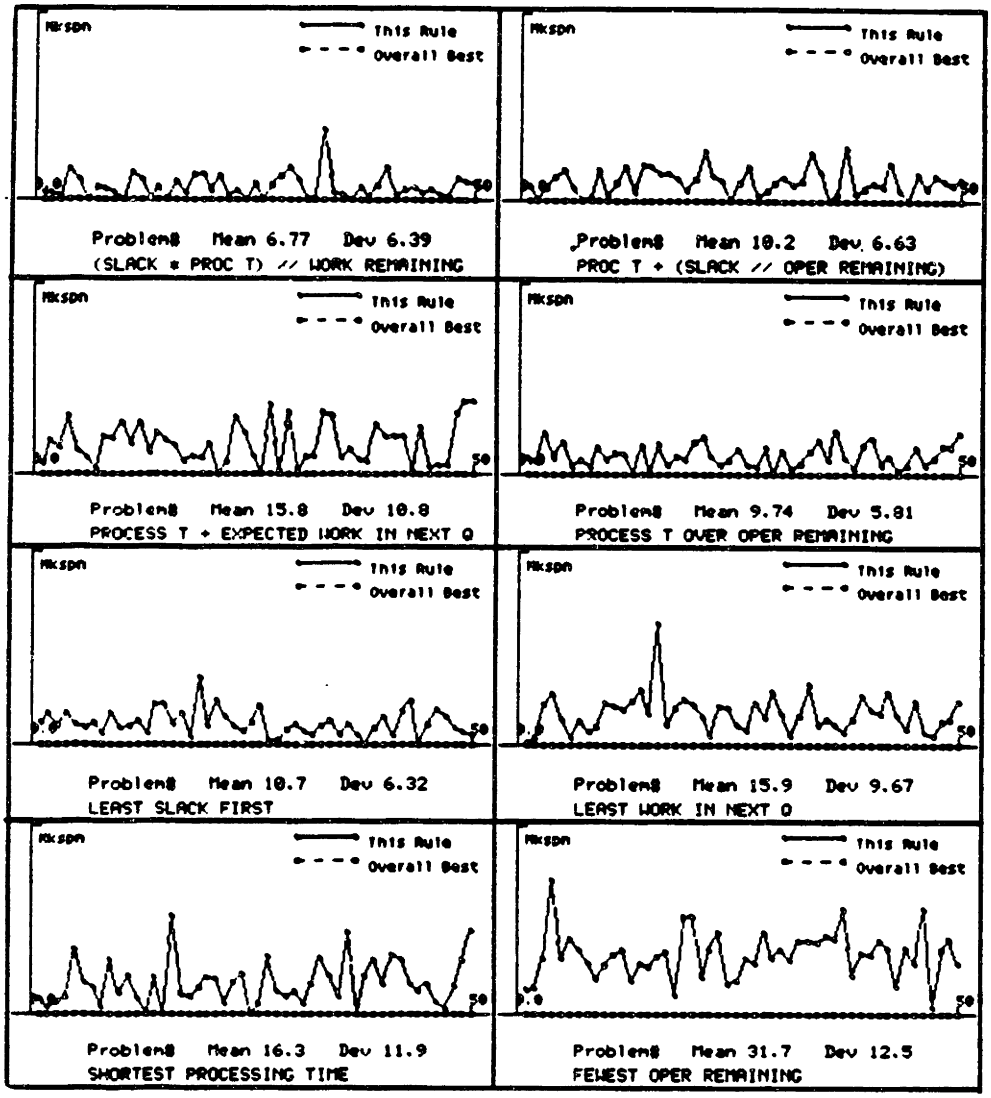


Figure 5.2: Relative performance under the makespan criterion (page 2 of 2)

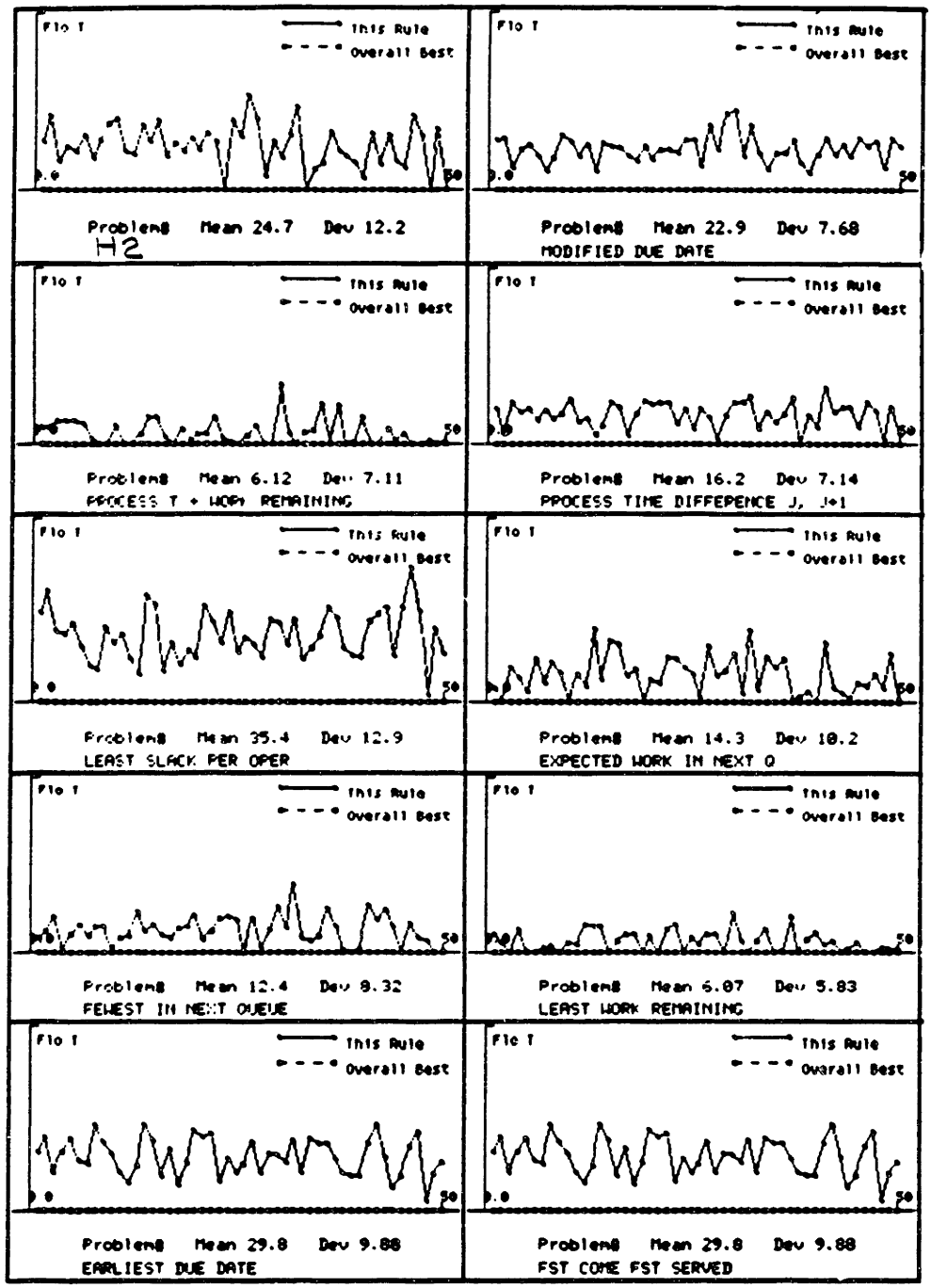


Figure 5.3: Relative performance under the flowtime criterion (page 1 of 2)

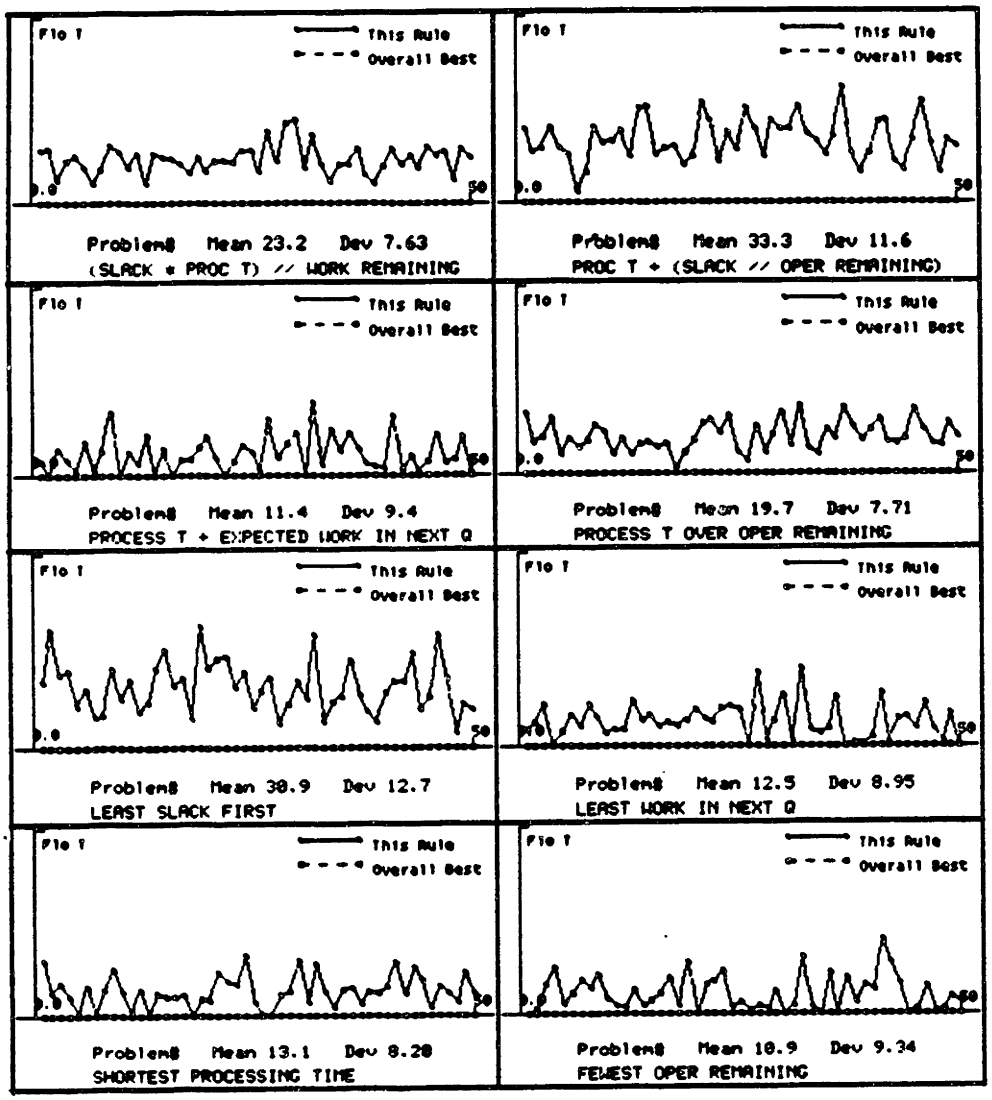


Figure 5.4: Relative performance under the flowtime criterion (page 2 of 2)

the outer regions of the space until a tube containing a single *Early Start Time* schedule remains. This method is relatively robust, because as the final tube shape is approached, there is a high density of low makespan schedules contained within the tube. This is not the case under the flowtime criterion. A high density of low flowtime schedules does not exist within such a tube. This accounts for the way heuristic H2 performs under the flowtime criterion.

Shown in Figures 5.5 and 5.6 is a statistical reduction performed on the experimental results which were shown in Figures 5.1 through 5.4. Each horizontal bar represents a 95% confidence interval for the expected average difference in objective function value over 50 $\{10, 10, \text{Gen}, \}$ problems. In these problems, all jobs arrive at the shop simultaneously. The heuristic and the dispatch rules were each used to generate a single solution to each problem. A random dispatching rule was run 500 times on each problem and the best values of makespan and flowtime were saved. In general, the schedule which had the best makespan was not the same one as had the best flowtime. For this reason, the random dispatching rule has an advantage over the rules which generate only one solution. Additionally, an active schedule generator was run 500 times on each problem and the best values of makespan and flowtime were saved.

As can be seen from the plots, the heuristic outperforms the dispatch rules under the criterion of makespan and comes close the the best makespans found by the randomized search technique. Under the criterion of flowtime, the rules which are based on amount of work remaining seem to do well.

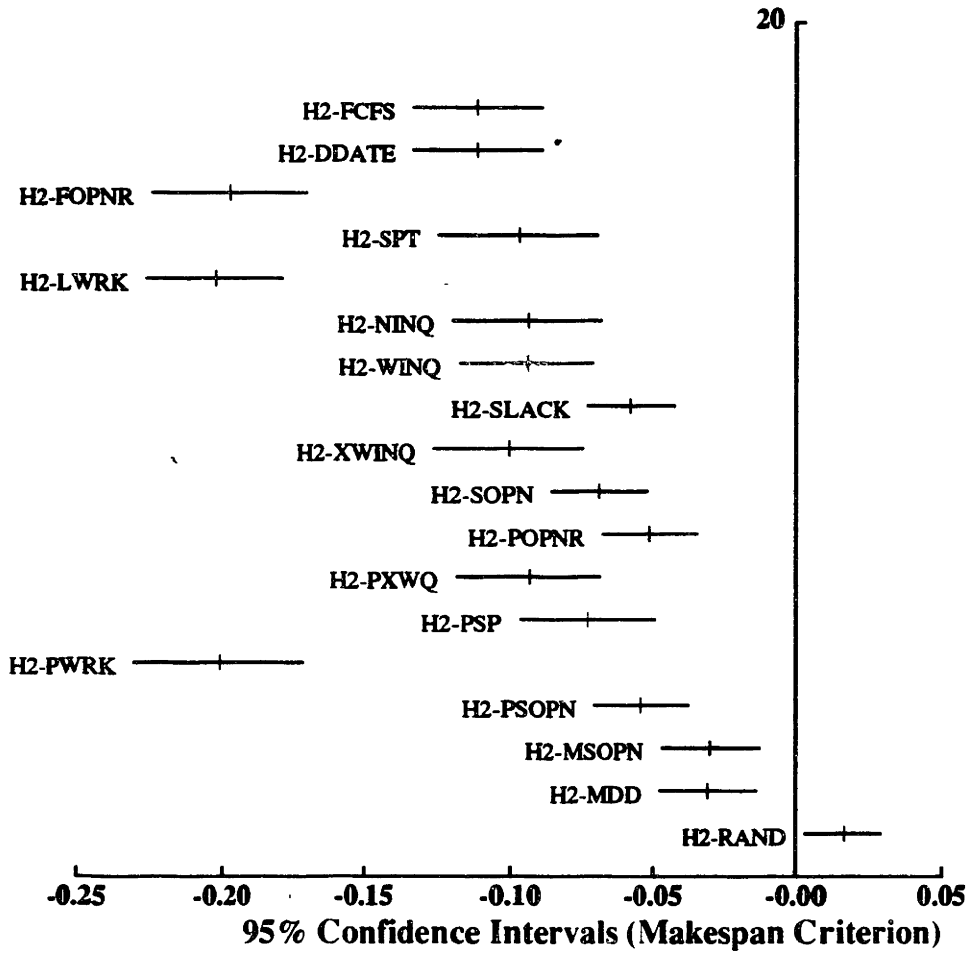


Figure 5.5: Comparison of H2 with Dispatch Rules under the makespan criterion

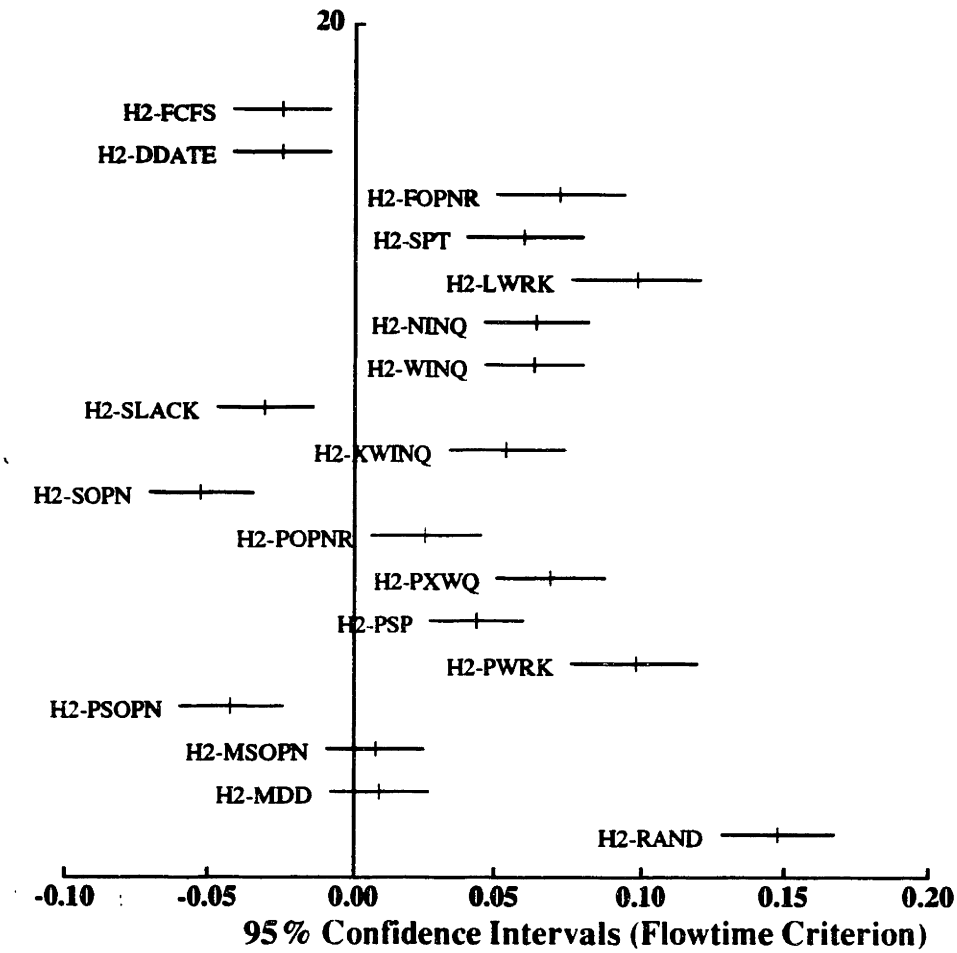


Figure 5.6: Comparison of H2 with Dispatch Rules under the flowtime criterion

5.5 H2 vs. Priority Dispatching on Problems with Due Dates

Another similar set of problems were explored. In these problems, arrival dates and due dates were imposed such that each subsequent job arrived 20% later than and was due 20% later than the previous job. Jobs were 10 tasks long with processing times chosen from a flat distribution in the range from 100 – 1000. The first job arrived at time 0, the second at time 1000, the third at time 2000 and so on. Shown in Figures 5.7 through 5.14 is the relative performance of heuristic H2, the priority dispatch rules, and two search strategies under the optimality criteria of max lateness, tardiness, makespan, and flowtime.

Figures 5.15 through 5.18 show statistical reductions of the results of the experimental data. Each horizontal bar represents a 95% confidence interval for the expected average difference in objective function value over 50 {10, 10, (Gen,Due-Date), } problems. In these test problems, heuristic H2 outperforms all of the dispatch rules and the two random searching strategies under the criterion of minimum maximum lateness (L_{max}). Heuristic H2 performs well above average under the two criteria of tardiness and makespan, and performs above average under the flowtime criterion.

5.6 Dimensionality Increasing

This experiment is designed to show how the performance of heuristic H2 varies with increasing problem size. This comparison would be easy to make if optimal solutions to all of the problems existed. For small problems an active schedule generation scheme [46] is used to exhaustively search for an *optimal* solution against which to

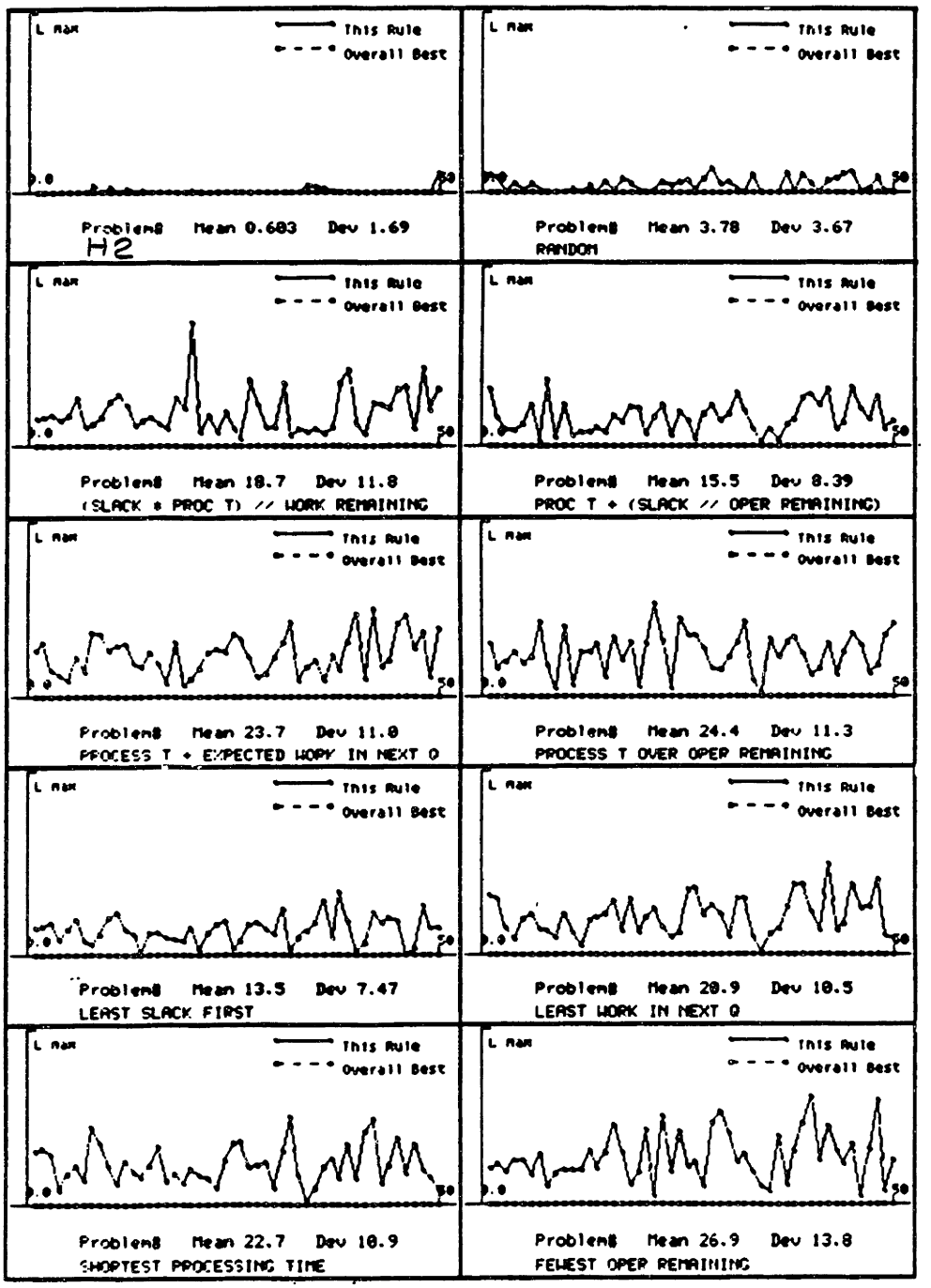


Figure 5.7: Relative performance under the max lateness criterion on problems with due dates (page 1 of 2)

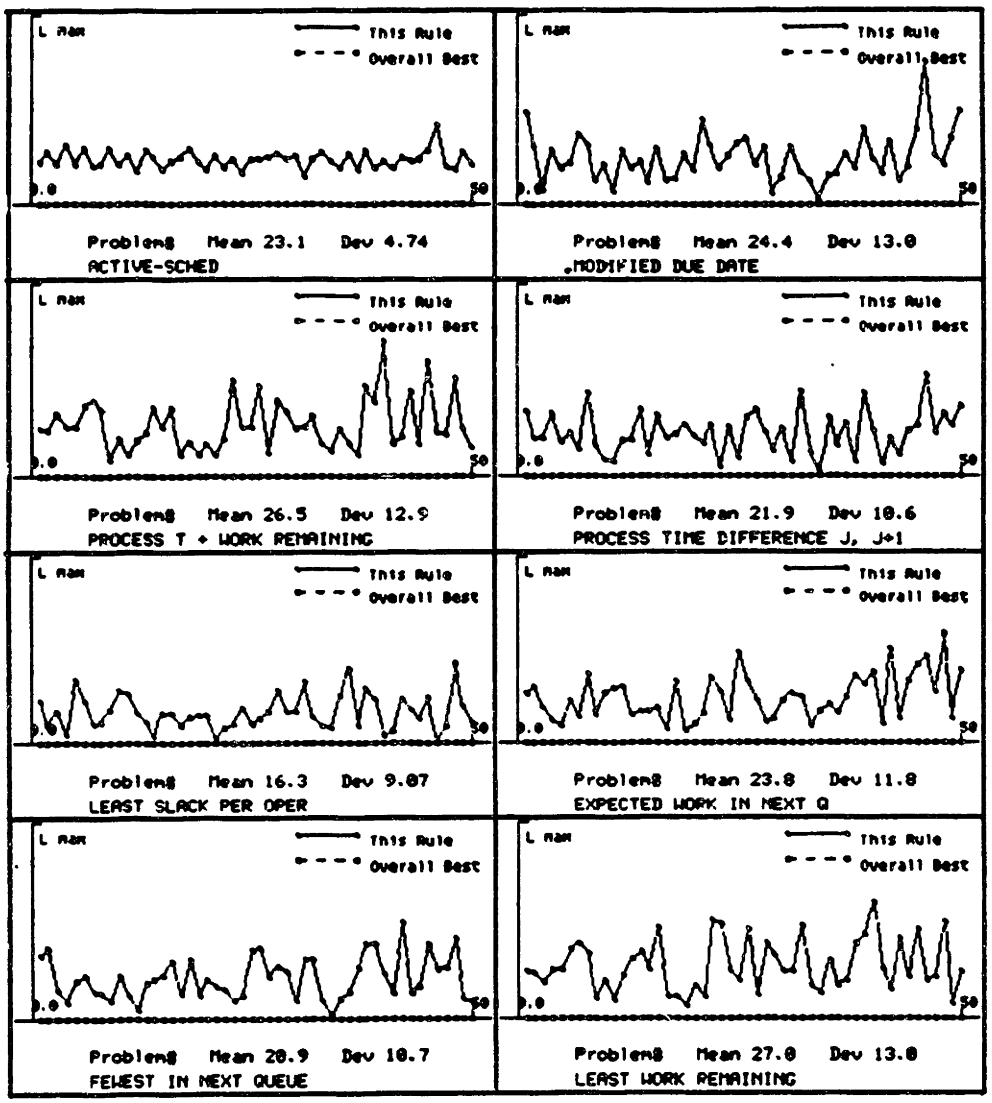


Figure 5.8: Relative performance under the max lateness criterion on problems with due dates (page 2 of 2)

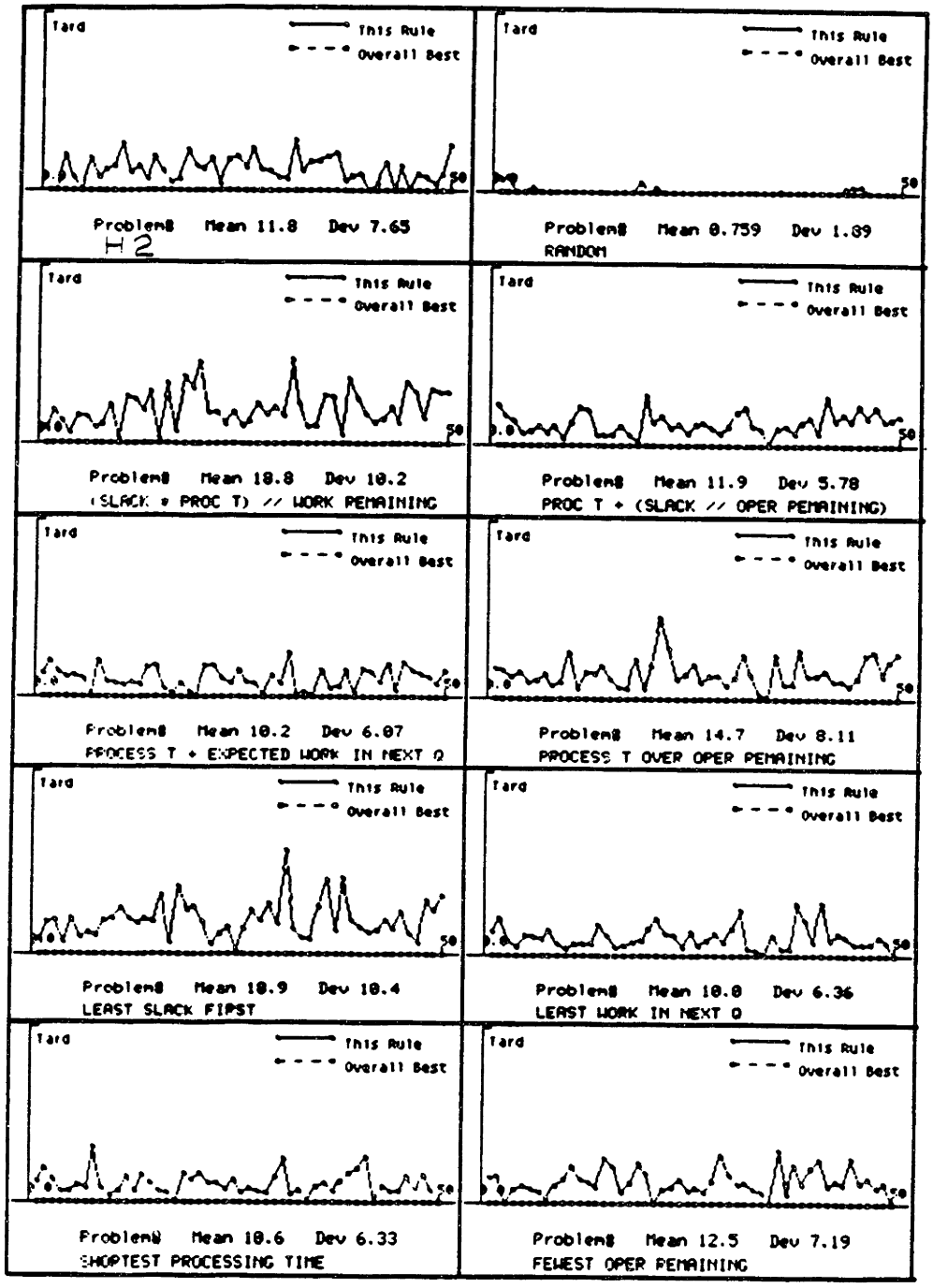


Figure 5.9: Relative performance under the tardiness criterion on problems with due dates (page 1 of 2)

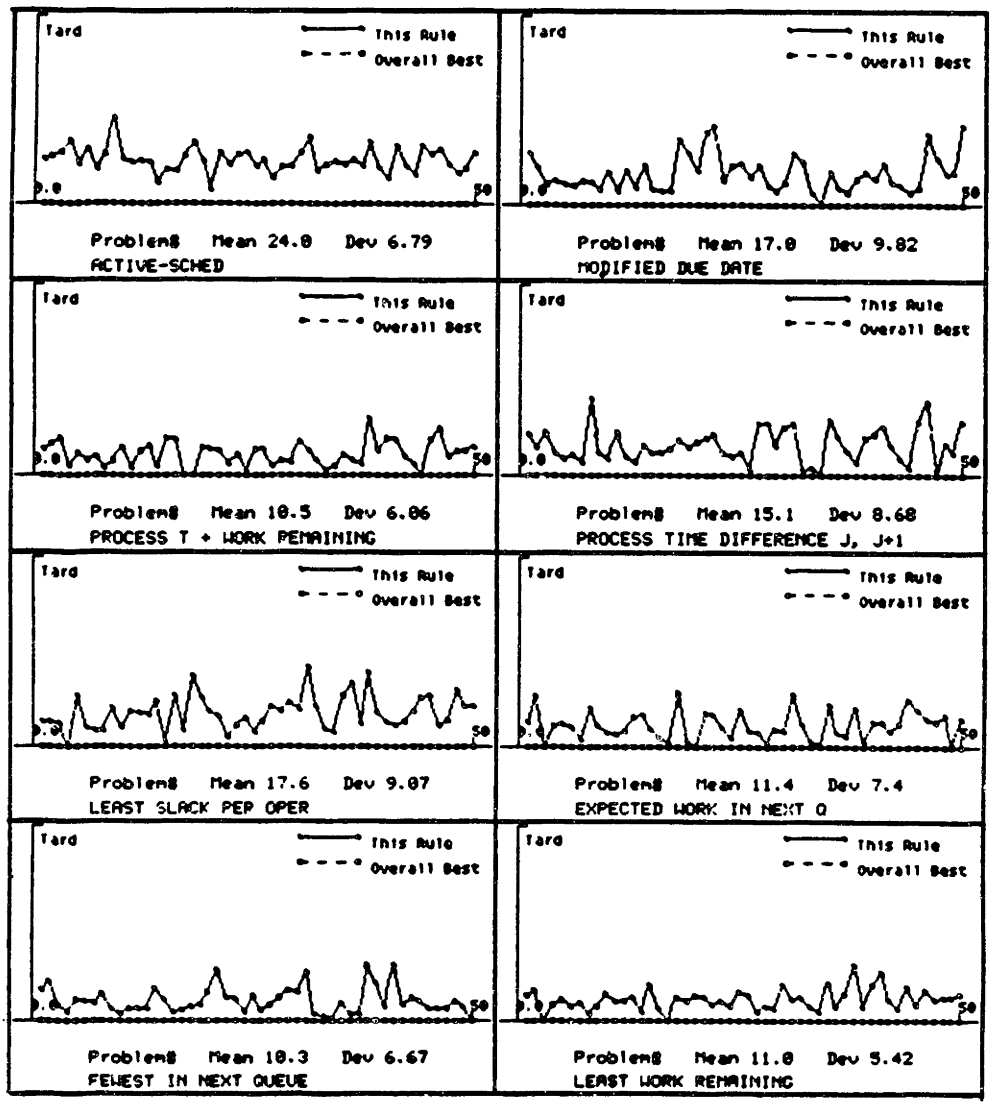


Figure 5.10: Relative performance under the tardiness criterion on problems with due dates (page 2 of 2)

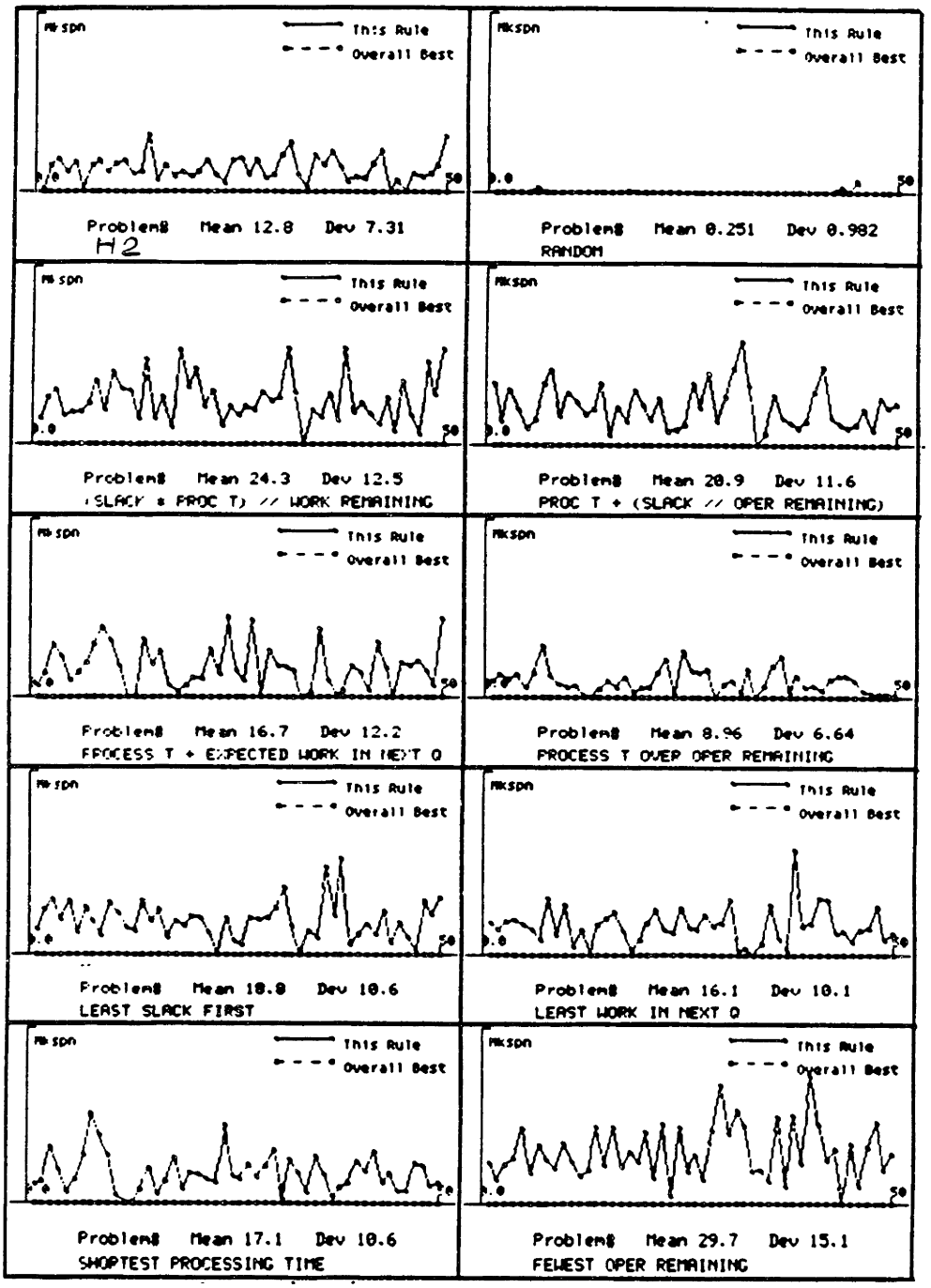


Figure 5.11: Relative performance under the makespan criterion on problems with due dates (page 1 of 2)

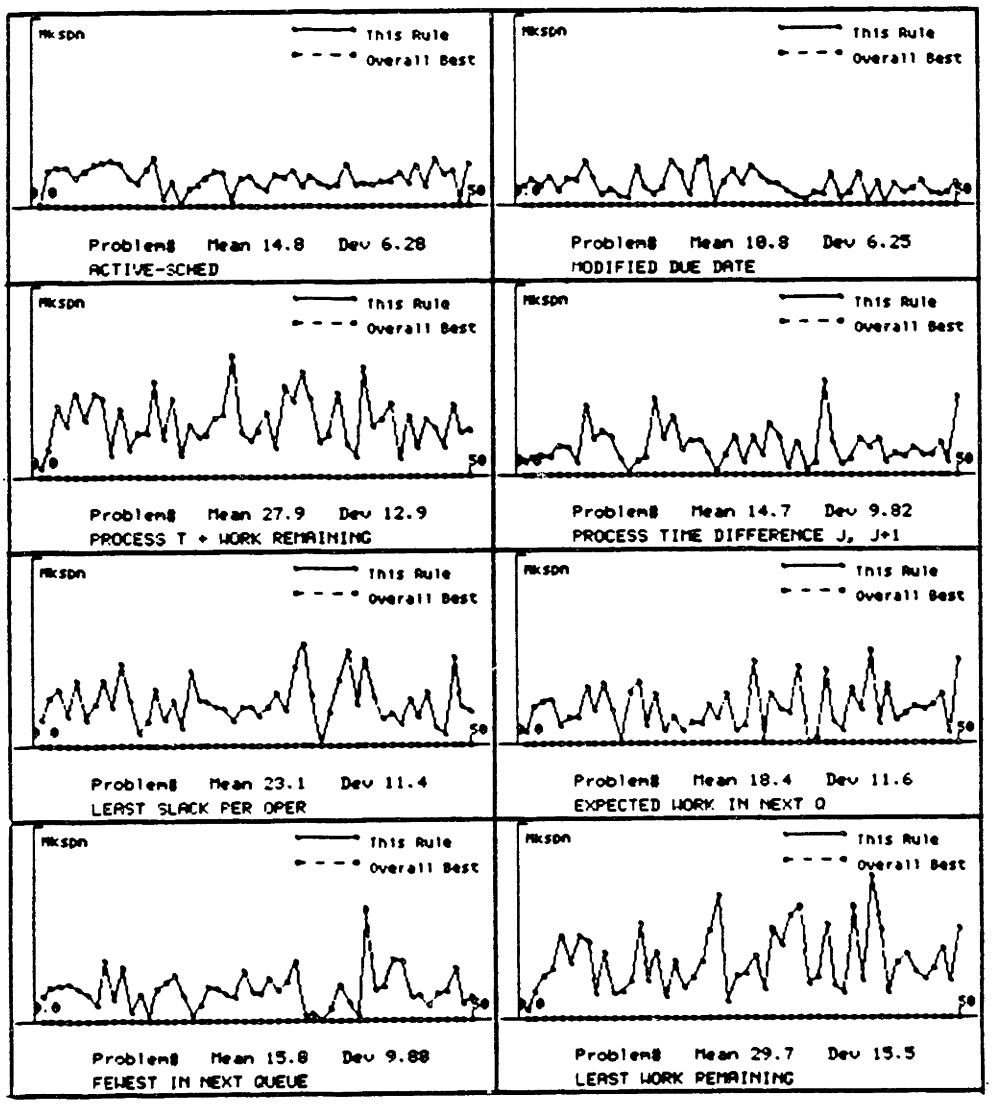


Figure 5.12: Relative performance under the makespan criterion on problems with due dates (page 2 of 2)

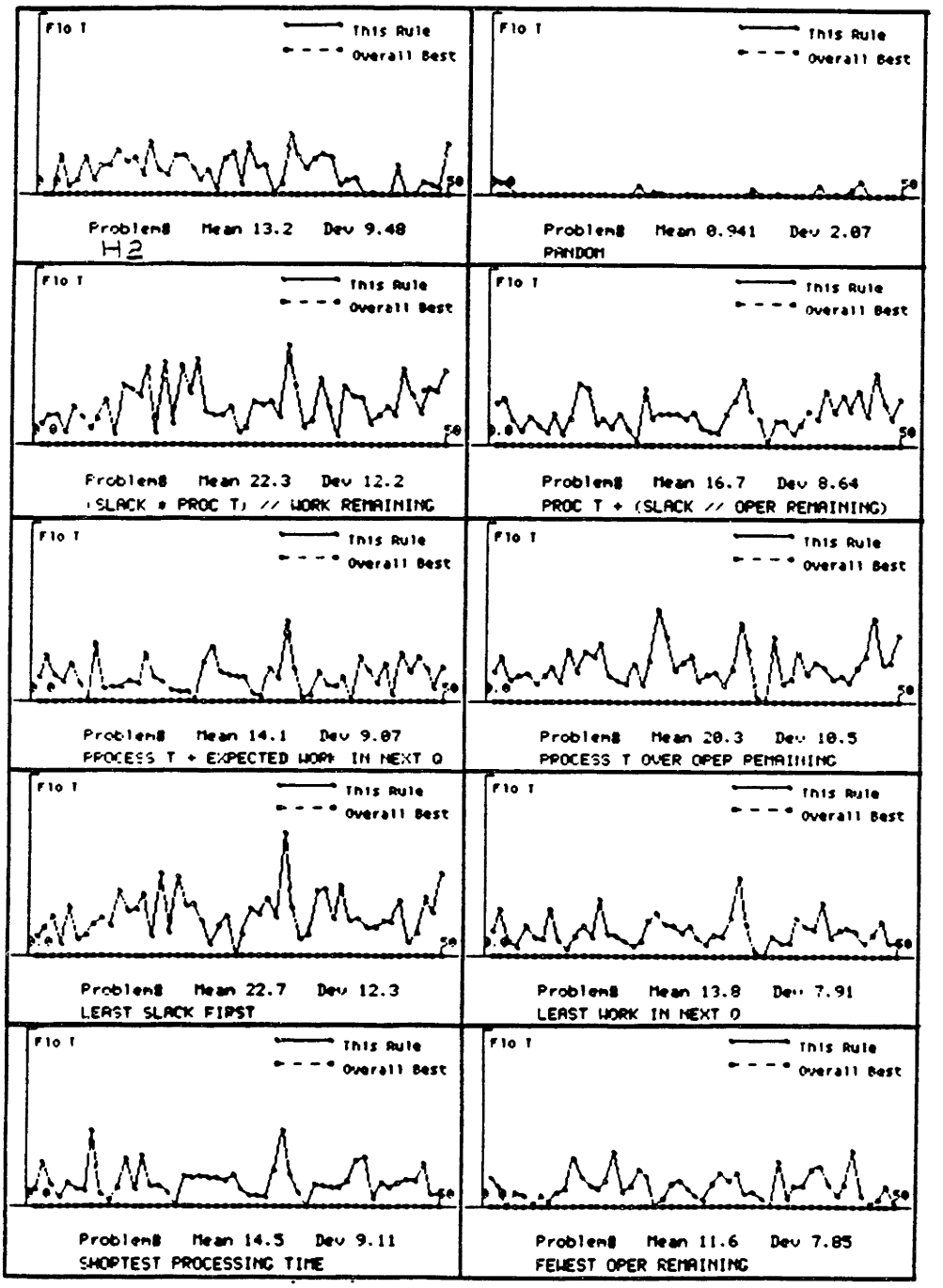


Figure 5.13: Relative performance under the flowtime criterion on problems with due dates (page 1 of 2)

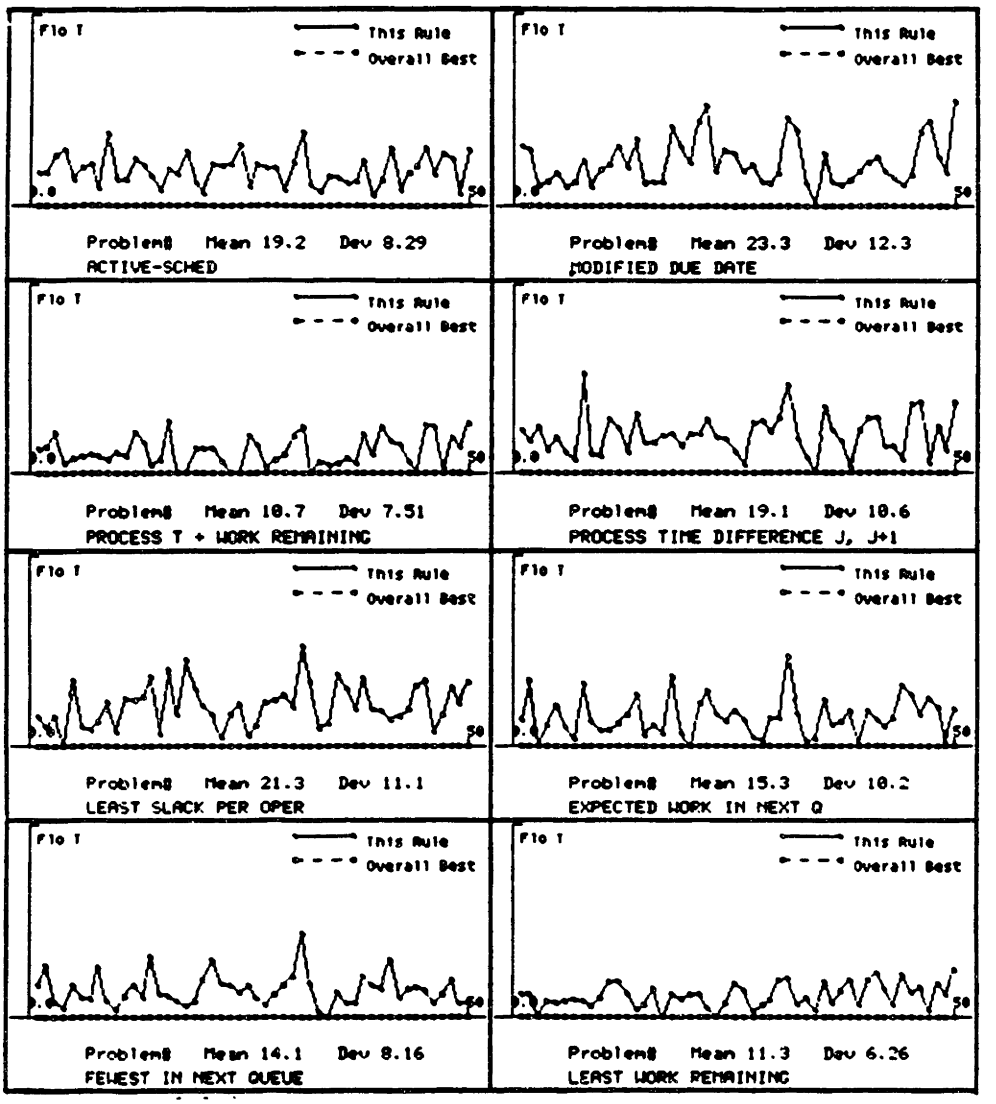


Figure 5.14: Relative performance under the flowtime criterion on problems with due dates (page 2 of 2)

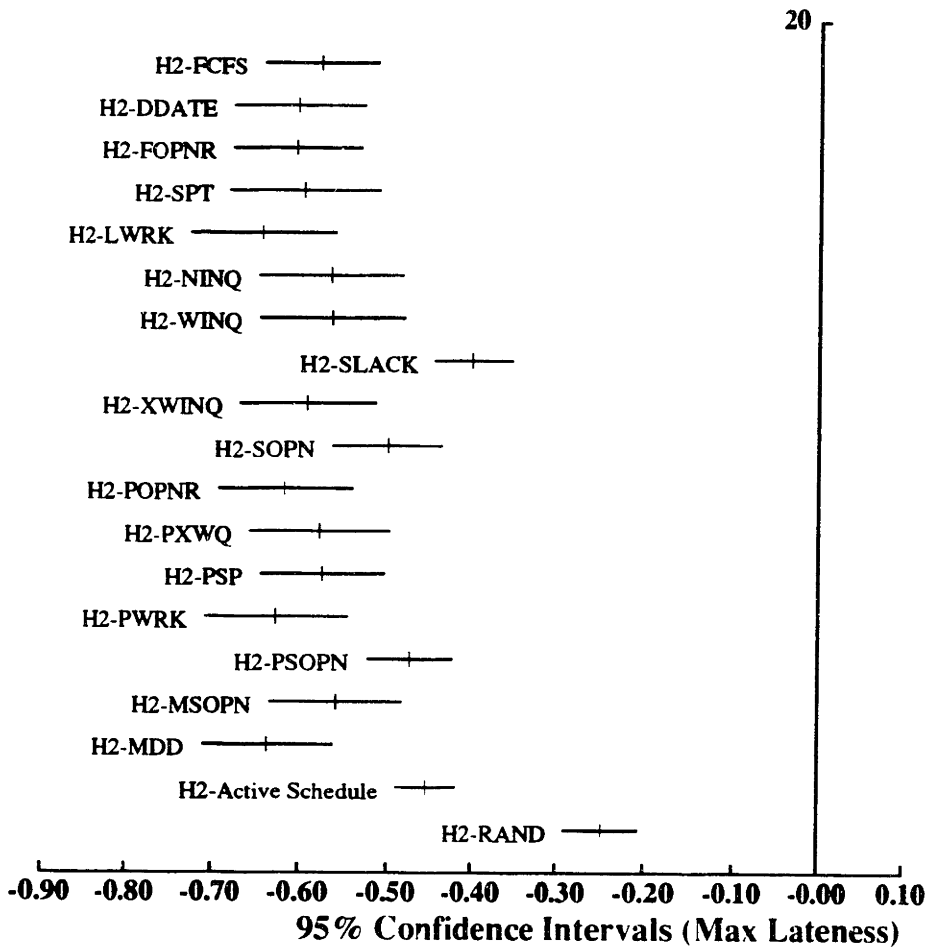


Figure 5.15: Comparison of H2 with Dispatch Rules under the max lateness criterion in problems with due dates

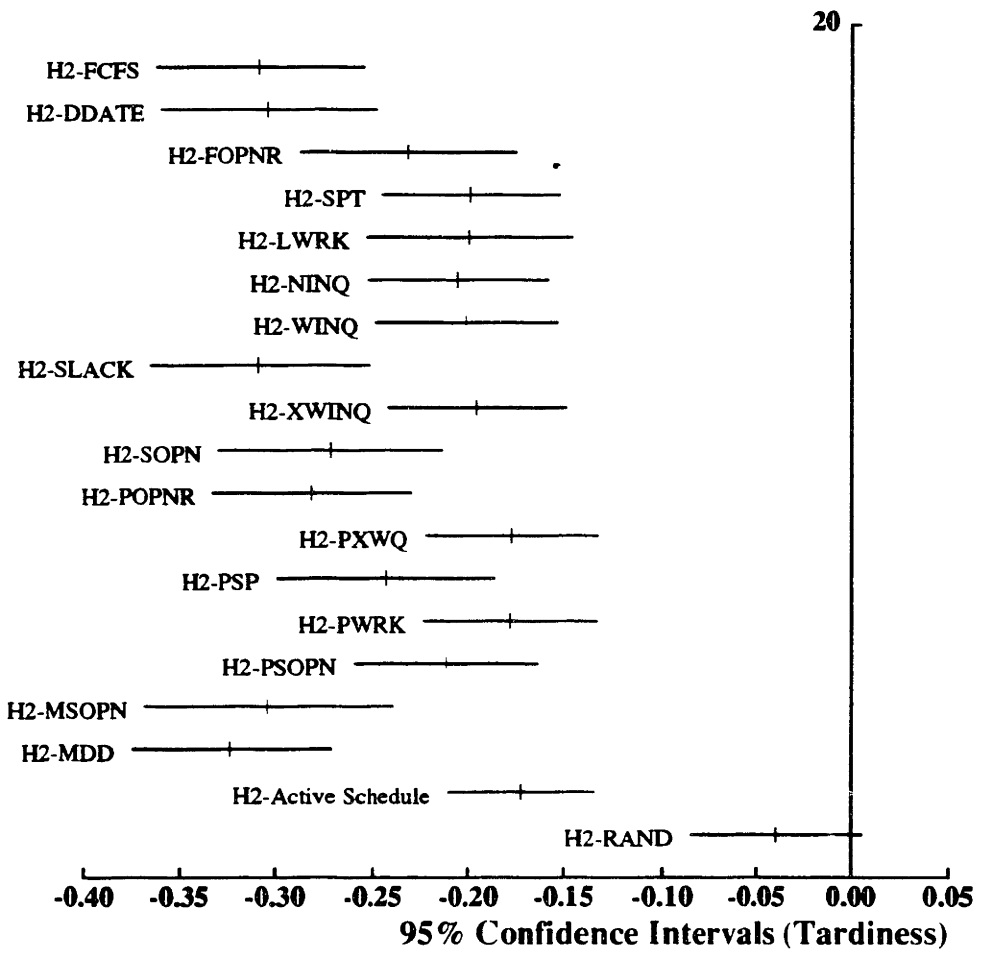


Figure 5.16: Comparison of H2 with Dispatch Rules under the tardiness criterion in problems with due dates

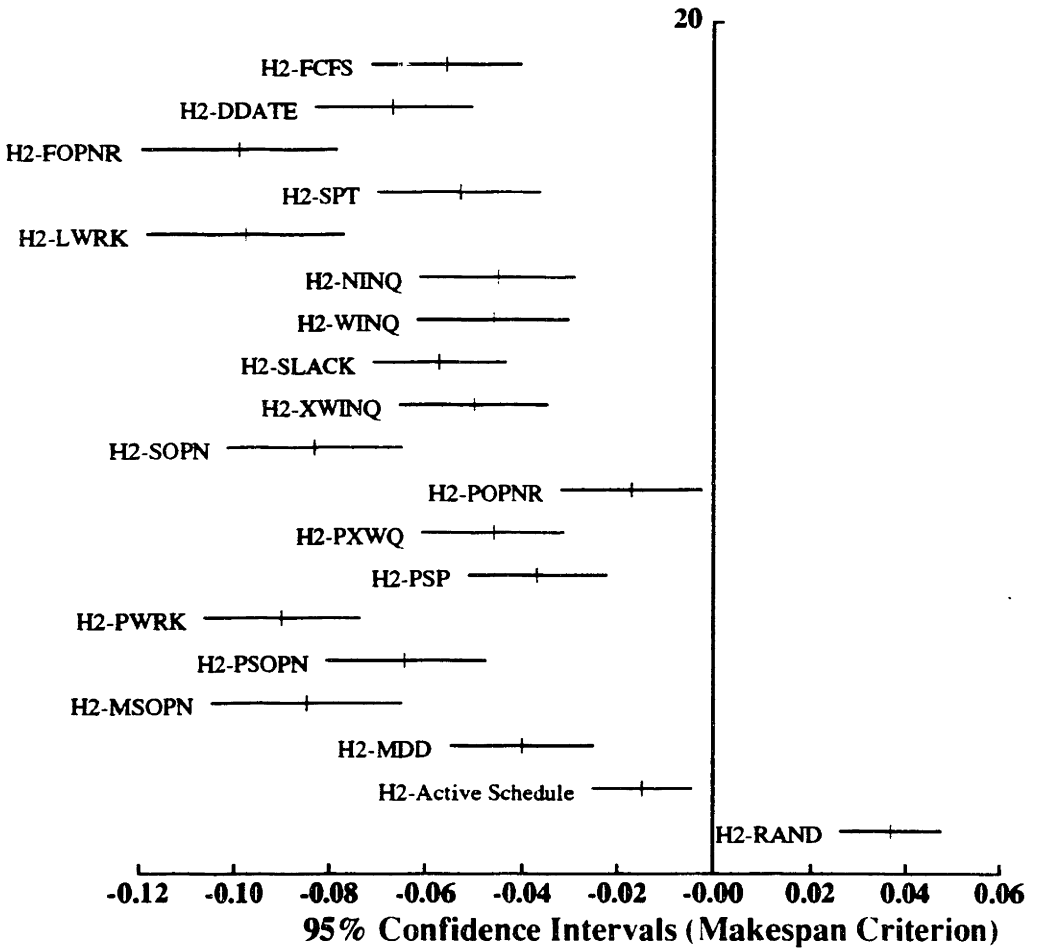


Figure 5.17: Comparison of H2 with Dispatch Rules under the makespan criterion in problems with due dates

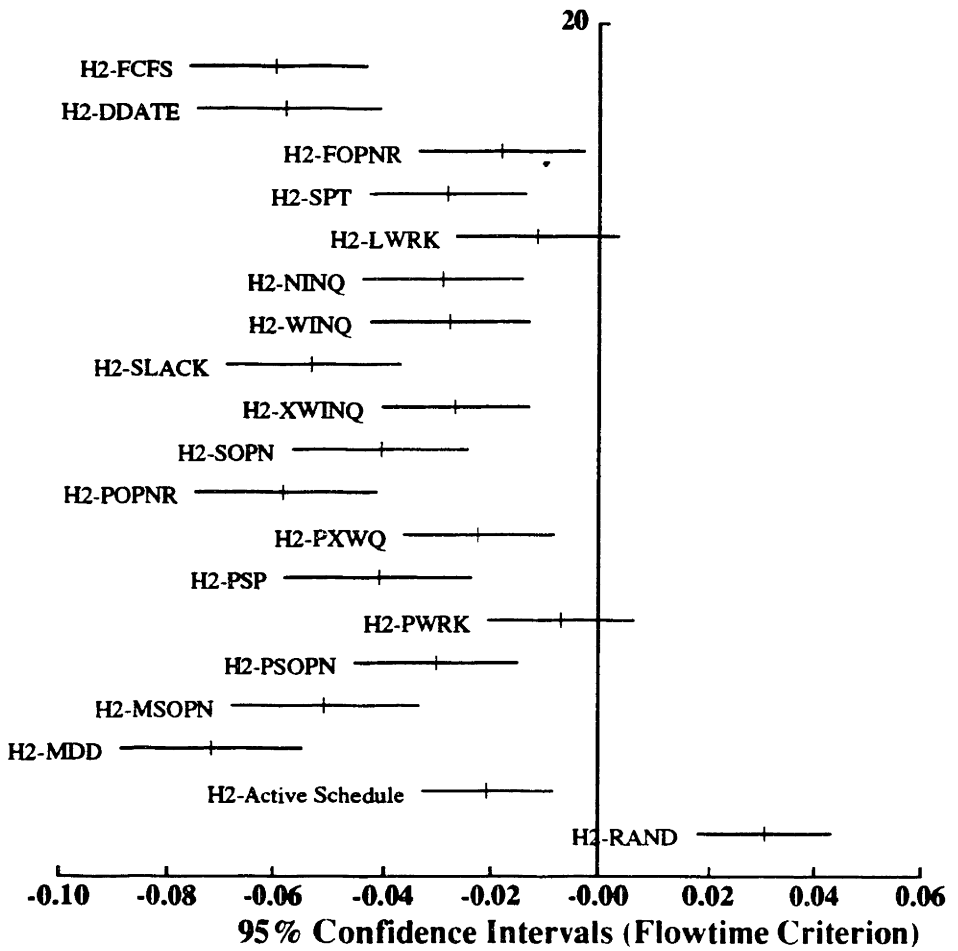


Figure 5.18: Comparison of H2 with Dispatch Rules under the flowtime criterion in problems with due dates

compare the heuristically obtained one. The maximum size problem which can be comfortably enumerated this way is $\{4, 4, .\}$.

For medium to large sized problems the solution spaces will be characterized by examining distributions of solutions obtained by generating 500 sample schedules using the *Random* priority dispatch rule. This sampling is a search technique which if continued for long enough would find the optimal solution. Here, 500 sample schedules were generated for each problem and the best one selected for comparison. See Appendix C for a discussion of the relative merits of sampling procedures.

Shown in Figure 5.19 is a characterization of how the quality of solutions generated by heuristic H2 vary as problem dimensionality is increased. Looking at the bottom of the figure, heuristic H2 found the optimal solution for all 50 of the $\{2, 5, \text{Gen}, \text{Makespan}\}$ problems. For the $\{3, 3, \text{Gen}, \text{Makespan}\}$ problems, heuristic H2 found the optimum 49 times out of 50. For the $\{4, 4, \text{Gen}, \text{Makespan}\}$ problems, the heuristic found the optimum 41 times out of 50. For the $\{6, 6, \text{Gen}, \text{Makespan}\}$ problems, the heuristic found the best solutions to 22 of the problems while the Random search found 32 of them (4 ties). And finally, for the $\{10, 10, \text{Gen}, \text{Makespan}\}$ case, heuristic H2 found the best solution 23 times, while the Random search found the best solution 27 times (no ties). When the heuristic failed to find the best solution, the deviation from best was greater than when the Random search failed to find the best. In summary, the makespans of schedules generated by the heuristic seem to get farther away from optimal as dimensionality of the problem increases, yet it still outperforms the dispatch rules under the criterion of makespan.

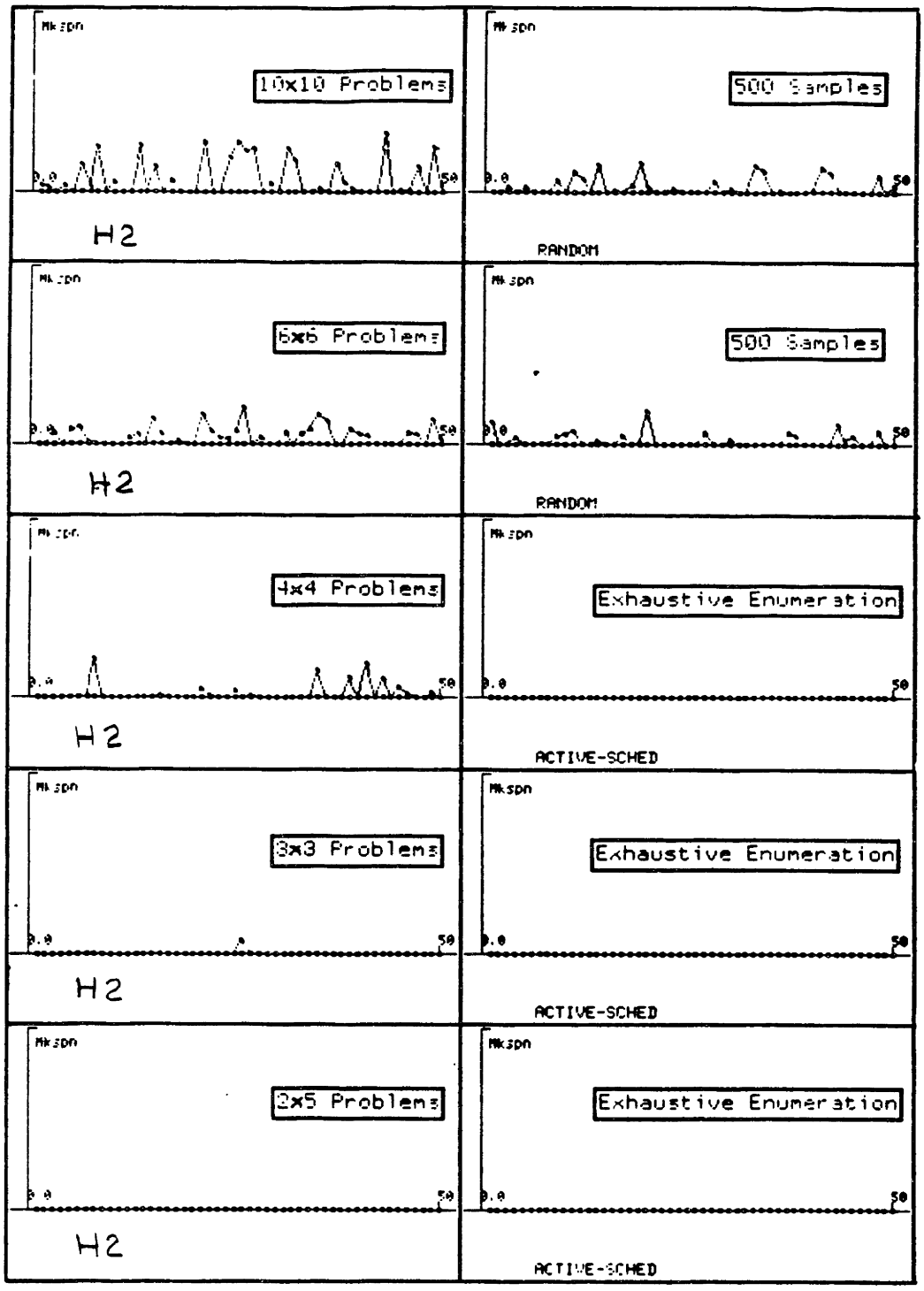


Figure 5.19: Performance of heuristic H2 as problem dimensionality is increased

Conclusions

Chapter 6

Visualizing algorithms in cartesian completion space helps to show some of the similarities of existing algorithms that would otherwise not be apparent. If the problem was represented as sequences, disjunctions or the vertices of a polytope the similarities of the pruning methods would be obscured. Furthermore, insights gained as to the location and adjacency of solutions helps to explain the performance characteristics of certain algorithms.

Heuristic H2 generates significantly better schedules than the priority dispatch rules on the *Job-Shop Scheduling Problem* when the optimality criterion is L_{max}. Using heuristic H2 to generate a solution to the famous 10x10 Job-Shop Problem in [47] results in a makespan of 985. The optimal makespan for this problem is known to be 930. A recent iterative relaxation procedure ("Shifting Bottleneck Procedure" [1]) SBI gives a makespan of 1015 for this problem.

The choice of optimality criteria affects the relative location of "good" solutions in the search space. In particular, under the *Makespan* criterion, the trajectories corresponding to "good" solutions were found to be geometric neighbors in the com-

pletion space. A large fraction of the "good" solutions were found to lie within a "tube" through the space. Under the *Flowtime* criterion, the trajectories corresponding to "good" solutions do not lie within a "tube" consequently an algorithm which tends to converge upon such a "tube" has little chance of succeeding. Therefore, the relationships of the locations of the "good" solutions in the solution space should be taken into account during algorithm design.

Typically, some natural (in the problem formulation) constraint (e.g. a disjunction or a time ordering) presents itself and is used as the basis of a solution generation process. There is no *a priori* reason to believe that using this particular process to prune the solution space will yield the desired result. A possible consequence of this is that many of the "good" solutions will be unavoidably pruned as the process proceeds. Two possible results of this follow. By throwing away some of the "good" solutions, it is now more difficult to continue the search. And, deciding which part of the space to prune may be more difficult as the properties of the set under consideration are less uniform thereby making the calculation of bounds more difficult.

Heuristic H1/CT can be used to test whether or not some set of constraints under consideration are capable of pruning with discrimination; that is, to selectively prune the space such that only the more desirable regions remain. This constraint testing can be accomplished without developing branching indicators and a lower bound (some necessary components for a branch and bound algorithm).

Symmetries in problem structure affect the spatial distribution of the trajectories corresponding to good schedules and should be taken into account during algorithm design. Under the L_{max} criterion, many "good" solutions were found to be adjacent in the cartesian space. Symmetries introduced into the problem cause certain "mirror image" trajectories to be equivalent. This makes it difficult to converge to a

relatively small region containing a set of "good" solutions. Therefore, either the set of constraints used should be capable of describing these symmetries, or they can be implicitly taken care of by having the resulting solutions map to other solutions via the symmetries.

Suggestions for Future Work

Chapter 7

7.1 Modify Heuristic H2

In heuristic H2, the constraint used to prune the space was the addition of a precedence constraint. This corresponds to the removal of a "2D corner" in completion space . In a 2D case, "removing a 2D corner" deletes the entire associated obstacle. In a 3D case, "removing a 2D corner" deletes 4 out of 6 of the bar shaped extensions of the obstacle shown in Figure 4.4.

The proposed modification is to use constraints which remove less than 4/6 of the bar shaped extensions of the obstacles ("a 3D corner"). The remainder of the obstacles is left for later consideration. Instead of pruning by adding a single precedence constraint, prune by imposing a boolean combination of precedence constraints. Given three activities which correspond to an obstacle consider the addition of constraints of the form $(A_i \prec A_j) \vee (A_i \prec A_k)$, $(A_j \prec A_k) \vee (A_j \prec A_i)$, $(A_k \prec A_i) \vee (A_k \prec A_j)$. These correspond to the three ways of deleting a "3D corner".

As in heuristic H2, choose the constraint who's compliment, if imposed, would

result in minimum slack in the resulting network. The compliments of the three constraints above are $(A_j \prec A_i) \wedge (A_k \prec A_i)$, $(A_k \prec A_j) \wedge (A_i \prec A_j)$, and $(A_i \prec A_k) \wedge (A_j \prec A_k)$. Implementing this would involve some method of encoding these constraints, and keeping track of the implications of their combinations.

In general, for an nD problem one could consider deleting an "nD corner", an "$(n-1)$D corner" and so on down to one plus the number of available machines of a given type. When there is more than one of each machine, say m, the lowest order constraint which need be considered is of the form $(A_i \prec A_j) \vee (A_i \prec A_k) \vee \ldots \vee (A_i \prec A_{i+m})$.

Note that this approach reduces to heuristic H2 when the simplest possible constraints are used; if $(A_i \prec A_j)$ generates the longest path then impose the constraint $(A_j \prec A_i)$.

One possible approach to utilizing a mixture of these constraints would be to consider addition of the highest dimensional constraints first, then those of the next highest dimension, down to the lowest order ones which need to be considered. Alternatively, one could consider starting from 1 or more dimensions higher than the lowest order ones which need to be considered.

7.2 Develop New Heuristics Modeled On Heuristics H1/CT and H2

Heuristic H1/CT uses the addition of selected precedence relations to prune undesirable regions of the search space. These selected precedence relations are capable of describing a set of relatively good minimum maximum lateness schedules. Under other objective criteria the distribution of good schedule trajectories through the

space cannot be described using such a set of selected precedence relations. This was demonstrated for the flowtime criterion.

Flowtime is minimized when there is a minimum of delay between successive activities in a job. By first processing all of the activities job 1, then all of the activities of job 2 and so on, one could generate a schedule of minimum flowtime. This schedule corresponds to a trajectory which traverses only the edges of the space. The addition of precedence relations tends to prune away the outside of the space and converges towards a tube, the opposite of what is needed under the flowtime criterion. Alternative sets of pruning constraints need to be proposed and tested. One method of pruning would be to remove solid regions from the interior of the space ("hollow it out" so to speak) in conjunction with some trimming from the outside. This type of pruning would tend to converge toward a shell like region.

Heuristic H2 uses the "length" of certain trajectories through the space as the basis of the pruning operations. These lengths were computed by summing up for each piecewise linear segment of the trajectory, the maximum of the segment's projections on to the coordinate axes. This is only one of many possible metrics that could be used to assign weights to trajectories in the space. The particular "length" metric used by H2 is effective when the objective criterion is that of minimum maximum lateness. Perhaps alternative metrics could be used by heuristic H2 to find good solutions under other objective criteria. An example alternative metric would be to weight the processing time of each activity (stretch the length along the coordinate axis associated with processing the given activity) by a work in process inventory cost.

7.3 Modify Heuristic H1/CT

Heuristic H1/CT operates by pruning as much of the search space as possible without eliminating the k best schedules found so far. It may be the case that potentially good regions of the space are being pruned before they are even explored. Modify the heuristic so that regions of the space are pruned only if both of the following conditions hold. First, as before, never prune regions containing the k best schedules. Second, only prune regions which are known (from the samples taken so far) to contain inferior schedules.

7.4 Veronoi Approach

One way of approaching a scheduling problem is to convert it into a shortest path problem and then solve it with some known optimization algorithm. Davis [25] did this with a certain class of resource constrained PERT network problems. To transform a PERT network first divide each activity into a chain of unit processing time activities (this is contingent upon having only integer processing times) inserting the necessary precedence relations between the unit activities. Any cut across this network corresponds to some (integer) feasible partial state of completion of the original tasks. Form a new network with these feasible states of completion as nodes. The "distance" between any two adjacent nodes is one time unit. Find the shortest path from start to end using a dynamic programming approach.

There are two problems with this. One, the activities must be of integer length. Two, the activities cannot be very long and there must be a fairly large ratio of precedence-relations to activities or else there will be a large number of partial states of completion generated. In other words, a combinatorial problem in n tasks can be

transformed to a polynomial time problem in the number of nodes but the problem happens to have an unacceptably large number of nodes.

The proposition is to generate a network which has the least possible complexity, yet has the property that any valid trajectory can be deformed to coincide with the network. The problem then reduces to searching this network for the optimal path. Consider the 3D cross shaped obstacle shown in Figure 4.4. If this was the only obstacle in the space, then the desired network could be defined as follows. Start with the rectangular network formed by the edges of the center of the 3D obstacle. Then attach two additional edges; one from the origin of the space to the closest vertex of the rectangular network; and one from the far corner of the space to the closest vertex of the rectangular network. If one deformed this network such that its edges were equidistant from the obstacle and the faces of the space then it would form a Veronoi diagram.

In this 3D case there are 14 edges, 12 of which are associated with the rectangular solid center of the obstacle and 2 connecting it to the rest of the space. In general there are $(n/2)2^n$ edges on an n dimensional rectangular solid. There are 2^n vertices from each of which emanates n edges. Each edge is connected to two vertices hence the factor of 1/2. For a $\{10, 10, \text{Flow},\}$ problem the desired network is formed by the edges around the center of each of the 10 10D obstacles. Since this is a flow type problem, the obstacles line up along the body diagonal of the 10D space. This total number of edges in this case is $10(10/2)2^{10} = 51{,}200$. This is a relatively small number of edges to search considering an upper bound on the number of distinct sequences associated with this problem is $(10!)^{10} = 3.96E + 65$. For the *General* structure problem in which the obstacles do not line up along the body diagonal, additional edges must be found to connect the various rectangular networks.

7.5 Normalize Path Probabilities For Random Sampler And Active Sampler

The random and active samplers that I have been using have a tendency to generate anti-symmetric schedules (when the symmetry considered is the inversion of all precedence relations in the problem specification). The solution sequences of this inverted problem should be the same as the original problem. Using either of these sampling schemes produces the various sequences with different probabilities for the inverted problem versus the non-inverted problem.

The source of this discrepancy can be visualized on a rectangular 2D grid. Here a sequence corresponds to a random walk from the lower left of the grid to the upper right, with the steps taken either one to the right or one up. The probabilities associated with the branches at each branch point are equal.

As an extreme case consider a 1 (vertical step) x 100 (horizontal steps) grid. There are two ways to leave the origin each with 1/2 probability. At the other end there are two ways to enter the destination. Note, that the probability of entering the destination from the bottom is $(1/2)^{100}$, a very small number. However, in the inverted problem, the probability of using this particular link is 1/2.

A normalization scheme wherein the probability of taking a particular branch is equal to the number of steps left in the branching direction divided by the total number of steps left to take resolves this anti-symmetric problem.

The *grid* associated with an actual scheduling problem is by no means a regular grid as assumed in the above example; it may be close enough such that this scheme might do some good. It seems plausible, in that when considering which task to schedule next, higher probability is given to the task which has many tasks

constrained to follow it. This should reduce the probability of generating a schedule in which one "job" is excessively delayed.

7.6 Cyclical Schedule Formulation

A slight modification of the heuristic might make it suitable for use in cyclical scheduling problems. In these problems (described in [50]), the same mix of activities is to be processed over some time period (e.g. daily) and there may be some partial ordering among these activities. The objective is to find an ordering of activities which is of minimum cost over the given time period when the Gantt chart of the final schedule is repeated periodically.

In the 2 job case, the usual 2D space is wrapped around a torroid as shown in Figure 7.1. A valid schedule corresponds to a continuous trajectory which avoids the obstacles while making 1 twist around the major axis and one twist around the minor axis.

Instead of computing the slacks between activity pairs by propagating arrival and due date information through the activity network, propagate information from one of the tasks under consideration around the network (consider the precedence network to be wrapped around a cylinder - head touching tail) to the other task in question. The slack between two tasks is how far apart two tasks can be displaced. The rest of the algorithm would remain the same.

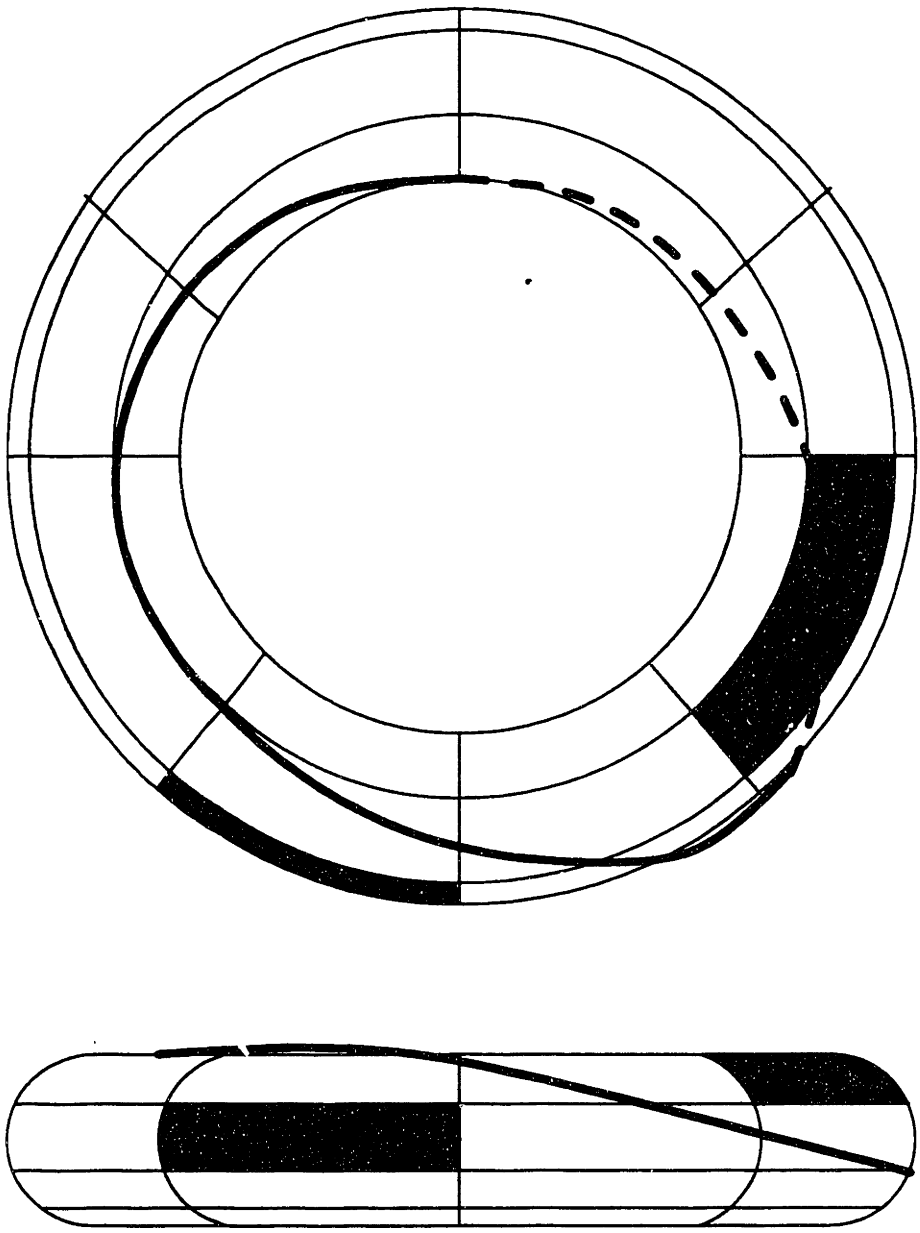


Figure 7.1: Torroidal Completion Space

7.7 Representation Transformation

Many algorithms that have been developed use as a basis of decision the setting of a precedence. Each decision can be interpreted as selecting a certain subset from the original set of solutions. Depending upon the optimality criterion used, the rejected subset may contain a disproportionate number of the *good* solutions.

It is proposed to use some transformation of the original precedence constraints which allows solutions which are similar in value to be in some sense adjacent (e.g. Adjacent integers along the number line differ by only a single bit when represented in Grey Code but may differ by many bits when represented in straight binary.) Hopefully, this would improve the performance of branch and bound algorithms by allowing easier characterization or bounds to be computed as the subsets under consideration have more uniform properties. In the case of relaxation algorithms, since *good* solutions are *adjacent*, local neighborhood type searches should yield improved solutions.

Consider representing a sequence as a binary string where each digit corresponds to the orientation of a precedence relation. In general, the *good* schedules will be scattered between 0...0 and 1...1. Another way of stating the goal of solving one of these problems is to find a 1 to 1 (bijective) mapping from the original binary strings to a set of string in which 0...001 corresponds to the best sequence, 0...010 to the next best and so on. Then, to find the *good* schedules simply map back starting from 0...001 and counting up.

A fully general mapping function should be able to map any element from the first set to any arbitrary element of the second set. There are 2^n elements in the first set (n digits), and therefore $(2^n)!$ possible transformations.

7.8 Learning Boolean Functions/Transformations

Pose the scheduling problem as one of learning the characteristics of a *good* schedule by observation and experimentation. Get some initial examples to learn from via sampling. Formulate some hypothesis in some domain of characteristics being considered. Then do planned experiments. This is different from the usual learning by example because it allows one to immediately test a current hypothesis instead of waiting for a counterexample to be presented.

Free Space Fraction

Appendix A

The following is an analysis to show what fraction of the completion space is occupied by obstacles. The approach is combinatorial in nature and resembles the multinomial theorem because it involves expanding an expression of the form $(a + b + c + ... + g)^n$. There are m (number of machines) terms inside of the parenthesis and the exponent n is the number of jobs. the ratio of free space (space not occupied by obstacles) to the total volume is given by the following formula assuming that all activity processing times are unity.

$$free/total = \begin{cases} \frac{(m)_n}{m^n} & n \leq m \\ 0 & n > m \end{cases} \tag{A.1}$$

The top of the first expression is read as m-falling-factorial-n which is like $m!$ but consists of only the first n terms (e.g. $(6)_3 = 6 * 5 * 4$).

A simple 3D example shows how the above expression arises. Say that we have four machines a, b, c, d. Each job visits each machine once, so for each job we can write a term of the form $(a + b + c + d)$ possibly with the elements permuted (i.e. $(b + a + c + d)$) but the order really doesn't matter under the addition operator. If

there are three jobs then we have three terms of the above form. Any rectangular block in the job-space corresponds to the processing of one activity from the first job, one activity from the second, and one activity from the third. There is a one to one correspondence between the individual terms in the expansion of $(a+b+c+d)^3$ and the 3-Dimensional rectangular blocks associated with the example 3-job problem.

Note that in the expansion there will be terms like a^3, bc^2 and abd. Only terms of the form $x^1y^1z^1$ correspond to free space. Any term with a factor's exponent greater than 1 means more than one machine of some given type is required if a schedule trajectory is to enter this particular block of the space.

The total number of blocks in this example is $4^3 = 64$. The number of free blocks can be computed by consideration of a series of choices involved in the expansion of $(a+b+c+d)^3$ which lead to terms of the form $x^1y^1z^1$. The object is to find the sum of coefficients of these terms in the final expansion. Note that this is equal to the sum of multinomial coefficients for terms of the form $a^ib^jc^kd^l$ where one of i,j,k,l is equal to 0 and the others are equal to unity. In the interest of generality, the multinomial theorem is being avoided.

Consider the following series of choices in the expansion.

Choose one term from $(a+b+c+d)$ that won't give a result with an exponent greater than 1 - there are 4 ways to do this. Choose a, b, c or d.

Next choose one term from $(a+b+c+d)$ that won't give a result with an exponent greater than 1 when multiplied by the previous choice. There are 3 ways to do this.

There are 2 ways to choose the next factor and one way to choose the last.

Then

$4 * 3 * 2 * 1/4^3 = 3/8$ is the free space fraction.

Note that if the number of choices (n = #jobs) is greater than m then an exponent greater than 1 is unavoidable - hence the $n > m$ case in the volume fraction expression.

This leads to the following question: If the volume fraction of free N-Space is 0 then what space is left in which to plot trajectories? What happens if one has 3 machines and 4 jobs. Obviously not all 4 jobs can ever be co-processed. In other words there is no free 4D space. But what about 3D subspaces. It turns out that the same formula is valid for this case also. Instead of letting n be the number of jobs, let n be equal to the dimensionality of the subspace of interest. Then for a given problem one can calculate how much of each dimensionality space is free.

Note that the above analysis was carried out for the one of each machine jobshop problem. It could also be easily modified for the case of more than one of each machine (look for terms of different form) or more than one of certain machines. Perhaps this type of calculation could be used to evaluate the usefulness of purchasing some new piece of equipment. If the new purchase would result in no increase in the free space fraction then there is no point in purchasing it. One might also consider free space increase per dollar spent as an indicator of relative merit of alternative purchases.

Coefficients could be put in front of each a, b, c etc to reflect processing times thereby improving the accuracy of the calculation. Also, instead of simply using a as a primitive term, use a term of the form aPT to represent the facts that we need machine a, person P, and tooling T to perform this operation. Of course this removes much of the symmetry from the calculations and mean that a computer might be required for the computation.

N-D Cone Fraction Derivation

Appendix B

We will now estimate the fraction of the space contained within the boundaries of a cone in n-space (completion space). Figure B.1 shows an example of a conical or pyramidal region in 3D space defined by two back to back pyramids. One has its apex at the origin (shown using dotted lines), the second has its apex at the far corner and they meet at their bases. For this example it is assumed that the cube is a units along each edge and the parameter b is allowed to vary from 0 to a thus defining the shallowness of the cone.

The volume V_{1p} of an n-D pyramid is defined as follows where B is the base and h is the height.

$$V_{1p} = \int_0^h (B)(\frac{x}{h})^{n-1} dx = \frac{Bh}{n} \tag{B.1}$$

The total volume V_t for the cube is given by a^n.

The volume of 2 back to back pyramids V_{2p} along the diagonal is

$$V_{2p} = \frac{B(h_1 + h_2)}{n} = \frac{B\sqrt{na^2}}{n} = \frac{Ba}{\sqrt{n}} \tag{B.2}$$

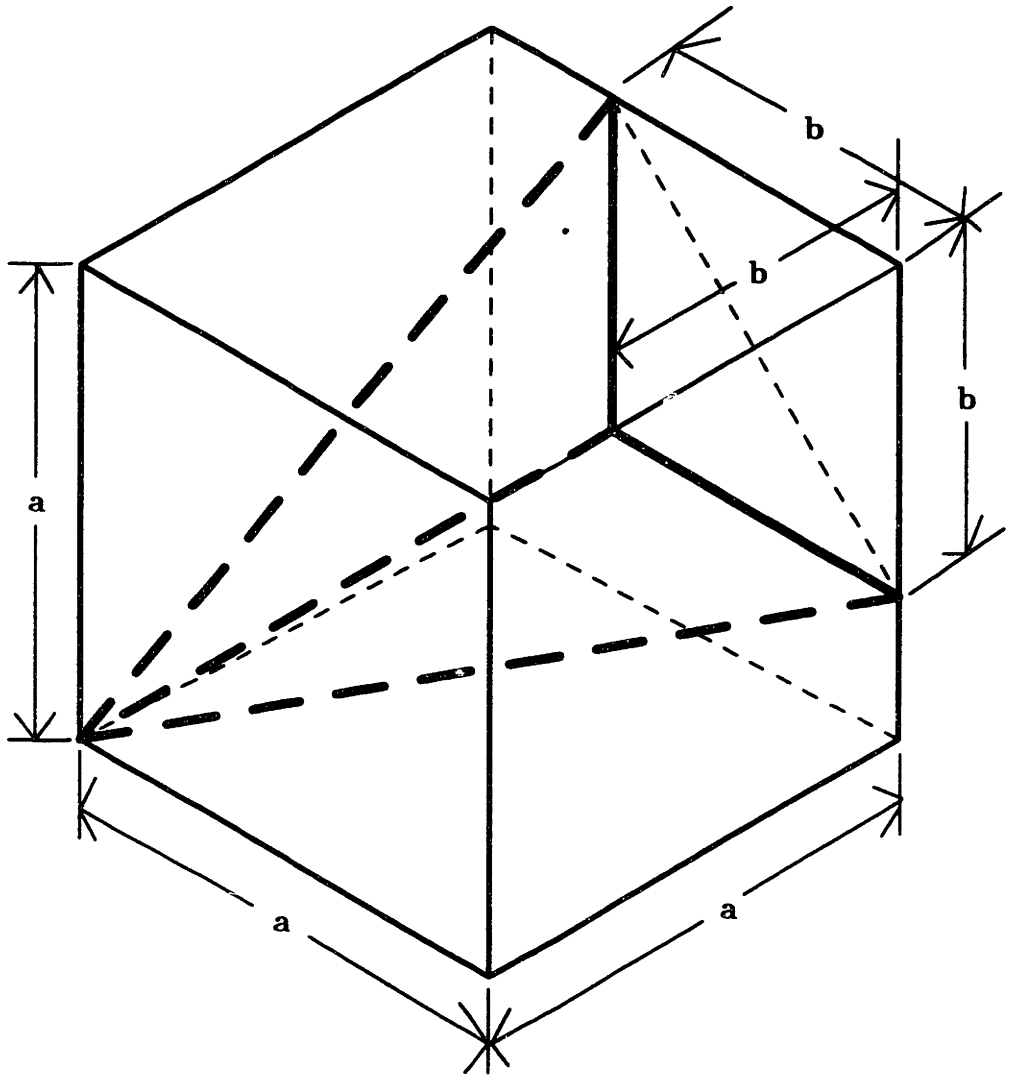


Figure B.1: Three D Cone Example

Then the volume fraction in terms of B the common base of the to pyramids, the dimensionality of the space n and the edge dimension a is

$$\frac{V_{2p}}{V_t} = \frac{B}{a^{n-1}\sqrt{n}} \tag{B.3}$$

What is needed now is an expression for the base B. In the 2D case the base B_2 is simply $b\sqrt{2}$ as shown in the top of Figure B.2.

The base in 3D B_3 is defined inductively in terms of B_2 as indicated in Figure B.2 as follows.

$$B_3 = 3B_2 \int_0^{|\vec{Centroid}(B_2) - \vec{Centroid}(B_3)|} \left(\frac{r}{|\vec{Centroid}(B_2) - \vec{Centroid}(B_3)|}\right)^2 dr \tag{B.4}$$

In general the base in n-space B_n is given by

$$B_n = C_n B_{n-1} \int_0^d \left(\frac{r}{d_n}\right)^{n-2} dr = C_n B_{n-1} \frac{d_n}{n-1} \tag{B.5}$$

Where C_n is the number of B_{n-1} which are involved in the integration, d is defined as $\left|\vec{Centroid}(B_n) - \vec{Centroid}(B_{n-1})\right|$ and the exponent $(n-2)$ reflects the dimensionality of B_{n-1}. This gives a recurrence relation for B_n with the additional terms C_n and d_n. These two quantities can be computed directly. Since C_n is the number of "faces" of B_n it is also equal to the number of unique sets of $n-1$ vertices which can be selected from a set of size n. Then

$$C_n = \binom{n}{n-1} = \frac{n!}{(n-1)!1!} = n \tag{B.6}$$

To compute d_n take the magnitude of the difference between two vectors defined in the coordinate system of the far corner pointing to the centroids of B_n and B_{n-1}.

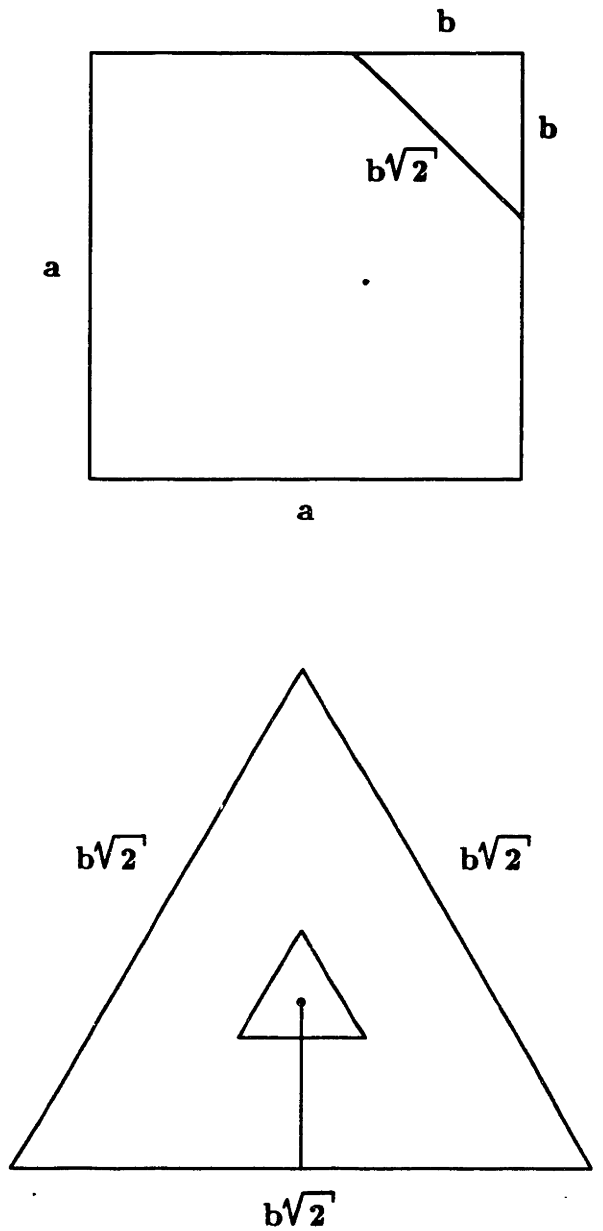


Figure B.2: Base Calculation

$$\begin{aligned} \vec{Centroid}(B_n) &= \tfrac{b}{n}\hat{i}_1 + \tfrac{b}{n}\hat{i}_2 + \cdots + \tfrac{b}{n}\hat{i}_n \\ &\text{and} \\ \vec{Centroid}(B_{n-1}) &= \tfrac{b}{n-1}\hat{i}_1 + \tfrac{b}{n-1}\hat{i}_2 + \cdots + \tfrac{b}{n-1}\hat{i}_{n-1} \end{aligned} \tag{B.7}$$

Then the Euclidean distance between these two centroids d_n is

$$d_n = \sqrt{(n-1)b^2(\frac{1}{n-1} - \frac{1}{n})^2 + (\frac{b}{n})^2} = \frac{b}{\sqrt{n(n-1)}} \tag{B.8}$$

Substituting C_n and d_n into Equation B.5 gives

$$B_n = \frac{bB_{n-1}n}{\sqrt{n(n-1)}(n-1)} \tag{B.9}$$

Starting from the boundary condition $B_2 = b\sqrt{2}$ and working out a few iterations gives

$$B_n = \frac{b^{n-1}\sqrt{n!}}{\sqrt{[(n-1)!]^3}} \tag{B.10}$$

Substituting this into Equation B.3 gives the fraction of the total volume enclosed within the two back to back pyramids as

$$\frac{V_{2p}}{V_t} = (\frac{b}{a})^{(n-1)}\frac{1}{(n-1)!} \quad \begin{cases} n & = 2, 3, 4, \cdots \\ & \text{and} \\ 0 & \leq \frac{b}{a} \leq 1 \end{cases} \tag{B.11}$$

Notes on Random Sampling

Appendix C

There are a few issues concerning random schedule generation which should be mentioned. The first issue is what class (set) of schedules one is sampling from. The second issue is how the sampling is biased within this set. The third is the distinction between the mode and the range of the sampling distribution. And the fourth issue concerns joint distributions.

Ideally, one would like to sample from a set of schedules which is guaranteed to contain all the schedules of interest. This could be accomplished either by sampling from the complete set or by sampling from another set whose members somehow dominate the complete set under the optimality criterion being used. One complete set would be the set of all unique sequences. This set has infinite cardinality if one allows delays in processing a given sequence. If one processes each activity as soon as possible consistent with the given sequence, then each sequence corresponds to an *Early Start Schedule*. These are the so called *Semi Active Schedules*.

The class of *Non Delay* schedules can be defined in the context of a *Job Shop* simulation. These schedules are generated by having each machine never wait (delay)

to process some activity if there is an activity in the machine's queue. The set of *Non Delay Schedules* is a subset of the set of *Early Start Schedules.*

Given a set to sample from, the distribution of objective function values varies depending on how the sampling operation is performed. For example, consider sampling with equal probability from the set of *Non Delay Schedules.* An experimental estimate of such a distribution was obtained by using the *Random* priority dispatch rule to generate sample schedules from a $\{10, 10, Gen, Makespan\}$ problem. The probability of obtaining any given sample schedule can be determined after the particular schedule has been generated. This is done by multiplying together the probabilities associated with each machine loading decision. Choosing one activity from a queue of five activities contributes a factor of $1/5$ to the overall probability of obtaining the particular sample schedule. These computed probabilities were used as weights on the samples obtained. This weighted distribution is plotted in Figure C.1. This is an estimation of the distribution of schedule makespans assuming that each *Non Delay* schedule is equally likely to be chosen.

Notice that using the *Random* priority dispatch rule (which assigns equal probability to choosing the various activities from a queue) introduces a biasing into the sampling procedure. The distribution from Figure C.1 is replotted in Figure C.2 in quantile format along with sample distributions obtained using the *Random* priority dispatch rule and an active schedule generator.

The biasing introduced by the *Random* dispatch rule significantly improves the expected makespan of the sample schedules in this case. This is because schedules having less conflicts which need to be resolved (shorter queues) have higher probabilities of being generated using this method.

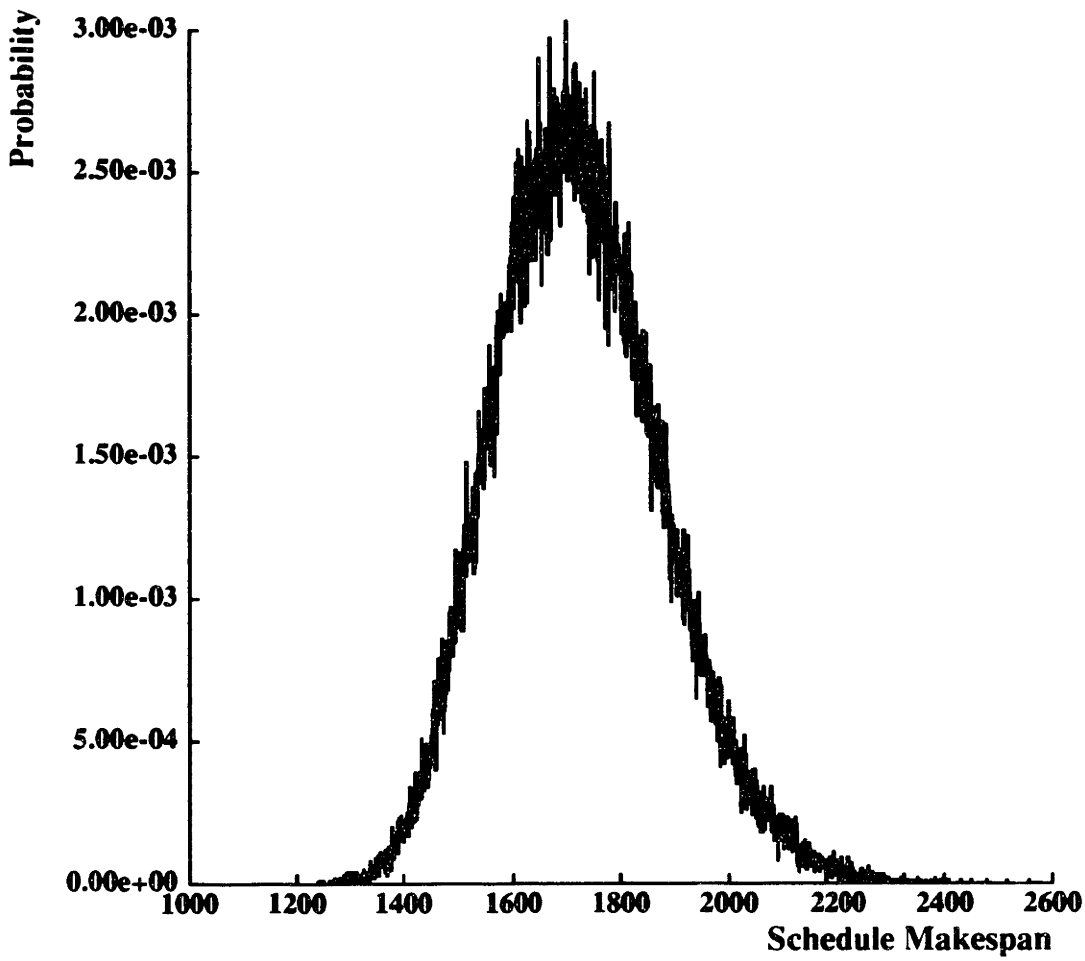


Figure C.1: Normalized Sample Distribution

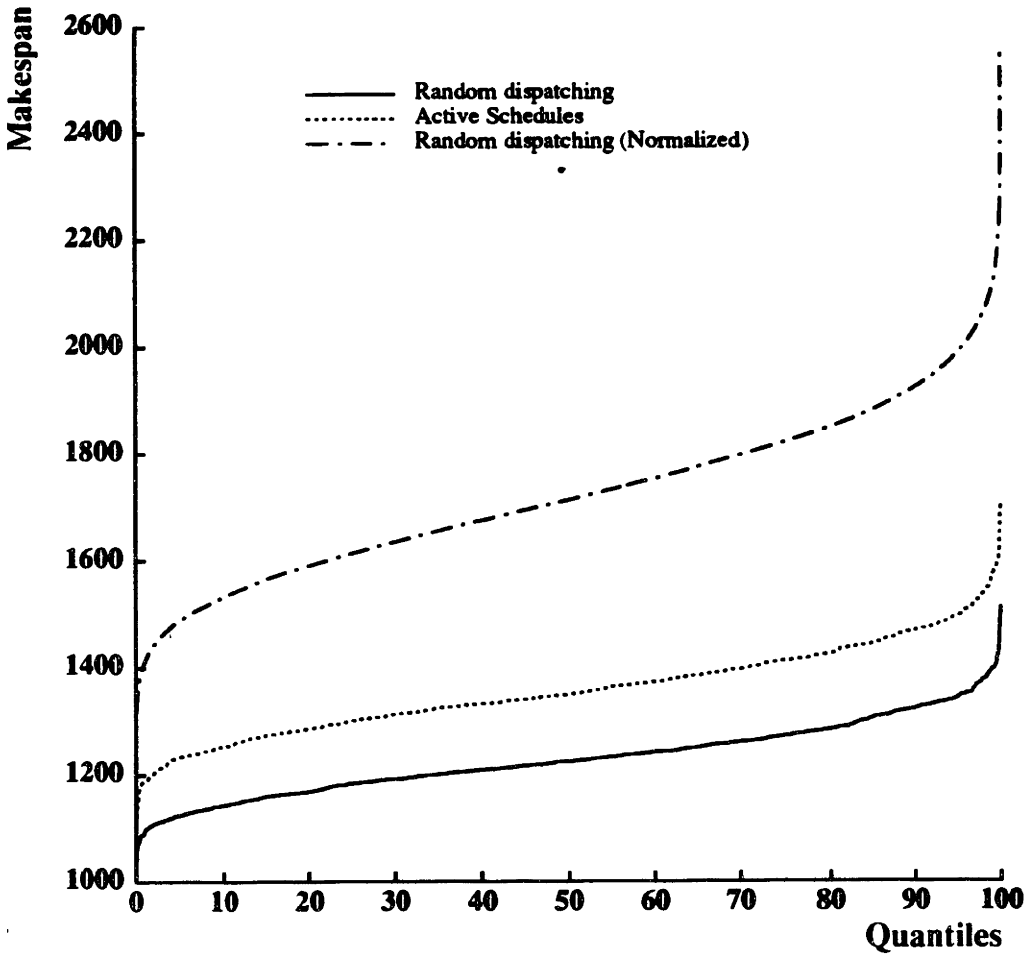


Figure C.2: Comparison of Distributions Obtained Using the *Random* Rule, a Normalized *Random* Rule and an Active Schedule Generator

The distributions shown in Figures C.1 and C.2 are approximately Gaussian. Although the tails of the distributions have finite length the probabilities associated with the tails gets very small before the end of the tails is encountered. In [47] is suggested a method based on sampling for generating a schedule which falls with some known confidence into a fraction of the area under the distribution. Suppose that the true distribution was known. One could partition the distribution into two parts. One part of area a (starting with the left side of the left tail) and one of area $1 - a$. Then any sample generated has probability a of falling within the range of the a area. The probability of at least one out of n samples falling within the a area is equal to $1-$*Probability of all samples falling in the* $(1 - a)$ *area.* This is simply $1-(1-a)^n$. Thus if a is 0.01 and n is 500 then the probability of generating at least one sample out of the 500 which lies within the top 0.01 of the area under the curve is $1 - (1 - 0.01)^{500} = 0.993$.

Suppose one of the samples lies within the range of the 0.01 area. This is not the same as being in the top 0.01 of the range of possible schedule values. Consider a true Gaussian distribution and define area a as the area under the left hand tail starting 3 standard deviations from the mean. Then a is 0.0015, and having a sample fall within the range of this area means that it falls somewhere between minus infinity and minus 3 sigma. This situation is somewhat exaggerated; in the case of the approximate Gaussian distribution mentioned above, having a sample fall within the top 0.01 of the area under the distribution translates to being approximately within the top 0.10 of the range of possible schedule values.

As illustrated, the distributions of *Makespans* values is approximately Gaussian; this is also true of the distribution of *Flowtime* values. As a matter of fact they seem to be almost independent as evidenced by the scatter plot shown in Figure C.3.

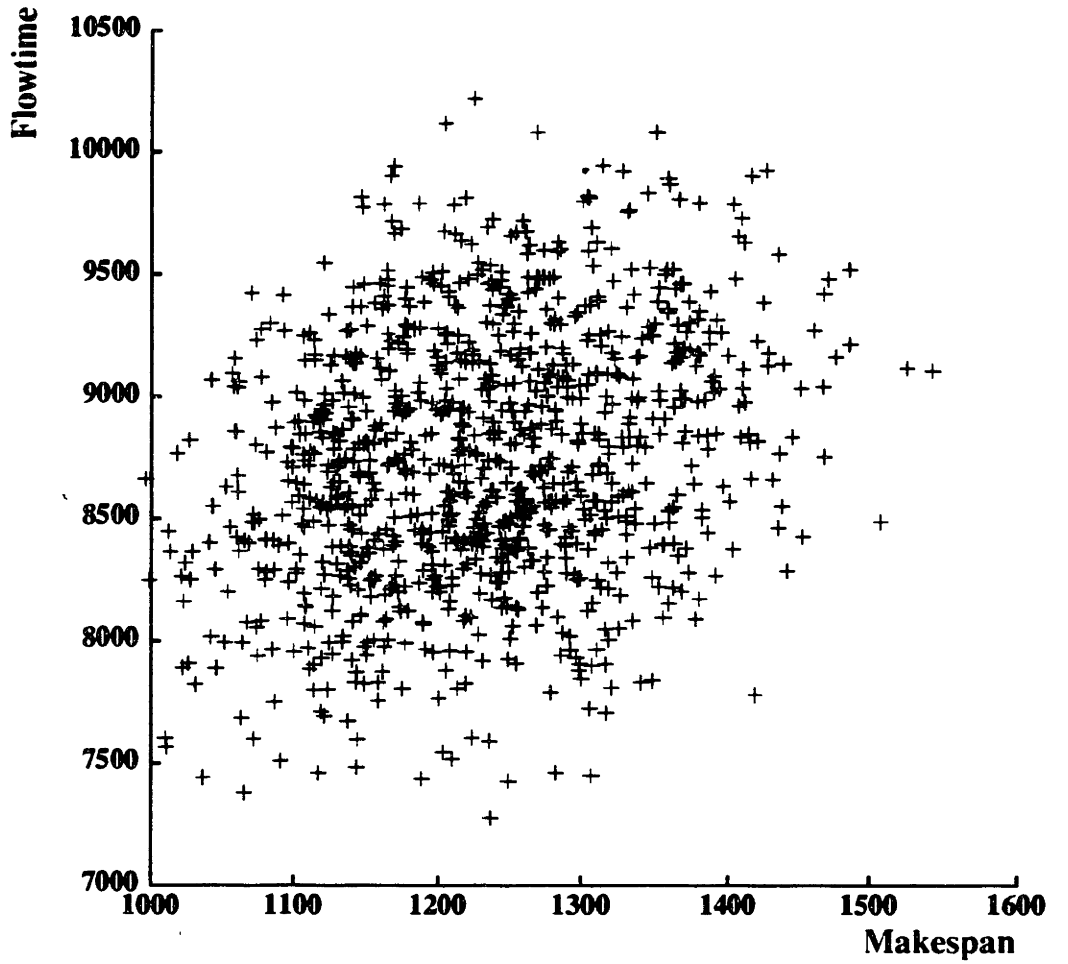


Figure C.3: Joint Distribution Scatter Plot, Schedule Makespan vs Schedule Flowtime

To generate this plot, first the makespan and flowtime of 1000 sample schedules generated using the *Random* priority dispatch rule were computed. Then one data point was plotted (flowtime vs makespan) for each schedule generated. Four percent of the range was added to the data so that duplicate data points would not overprint.

Applying H1/CT to a Problem with Symmetry

Appendix D

A 12 job by 6 machine problem was generated by making a copy of each of the jobs in a 6 job 6 machine problem ($\{6, 6, ,\}$). Experiments similar to those of Chapter 3.3 using heuristic H1/CT were performed on this new problem using the criteria of makespan and flowtime. For these experiments, first 200 sample schedules were randomly generated, then the top ten schedules according to the optimality criteria were selected. If all ten of these schedules had precedence relations in common, these common precedence relations were used to prune the original problem and the sampling repeated. The results of these experiments are shown in Figures D.1 and D.2. The procedure produced mixed results in the case of the makespan criterion. After 242 of the 450 possible precedence relations were used to prune the space the probabilities of generating very low makespan schedules was reduced. This situation improved during a subsequent iteration as shown by the solid curve.

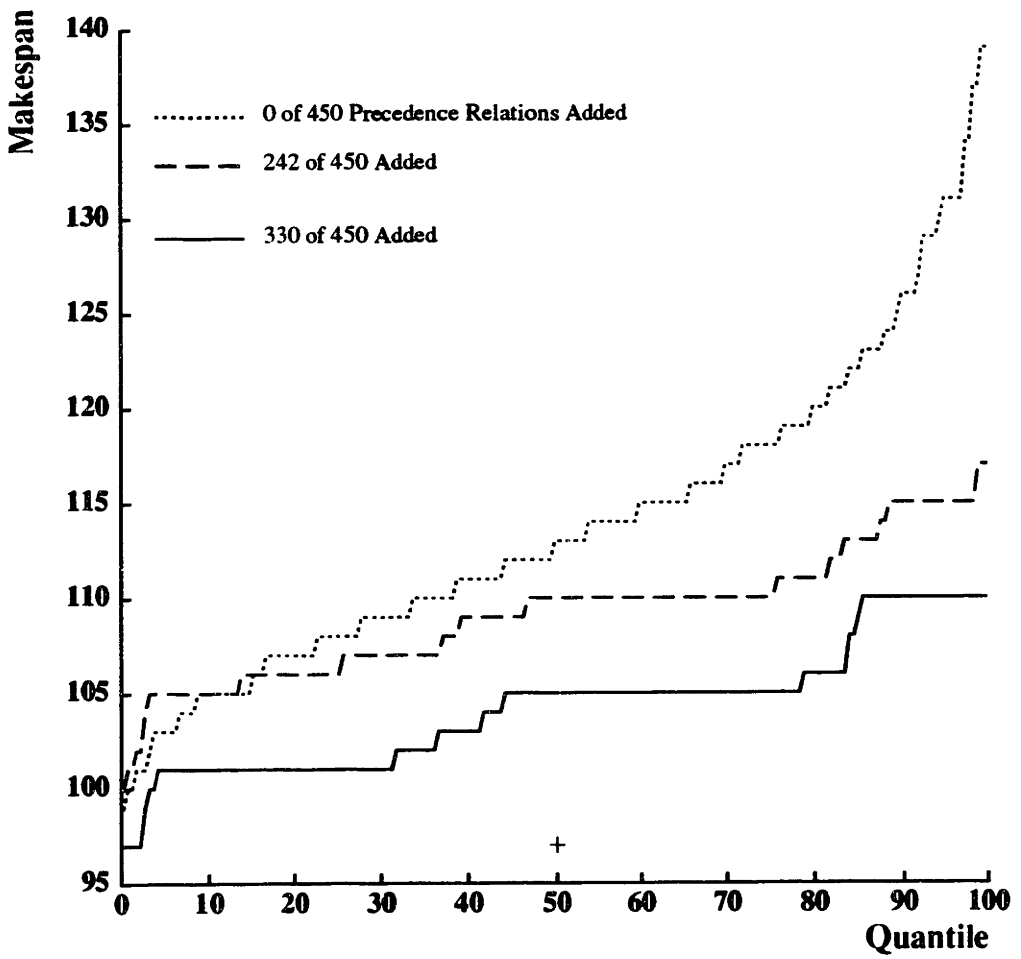


Figure D.1: H1/CT applied to a 12 job 6 machine problem (makespan criterion)

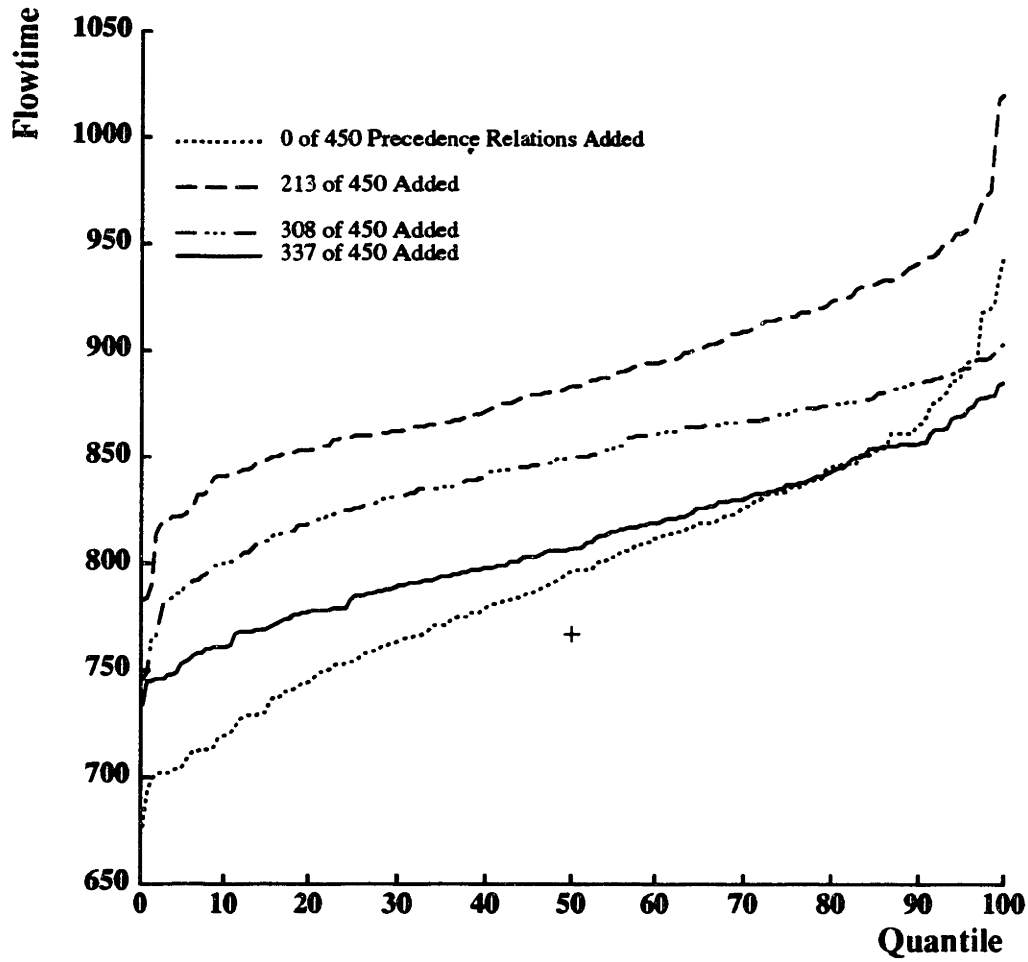


Figure D.2: H1/CT applied to a 12 job 6 machine problem (flowtime criterion)

The mixed results in the above$\{12, 6, G, \text{Makespan}\}$ experiment were unexpected. One possible explanation for this is that a number of symmetries were introduced into the state space by including a copy of each job. As sample schedules were generated and compared these symmetries were not taken into account thereby causing a lack of adjacency between schedule trajectories which would mirror on to one another.

Consider a 2-D subspace of the state space defined by a job and its copy as shown in Figure D.3. Notice that there is an axis of symmetry along the main diagonal. Any trajectory segment which crossed the diagonal could be mapped into an equivalent (in terms of objective function value) trajectory on the opposite side of the diagonal.

An experiment was performed in order to verify the above supposition. There would be two ways in which to take these symmetries into account. One, the sample schedules could be generated as before and some post sampling processing could be done to mirror the trajectories into a canonical form suitable for the comparison step. Alternately, schedule generation could be restricted such that trajectories were confined to one of the two regions defined by the diagonal of Figure D.3. The latter of these two was chosen on the basis of ease of implementation.

Notice that the obstacles all lie on the diagonal in Figure D.3 and that in order to pass from one region to the other the trajectory must pass by one of these obstacles. But, negotiating these obstacles corresponds to making a machine loading decision between two identical (in terms of objective function value of the completed schedule) alternatives. The sampling scheme was modified such that when this situation occurred a unique choice was made (i.e. stay in the lower region or equivalently give activities from Job 1 priority over those from Job 2).

The correlation procedure was then repeated on the $\{12, 6, G, \text{Makespan}\}$ problem with the modified (to incorporate mirroring) random schedule generator. The results

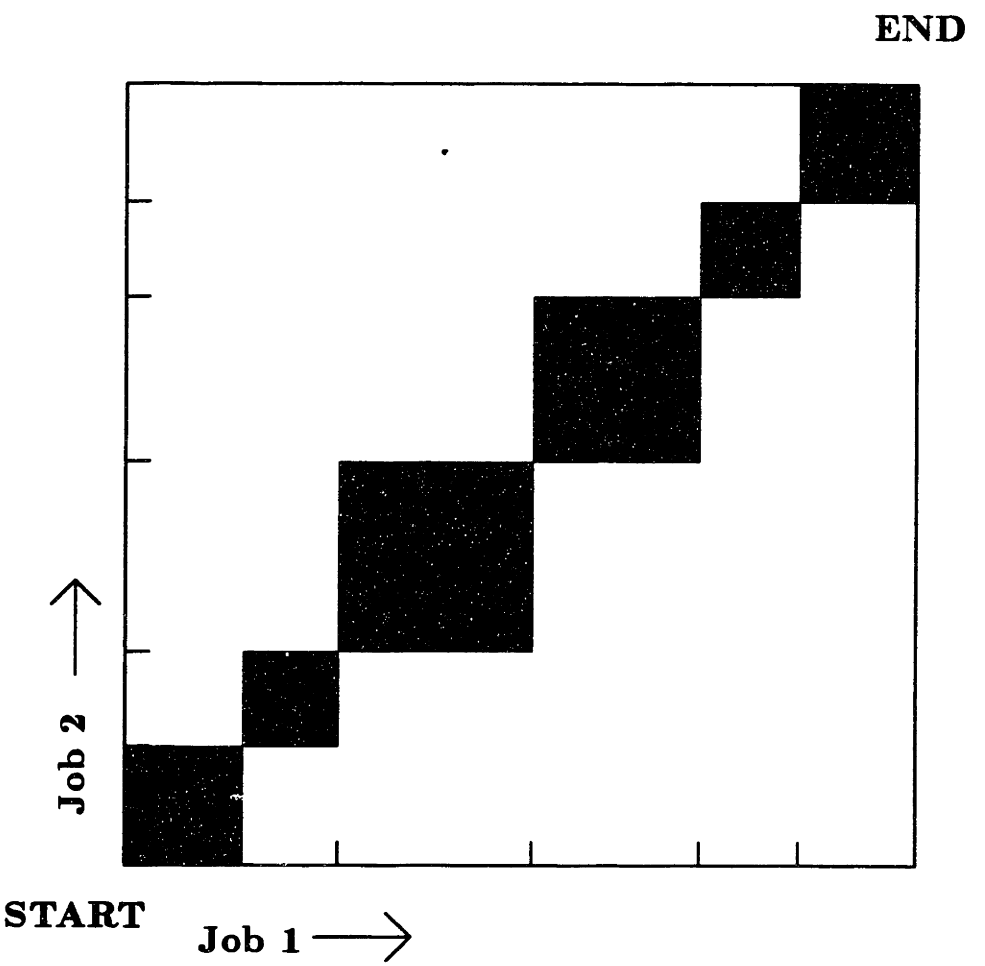


Figure D.3: Symmetric 2-D Subspace

of these experiments are shown in Figures D.4,D.5. These results are very similar to those of the $\{6, 6, G, \text{Makespan}\}$ problem from which the $\{12, 6, G, \text{Makespan}\}$ was composed. The distributions are seen to uniformly improve during successive iterations.

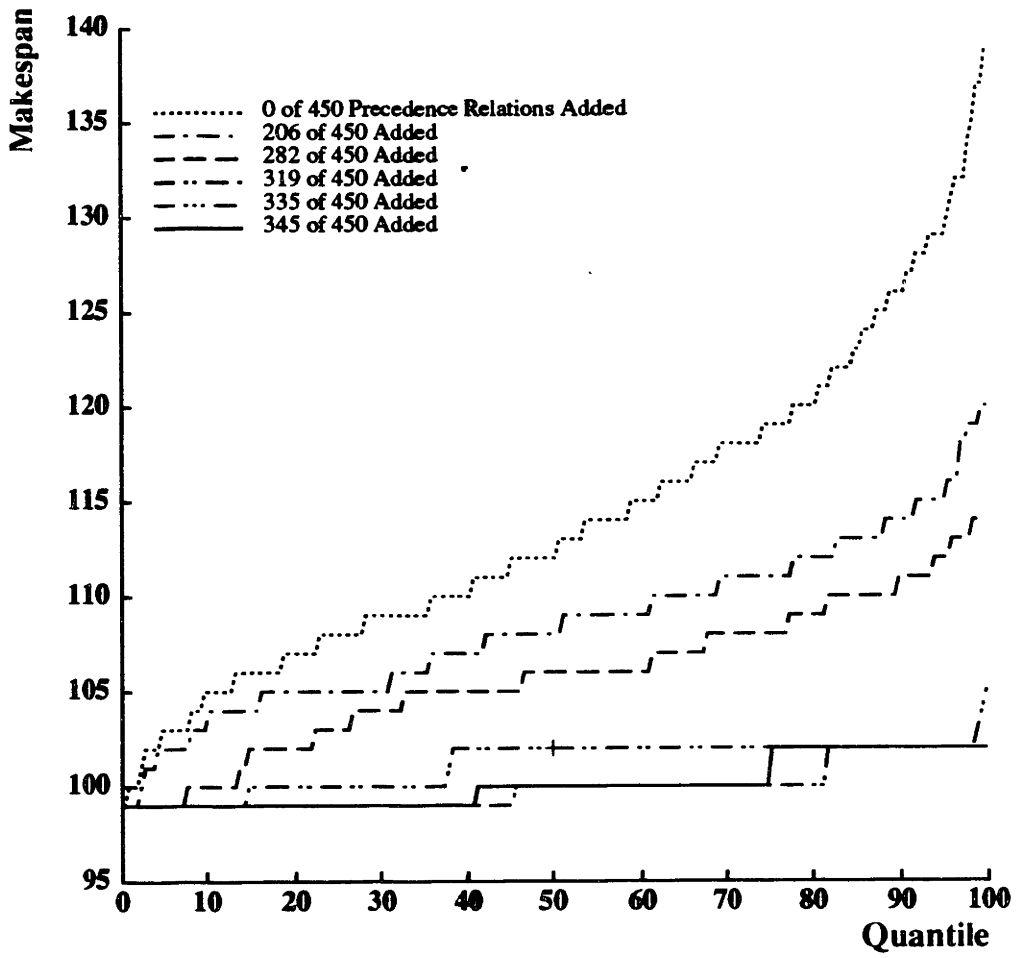


Figure D.4: H1/CT modified to account for symmetry applied to a 12 job 6 machine problem (makespan criterion)

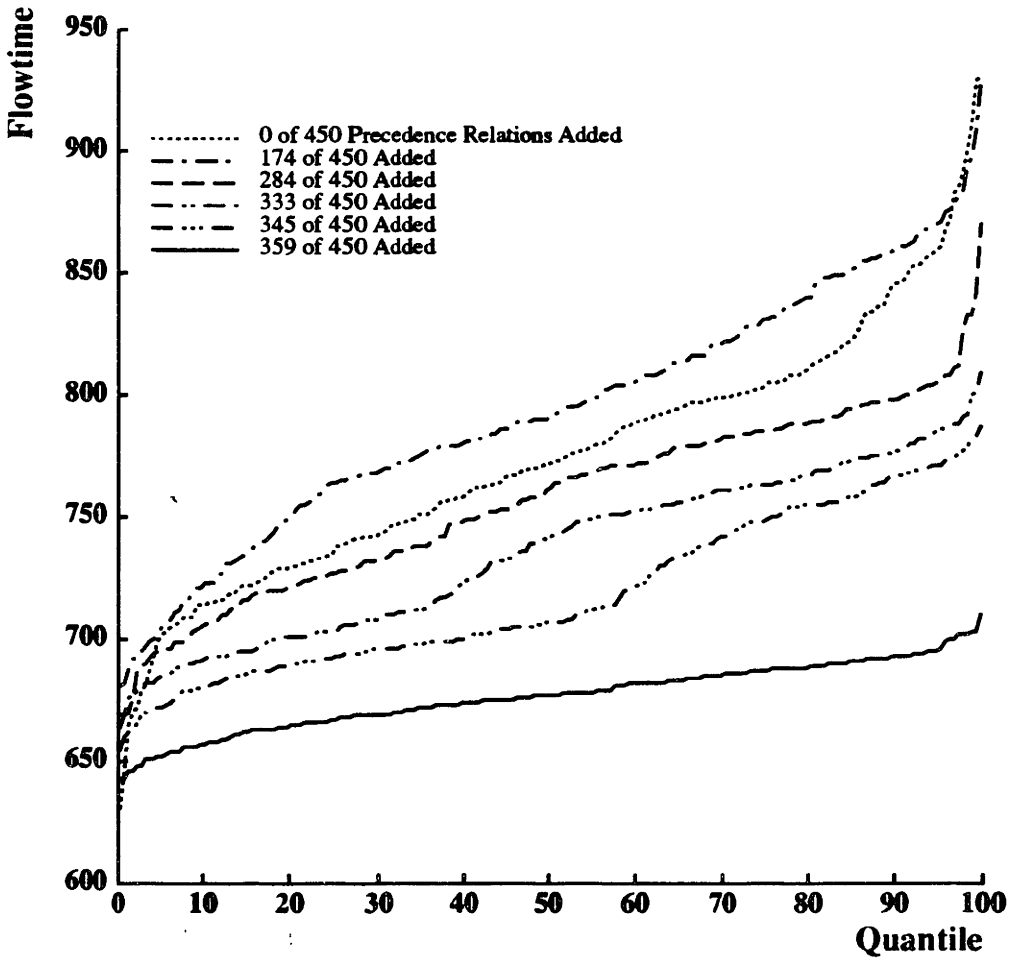


Figure D.5: H1/CT modified to account for symmetry applied to a 12 job 6 machine problem (flowtime criterion)

Bibliography

[1] **Adams, Joseph; Balas, Egon; and Zawack, Daniel,**
"The Shifting Bottleneck Procedure for Job Shop Scheduling", Management Science Research Report No. MSRR-525, Carnegie-Mellon University, Pittsburgh, Pennsylvania, *July 1986.*

[2] **Aho, A. V.; Hopcroft, J. E.; and Ullman, J. D.,**
The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA; Menlo Park, Ca.; London; Amsterdam; Don Mills, Ontario; Sydney, *1974.*

[3] **Akella, R.; and Kumar, P. R.,**
"Optimal Control of Production Rate in a Failure Prone Manufacturing System", Series LIDS-P; 1427, Laboratory for Information and Decision Systems, MIT, *January 1985.*

[4] **Akella, Ramakrishna; Choong, Yong; and Gershwin, Stanley B.,**
"Performance of Hierarchical Production Scheduling Policy", Report # LIDS-

FR-1357, Laboratory for Information and Decision Systems, MIT, *February 1984.*

[5] **Akers, Sheldon B.; and Friedman, Joyce,**
"A Non-Numerical Approach to Production Scheduling Problems", Operations Research, *Vol. 3, No. 4, pp 429-442, November 1955.*

[6] **Akers, S. B.,**
"A Graphical Approach to Production Scheduling Problems", Operations Research, *Vol. 4, pp 244-245, 1956.*

[7] **Allen, James F.,**
"Maintaining Knowledge about Temporal Intervals", Communications of the ACM, *Vol. 26, No. 11, pp 832-843, 1983.*

[8] **Baker, Kenneth R.,**
"A Comparative Study of Flow-Shop Algorithms", Operations Research, *Vol. 23, No. 1, pp 62-73, January-February 1975.*

[9] **Bakshi, Mahendra S.; and Arora, Sant Ram,**
"The Sequencing Problem", Management Science, *Vol. 16, No. 4, pp B247-B263, December 1969.*

[10] **Balas, Egon,**
"Discrete Programming by the Filter Method", Operations Research, *Vol. 15, pp 915-957, 1967.*

[11] **Barr, A.; and Feigenbaum, E. A.,**
Handbook of Artificial Intelligence, *Vol. 1,* William Kaufmann, Inc., Los Altos, California, *1981.*

[12] **Bourne, David A.; and Fox, Mark S.,**

"Autonomous Manufacturing: Automating the Job-Shop", Computer (USA), *Vol. 17, No. 9, pp 76-86, September 1984.*

[13] **Brooks, George H.; and White, Charles R.,**

"An Algorithm for Finding Optimal or Near Optimal Solutions to the Production Scheduling Problem", The Journal of Industrial Engineering, *Vol. XVI, No. 1, pp 34-40, January-February 1965.*

[14] **Buer, H.; and Möhring,**

"A Fast Algorithm for the Decomposition of Graphs and Posets", Math. Opns. Res., *Vol. 8, pp 170-184, 1983.*

[15] **Bunnag, Panit; and Smith, Spencer B.,**

"A Multifactor Priority Rule for Jobshop Scheduling Using Computer Search", IIE Transactions, *pp 141-146, June 1985.*

[16] **Campbell, Herbert G.; Dudek, Richard A.; and Smith, Milton L.,**

"A Heuristic Algorithm for the n Job m Machine Sequencing Problem", Management Science, *Vol. 16, No. 10, pp B630-B637, June 1970.*

[17] **Carlier, Jacques,**

"The One-Machine Sequencing Problem", European Journal of Operations Research, *Vol. 11, pp 42-47, 1982.*

[18] **Carlier,J.; Chretienne, Ph.; and Girault, C.,**

"Modelling Scheduling Problems with Timed Petri Nets", Advances in Petri Nets 1984, Edited by **G. Rozenberg with the cooperation of H. Genrich and G. Raucairol ,**

Sringer-Verlag , Berlin, Heidelberg, New York, Tokyo, *Lecture Notes in Computer Science - 188, pp 62-82, 1985.*

[19] **Charlton, John M.; Death, Carl C.,**
"A Method of Solution for General Machine-Scheduling Problems", Operations Research, *Vol. 18, pp 689-707, 1970.*

[20] **Chow, We-Min; MacNair, Edward A.; and Sauer, Charles H.,**
"Analysis of Manufacturing Systems by the Research Queueing Package", IBM J. Res. Develop., *Vol. 29, No. 4, pp 330-342, July 1985.*

[21] **Conway, R. W.,**
"An Experimental Investigation of Priority Assignment in a Job Shop", RM-3789-PR, The Rand Corporation, Santa Monica, CA, *1964.*

[22] **Chryssolouris, G.; Wright, K.; Pierce, J.; and Cobb, W.,**
"Manufacturing Systems Operation: Dispatch Rules Versus Intelligent Control", To be published in "Robotics and Computer-Integrated Manufacturing", *1987.*

[23] **Clemmer, George L.,**
"An Artificial Intelligence Approach to Job-Shop Scheduling", Master Thesis, Sloan School of Mgmt., MIT, *September 1984.*

[24] **Conway, Richard W.; Maxwell, William L.; and Miller, Louis W.,**
"Theory of Scheduling", Addison-Wesley, Reading, MA; Menlo Park, Ca.; London; Amsterdam; Don Mills, Ontario; Sydney, *1967.*

[25] **Davis, Edward W.; and Heidorn, George E.,**
"An Algorithm for Optimal Project Scheduling Under Multiple Resource Con-

straints ", Management Science, *Vol. 17, No. 12, pp B803-B816, August 1971.*

[26] **Davis, Edward W.,**

"Project Scheduling under Resource Constraints - Historical Review and Categorization of Procedures", AIEE Transactions, *May 1973.*

[27] **Davis, Edward W.; and Patterson, James H.,**

"A Comparison of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling", Management Science, *Vol. 21, No. 8, pp 945-955, 1975.*

[28] **Dempster, M. A. H.; Fisher, M. L.; Jansen, L.; Lageweg, B. J.; Lenstra, J. K.; and Rinnooy Kan, A. H. G.,**

Mathematics of Operations Research, *Vol. 8, No. 4, pp 525-537, November 1983.*

[29] **Dewdney, A. K.,**

"On the Spaghetti Computer and other Analog Gadgets for Problem Solving", Scientific American, *pp 19-26, June 1984.*

[30] **Dewdney, A. K.,**

"Analog Gadgets that Solve a Diversity of Problems and Raise an Array of Questions", Scientific American, *pp 18-28, June 1985.*

[31] **Engelke, H.; Grotrian, J.; Scheuing, C.; Schmackpfeffer, A.; Schwarz, W.; Solf, B.; and Tomann, J.,**

"Integrated Manufacturing Modeling System", IBM J. Res. Develop., *Vol. 29, No. 4, pp 343-355, July 1985.*

[32] **Erschler, J.; Fontan, G.; Merce, C.; and Roubellat, F.,**

"A New Dominance Concept in Scheduling n Jobs on a Single Machine with Ready Times and Due Dates", Operations Research, *Vol. 31, No. 1, pp 114-127, January-February 1983.*

[33] **Fisher, Marshall I.,**

"Optimal Solution of Scheduling Problems Using Lagrange Multipliers: Part I", Operations Research, *Vol. 21, pp 1114-1127. 1973.*

[34] **Fox, B. R.; and Kempf K. G.,**

"A Representation for Opportunistic Scheduling", 3rd ISRR, Edited by **Faugeras and Giralt,**

pp 111-117, 1986.

[35] **Fox, B. R.; and Kempf, K. G.,**

"Opportunistic Scheduling for Robotic Assembly", Proc. Inter. Conf. on Robotics and Automation, IEEE, *pp 880-889, 1985.*

[36] **Fox, B. R.; and Ho, C. Y.,**

"A Relational Control Mechanism for Flexible Assembly", Proceedings of Advanced Software in Robotics, Liege, Belgium, *pp 2.A/1-11, May 1983.*

[37] **Fox, Mark S.; Allen, Brad; and Strohm, Gary,**

"Job-Shop Scheduling: An Investigation in Constraint-Directed Reasoning ", AAAI-82, *Vol. 1, pp 155-158, 1982.*

[38] **Fox, Mark S.,**

"The Intelligent Management System: An Overview", Technical Report, Robotics Institute, Carnegie-Mellon Univ., *December 1982.*

[39] **Fox, Mark S.; Smith, Stephen F.; Allen, Bradley P.; Strohm, Gary A.; and Winberly, Francis C. ,**
"ISIS: A Constraint-Directed Reasoning Approach to Job Shop Scheduling", IEEE Paper CH1887, *pp 76-81, 1983.*

[40] **Fox, Mark S.,**
"Constraint-Directed Search: A Case Study of Job-Shop Scheduling", CMU-RI-TR-83-22,CMU-CS-83-161, Carnegie-Mellon University, *1983.*

[41] **Fox, M. S.; and Smith, S. F.,**
"ISIS - A Knowledge Based System for Factory Scheduling", Expert Syst. (GB), *Vol. 1, No. 1, pp 25-49, 1983.*

[42] **Gershwin, Stanley B.,**
"Material and Information Flow in an Advanced Automated Manufacturing System", LIDS-P-1199, Laboratory for Information and Decision Systems, MIT, Cambridge, MA., *May 1982.*

[43] **Gershwin, Stanley B.,**
"An Efficient Decompostion Method for the Approximate Evaluation of Production Lines with Finite Storage Space and Blocking", Report # LIDS-P-1309, Laboratory for Information and Decision Systems, MIT, *December 1983.*

[44] **Gershwin, Stanley B.; Akella, Ramakrishna; and Choong, Yong,**
"Short Term Production Scheduling of an Automated Manufacturing Facility", Report # LIDS-FR-1356, Laboratory for Information and Decision Systems, MIT, *February 1984.*

[45] **Gershwin, Stanley B.,**

"An Efficient Decompostion Method for the Approximate Evaluation of Production Lines with Finite Storage Space", Report # LIDS-P-1308, Laboratory for Information and Decision Systems, MIT, *July 1983, revised September 1983, March 1984.*

[46] **Giffler, B.; and Thompson, G. L. ,**

"Algorithms for Solving Production Scheduling Problems", Operations Research, *Vol. 8, pp 487-503, 1960.*

[47] **Giffler, B.; Thompson, G. L.; and Van Ness, V. ,**

Edited by **Muth, J. F., G. L. Thompson,**

"Numerical Experience with Linear and Monte Carlo Algorithms for Solving Production Scheduling Problems ", Industrial Scheduling, Chap. 3, *pp 21-38*, Prentice-Hall, Englewood Cliffs, N. J., *1963.*

[48] **Grabowski, Jozef; Skubalska, Ewa; and Smutnicki, Czslaw,**

"On Flow Shop Scheduling with Release and Due Dates to Mimimize Maximum Lateness", *pp 615-620,*

[49] **Graves, Stephen C.,**

"A Review of Production Scheduling", Operations Research, *Vol. 29, No. 4, pp 646-675, July-August 1981.*

[50] **Graves, Stephen C.; Meal, Harlan C.; Stefek, Daniel; and Zeghmi, Abdel Hamid,**

"Scheduling of Re-Entrant Flow Shops", Operations Management, *Vol. 3, No. 4, pp 197-207, August 1983.*

[51] **Graves, Stephen C.; and Lamar, Bruce W.,**
"An Integer Programming Procedure for Assembly System Design Problems", Operations Research, *Vol. 31, No. 3, pp 522-545, May-June 1983.*

[52] **Hardgrave, William W.; and Nemhauser, George L.,**
"A Geometric Model and a Graphical Algorithm for a Sequencing Problem ", Operations Research, *Vol. 11, No. 6, pp 889-900, 1963.*

[53] **Hopcroft, J. E.; and Tarjan, R. E.,**
"Dividing a Graph into Triconnected Components", SIAM Journal of Computing, *Vol. 2, No. 3, pp 135-158, September 1973.*

[54] **Kimemia, Joseph G.; and Gershwin, Stanley B.,**
"An Algorithm for the Computer Control of Production in a Flexible Manufacturing System", LIDS-P-1134 Revised, Laboratory for Information and Decision Systems, MIT, Cambridge, MA., *Revised January 1982.*

[55] **Kirkpatrick, S.; Gelatt, Jr., C. D.; and Vecchi, M. P.,**
"Optimization by Simulated Annealing", Science, *Vol. 220, No. 4598, pp 671-680, 13 May, 1983.*

[56] **Krone, Martin J.; and Steiglitz, Kenneth,**
"Heuristic-Programming Solution of a Flowshop-Scheduling Problem", pp 629-638,

[57] **Kurtulus, I.; and Davis, E. W.,**
"Multi-Project Scheduling: Categorization of Heuristic Rules Performance", Management Science, *Vol. 28, No. 2, pp 161-172, February 1982.*

[58] **Lageweg, B. J.; Lenstra, J. K.; and Rinnooy Kan, A. H. G.,**
"Job-Shop Scheduling by Implicit Enumeration", Management Science, *Vol. 24, No. 4, pp 441-450, December 1977.*

[59] **Lawler, E. L.,**
"Sequencing Jobs to Minimize Total Weighted Completion Time Subject to Precedence Constraints", Annals of Discrete Math, North-Holland Publishing Company, *Vol. 2, pp 75-90, 1978.*

[60] **Lipton, Michael J.,**
"Integrating Real Time Scheduling into a Flexible Automated Electronics Plant", SME Technical Paper # EE85-133, FMS for Electronics, *February 1985.*

[61] **Lipton, Michael J.,**
"The Development of a Real Time Scheduling System for Automated Production", SME Technical Paper # MS85-1095, Autofact , *November 1985.*

[62] **Marcus, Robert,**
"An Application of Artificial Intelligence to Operations Research", Communications of the ACM, *Vol. 27, No. 10, pp 1044-1047, October 1984.*

[63] **McMahon, Graham; and Florian, Michael,**
"On Scheduling with Ready Times and Due Dates to Minimize Maximum Lateness ", Operations Research, *Vol. 23, No. 3, pp 475-482, May-June 1975.*

[64] **Metropolis, Nicholas; Rosenbluth, Arianna W.; Rosenbluth, Marshall N.; Teller, Augusta H.; and Teller, Edward,**

"Equation of State Calculations by Fast Computing Machines", The Journal of Chemical Physics, *Vol. 21, No. 6, pp 1087-1092, June, 1953.*

[65] **Moore, J. M.; and Wilson, R. C.,**
"A Review of Simulation Research in Job Shop Scheduling", Production and Inventory Management, *pp 1-10, January 1967.*

[66] **Nugent, C. E.,**
"On Sampling Approached to the Solution of the n-by-m Static Sequencing Problem", PhD Thesis, Cornell University, *September 1964.*

[67] *"Combinitorial Optimization: Annotated Bibliographies"*, Edited by **O'hEigeartaigh, M.; Lenstra, J. K.; and Rinooy Kan, A. H. G.,**
John Wiley & Sons, Chichester, New York, Brisbane, Toronto, Singapore, *1985.*

[68] **Patterson, James H.,**
"Alternate Methods of Project Scheduling with Limited Resources", RDTR, Research and Development, Naval Amunition Depot, Crane, Indiana, *No. 174, 1970.*

[69] **Peterson, James L. ,**
"Petri Net Theory and the Modeling of Systems", Prentice-Hall, INC., Englewood Cliffs, N.J. 07632, *1981.*

[70] **Picard, Jean-Claude; and Queyranne, Maurice,**
"The Time-Dependent Traveling Salesman Problem and Its Application to the Tardiness Problem in One-Machine Scheduling", Operations Research, *Vol. 26, No. 1, pp 86-110, January-February 1978.*

[71] **Reddy, Y. V.; and Fox, Mark S.,**

"KBS: An Artificial Intelligence Approach to Flexible Simulation", Technical Report, Robotics Institute, Carnegie-Mellon Univ., *September 1982.*

[72] **Reisig, W.,**

"Petri Nets, an Introduction", Edited by **W. Brauer, G. Rozenberg, A. Salomaa,**

1982 Springer-Verlag , Berlin, Heidelberg, New York, Tokyo, *EATCS Monographs on Theoretical Computer Science, 1985.*

[73] **Rinnooy Kan, A. H. G. ,**

"Machine Scheduling Problems", Martinus Nijhoff / The Hague. *1976.*

[74] *"Advances in Petri Nets 1984"*, Edited by **G. Rozenberg with the cooperation of H. Genrich and G. Raucairol ,**

Sringer-Verlag , Berlin, Heidelberg, New York, Tokyo, *Lecture Notes in Computer Science - 188, 1985 .*

[75] **Alla, H.; Ladet, P.; Martinez, J.; and Silva-Suarez, M.,**

"Modeling and Validation of Complex Systems by Colored Petri Nets - Application to a Flexible Manufacturing System", Advances in Petri Nets 1984, Edited by **G. Rozenberg with the cooperation of H. Genrich and G. Raucairol ,**

Sringer-Verlag , Berlin, Heidelberg, New York. Tokyo, *Lecture Notes in Computer Science - 188, pp 15-31, 1985 .*

[76] **Schrage, Linus and Baker, Kenneth R.,**

"Dynamic -Programming Solution of Sequencing Problems with Precedence

Constraints", Operations Research, *Vol. 26, No. 3, pp 444-449, May-June 1978.*

[77] **Sen, Tapan; and Gupta, Sushil K.,**

"A State-of-Art Survey of Static Scheduling Research Involving Due Dates ", OMEGA, *Vol. 12, No. 1, pp 63-76, 1984.*

[78] **Sidney, Jeffrey B.; and Steiner, George,**

"Optimal Sequencing by Modular Decomposition: Polynomial Algorithms ", Operations Research, *Vol. 34, No. 4, pp 606-612, July-August 1986.*

[79] **Smith, Stephen F.; and Ow, Peng Si,**

"The Use of Multiple Problem Decompositions in Time Constrained Planning Tasks", CMU-RI-TR-85-11, Carnegie-Mellon University, *1985.*

[80] **Szwarc, Wlodzimierz,**

"Solution of the Akers-Friedman Scheduling Problem", Operations Research, *Vol. 8, No. 6, pp 782-788, November 1960.*

[81] **Valdes, Jacobo,**

"Parsing Flowcharts and Series-Parallel Graphs", Stanford University Computer Science Department Technical Report, STAN-cs-78-682, *December 1978.*

[82] **Valdes, Jacobo; Tarjan, Robert E.; and Lawler, Eugene L.,**

"The Recognition of Series Parallel Digraphs", SIAM Journal of Computing, *Vol. 11, No. 2, pp 298-313, May 1982.*

[83] **Villa, A.; Fassino, B.; and Rosetto, S.,**

"Performance Evaluation of Series Manufacturing Processes by Dynamic Ag-

gregate Model. ", Computer Integrated Manufacturing, *Vol. 8, pp 33-38, 1983.*

[84] **Villa, A.; Fassino, B.; and Rosetto, S.,**
"Discrete Event Dynamic Aggeregate Model of Series Manufacturing Processes", *Vol. 8, pp 9-17, 1983.*

[85] **Weeks, James K.; and Fryer, John S.,**
"A Simulation Study of Operating Policies in a Hypothetical Dual Constrained Job Shop", Management Science, *Vol. 22, No. 12, pp 1362-1371, August 1976.*

[86] **Wiest, Jerome D.; and Levy, Ferdinand K.,**
"A Management Guide to PERT/CPM : with GERT/PDM/DCPM and other Networks", Prentice-Hall Inc., Englewood Cliffs, N.J. 07632, *c1977.*

[87] **Wittrock, Robert J.,**
"Scheduling Algorithms for Flexible Flow Lines", IBM J. Res. Develop., *Vol. 29, No. 4, pp 401-412, 1985.*