LOOK-AHEAD TECHNIQUES FOR MICRO-OPPORTUNISTIC JOB SHOP SCHEDULING

Norman Sadeh

March 1991

CMU-CS-91-102

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This research was supported, in part, by the Defense Advanced Research Projects Agency under contract #F30602-88-C-0001, and in part by grants from McDonnell Aircraft Company and Digital Equipment Corporation. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency, McDonnell Aircraft, or Digital Equipment Corporation.

To my parents,
Denise and Leon,
and my wife, Patricia.

Acknowledgements

I would like to express my gratitude to the many people who helped make this dissertation possible. In particular, I would like to thank:

> My parents and my wife for their love and support throughout all these years;
>
> My advisor, Mark Fox, for his guidance, enthusiasm and unfailing support;
>
> The other members of my thesis committee, Tom Mitchell, Tom Morton, Judea Pearl, and Steve Smith, for many helpful comments on this research;
>
> All present and past members of the CORTES project for many stimulating discussions - Special thanks to Bob Frederking, who volunteered to proofread parts of this dissertation;
>
> All my friends and colleagues at the Center for Integrated Manufacturing Decision Systems and in the School of Computer Science for having made these past four years such a unique experience;
>
> Les Gasser for his friendship and support during my earlier graduate life in Los Angeles.

I would also like to acknowledge the financial support I received directly or indirectly from DARPA, Digital Equipment Corporation, and McDonnell Douglas, while a student at Carnegie Mellon, as well as the financial support of the Belgian American Educational Foundation during my first year in the U.S..

Abstract

Scheduling deals with the allocation of resources over time to perform a collection of tasks. Scheduling problems arise in domains as diverse as manufacturing, computer processing, transportation, health care, space exploration, education, etc. Scheduling problems are conveniently formulated as Constraint Satisfaction Problems (CSPs) or Constrained Optimization Problems (COPs). A general paradigm for solving CSPs and COPs relies on the use of backtrack search. Within this paradigm, the scheduling problem is solved through the iterative selection of a subproblem and the tentative assignment of a solution to that subproblem. Because most scheduling problems are NP-complete, even finding a solution that simply satisfies the problem constraints could require exponential time in the worst case. This dissertation demonstrates that the granularity of the subproblems selected by the backtrack search procedure critically affects both the efficiency of the procedure and the quality of the resulting solution. A so-called micro-opportunistic search procedure is developed, in which subproblems can be as small as a single operation. Look-ahead techniques are presented that constantly track the evolution of so-called bottleneck resources. These look-ahead techniques enable the scheduler to take advantage of the fine granularity of its search procedure by opportunistically revising its scheduling strategy as bottlenecks shift from one part of the problem space to another.

More specifically, two variations of the job shop scheduling problem are successively studied:

1. The first variation is one in which operations have to be performed within non-relaxable time windows. Heuristics to guide a micro-opportunistic scheduler are presented that are shown to outperform both generic CSP heuristics as well as specialized heuristics developed for similar scheduling problems.

2. The second part of this work deals with the factory scheduling problem. A micro-opportunistic factory scheduler, called MICRO-BOSS, is described that explicitly accounts for both tardiness and inventory costs. MICRO-BOSS is shown to outperform several competing scheduling techniques.

Experimental results also indicate that schedule quality deteriorates as the granularity of the search procedure increases, thereby suggesting the superiority of a micro-opportunistic approach to job shop scheduling over coarser search procedures such as those implemented in ISIS, OPT, and OPIS. They also indicate that the ability of the micro-opportunistic approach to constantly revise its search strategy is instrumental in efficiently solving problems in which some operations have to be performed within non-relaxable time windows.

Chapter 1
Introduction

1.1. Overview

This dissertation is concerned with dynamically revising a search procedure to focus on the most critical decision points and the most promising decisions at these points. Heuristics to redirect search are presented in the context of the job shop scheduling domain that have yielded important increases in both search efficiency and schedule quality over a variety of competing techniques.

Scheduling deals with the allocation of resources over time to perform a collection of tasks [Baker 74, Rinnooy Kan 76, French 82]. Scheduling problems arise in many domains. In the manufacturing domain, tasks, often referred to as jobs, correspond to parts or batches of parts that need to be processed on a set of machines [Muth 63, Johnson 74, Graves 81, Silver 85, Rodammer 88]. In hospitals, tasks are patients and resources are nurses, hospital beds or medical equipment required to treat the patients. Scheduling problems arise in schools, where tasks are classes and resources can be teachers, classrooms, and students [Feldman 89, Dhar 90]. Other examples include transportation-related problems (e.g. troop transportation, airport terminal scheduling, train scheduling [Fukumori 80], etc.), computer scheduling problems (e.g. CPU scheduling [Peterson 85]), space telescope scheduling [Muscettola 89, Johnston 90], appointment scheduling [Goldstein 75], etc.

Scheduling problems are conveniently formulated as either Constraint Satisfaction Problems (CSPs) or Constrained Optimization Problems (COPs). A CSP is defined by a set of variables and a set of constraints that restrict the values that can simultaneously be assigned to these variables [Montanari 71, Mackworth 77, Dechter 88]. A COP is a CSP with an objective function to be optimized subject to the problem constraints [Papadimitriou 82, Nemhauser 88, Fox 89, Dechter 90]. A general paradigm for solving CSPs and COPs relies on the use of *backtrack search* [Walker 60, Golomb 65, Bitner

75]. Within this paradigm, the scheduling problem is solved through the iterative selection of a subproblem and the tentative assignment of a solution to that subproblem. If in the process of building a solution, a partial solution is reached that cannot be completed without violating a problem constraint, one or several earlier assignments need to be undone. This process of undoing earlier assignments is called *backtracking*. It deteriorates the efficiency of the search procedure, and hence increases the time required to come up with a solution. Because of its mathematical structure [Garey 79], the general version of the scheduling problem studied in this dissertation (known as the *job shop scheduling problem*) can potentially require very large amounts of backtracking.

Traditionally, scheduling techniques have dealt with the backtracking issue by transforming the mathematical structure of the problem, and allowing some constraints to be relaxed as needed. This approach is commonly used in factory scheduling, where rather than requiring that all jobs be completed by their due dates, job due dates are relaxed as much as necessary in order to efficiently come up with a schedule. While producing efficient scheduling procedures, this approach often results in fairly poor solutions. In problems such as space telescope scheduling, where some due dates cannot be relaxed (e.g. scheduling the operations required to take snapshots of an eclipse), this approach simply does not work.

Instead, this dissertation investigates new scheduling techniques, which, short of guaranteeing backtrack-free search, provide quality schedules while generally maintaining search (i.e. backtracking) at a low level. Additionally, these techniques are capable of dealing with scheduling problems in which constraints such as due dates are not always relaxable. A key feature of the search techniques that will be discussed lies in their ability to dynamically adapt during search. In the Artificial Intelligence (AI) literature, this ability to *dynamically revise the search procedure* has been termed *opportunistic search* [HayesRoth 79, Erman 80, Stefik 81a]. While earlier opportunistic schedulers have relied on coarse problem decompositions, this work presents a so-called *micro-opportunistic scheduling* approach that allows for much finer subproblems. It is shown that the extra flexibility of a micro-opportunistic scheduling procedure can be exploited to constantly redirect the scheduling effort towards those resources that are likely to be the most difficult to schedule (so-called *bottleneck resources*). *Look-ahead techniques* are described that help the micro-opportunistic scheduler identify critical subproblems and promising solutions to these subproblems. These techniques are shown to allow the micro-opportunistic scheduler to perform particularly well both with respect to search efficiency and schedule quality.

More specifically, two studies of the micro-opportunistic scheduling paradigm are successively presented:

1. The first study is concerned with a variation of the generic Job Shop CSP, in which operations have to be performed within non-relaxable time windows (e.g. non-relaxable due dates and release dates). The study indicates that often generic CSP heuristics are not sufficient to guide the search for a solution to this problem. This is because these heuristics fail to properly account for constraint tightness and for the connectivity of the constraint graph. Instead, a probabilistic model of the search space is introduced. New heuristics are developed based on this model, that are shown to significantly speedup search.

2. The second part of this work deals with the factory scheduling problem. This optimization problem allows us to simultaneously study both the quality and efficiency performance of the micro-opportunistic approach. A micro-opportunistic factory scheduling system, called **MICRO-BOSS** (Micro-Bottleneck Scheduling System) is described that attempts to simultaneously reduce *both tardiness and inventory costs*. A large scale computational study indicates that the micro-opportunism embedded in MICRO-BOSS enables the scheduler to outperform a variety of competing scheduling techniques, both from Operations Research and Artificial Intelligence, under a wide range of scheduling conditions. The study also indicates that schedule quality deteriorates as the granularity of the subproblems used in the search procedure increases, thereby suggesting the superiority of a micro-opportunistic approach over coarser search procedures such as those implemented in the ISIS [Fox 83], OPT [Goldratt 80, Jacobs 84, Fox 87], and OPIS [Smith 86a, Ow 88a] scheduling systems.

The balance of this introduction gives a more formal definition of the job shop scheduling problem and reviews relevant work both in job shop scheduling and in Constraint Satisfaction/Constrained Optimization.

1.2. The Job Shop Scheduling Problem

The job shop scheduling problem requires scheduling a set of jobs on a finite set of resources. Each job is a request for the scheduling of a set of operations according to a process plan (often referred to as process routing in the manufacturing domain) that specifies a partial ordering among these operations. In order to be successfully performed, each operation requires one or several resources (e.g. a machine, a human operator, a set of fixtures), for each of which there may be several alternatives (e.g. several machines of the same type). Operations are atomic: once started they cannot be interrupted. In the simplest situation, each operation has a fixed duration, and each resource can only process one operation at a time.

In manufacturing, jobs typically have release dates, before which they cannot start (e.g. because the raw materials required to process a job are not scheduled to arrive before that date) and due dates by which, ideally, they should be completed. These dates are generally provided by a master scheduling module [Silver 85]. In make-to-order environments, also referred to as *open shops*, due dates correspond to delivery dates of customer orders. In make-to-stock environments, also referred to as *closed shops*, release and due dates are artificially generated to reduce the complexity of the problem, prevent the shop from being overflooded with inventory, and avoid stockouts (i.e. avoid running out of finished-goods to meet customer demand).

Job shop scheduling is a Constraint Satisfaction Problem (CSP) or Constrained Optimization Problem (COP). The variables of the problem are the operation start times, and the resources assigned to each operation, when there is a choice. The constraints of the problem include *precedence constraints* specified by the process routings and *capacity constraints* that prevent resources from being allocated to more operations than they can process at one time (resource capacity). Job release dates and due dates are constraints that restrict the domains of acceptable operation start times. Additional constraints may further restrict these domains such as constraints in factory scheduling that require some operations to be performed over a single shift. Similar constraints are often found in a variety of other scheduling problems, in which the domains of legal operation start times can be made up of disjoint time intervals (e.g. [Muscettola 89]).

Figure 1-1 depicts a small job shop problem with four jobs. Each operation is represented by a box labeled by a triple consisting of the name of the operation (e.g. O_1^1),

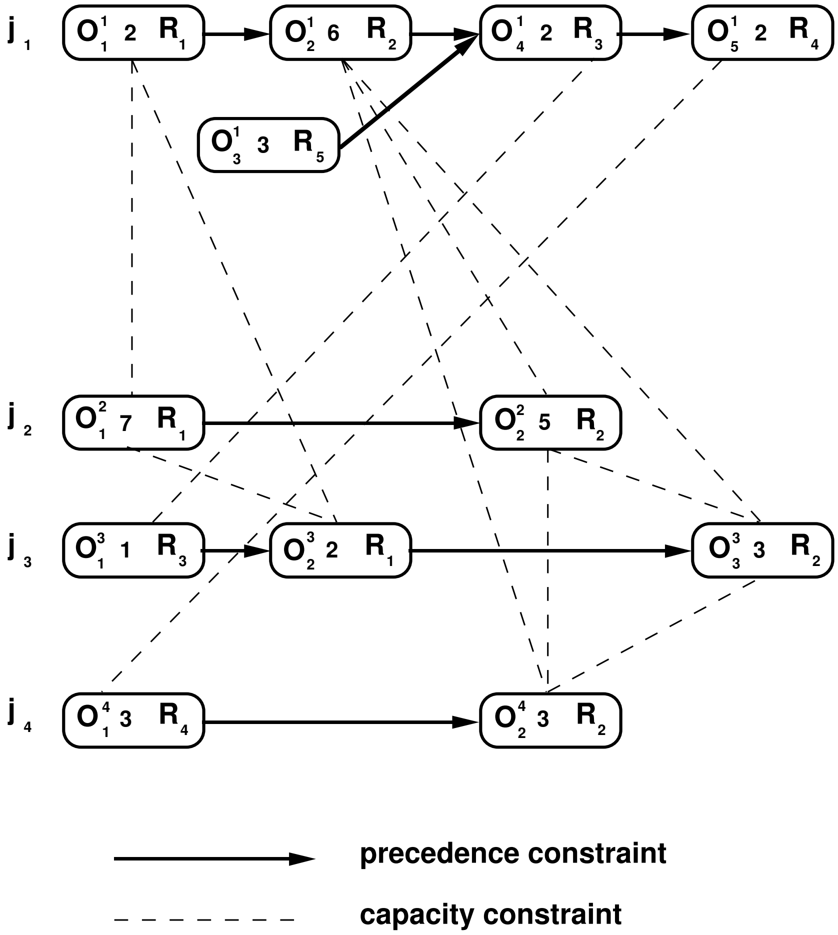


Figure 1-1: A simple job shop problem with 4 jobs. Each node is labeled by the operation that it represents, its duration, and the resource that it requires.

the duration of that operation (e.g. 2), and its resource requirement (e.g. R_1). The arrows represent precedence constraints. For instance, job j_1 requires 5 operations O_1^1, O_2^1,..., O_5^1. O_1^1 has to be performed before O_2^1, O_2^1 before O_4^1, etc. The other arcs in the graph represent capacity constraints. In this example, each resource is assumed to be capable of processing only one operation at a time. A capacity constraint between two operations expresses that these two operations cannot overlap in time. Clearly there is a capacity constraint between each pair of operations that require the same resource (i.e. there is a clique of capacity constraints for each resource). For instance, the clique for resource R_2 involves operations O_2^1, O_2^2, O_3^3 and O_2^4. Typically, each job also has a release date and a due date, which are not represented in Figure 1-1.

In some scheduling problems, it is sufficient to come up with a solution that satisfies the problem constraints (job shop CSP). Often however, not all solutions are equally preferred. When this is the case, job shop scheduling becomes a COP with an objective function to optimize. Different scheduling domains generally entail different scheduling criteria. There exists however a small set of criteria that pervades the scheduling literature. Because these metrics will be referred to later on in this work, they are now briefly defined. A more complete set of scheduling metrics is described in [Rinnooy Kan 76] along with equivalence relations between them.

- **Tardiness**: Job tardiness is defined as the amount of time a job completes past its due date[1] . The total (average) tardiness of a schedule is the total (average) job tardiness in that schedule.

- **Flowtime**: Job flowtime is the time spent by a job in the shop while being processed. It is the length of the time interval that spans from the release of the job to its completion. The total (average) flowtime of a schedule is the total (average) job flowtime in that schedule. In factory scheduling, average job flowtime is an indicator of the time required to produce a part. A schedule with a larger average flowtime will result, on the average, in more parts sitting in the shop waiting to be completed. In other words, there will be more *in-process inventory* (also referred to as work-in-process or work-in-progress).

- **Earliness**: Job earliness is defined as the amount of time a job completes before its due date. The total (average) earliness of a schedule is the total (average) job earliness in that schedule. In factory scheduling problems where parts completed before their due dates have to wait to be shipped, average job earliness is a measure of *finished-goods inventory*.

[1] Job tardiness should not be confused with job lateness, which is negative when a job completes before its due date. By definition the tardiness of a job completing before its due date is always zero.

- **Makespan**: The makespan of a scheduling problem is the length of the time interval that spans from the start time of the first released job to the completion time of the last completed job. This measure is appropriate in project scheduling, where there is a finite number of jobs to be carried out. In manufacturing domains, where new jobs arrive every day or every week, it is more natural to speak in terms of **throughput** (i.e. the number of jobs processed per unit of time) or in terms of **resource utilization** (i.e. the fraction of its time that a resource is active). These three metrics are equivalent in the sense that a reduction in the makespan of a schedule produces a proportionate increase in throughput and resource utilization [Rinnooy Kan 76].

There are many other possible ways to evaluate the quality of a schedule. In factory scheduling, each of the measures defined above looks only at one important source of costs in the production process. In Chapter 4, it will be argued that, instead of relying on any one of these measures, one should attempt to *simultaneously* account for the different costs hidden behind these measures. More formal definitions of the scheduling problems studied in this thesis are provided at the beginning of Chapters 3 and 4.

The next section briefly reviews the state of the art in scheduling and CSP/COP.

1.3. Related Work

The first part of this section reviews the state of the art in job shop scheduling. Rather than attempting to give a comprehensive survey of the field, the review focuses on a couple of recent techniques closely related to the work presented in this dissertation. In the process, the review attempts to point out shortcomings of these techniques and motivates the micro-opportunistic approach investigated in this thesis. The second part of the section summarizes relevant work in CSP/COP. Weaknesses of current CSP/COP techniques are briefly identified that need to be remedied in order to successfully apply the CSP/COP paradigm to real life problems such as job shop scheduling.

1.3.1. The State of the Art in Job Shop Scheduling

Job shop scheduling is an NP-hard problem [Garey 79, Lawler 82]. While *mathematical programming* techniques developed in Operations Research have proved particularly useful for aggregate planning [Lawrence 84], they are overwhelmed by the combinatorial number of discrete variables[2] required to represent job shop scheduling problems [Nemhauser 88]. More generally, with the exception of a couple of one-, two-, and three-machine scheduling problems, for which there exist efficient algorithms [Rinnooy Kan 76], all attempts to guarantee an optimal solution have failed.

Instead, job shop scheduling problems have traditionally been solved using *priority dispatch rules* [Baker 74, Panwalkar 77, French 82]. These are local decision rules of the greedy type that build schedules via a forward simulation of the shop. Because these rules lack a global view of the shop, they usually build up large amounts of inventory in front of bottleneck resources[3].

More recently, with the advent of more powerful computers, a couple of more sophisticated scheduling techniques have been developed [Goldratt 80, Fox 83, Ow 85, Adams 88, Ow 88a, Morton 88].

The first and by far most publicized of these techniques is the one developed by Eliyahu Goldratt and his colleagues in the late seventies and early eighties within the context of the **OPT** factory scheduling system [Jacobs 84, Fox 87][4]. Among other things, this system emphasized the need to distinguish between bottleneck and non-bottleneck machines. In OPT, bottlenecks drive the entire schedule as they determine the throughput of the plant. More specifically, a module called SERVE produces an initial infinite capacity schedule by working backwards from the job due dates. This initial schedule helps detect potential bottlenecks. The OPT module itself is then called upon to generate a forward finite capacity schedule that optimizes the utilization of these bottlenecks. The

---

[2]A simple job shop problem with *n* jobs and *m* resources of unary capacity, in which each job needs to be processed by each of the *m* resources, produces *m* cliques of $\binom{n}{2}$ capacity constraints. Because these capacity constraints are *disjunctive*, they each translate into a binary variable in a Mixed Integer Programming model.

[3]Informally, a *bottleneck* is a resource whose utilization is expected to be close to or larger than its available capacity.

[4]See also [Goldratt 86] for a lively description of the philosophy behind OPT.

resulting bottleneck schedules are passed back to the SERVE module, which schedules the non-bottleneck operations while trying to minimize inventory.

At about the same time, the **ISIS** factory scheduling system developed by Mark Fox and his team first demonstrated the potential of AI modeling and heuristic search techniques to help solve production scheduling problems [Fox 83, Smith 86b]. For the first time, rather than relying on a simplified model of the shop, ISIS attempted to deal with the full range of constraints and objectives encountered in the manufacturing domain. Unfortunately, the overall performance of the system was somewhat mitigated by the rigidity of its search procedure, which required jobs to be scheduled one by one (so-called *job-centered* approach). While this search procedure was particularly efficient at reducing inventory, it had problems optimizing utilization of bottleneck resources. As a result, a new system, called **OPIS**, was developed by Steve Smith, Peng Si Ow, and their colleagues [Smith 86a, Smith 86b, Ow 88a]. In OPIS, the notion of bottleneck resource was pushed one step further, as it was recognized that new bottlenecks can appear during the construction of the schedule. The OPIS scheduler combines two scheduling perspectives: a *resource-centered perspective* for scheduling bottleneck resources, and a *job-centered perspective* to schedule non-bottleneck operations on a job by job basis. Rather than relying on its initial bottleneck analysis, OPIS typically repeats this analysis each time a resource or a job has been scheduled. This ability to detect the emergence of new bottlenecks during the construction of the schedule and revise the current scheduling strategy has been termed *opportunistic scheduling* [Ow 88a]. Nevertheless, the opportunism in this approach remains limited in the sense that it typically requires scheduling an entire bottleneck (or at least a large chunk of it) before being able to switch to another one. For this reason, such scheduling techniques should in fact be called *macro-opportunistic*.

In reality, bottlenecks do not necessarily span over the entire scheduling horizon. Moreover they tend to shift before being entirely scheduled. A scheduler that can only schedule entire resources will not be able to take advantage of these considerations. Often it will overconstrain its set of alternatives before having worked on the subproblems that will most critically determine the quality of the entire schedule. This in turn will often result in poorer solutions. A more flexible approach would allow to quit scheduling a resource as soon as another resource is identified as being more constraining. In fact, in the presence of multiple bottlenecks, one can imagine a technique that constantly shifts attention from one bottleneck to another rather than focusing on the optimization of a single bottleneck at the expense of others. For these reasons, it seems desirable to investigate a more flexible approach to scheduling, or a *micro-opportunistic* approach, in which the evolution of bottlenecks is continuously monitored during the construction of the schedule, and the problem solving effort constantly redirected towards the most serious bottleneck. In its simplest form, this micro-opportunistic approach results in an *operation-centered* view of scheduling, in which each operation is considered an independent decision point and can be scheduled without requiring that other operations using the same resource or belonging to the same job be scheduled at the same time. This is the approach adopted in this thesis.

An alternative approach for dealing with the emergence of new bottlenecks has been recently proposed by Adams, Balas and Zawack [Adams 88] (See also [Dauzere-Peres 90]). This approach, known as the *Shifting Bottleneck Procedure* (**SBP**), sequences resources one by one, while continuously reoptimizing the schedule of resources sequenced earlier. SBP has allowed for the production of schedules with near-optimal makespan for problems with up to 500 operations. Attempts to generalize the procedure to account for due dates and more complex objectives seem to have been less successful so far [Serafini 88]. It should be pointed out that the idea of continuously reoptimizing the current partial schedule is not incompatible with the micro-opportunistic approach[5].

The **SCHED-STAR** scheduling module developed by Morton, Lawrence, Rajagopolan and Kekre takes yet another approach to dealing with bottleneck resources [Morton 88]. Rather than relying on a simplified objective function, this price-based factory scheduler accounts directly for both tardiness and inventory costs in order to minimize the net present value of cash flows in the plant. Based on these exogenous costs, implicit *resource prices* are derived via internal simulation that reflect resource contention in function of time. These prices are used to determine job releases and attribute priorities to competing jobs at each machine. The MICRO-BOSS factory scheduling system described in Chapter 4 also uses tardiness and inventory costs to help identify bottleneck resources. The design of SCHED-STAR prevents however the system to take full advantage of its bottleneck analysis. In particular, the system builds schedules via a forward simulation of the shop. As a result, release decisions are always made before

---

[5]By only scheduling those operations that appear to be most critical, a micro-opportunistic approach should in fact allow for more effective reoptimization procedures.

bottleneck sequencing decisions. This is contrary to the lessons taught by OPT[6]. SCHED-STAR makes up for this potential weakness by iterating its simulation, using its previous schedule to derive new resource prices and generate new schedules. It is not clear how effective this iterative approach can be and how dependent it is on the ability to guess a good initial schedule.

The main purpose of this brief review was to emphasize the need for a micro-opportunistic scheduling approach in order to produce better schedules. As will be demonstrated in this thesis, the extra flexibility of the micro-opportunistic search procedure advocated here, not only helps produce better schedules but also enables a scheduling system to deal more effectively with operations that need to be performed within non-relaxable time windows (e.g. non-relaxable release dates and due date constraints).

More comprehensive reviews of the job-shop scheduling literature can be found in [Baker 74, Rinnooy Kan 76, French 82, Lawler 82]. For recent surveys of the production scheduling literature, the reader is referred to [Graves 81] and [Rodammer 88]. Conventional approaches to production planning and scheduling are discussed at length in [Johnson 74, Hax 84, Silver 85].

The second part of this review deals with earlier work in CSP/COP. It includes references to other applications of the CSP paradigm to job shop scheduling problems.

1.3.2. Relevant Work in CSP/COP

The general CSP is NP-complete [Garey 79]. Techniques for solving the general CSP extend the depth-first backtrack search procedure [Walker 60, Golomb 65, Bitner 75, Pearl 84], in which a solution is incrementally built by instantiating one variable (or more generally one subproblem) after another. Every time a variable is instantiated, a new *search state* is created, where new constraints are added to account for the value assigned to that variable. If a partial solution is built that cannot be completed, the current search state is said to be a *deadend*. The system needs to *backtrack* to an earlier less complete solution, and try alternative variable assignments. Search typically stops

[6]Indeed, by first sequencing bottleneck machines, systems like OPT determine how much can be produced by the plant. Bottleneck schedules are then used to determine when to release jobs without starving the bottleneck and without building excess inventory.

when a first solution has been found, or when all alternatives have been tried without success. In the latter case, the CSP is said to be infeasible.

Because the general CSP is NP-complete, backtrack search may require exponential time in the *worst case*. Research in CSP has produced four types of techniques that can help improve the *average* efficiency of the basic backtrack search procedure [Dechter 91]:

1. *Consistency Enforcing (Checking) Techniques*: These techniques are meant to prune the search space by eliminating local inconsistencies that cannot participate in a global solution [Mackworth 85]. This is done by inferring new constraints and adding them to the current problem formulation. If during this process the domain of a variable becomes empty, a deadend situation has been identified. Consistency enforcing techniques can be applied either before or during search. In general, achieving higher levels of consistency reduces backtracking. There is however a tradeoff [Haralick 80, Mackworth 85, Nadel 88] between the amount of computation spent enforcing consistency and the savings achieved in the actual search. Partial consistency enforcing algorithms have been classified according to the degree of consistency that they achieve between variables. In particular, consistency enforcing algorithms that achieve consistency among subsets of *k* variables are said to enforce *k-consistency* [Freuder 82]. In general *k-consistency* algorithms have a complexity exponential in *k*.

2. *Variable and Value Ordering Heuristics*: These heuristics are concerned with the order in which variables are instantiated and values assigned to each variable. A good *variable ordering* is one that starts with the variables that are the most difficult to instantiate. By first instantiating these critical variables, one hopes to avoid wasting a lot of time building partial solutions that cannot be completed. A good *value ordering* heuristic is one that leaves open as many options as possible to the remaining uninstantiated variables (i.e. a so-called *least constraining value* ordering heuristic). These heuristics are meant to reduce the chances of backtracking and its cost, when it cannot be avoided. Both theoretical and experimental studies show

that variable and value ordering heuristics can significantly reduce search [Haralick 80, Purdom 83, Stone 86, Dechter 88, Zabih 88, Dechter 89a, Fox 89].

3. *Deadend Recovery Techniques*: These techniques help decide which earlier assignments to undo in order to recover from a deadend. The simplest such strategy is known as *chronological backtracking*. It consists in undoing the last assignment, and trying another one (if there is one left). More sophisticated deadend recovery strategies have been designed that attempt to go back to the source of failure and undo one or several of the assignments that prevent the current partial solution from being successfully completed [Stallman 77, Doyle 79, Gaschnig 79, Dechter 89b]. Techniques have also been developed that attempt to "learn" from deadends by abstracting from these situations a set of partial assignments that are inconsistent and should therefore be avoided in the future[7] [Dechter 89b].

4. *Hierarchical Reformulation Techniques*: These are techniques to automatically define abstractions in a CSP. If carefully chosen, such abstractions have been known to significantly reduce search [Sacerdoti 74, Sussman 80, Stefik 81b, Fox 86]. With the exception of [Dechter 89c, Knoblock 91], very little formal work has been done in this area.

At the time this research started, a good deal of experimental results reported in the CSP literature had been obtained on toy problems such as N-queens[8]. The problems used in these experiments generally involved 10 to 20 variables, each with at most 10 to 20 values. How the CSP paradigm would scale up on harder and larger problems such as job shop scheduling remained an open issue. This thesis demonstrates that while the CSP paradigm itself scales up pretty well, the particular heuristics that have been proposed so far in the literature are often too weak to produce good results. Chapter 3 points to the

[7]This last technique can also be seen as a form of dynamic consistency enforcement.

[8]The N-queens problem requires positioning N queens on an $N \times N$ chess board so that they cannot attack each other according to chess rules [Kraitchick 42]. This problem is not NP-hard [Yaglom 64, Abramson 89].

shortcomings of popular variable and value ordering heuristics. A new probabilistic model is also introduced that allows for the definition of more powerful variable and value orderings. This model was influenced by the work of Bernard Nadel [Nudel 83, Nadel 86a, Nadel 86b, Nadel 86c], who himself generalized a probabilistic model introduced earlier by Haralick and Elliott [Haralick 80]. In his work, Nadel identified a small set of measures that he used to select between alternative search orderings based on complexity estimates. A key measure in Nadel’s work is that of *constraint satisfiability*, namely the number of ways in which a constraint can be satisfied. The probabilistic estimates developed by Nadel solely account for these constraint satisfiabilities and the ways in which constraints in a problem are connected to each other[9]. It seems however that, in hard problems like job-shop scheduling, measures of constraint satisfiability are not sufficient. The difficulty in satisfying a constraint, which will from now on be referred to as the **tightness** of that constraint, critically depends on the *specific ways* in which that constraint interacts with other problem constraints, i.e. the tightness of a constraint is generally determined by the specific pairs of values allowed by that constraint and the other interacting constraints. For instance, in job shop scheduling, interactions between precedence constraints can be very different from interactions between capacity constraints, even if these constraints form similar constraint graphs and have similar satisfiabilities.

Recently several applications of CSP techniques to job shop scheduling problems have been reported in the literature [Collinot 88, LePape 88, Dincbas 88, Keng 89, Burke 89, Prosser 89, Elleby 89, Johnston 90, Badie 90, Minton 90]. Of particular relevance to this dissertation is the work of Naiping Keng [Keng 89], who developed a pair of variable and value ordering heuristics that attempts to account for interactions between capacity constraints. His heuristics are described in more detail in Chapter 3. Experimental results are also presented indicating that, although more powerful than generic CSP heuristics, Keng’s heuristics still fail to account for some important constraint interactions. Recently, Minton proposed a so-called repair heuristic approach to job-shop scheduling [Minton 90]. Within this approach, each variable is initially assigned a tentative value. This initial assignment is then refined in order to get rid of all constraint violations. The repair heuristic suggests to first work on the variable whose current tentative assignment

[9]The term "connected" refers to a graphical representation of a CSP with binary constraints, known as the *constraint graph* of the problem. In a constraint graph, each variable is represented by a node, and binary constraints are represented by arcs between two nodes.

violates the largest number of constraints, and to replace this assignment with one that minimizes the number of remaining conflicts. While this technique has performed particularly well on the N-queens problem, its performance on more difficult problems, such as job-shop scheduling, remains to be assessed. A potential drawback of Minton's technique is its reliance on a single tentative assignment to identify critical variables and promising values for these variables. Indeed, partial solutions produced by the procedure may in fact bear little resemblance to earlier tentative assignments. As a consequence, the look-ahead provided by the repair heuristic will generally be limited to a couple of search states, rather than applying to the entire problem.

So far very little work has been done to extend the CSP paradigm to deal with optimization problems[10]. While mathematical programming techniques, both continuous and discrete, have provided elegant solutions to many COPs, there remain problems on which these techniques have had very little impact so far. As pointed out earlier, job shop scheduling belongs to this class of more difficult problems. The work reported in Chapters 4 and 5, within the context of the MICRO-BOSS factory scheduler, indicates that the CSP paradigm can sometimes provide a viable alternative to traditional Mixed Integer Programming techniques.

Extending the CSP approach to deal with optimization problems is far from trivial. Good variable and value ordering heuristics to find a feasible solution will often perform poorly in the presence of an objective function. Earlier experiments reported in [Sadeh 89a, Sadeh 90], in the context of job shop scheduling problems, indicate that, although particularly effective to reduce search, least constraining value ordering heuristics, such as those advocated by Keng [Keng 89], tend to produce poor solutions. Looking for a good solution generally requires more constraining value orderings, and may therefore result in more backtracking. This will generally require more powerful consistency enforcing mechanisms (see Chapter 2), and variable ordering heuristics that account for the bias of the value ordering heuristic towards the selection of better values (see Chapter 4).

[10]An exception is the work of Rina Dechter, Avi Dechter, and Judea Pearl who identified a class of COPs with acyclic constraint graphs that can be solved to optimality in polynomial time using a dynamic programming technique [Dechter 90].

1.4. Summary of Contributions

The main contributions of this dissertation can be summarized as follows:

- *A Micro-opportunistic Approach to Job Shop Scheduling*: While earlier schedulers such as OPT, ISIS, and OPIS have relied on coarse problem decompositions, this dissertation presents a micro-opportunistic approach to job shop scheduling that allows for much finer subproblems. It is shown that the extra flexibility of this approach can be exploited to constantly redirect the scheduling effort towards those bottleneck operations that appear to be the most critical. Experimental results indicate that the ability of the micro-opportunistic approach to constantly revise its search procedure allows for significant increases in schedule quality and is instrumental in efficiently solving problems in which some operations have to be performed within non-relaxable time windows (e.g. non-relaxable release and due dates).

- *Application of the CSP Paradigm to Job Shop Scheduling/A Probabilistic Model of the Search Space*: This thesis demonstrates that, while the CSP paradigm (i.e. combining consistency enforcing techniques with variable and value ordering heuristics) scales up to larger and harder problems such as job shop scheduling, the particular heuristics that had been proposed earlier are not sufficient for problems such as job shop scheduling. This is because these heuristics fail to properly account for constraint tightness and for the connectivity of the constraint graph. Instead, a new probabilistic model of the search space is defined that allows for the definition of more powerful variable and value ordering heuristics. These heuristics have allowed for the solution of scheduling problems with non-relaxable time windows that could not be solved efficiently by prior techniques.

- *Extension of the CSP Paradigm to deal with COPs*: This dissertation also extends the CSP paradigm to deal with job shop scheduling as an optimization problem. While least constraining value ordering heuristics used to solve CSPs are particularly good at reducing backtracking, they typically fail to provide good solutions. Instead, more constraining value ordering heuristics are required. This in turn requires the use of stronger consistency enforcing mechanisms and more accurate variable ordering heuristics in order to maintain backtracking at a low level. This thesis describes such mechanisms within the framework of the job shop scheduling problem. In particular, a new variable ordering heuristic is described that identifies critical operations as those involved in important tradeoffs. The resulting scheduler implements a two-step optimization procedure. In the first step, reservation assignments are optimized within each jobs, and critical operations are identified as those whose good reservations conflict most with the good reservations of other operations. Reservations for the critical operations are then ranked according to their ability to minimize the costs incurred by the operation itself and the operations with which it competes.

- *The MICRO-BOSS Factory Scheduling System*: One of the most tangible contribution of this thesis is certainly the MICRO-BOSS factory scheduling system itself, which is able to deal explicitly with tardiness costs, in-process inventory costs, and finished-goods inventory costs. This system has outperformed a variety of competing scheduling techniques under a wide range of scheduling conditions. MICRO-BOSS introduces the notion of bottleneck operation, which is directly formalized in terms of tardiness and inventory costs and accounts for earlier scheduling decisions.

A more complete summary of contributions is provided in Chapter 6.

1.5. Thesis Outline

Chapter 2 describes the micro-opportunistic search procedure studied in this dissertation with a special emphasis on consistency enforcing mechanisms. Chapter 3 deals with a generic version of the job shop CSP, in which operations may require several resources for which there can be alternatives, and some operations have to be performed within one or several non-relaxable time windows. The chapter starts by reviewing some popular variable and value ordering heuristics, and explains why these heuristics typically fail when applied to job shop scheduling. A probabilistic model of the search space is then introduced that allows for the definition of more powerful variable and value ordering heuristics. Experimental results presented at the end of this chapter demonstrate that these new variable and value ordering heuristics outperform a variety of other CSP heuristics (both generic heuristics and specialized CSP heuristics designed for job shop scheduling). Chapter 4 deals with job shop scheduling as a COP. This chapter presents the look-ahead techniques developed within the context of the MICRO-BOSS factory scheduler to help the system decide which operation to schedule next and which reservation to assign to that operation. Several experimental studies are reported in Chapter 5 that demonstrate the superiority of MICRO-BOSS over both traditional priority dispatch rules and coarser opportunistic scheduling techniques. Chapter 6 concludes this dissertation with a set of final remarks.

Chapter 2
The Micro-opportunistic Search Procedure

2.1. Overview

In the micro-opportunistic (or operation-centered) approach studied in this dissertation, each operation is considered an independent decision point. Any operation can be scheduled at any time, if deemed appropriate by the scheduler. There is no obligation to simultaneously schedule other operations upstream or downstream within the same job, nor is there any obligation to schedule other operations competing for the same resource.

The micro-opportunistic scheduler proceeds according to the generic backtrack search procedure by iteratively selecting an operation to be scheduled and a reservation to be assigned to that operation (i.e. a start time and a specific resource for each resource requirement for which there are several alternatives). Every time an operation is scheduled, a new *search state* is created, where new constraints are added to account for the reservation assigned to that operation. If an unscheduled operation is found to have no possible reservations left, a *deadend state* has been reached: the system needs to *backtrack* (i.e. it needs to undo some earlier reservation assignments in order to be able to complete the schedule). If the search state does not appear to be a deadend, the scheduler moves on and looks for a new operation to schedule and a reservation to assign to that operation. This process goes on until all operations have been scheduled, or until the scheduling problem has been found to be infeasible.

Because job shop scheduling is NP-complete, this procedure could require exponential time in the *worst case*. In practice, as demonstrated by the experimental studies presented in this thesis, it is generally possible to maintain the *average complexity* of the procedure at a very low level while producing quality schedules. This is achieved by interleaving search with the application of consistency enforcing techniques and a set of look-ahead techniques that help decide which operation to schedule next (so-called *variable ordering heuristic*) and which reservation to assign to that operation (so-called *value ordering heuristic*).

1. **Consistency Enforcement**: Consistency enforcing techniques are used to prune possible reservations (of unscheduled operations) that have become unavailable due to earlier reservation assignments. By constantly inferring new constraints resulting from earlier scheduling decisions, these techniques reduce the chance of selecting reservation assignments that are inconsistent with earlier scheduling decisions. This reduces the chances of backtracking. Additionally, by allowing for the early detection of deadend states, these techniques limit the amount of work wasted in the exploration of fruitless alternatives. In other words these techniques reduce both the frequency and the amount of backtracking.

2. **Look-ahead Analysis**: The purpose of the look-ahead analysis is to help identify critical operations and promising reservations for these operations. Operation criticality and reservation goodness are measures that are not intrinsic to a problem. They depend on earlier scheduling decisions (i.e. they change from one search state to another), and on the objective to be optimized. In this dissertation, two variations of the job shop scheduling problem are successively studied, one in which the only concern is to come up with a feasible solution as fast as possible (job shop CSP), and one in which the objective is to efficiently come up with as good a solution as possible (job shop COP). In the job shop CSP, a critical operation is one whose reservations are likely to become unavailable if other operations were scheduled first. By scheduling these operations first, the scheduler avoids building partial schedules that it will not be able to complete later on. Similarly a good reservation, in the job shop CSP, is one that leaves enough room to other unscheduled operations so that the schedule can be completed with minimal backtracking. In the case of the job shop COP, the notions of operation criticality and reservation goodness appear to be more complex. This dissertation suggests that, *in COPs, critical variables (e.g. critical operations in job shop scheduling) are variables participating in important tradeoffs and promising values for these variables are values that optimize these tradeoffs*. An important tradeoff is one that critically impacts the quality of the *entire* solution. By first optimizing the most

important tradeoffs in a problem, the system can later use the solutions to these tradeoffs to help solve the remainder of the problem. Indeed, once critical tradeoffs have been worked out, the remainder of the problem tends to become more decoupled, and hence easier to optimize. Chances of backtracking tend to simultaneously subside as well. A system, that does not attempt to work out critical tradeoffs first, runs the risk overconstraining its set of alternatives before having worked on the subproblems that will impact most the quality of the entire solution.

This thesis describes a unifying framework in which measures of **reservation reliance** (i.e. the reliance of an operation on the availability of a reservation) and **resource contention** are computed to identify critical operations and promising reservations for these operations. This framework is applied to both the CSP and COP versions of the job shop scheduling problem studied in this dissertation. In job shop CSPs, the reliance of an operation on the availability of a specific reservation (in a search state) is defined as a function of the number of alternative reservations still available to that operation (in that search state). Operations with a small number of alternative reservations left are said to rely more on each one of their remaining possible reservations. In job shop COPs, the reliance of an operation on a reservation is defined as a function of the expected merit of assigning that reservation to the operation compared to the merit of alternative reservations still available to the operation, and compared to differences in merit between the reservations still available to other operations. In other words, operations with only a small fraction of their remaining reservations expected to result in a high value of the objective are said to highly rely on this small fraction of good reservations. Once reservation reliance has been evaluated, critical resource/time intervals are identified as highly relied upon resource/time intervals, and critical operations as those that most heavily rely on the availability of these critical resource/time intervals. While measures of reservation reliance depend on whether the scheduling problem is a CSP or a COP, the procedure used to measure resource contention and identify critical operations remains the same.

The next section gives a more formal description of the top-level procedure embedded in the micro-opportunistic approach.

2.2. The Search Procedure

Concretely, the micro-opportunistic search procedure starts in a search state in which no operation has been scheduled yet, and proceeds according to the following steps:

1. If all operations have been scheduled then stop, else go on to 2;
2. Apply the **consistency enforcing** procedure;
3. If a deadend is detected then **backtrack** (i.e. select an alternative if there is one left and go back to 1, else stop and report that the problem is infeasible), else go on to step 4;
4. Perform the **look-ahead** analysis: evaluate the reliance of each unscheduled operation on the availability of its remaining possible reservations, and measure resource contention over time;
5. Select the next operation to be scheduled (so-called **operation ordering** heuristic): select the operation that relies most on the most contended (i.e. most highly relied upon) resource/time interval;
6. Select a promising reservation for that operation (so-called **reservation ordering** heuristic)
7. Create a **new search state** by adding the new reservation assignment to the current partial schedule. Go back to 1.

In this search procedure, the so-called opportunistic behavior results from *the ability of the scheduler to constantly revise its search strategy and redirect its effort towards the scheduling of the operation that appears to be the most critical in the current search state*. This degree of opportunism differs from that displayed by other approaches where the scheduling entity is an entire resource or an entire job [Ow 88a], i.e. where an entire resource (or at least a large chunk of it) or an entire job (or at least a large portion of it) needs to be scheduled before the scheduler is allowed to revise its current scheduling strategy.

The results reported in this dissertation were obtained using a simple chronological backtracking scheme. The remainder of this section gives a more detailed description of the consistency enforcing procedure.

2.3. Enforcing Consistency

Clearly there is a tradeoff between the time spent enforcing consistency in each search state and the actual savings achieved in search time. Furthermore, because job shop scheduling is NP-complete, a consistency enforcing mechanism that would immediately detect all deadend states (i.e. all partial schedules that cannot be completed) is likely to require exponential time. Overall, it is usually a better idea to try to catch certain types of deadend states that are easy to detect. Deadend states that are not directly caught by the consistency enforcing mechanism require some amount of search to be identified: in the process of attempting to complete the partial schedule of such hard-to-detect deadend states, the scheduler eventually reaches deadend states that are easier to recognize, and backtracks. After having exhaustively tried all possible ways in which to complete the partial schedule of a hard-to-detect deadend state, the scheduler backs up to that state, and labels it as a deadend. By always instantiating operations that are difficult to schedule, the scheduler is able to greatly reduce the amount of search necessary to detect these more difficult deadend states. Indeed, by instantiating difficult variables (i.e. highly constraining variables) in a hard-to-detect deadend state, the scheduler creates a new deadend state that is even more constrained, and hence easier to detect.

Both theoretical and empirical studies [Montanari 71, Haralick 80, Nudel 83, Mackworth 85, Dechter 88, Dechter 89a, Nadel 88] indicate that, by and large, it is not a good idea to seek very high levels of consistency in a search state. Simply achieving 2-consistency (also referred to as *arc-consistency* [Waltz 75, Mackworth 77]) levels in a *forward checking* [Haralick 80] fashion often appears to be a good tradeoff. A binary constraint (i.e. "arc") restricting two variables is said to be *arc consistent* when the sets of remaining possible values of both variables are such that any value in the set of one variable is supported by/compatible with at least one value in the set of the other. Achieving arc consistency with respect to a binary constraint requires pruning all values that do not meet this condition. In general, for two variables with k possible values each, this requires at most $O(k^2)$ consistency checks. A search state is said to be totally arc-consistent if all its constraints have been made arc consistent. Forward checking is a form of partial arc-consistency [McGregor 79, Haralick 80]. It only requires achieving arc-consistency with respect to binary constraints connecting non-instantiated variables to instantiated ones. Forward checking does not attempt to achieve arc-consistency between non-instantiated variables.

In job shop scheduling problems, it is possible to achieve complete arc consistency with respect to precedence constraints in $O(\alpha)$ time, where α is the number of precedence constraints in the problem [Tarjan 83a]. As in PERT/CPM [Johnson 74], this is done using a longest path algorithm that takes advantage of the acyclicity of the precedence graph to produce an efficient order in which to update pairs of earliest/latest possible start times for each unscheduled operation[11]. It turns out that this method actually guarantees decomposability[12] [Davis 87, Dechter 89d]. Hence, in the absence of capacity constraints (e.g. problems in which no two operations require the same resource), updating pairs of earliest/latest possible start times for each unscheduled operation in each search state is sufficient to guarantee backtrack-free search.

Enforcing consistency with respect to capacity constraints appears to be more difficult due to the disjunctive nature of these constraints. For these constraints, a forward checking type of consistency enforcement is carried out with respect to capacity constraints [LePape 87]. In other words, whenever a resource is allocated to an operation over some time interval, this procedure marks that time interval as unavailable to all other operations requiring that same resource.

Because this technique achieves only partial consistency with respect to capacity constraints, it is not possible to guarantee backtrack-free search. Sometimes the scheduler reaches a search state, in which several unscheduled operations competing for a resource appear to each have some possible reservations left, while the total capacity available on the resource is actually insufficient to accommodate all these operations together. In order to catch some of these situations more rapidly, it was found useful to add a set of redundant binary constraints to the problem formulation. These constraints express that, if two operations, O_i^k and O_j^l, require the same resource and are constrained in such a way that they each totally rely on the availability of some time interval on that

---

[11]See also [Smith 83] for an incremental version of this procedure, as new operations are scheduled.

[12]A constraint network is said to be decomposable iff every assignment of values to any subset of K variables that satisfies all the constraints among these K variables can be extended by an assignment of a value to any variable not in the subset, in such a way that the resulting set of $K+1$ assignments satisfies all the constraints among the $K+1$ variables [Dechter 89d]. Decomposability is sufficient to ensure backtrack-free search.

resource's calendar (even though they may still have several possible start times left), then these two time intervals cannot overlap. Let R_{ip}^k denote the *p*-th resource required by O_i^k and R_{jq}^l the *q*-th resource required by O_j^l . Let also est_i^k, lst_i^k and du_i^k respectively denote the earliest possible start time, latest possible start time, and duration of O_i^k, and est_j^l, lst_j^l and du_j^l denote those of O_j^l. The binary constraint between two operations, O_i^k and O_j^l, can then be formulated as:

$$(\forall p\,\forall q\, R_{ip}^k \neq R_{jq}^l) \vee (lst_i^k \geq est_i^k + du_i^k) \vee (lst_j^l \geq est_j^l + du_j^l)$$
$$\vee (lst_j^l \geq est_i^k + du_i^k) \vee (lst_i^k \geq est_j^l + du_j^l)$$

Figure 2-1 illustrates a simple situation where two operations O_i^k and O_j^l violate one such constraint. Both operations are assumed to require the same resource, say $R = R_{i1}^k = R_{j1}^l$. $eft_j^l = est_j^l + du_j^l$ is O_j^l's earliest possible finish time, and $lft_j^l = lst_j^l + du_j^l$ its latest possible finish time. Similarly eft_i^k is O_i^k's earliest possible finish time, and lft_i^k its latest possible finish time. Whichever start time is assigned to O_j^l, O_j^l will need resource *R* between lst_j^l and eft_j^l (this would not be the case if $lst_j^l \geq eft_j^l$). Similarly O_i^k will need that same resource between lst_i^k and eft_i^k. In Figure 2-1, these two time intervals, namely [lst_j^l, eft_j^l[and [lst_i^k, eft_i^k[, overlap. This indicates that the resource is oversubscribed: a deadend state has been detected.

These types of conflicts can be efficiently avoided by maintaining for each resource a calendar (e.g. a bit vector) that records each of the time intervals on which an *unscheduled* operation totally relies (a similar but separate calendar is used to keep track of actual reservations). As soon as two operations totally rely on overlapping resource/time intervals, a deadend state is detected[13].

Clearly there remain more complex deadend situations that will not be immediately caught by the consistency checking mechanism just described. These deadends are the ones that will produce backtracking. In practice, it has been our experience, that the

---

[13]Notice that this is not equivalent to achieving full arc-consistency. Full arc-consistency would require pruning all start times that have become unavailable due to unscheduled operations that totally rely on the availability of some resource/time intervals. This would require a more complex procedure in which some operations may have to be inspected several times: as their earliest/latest possible start time intervals shrink, new operations may start to totally rely on the availability of some resource/time intervals, or operations that already relied totally on some resource/time intervals may see these intervals grow longer. This in turn may affect the earliest and/or latest possible start times of other operations, and so on. Although these computations can still be performed efficiently, it is not clear whether, on the average, this would reduce total processing time.

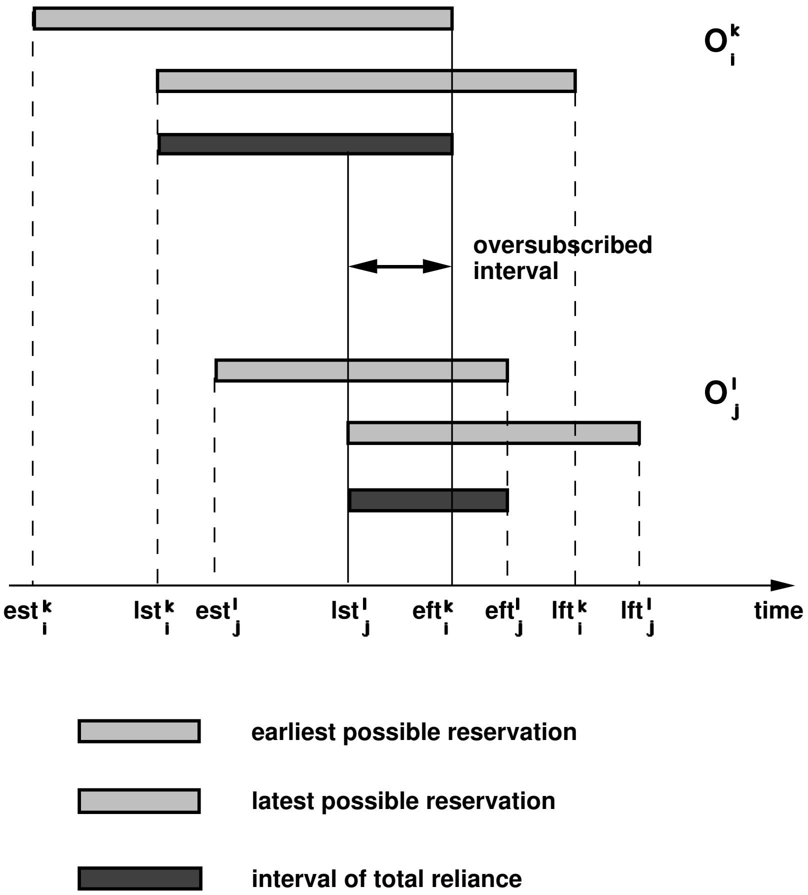


Figure 2-1: A situation with an oversubscribed resource that can easily be detected.

consistency enforcing techniques described in this section generally provide a good compromise between the amount of consistency enforcement performed in each search state and the amount of backtracking resulting from the deadends that are not immediately detected. This is because the micro-opportunistic approach does not rely solely on these consistency enforcing techniques to maintain backtracking at a low level. Instead, the approach also relies on powerful look-ahead techniques that help identify critical operations and promising reservations for these operations. Experimental results presented in Chapters 3 and 5 indicate that these techniques play a critical role in keeping backtracking at a low level.

Chapter 3
The Job Shop Constraint Satisfaction Problem

3.1. Introduction

This chapter studies a variation of the job shop scheduling problem, referred to as the job shop CSP, in which operations have to be performed within non-relaxable time windows. Examples of such problems include factory scheduling problems, in which some operations have to be performed within one or several shifts, spacecraft mission scheduling problems, in which time windows are determined by astronomical events over which we have no control, factory rescheduling problems, in which a small set of operations need to be rescheduled without revising the schedule of other operations, etc. The objective assumed in this chapter requires finding a feasible schedule as fast as possible. A COP variation of this problem is studied in Chapter 4.

Because this version of the job shop scheduling problem is NP-complete, the worst-case complexity of any procedure is expected to be exponential. CSP techniques that interleave search with consistency enforcing techniques and variable/value ordering heuristics have been reported to generally allow for important increases in search efficiency when applied to other NP-complete CSPs. This chapter aims at determining if similar savings can be obtained in job shop scheduling, and, more generally, if, *on the average*, the CSP paradigm is sufficient to efficiently solve job shop scheduling problems. In order to address this difficult question, the chapter first reviews generic variable and value ordering heuristics that have been reported to perform particularly well on other CSPs. The review suggests that these heuristics are too weak to solve hard problems like job shop scheduling. This is because these heuristics fail to properly account for the interactions induced by the high connectivity of the constraint graphs often encountered in job shop scheduling problems. The chapter then introduces a new probabilistic framework, within which variable and value ordering heuristics are defined that better account for these interactions. A key to defining these more powerful heuristics lies in the ability of the probabilistic framework to provide estimates of the **reliance** of an operation on the availability of a reservation, and measures of **resource contention** between unscheduled operations.

Experimental results indicate that these new heuristics outperform both generic CSP heuristics as well as more specialized heuristics recently developed for similar scheduling problems. The results also suggest that, despite its exponential worst-case complexity, the job shop scheduling problem admits many instances that can be solved efficiently. There remain however some particularly difficult problems that require larger amounts of search. A second set of experiments is also reported that attempts to assess the impact of the granularity of the micro-opportunistic approach on the efficiency of the backtrack search procedure. The results confirm our intuition that the fine granularity of the micro-opportunistic approach is indeed instrumental in achieving the high search efficiency observed in the first set of experiments.

Earlier discussions of different variations of the techniques reported in this chapter can be found in [Sadeh 88, Sadeh 89a, Sadeh 89b, Sadeh 89c, Fox 89, Sadeh 90].

3.2. Problem Definition

The job shop scheduling problem requires scheduling a set of jobs $J=\{j_1,\ldots,j_n\}$ on a set of physical resources $RES=\{R_1,\ldots,R_m\}$. Each job j_l consists of a set of operations $O^l=\{O^l_1,\ldots,O^l_{n_l}\}$ to be scheduled according to a process routing that specifies a partial ordering among these operations (e.g. O^l_i BEFORE O^l_j). This chapter assumes job shop CSPs with *tree-like* process routings. A tree-like process routing is one whose graph of precedence constraints forms a tree[14]. Examples of tree-like process routings are represented in Figure 3-1.

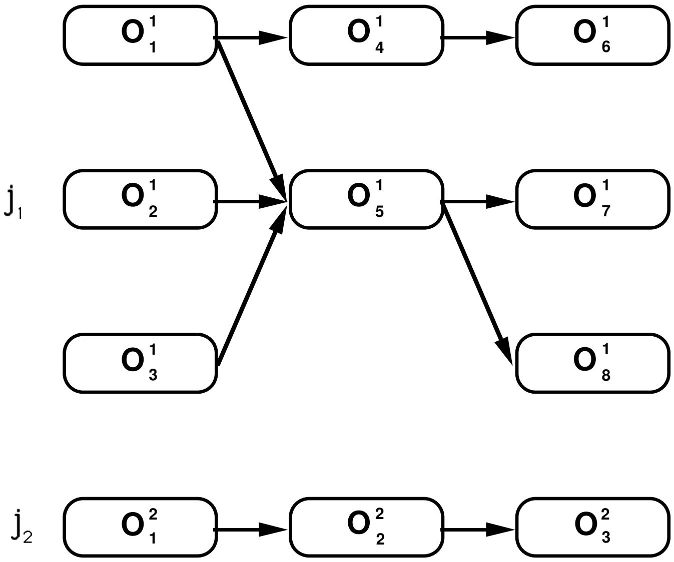


Figure 3-1: Examples of tree-like process routings.

In the job shop CSP studied in this chapter, each job j_l has a release date rd_l and a due-date dd_l between which all its operations have to be performed. Each operation O^l_i has a fixed duration du^l_i and a variable start time st^l_i. The domain of possible start times of each operation is initially constrained by the release and due dates of the job to which the operation belongs. If necessary, the model allows for additional unary constraints that further restrict the set of admissible start times of each operation, thereby defining one or several time windows within which an operation has to be carried out (e.g. a specific shift in factory scheduling). In order to be successfully executed, each operation O^l_i requires p^l_i

---

[14]This is by far the most common situation, especially in factory scheduling. Extensions of the techniques presented in this chapter to more general types of process routings will be briefly discussed as well.

different resources (e.g. a milling machine and a machinist) R^l_{ij} $(1 \le j \le p^l_i)$, for each of which there may be a pool of physical resources from which to choose, $\Omega^l_{ij}=\{r^l_{ij1},...,r^l_{ijq^l_{ij}}\}$, with $r^l_{ijk} \in RES$ $(1 \le k \le q^l_{ij})$ (e.g. several possible milling machines).

More formally, the problem can be defined as follows:

VARIABLES:

The variables of the problem are:

1. the **operation start times**, st^l_i, $(1 \le l \le n,\ 1 \le i \le n_l)$, and

2. the **resources**, R^l_{ij}, $(1 \le l \le n,\ 1 \le i \le n_l,\ 1 \le j \le p^l_i)$ selected for those resource requirements for which an operation has several alternatives.

CONSTRAINTS:

The non-unary constraints of the problem are of two types:

1. **Precedence constraints** defined by the process routings translate into linear inequalities of the type: $st^l_i + du^l_i \le st^l_j$ (i.e. O^l_i BEFORE O^l_j);

2. **Capacity constraints** that restrict the use of each resource to only one operation at a time translate into disjunctive constraints of the form: $(\forall p\, \forall q\ R^k_{ip} \ne R^l_{jq}) \vee st^k_i + du^k_i \le st^l_j \vee st^l_j + du^l_j \le st^k_i$. These constraints simply express that, unless they use different resources, two operations O^k_i and O^l_j cannot overlap[15].

Additionally, there are unary constraints restricting the set of possible values of individual variables. These constraints include non-relaxable due dates and release dates, between which all operations in a job need to be performed. The model actually allows any type of unary constraint that further restricts the set of possible start times of an operation. Time is assumed discrete, i.e. operation start times and end times can only take integer values. Finally, each resource requirement R^l_{ij} has to be selected from a set of resource alternatives, $\Omega^l_{ij} \subseteq RES$.

[15]These constraints have to be generalized when dealing with resources of capacity larger than one.

OBJECTIVE:

In the job shop CSP studied in this chapter, the objective is to come up with a feasible solution as fast as possible. Notice that this objective is different from simply minimizing the number of search states visited. It also accounts for the time spent by the system deciding which search state to explore next.

EXAMPLE:

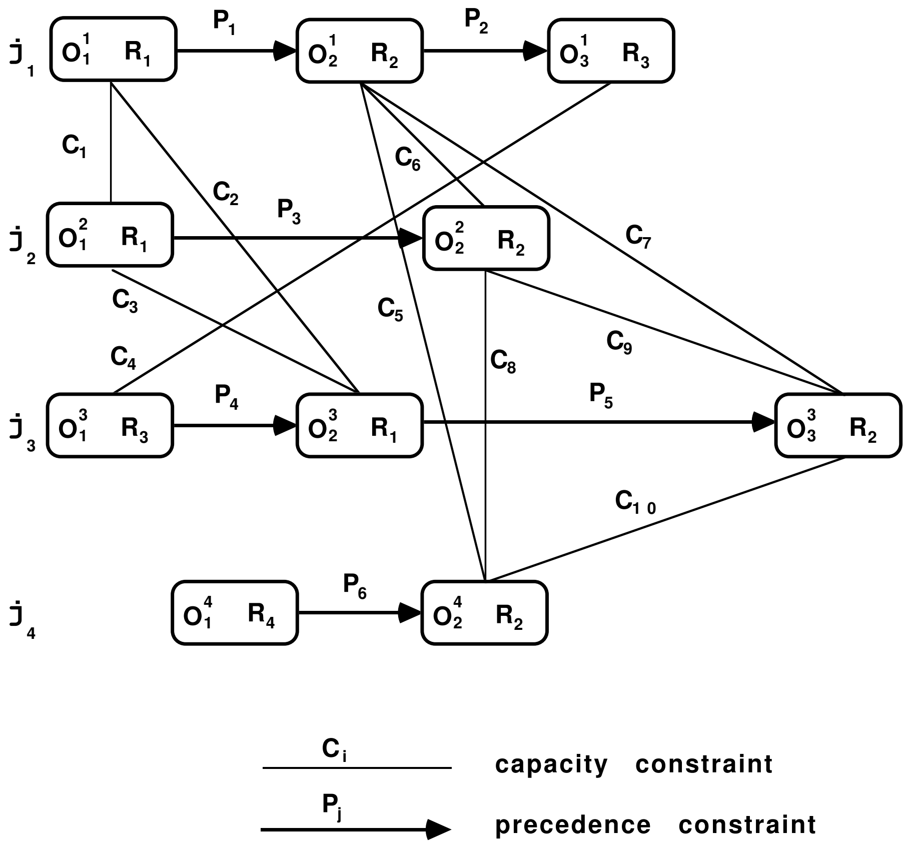


Figure 3-2: A simple job shop problem with 4 jobs. Each node is labeled by the operation that it represents and the resource required by this operation.

Figure 3-2 depicts a simple job shop scheduling problem with four jobs $J=\{j_1,j_2,j_3,j_4\}$ and four physical resources $RES=\{R_1,R_2,R_3,R_4\}$. In this simple example each operation has a single resource requirement with a single possible value. Operation start times are the only variables. For the sake of simplicity, it is assumed that all operations have the same duration, namely 3 time units, that all jobs are released at time 0 and have to be completed by time 15 (the minimum makespan of this problem). None of these simplifying assumptions is required by the techniques that will be discussed: jobs usually have different release and due dates, operations can have different durations, several resource requirements, and several alternatives for these requirements. However simple, this example will often turn out to be sufficient to highlight the shortcomings of some popular CSP heuristics. If necessary, the example will be slightly complicated in order to emphasize other shortcomings that would not be immediately visible otherwise.

Notice that, in this problem, resource R_2 is the only one to be required by four operations (one from each job). Since all operations in the example have the same duration, resource R_2 can be expected to constitute a small bottleneck. The next two sections attempt to see how generic CSP heuristics as well as more sophisticated heuristics designed for scheduling problems deal with such bottlenecks as well as with other constraint interactions common to job shop scheduling problems. The first section focuses on variable ordering heuristics, while the second is concerned with value ordering heuristics[16].

3.3. Shortcomings of Popular Variable Ordering Heuristics

A powerful way to reduce the average complexity of backtrack search consists in judiciously selecting the order in which variables are instantiated. The intuition is that, by instantiating difficult variables first, backtrack search will generally avoid building partial solutions that it will not be able to complete later on. This reduces the chances (i.e. the frequency) of backtracking. Instantiating difficult variables first can also help reduce the amount of backtracking when the system is in a deadend state that is not immediately detected by its consistency checking mechanism. Indeed, by instantiating difficult variables, the system moves to more constrained deadend states that are easier to detect. This reduces the time the system wastes attempting to complete partial solutions that cannot be completed.

---

[16]As pointed out in Chapter 2, in the micro-opportunistic approach, the term *variable ordering heuristic* refers to the selection of the next *operation* to be scheduled, and the term *value ordering heuristic* refers to the selection of a promising reservation for that operation (i.e. a start time and a resource for each resource requirement for which there are several possibilities). Since an operation generally involves several variables (i.e. a start time and a set of resource requirements), it might have been more appropriate to use the term *subproblem selection heuristic* or *aggregate variable ordering heuristic* instead of simply *variable ordering heuristic*. Similarly the term *value ordering heuristic* may not be the most appropriate. We keep this terminology because of its common use in the CSP literature.

Two types of variable ordering heuristics are usually distinguished:

1. **Fixed variable ordering heuristics**: A unique variable ordering is determined prior to starting the search and used in each branch of the search tree;

2. **Dynamic variable ordering heuristics**: The ordering is dynamically revised in each search state in order to account for earlier assignments. Different branches in the search tree generally entail different variable orderings.

Clearly fixed variable orderings require less computation since they are determined once and for all. On the other hand, dynamic variable ordering heuristics are potentially more powerful because of their ability to identify difficult variables within specific search states rather than for the overall search tree. Many CSP studies performed on simple problems such as N-queens or on moderate-size problems have found that dynamic variable ordering heuristics are too expensive (e.g. [Dechter 89a]). There are however more difficult problems, for which dynamic variable ordering heuristics can be expected to achieve exponential savings in the average amount of search required to come up with a solution [Purdom 83]. For these more difficult problems, it has often been suggested that a simple heuristic known as the **Dynamic Search Rearrangement** heuristic (DSR) would be sufficient [Bitner 75, Purdom 83, Dechter 89a, Ginsberg 90]. In each search state, DSR looks for the variable with the smallest number of remaining values, and selects this variable to be instantiated next. DSR has often been used as a benchmark to determine whether it is worthwhile using a dynamic variable ordering heuristic for a given class of problems. The experiments presented at the end of this chapter clearly show that job shop scheduling belongs to the class of more difficult problems for which a dynamic variable ordering is justified. Furthermore these experiments show that even DSR is often insufficient to solve realistic job shop CSPs.

The scheduling problem introduced in the previous section helps understand the shortcomings of DSR. Figure 3-3 depicts the same problem after applying the consistency enforcing techniques described in Chapter 2. According to DSR, there are six operations that are equally good candidates to be assigned a reservation first: O_1^1, O_2^1, O_3^1, O_1^3, O_2^3, and O_3^3. Indeed, these six operations all appear equally difficult to DSR, as they each have seven possible start times left. The other four operations in the problem

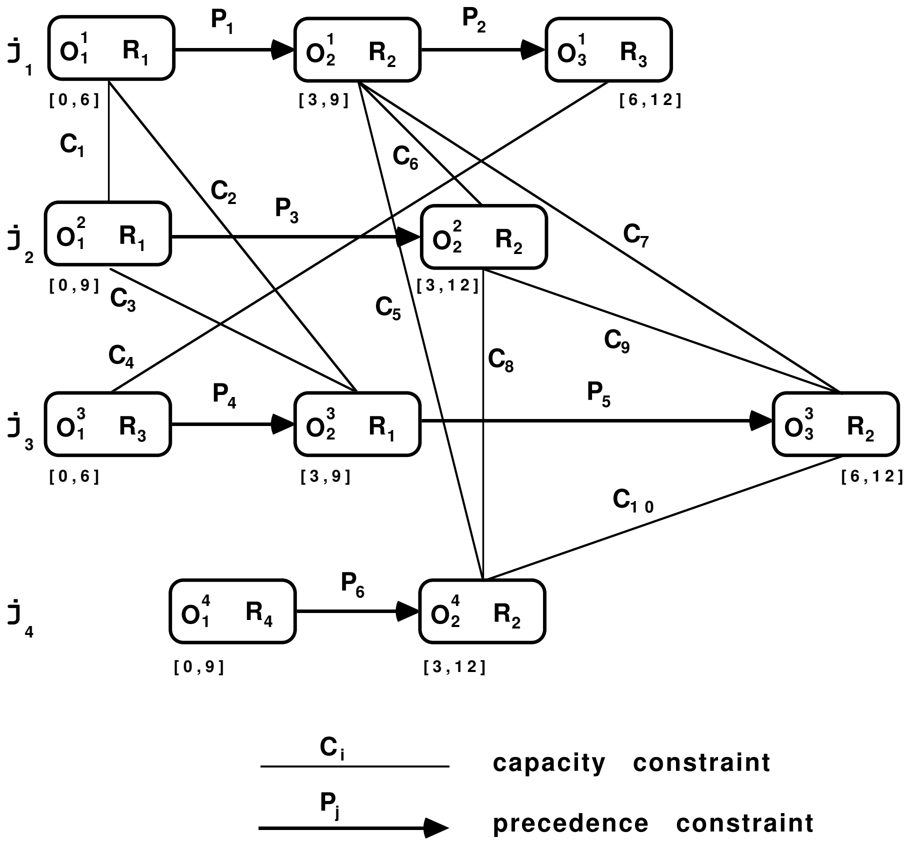


Figure 3-3: The same job shop CSP after consistency labeling. Start time labels are represented as intervals. For instance, [0,6] represents all start times between time 0 and time 6, as allowed by the time granularity, namely {0,1,2,3,4,5,6}.

appear easier as they each have ten possible start times left. It is easy to see that some of the six operations that appear equally difficult to DSR are in fact more difficult to schedule than others. Consider operations O_2^1 and O_3^3: both operations require resource R_2, which is required by a total of four operations. Moreover, in three cases out of four, the operation requiring resource R_2 is the last operation in its job. This high contention for resource R_2 indicates that O_2^1 and O_3^3 will probably be difficult to schedule. On the other hand, an operation like O_1^3 competes only with one other operation for resource R_3, namely operation O_3^1. Moreover, the fact that O_1^3 is the first operation in job j_1, while O_3^1 is the last operation in job j_3, suggests that these two operations are not very likely to compete with each other. Operations O_3^1 and O_1^3 can be expected to be easier to schedule than operations O_2^1 and O_3^3. Unfortunately DSR is not able to account for these observations. This is because DSR simply counts the number of remaining values of each variable, but fails to estimate the likelihood that these values remain available later on. Clearly start times of operations competing for highly contended resources are more likely to become unavailable than those of other operations.

In this example, the bottleneck resource R_2 also corresponds to the largest clique of capacity constraints. Therefore, a variable ordering heuristic that identifies difficult variables (i.e. nodes in the constraint graph) as those with many incident constraints might actually provide better advice than DSR. Several such variable ordering heuristics have been proposed in the literature. These heuristics are generally fixed variable ordering heuristics, unless new constraints are added to the problem as it is solved. One such heuristic is the **Minimum Width** (MW) heuristic [Freuder 82, Dechter 89a]. MW "*orders the variables from last to first by selecting, at each stage, a node in the constraint graph which has a minimal degree*[17] *in the graph remaining after deleting from the graph all nodes which have been selected already*" [Dechter 89a]. A variation of this heuristic known as the **Minimum Degree** (MD) heuristic simply ranks variables according to their degree in the initial constraint graph [Dechter 89a]. In the example depicted in Figure 3-2, MD would select O_2^1 to be scheduled first. There are also MW orderings starting with that operation. In general, scheduling problems are not that simple, and fixed variable ordering heuristics like MD or MW do not provide very good advice either. This is best illustrated by slightly modifying the scheduling problem depicted in Figure 3-2.

Suppose, for instance, that we change the problem and introduce a fifth resource, say R_5. Suppose also that we allow any of the operations requiring R_1 or R_3 in the original problem to use R_5 as an alternative resource. Now we have:

- $\Omega_{11}^1 = \Omega_{11}^2 = \Omega_{21}^3 = \{R_1, R_5\}$

- $\Omega_{31}^1 = \Omega_{11}^3 = \{R_3, R_5\}$

In this case, the two cliques of capacity constraints corresponding to R_1 and R_3 are subsumed by a larger clique of capacity constraints[18] involving five operations: O_1^1, O_3^1, O_1^2, O_1^3, and O_2^3 (Figure 3-4). Due to the additional capacity constraints resulting from the

[17] The degree of a node is the number of constraints incident to that node.

[18] Capacity constraints between operations belonging to the same job have been subsumed by precedence constraints in that job.

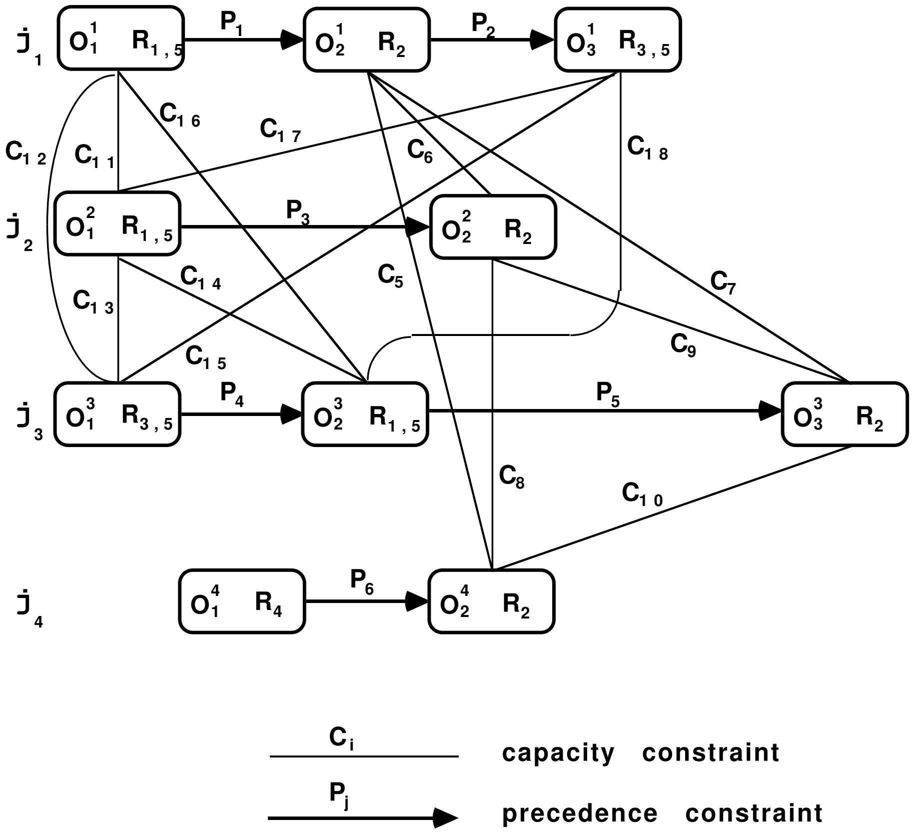


Figure 3-4: A new resource R_5 is added to the problem.
$R_{1,5}$ stands for R_1 or R_5. $R_{3,5}$ stands for R_3 or R_5.

introduction of R_5, there are now MW orderings and MC orderings starting with some of these five operations, while in fact the addition of R_5 has made these five operations even easier to schedule. This is because MW and MC do not account for **constraint tightness**, namely the difficulty of satisfying a specific constraint [Nadel 86c, Fox 89]: the addition of R_5 has significantly loosened the capacity constraints participating in the new clique. Hence operations connected by these constraints have become easier to schedule[19].

Nadel has proposed more sophisticated variable ordering heuristics based on probabilistic estimates of the expected number of nodes explored or based on estimates of the number of consistency checks performed by the system [Nudel 83, Nadel 86c]. His

[19]Another example of a variable ordering heuristic that does not account for constraint tightness is the **Max-cardinality search order** which arbitrarily selects the first variable to be instantiated, and then at each stage picks the variable connected to the largest number of already instantiated variables [McGregor 79, Dechter 89a]. This heuristic can also be seen as a fixed variation of DSR.

measures of variable criticality are very expensive to evaluate in their general form, and rely solely on measures of constraint satisfiabilities. On the other hand, resource contention is the result of a very specific type of interactions between many variables (essentially interactions between many inequality constraints), which cannot be solely captured by measures constraint satisfiabilities.

Finally, variable ordering heuristics described in the CSP literature treat all problem constraints uniformly. In job shop scheduling problems, there are however two very different types of constraints: capacity constraints and precedence constraints. As pointed out in Chapter 2, the consistency enforcing techniques implemented in the micro-opportunistic approach take advantage of particular algebraic properties of precedence constraints to efficiently ensure that backtracking only occurs as a result of the violation of capacity constraints. This particular feature of the consistency enforcing mechanism can in turn be exploited to design a more effective variable ordering heuristic. Indeed, because of the consistency enforcement mechanism implemented in the micro-opportunistic search procedure, the criticality of an operation is uniquely a function of the difficulty of finding a reservation for that operation that will not violate some capacity constraints.

A specialized variable ordering heuristic that takes advantage of this observation is that of Keng and Yun [Keng 89], though its authors apparently failed to relate the strength of their heuristic to this observation. Keng and Yun suggested a generalization of DSR in which each operation reservation (i.e. each value) is assigned a survivability measure reflecting its chance of satisfying the capacity constraints (i.e. its chance of surviving the competition with other operations for the possession of a resource). The operation to be scheduled next is the one with the smallest global survivability, as determined by the sum of the survivabilities of each of its (remaining) possible reservations. Experiments presented at the end of this chapter, show that this heuristic performs better than all the generic heuristics described above. They also show that this heuristic is quite expensive, as it requires inspecting all the remaining reservations (i.e. values) of all unscheduled operations[20]. In scheduling problems with several hundred operations, each with several

[20]Notice also that this heuristic may still identify operations with just a few remaining possible reservations as being critical while in fact these reservations may not be conflicting with the reservations of any other operation. This could be the case if, for instance, operation O_1^4 in the example in Figure 3-2 had only a small number of possible start times. In fact, the consistency enforcing technique ensures that backtracking will never be caused by this operation, since there is no capacity constraint incident on it.

hundred possible start times and several possible resources, this heuristic may not be cost effective. More efficient heuristics can be obtained by focusing on one or a small number of cliques of tight capacity constraints. A good variable ordering heuristic consists in selecting the operation that is the most likely to violate a constraint in these cliques. A heuristic based on this idea is described in Section 5, which runs faster than Keng and Yun's heuristic while achieving an even higher search efficiency.

3.4. Shortcomings of Popular Value Ordering Heuristics

Another powerful way to reduce the average complexity of backtrack search relies on judiciously selecting the order in which value assignments are tried for a variable. A good value ordering heuristic to minimize backtracking consists in assigning so-called *least constraining values*. A least constraining value is one that is expected to participate in many solutions to the problem (or to the subproblem defined by the current search state). By first trying least constraining values, the system will generally maximize the number of values left to variables that still need to be instantiated, and hence it it will avoid building partial solutions that cannot be completed.

Attempting to exactly compute the number of global solutions in which a given assignment (i.e. value) participates would be futile as it would require finding all solutions to the problem. Instead Dechter and Pearl developed a heuristic, called **ABT**[21], that relies on tree-like relaxations of the problem to approximate the goodness of a value. A tree-like relaxation of a CSP is one whose constraint graph is a tree that spans some or all the nodes (i.e. variables) of the original CSP. Within such relaxations, the number of solutions in which a value participates can be efficiently computed in $O(nk^2)$ steps, where n is the number of variables in the CSP, and k the maximum number of possible values of a variable. The intuition is that, if one can find a tree-like relaxation that is close enough to the original CSP, a good value for the relaxation should also be a good value for the original CSP. One way to obtain tight tree-like relaxations is by associating with each (binary) constraint C in the original constraint graph a weight $w(C)$ equal to the satisfiability of that constraint (i.e. the number of pairs of values that satisfy the constraint). A tight tree-like relaxation corresponds to a Minimum Spanning Tree (MST) in the resulting network (see also [Chow 68]).

---

[21] ABT stands for Advised Backtracking.

While ABT has performed particularly well on some CSPs, it does not appear appropriate for CSPs such as job shop scheduling. Indeed, even for a fixed variable ordering, the heuristic generally requires the computation of a fixed MST for each of the n levels in the search tree. This amounts to n MST computations, each of which typically requires $O(n^2)$ elementary computations [Tarjan 83b] (hence a total of $O(n^3)$ elementary computations). The experimental results presented at the end of this chapter indicate that a fixed variable ordering is generally not enough to efficiently solve job shop scheduling problems. Under these conditions, it might even be necessary to identify new tree-like relaxations in each search state[22]. These computations may become quite expensive for large CSPs. There is however a more important problem with this heuristic, whether using minimum spanning tree relaxations or not: **there is no guarantee that there even exists a tight enough tree-like relaxation of the CSP**, namely a tree-like relaxation that will provide sufficiently good advice to guide search[23]. This is most likely to be the case with job shop scheduling problems, as explained below with an example.

[22]This would also require updating the weights of each constraint in each search state.

[23]The experiments reported in [Dechter 88] seemed to indicate the opposite. In these experiments, it appeared that often the advice provided by ABT was too expensive and too accurate. Instead advice provided by looser relaxations ended up being more cost-effective. However, these results involved fairly easy problems with 15 variables, 5 values per variable, and a relatively high density of solutions.

Consider constraint P_1 in the scheduling problem depicted in Figure 3-3. P_1 is a precedence constraint between operation O_1^1 and operation O_2^1. The set of start time pairs (st_1^1, st_2^1) that satisfy constraint P_1 is:

$$\{(0,3),(0,4),...,(0,9),(1,4),(1,5),...,(1,9),...,(6,9)\}$$

In order to identify a tight tree-like relaxation, P_1 is assigned a weight, $w(P_1)$, equal to the cardinality of that set, namely $w(P_1) = 7+6+5+4+3+2+1 = 28$. Similar computations can be performed to compute the weights of other constraints. These weights are as follows:

- $w(P_1)=w(P_2)=w(P_4)=w(P_5)=28$
- $w(P_3)=w(P_6)=55$
- $w(C_1)=38$, $w(C_2)=29$, $w(C_3)=38$
- $w(C_4)=43$
- $w(C_5)=w(C_6)=38$, $w(C_7)=29$, $w(C_8)=56$, $w(C_9)=w(C_{10})=38$

Figure 3-5 shows an MST relaxation of the scheduling problem obtained using these weights. It appears that the MST relaxation includes 10 out of the 16 constraints present in the original CSP. The loss of information initially contained in the cliques of capacity constraints is even more dramatic. Only 2 out of the 6 constraints in the clique corresponding to R_2 have been preserved. This is not an accident. **In general a resource required by *M* operations will result in a clique of $\binom{M}{2}$ capacity constraints. At most *M-1* of these capacity constraints can be preserved in any tree-like relaxation of the problem**. Under these conditions, we should not be surprised if the advice provided by ABT for job shop CSPs is not very effective. Suppose for instance that the system selects O_2^1 to be instantiated first[24]. Using the MST relaxation represented in Figure 3-5, ABT would recommend assigning start time 4 to this operation. A careful examination of the scheduling problem reveals however that there is no feasible schedule with O_2^1 starting at 4. Indeed it appears that if O_2^1 were to start at time 4, the three other operations requiring

[24]It should now be clear that this is a good choice, since this operation has only seven possible start times and requires resource R_2, the main bottleneck of the problem.

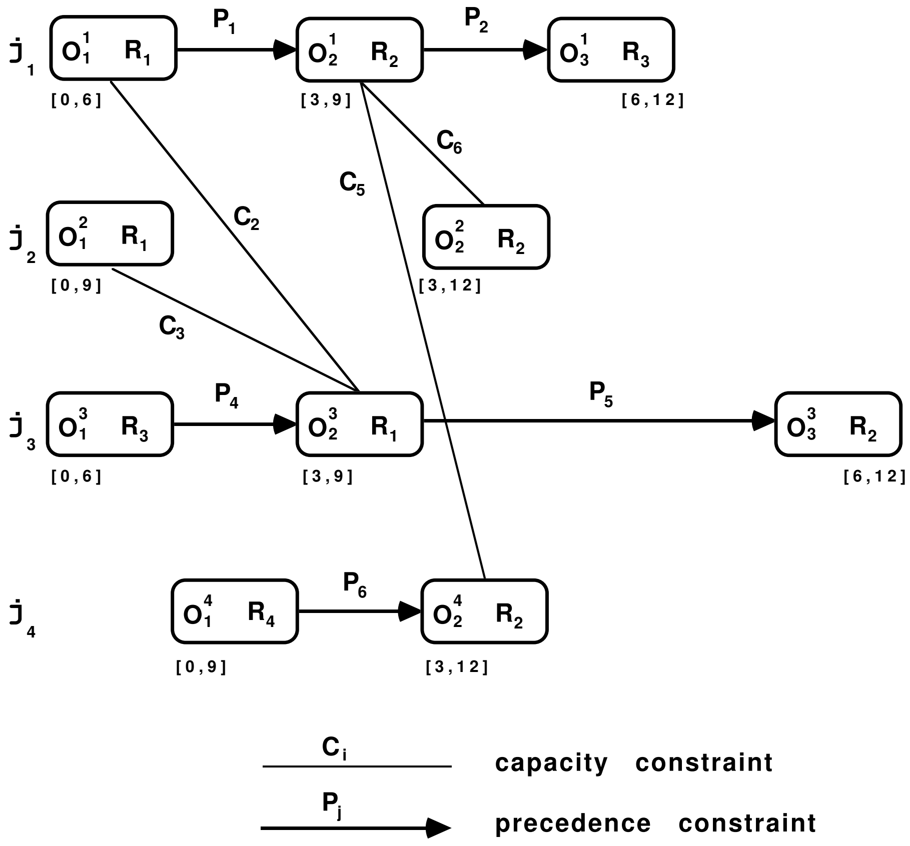


Figure 3-5: An MST relaxation of the scheduling problem.

resource R_2 would all have to be scheduled between time 7 and time 15. In other words, there would be 8 time units left to fit 3 operations that each have a duration of 3 time units. This is clearly impossible.

Keng and Yun have developed a specialized value ordering heuristic that can deal more effectively with cliques of capacity constraints [Keng 89]. This heuristic first estimates the overall need for each resource in function of time. Based on these estimates, operation reservations are ranked according to how well they are expected to prevent contention with the resource requirements of other operations. As the results reported later in this chapter indicate, Keng and Yun's value ordering heuristic generally outperforms ABT. However their heuristic omits to leave enough room to other operations within the same job so that they can be assigned least constraining reservations as well. In other words, Keng and Yun's heuristic accounts only for the capacity constraints incident at the current operation, but fails to account for capacity constraints at other operations connected by precedence constraints to the current operation.

The next section describes a probabilistic model of the search space that better accounts for the high connectivity of constraint graphs typically found in job shop scheduling, and for the constraint interactions induced by these graphs. New variable and value ordering heuristics are defined within this framework that attempt to remedy the shortcomings identified above.

3.5. New Variable and Value Ordering Heuristics

3.5.1. Underlying Assumptions

Rigorously speaking, good variable and value ordering heuristics are heuristics that minimize the time required for search to complete (i.e. either with a solution, if one exists, or with the answer that the problem is overconstrained). If the problem is infeasible, search time is independent of the value ordering heuristic (except for the time spent by the system ordering values according to the heuristic): once a variable has been selected, the system will have to try each one of its remaining values before being able to conclude that the current partial solution cannot be completed. In general variable and value ordering heuristics affect the number of search states that are explored, the average amount of time spent enforcing consistency in each search state, and the amount of time spent by the system ordering variables and values according to these heuristics. Variable and value ordering heuristics may even affect each other’s performance. The complexity of these interactions precludes the design of heuristics that directly minimize the expected search time. One can however attempt to design heuristics that aim at reducing some of the factors identified above. The approach taken in this section aims at developing heuristics that *efficiently reduce the expected number of search states explored by the system*. Because in the micro-opportunistic approach the time spent by the system enforcing consistency is mainly a function of the number of operations that have already been scheduled (i.e. the depth in the search tree) rather than a function of the specific operations that have been scheduled, such heuristics are expected to effectively reduce search time as well.

In this section, it is postulated that **a critical variable is one that is expected to cause backtracking**, namely one whose remaining possible values are expected to conflict with the remaining possible values of other variables. Under a set of simplifying independence assumptions, Haralick and Elliott have shown that such a measure of criticality will minimize the expected length of branches in the search tree, and hence the total number of search states that need to be visited to come up with a solution [Haralick

80][25]. It is also postulated that a **good value is one that is expected to participate in many solutions**.

In the next subsection, a probabilistic model of the search space is introduced that can be used to approximate variable criticality and value goodness.

3.5.2. A Probabilistic Model of the Search Space

In the micro-opportunistic approach, critical operations are those that are likely to violate capacity constraints. These operations can be identified as operations that heavily rely on the availability of highly contended resource/time intervals. The consistency checking mechanism embedded in the micro-opportunistic search procedure ensures that the only reservations left to unscheduled operations are reservations that will not directly conflict with reservations already allocated to other operations. Constraint violations that will force the scheduler to backtrack are violations that will occur between operations that have not yet been scheduled. **Contention between (unscheduled) operations for a resource over some time interval is determined by the number of (unscheduled) operations competing for that resource/time interval and the reliance of each one of these operations on the availability of this resource/time interval**. Typically, operations with few possible reservations left will heavily rely on the availability of any one of these reservations, whereas operations with a handful of remaining reservations will rely much less on any one of these reservations.

In order to measure resource contention in a given search state, a probabilistic framework is assumed, in which each reservation ρ that remains possible for an unscheduled operation O_i^l is assigned a *subjective probability* $\sigma_i^l(\rho)$ to be allocated to that operation. Because a priori there is no reason to believe that one reservation is more likely to be selected than another, each operation reservation is assigned an equal probability to be selected. Clearly, in any schedule, each operation will be assigned only one reservation. Hence, the reservation distributions are chosen to be of the form:

---

[25]See [Haralick 80] pp. 307-312. At the end of their proof, the authors make the unnecessary assumption that each variable value is equally likely to become unavailable. Under this assumption, the variable with what they call the *smallest success probability* (or equivalently the variable most likely to create backtracking) is the one with the smallest number of remaining values. The authors exploit this result to motivate their use of the Dynamic Search Rearrangement heuristic. When this last assumption is omitted, Haralick and Elliott's proof shows that (under several other simplifying assumptions made earlier in their proof) choosing the variable most likely to create backtracking will minimize the expected length of each branch in the search tree.

$$\sigma_i^l(\rho) = \frac{1}{NBR_i^l}$$

where NBR_i^l is the number of remaining reservations of O_i^l in the current search state. This distribution mirrors our intuition that an operation with many possible reservations does not heavily rely on any single one of its remaining reservations, and hence the probability of any single one of these reservations to be selected is rather low. On the other hand, operations with few remaining possible reservations are more likely to have to use any one of these reservations. Using these subjective reservation distributions, it is possible to estimate the *reliance* of an operation O_i^l on the availability of a resource $R_k \in RES$ at time τ as the probability that the reservation allocated to this operation will require that resource at that time. This probability will be referred to as the *individual demand* of operation O_i^l for resource R_k at time τ. It will be denoted $D_i^l(R_k, \tau)$. This probability can be computed as the sum of the probabilities $\sigma_i^l(\rho)$ of all remaining reservations ρ of operation O_i^l that require resource R_k at time τ. Finally, by adding the individual demands of all unscheduled operations requiring resource R_k, an *aggregate demand profile*, $D_{R_k}^{aggr}(\tau)$, is obtained that indicates contention between unscheduled operations for resource R_k as a function of time. Alternatively, one can postulate a stochastic mechanism that completes the current partial schedule (in the current search state) by randomly assigning a reservation to each unscheduled operation O_i^l according to its σ_i^l distribution. $D_i^l(R_k, \tau)$ is the probability that the stochastic mechanism assigns O_i^l a reservation that requires R_k at time τ, and $D_{R_k}^{aggr}(\tau)$ is the expected number of reservations made by the stochastic mechanism for R_k at time τ (or the expected number of operations requiring that resource at that time).

Similar demand profiles are built by Keng and Yun's variable and value ordering heuristics [Keng 89]. The heuristics that will be presented differ from those of Keng and Yun in the way they exploit these demand profiles[26]. Earlier Muscettola and Smith also proposed techniques to build probabilistic demand profiles, based on a *predefined variable ordering* [Muscettola 87].

The following illustrates the construction of these profiles for the example introduced in Figure 3-2.

---

[26]The work presented here was performed concurrently to that of Keng and Yun [Sadeh 88, Sadeh 89a, Fox 89]. Notice that the interpretation given by Keng and Yun for their demand profiles is not a probabilistic one.

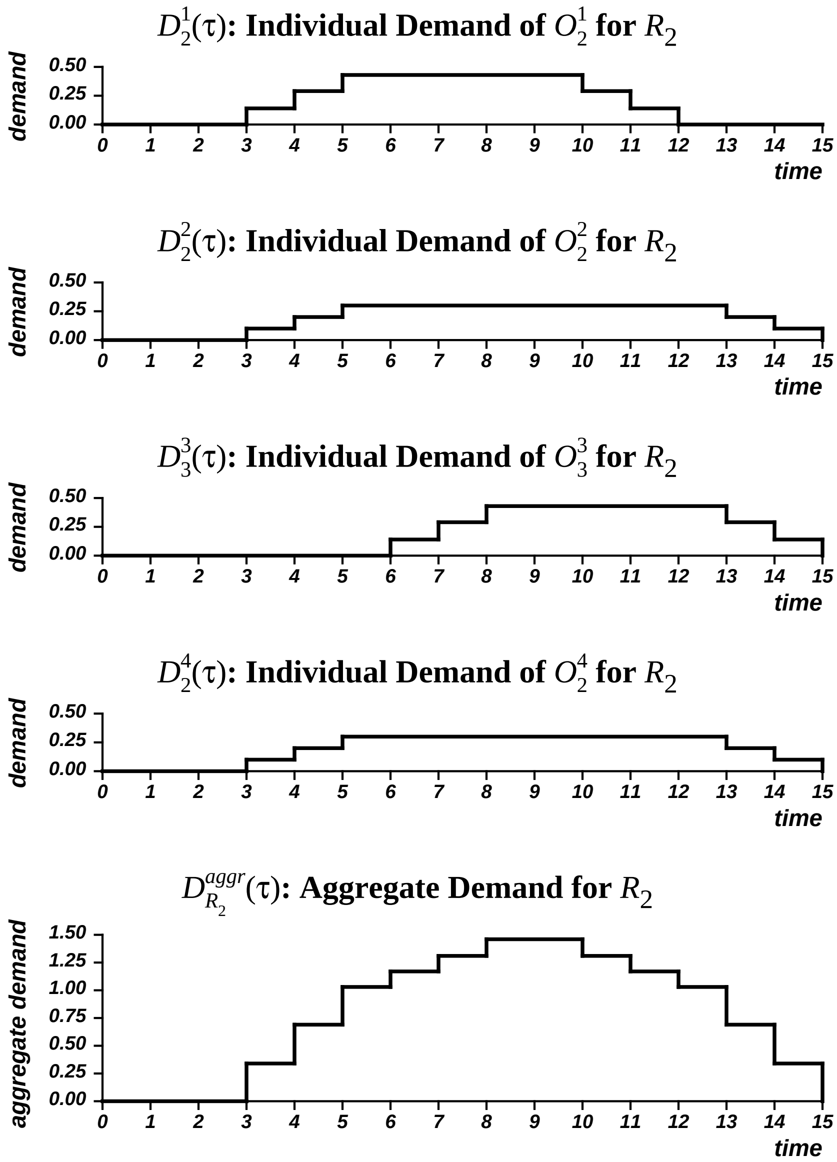


Figure 3-6: Building R_2's aggregate demand profile in the initial search state.

Consider operation O_2^1 in the initial search state depicted in Figure 3-3. After enforcing consistency, this operation has 7 possible reservations (i.e. start times $st_2^1 = 3, 4, \ldots, 9$), each with a subjective probability $\sigma_2^1(st_2^1) = \frac{1}{7}$ to be selected. On the other hand, O_2^2 has 10 possible start times: $st_2^2 = 3, 4, \ldots, 12$. Therefore the subjective probability that any one of these 10 possible start times will be selected is $\sigma_2^2(st_2^2) = \frac{1}{10}$. The individual demand of an operation O_i^l for resource R_2 at some time t can be computed by simply adding the probability of each reservation that would require using resource R_2 at time τ, i.e. by adding the probabilities of all reservations starting between t and $t - du_i^l$. For instance:

$$D_2^1(R_2, t) = \sum_{t - du_2^1 < \tau \le t} \sigma_2^1(\tau)$$

In particular $D_2^1(R_2, t) = \frac{1}{7}$ for all times t such that $3 \le t < 4$. This is because there is only one possible start that would cause operation O_2^1 to use resource R_2 at any of these times, namely $st_2^1 = 3$. Between time 4 and time 5, things are different as there are two possible start times that would cause O_2^1 to use R_2 over that time interval:$st_2^1 = 3$ and $st_2^1 = 4$. Consequently the demand of O_2^1 for resource R_2 between time 4 and time 5 is $\frac{2}{7}$. Similar computations can be performed for the other time intervals over which O_2^1 may require resource R_2. Figure 3-6 shows the individual demands of all four operations requiring resource R_2, as well as the aggregate demand for that resource obtained by adding the four individual demands over time. As expected, the two operations with only seven possible start times (namely O_2^1 and O_3^3) have more compact individual demands than the two operations with ten possible start times (namely O_2^2 and O_4^2). Notice also, that, because of the normalization of the $\sigma_i^l(\rho)$ distributions, the total individual demand of an operation with only one possible resource (like all the operations in this example) is always equal to the duration of that operation. This total demand is simply spread differently over time, depending on the number of start times still available to the operation.

Figure 3-7 displays the aggregate demands for the four resources of the example. As anticipated, resource R_2 appears to be the resource that is the most contended for.

In general, building these demand profiles will require to look at each remaining reservation of each unscheduled operation. Hence the worst-case complexity of the procedure is $O(Nk)$ in each search state, where N is the number of unscheduled operations and k the number of remaining reservations of an unscheduled operation. In practice, the sets of remaining reservations of many operations do not change from one search state to another and it is more efficient to only update the individual demands of those operations whose sets have shrunk. The old individual demands of operations whose sets of possible reservations have shrunk are subtracted from the aggregate

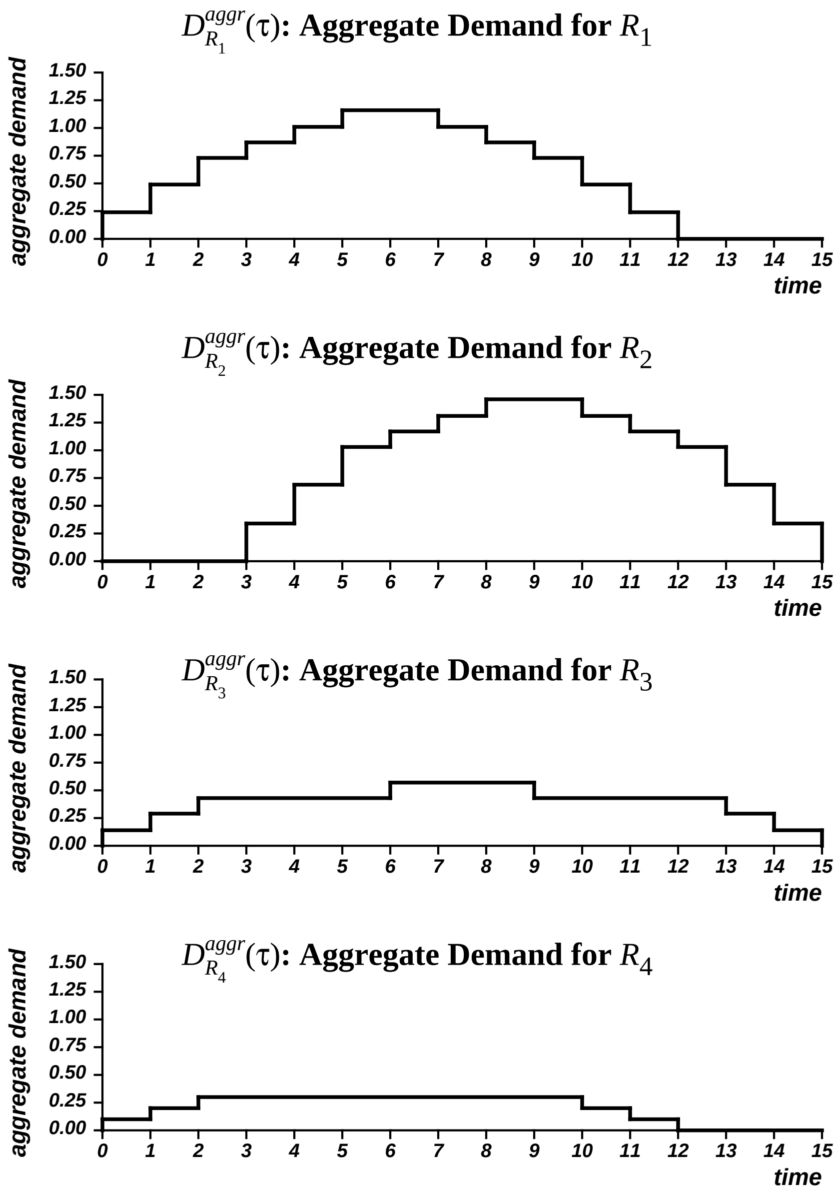


Figure 3-7: Aggregate demands in the initial search state for each of the four resources.

demand profiles, and the new individual demands are added instead. It is possible to perform similar updates when the system backtracks.

3.5.3. A Variable Ordering Heuristic Based on Measures of Resource Contention

The aggregate demand for a resource over a time interval is a measure of contention between unscheduled operations for that resource/time interval. In general, the resource/time interval with the highest demand (i.e. the one that is the most contended for) can be expected to be the one where capacity constraints are most likely to be violated[27]. As explained earlier, the individual contribution of an operation to the demand for a resource/time interval (i.e. its individual demand for that resource/time interval) is also a measure of the reliance of that operation on the availability of that resource/time interval. Accordingly, **the operation with the highest contribution to the demand for the the most contended resource/time interval is considered the most likely to violate a capacity constraint**, since it is the one that relies most on the availability of that highly contended resource/time interval.

Several variations of this variable ordering heuristic have been implemented. The simplest and often most effective one inspects each resource’s aggregate demand profile using time intervals of duration equal to the average duration of the operations requiring that resource. The heuristic then picks the operation with the largest contribution (i.e. the largest individual demand) to the demand for the most contended of these time intervals. This is the variable ordering heuristic used in the experiments reported at the end of this chapter. It will be referred to as **ORR**, which stands for "Operation Resource Reliance" heuristic.

Figure 3-7 displays the demand profiles for R_1, R_2, R_3, and R_4, the four resources of the problem introduced in Section 3.2. The largest demand peak identified by ORR is that for resource R_2 between time 8 and 11, which corresponds precisely to the clique of tight capacity constraints identified earlier. Figure 3-8 indicates that the operation with the largest contribution to that demand is O_3^3. This is not a coincidence, this operation competes for the most contended resource and belongs to the group of six operations that have only seven possible start times left after consistency checking. Notice that, in this example, there are actually two intervals in the demand profile of R_2 that qualify as most contended for: [7,10[and [8,11[. Had the scheduler chosen [7,10[instead of [8,11[, it

[27]These tight capacity constraints are those connecting operations that contribute to the demand for the highly contended resource/time interval.

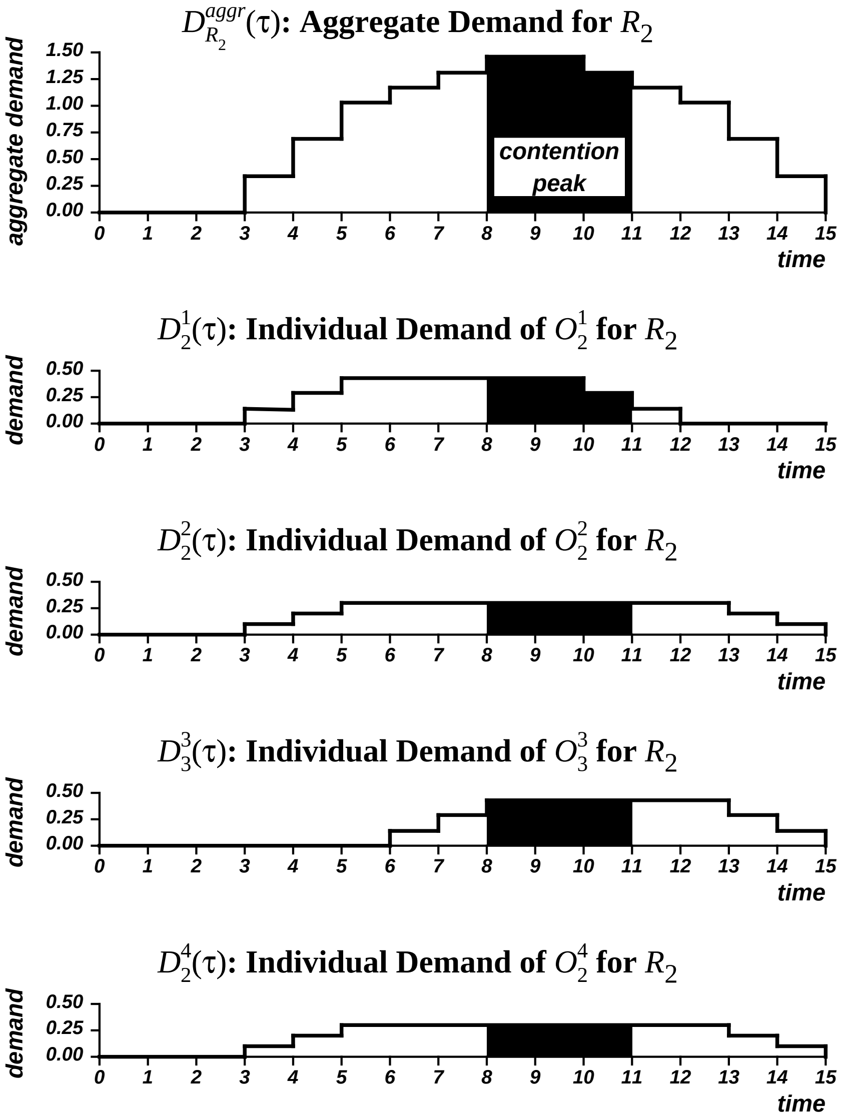


Figure 3-8: ORR Heuristic: the most critical operation is the one that relies most on the most contended resource/time interval.

would have selected O^1_2 as the operation to be scheduled next. In fact O^3_3 and O^2_2 appear equally critical in this example.

This heuristic requires to successively look at each resource, and each time interval in that resource's calendar, in order to identify the one that is the most contended for. If there are m resources and if the scheduling horizon is H, this requires $O(Hm)$ elementary computations.

3.5.4. A Value Ordering Heuristic Avoiding Resource Contention

In Section 3.4, ABT was found to have two major weaknesses when applied to job shop scheduling. The first weakness had to do with the computational overhead involved in the determination of tight tree-like relaxations of the problem, while the second one lay in the inability of tree-like relaxations to properly account for cliques of capacity constraints. In contrast, a value ordering heuristic is now described that attempts to avoid resource contention while relying on predetermined tree-like relaxations. The tree-like relaxations are comprised of some or all the precedence constraints of the process routing to which the current critical operation belongs (i.e. the operation selected to be scheduled next by the variable ordering heuristic). Rather than simply counting the number of solutions to the relaxation in which a given reservation assignment participates, **the value ordering heuristic makes up for the lack of information within this predefined relaxation by accounting for the probability that a solution to the relaxation also satisfies the cliques of capacity constraints**. The probability that a solution satisfies the cliques of capacity constraints (i.e. survives resource contention) is estimated using the same demand profiles that were constructed for the variable ordering heuristic.

For job shop CSPs with tree-like process routings, the tree-like relaxation adopted by the heuristic is comprised of all the unscheduled operations connected by precedence constraints to the current critical operation, along with these precedence constraints. Each candidate reservation assignment (for the critical operation) is ranked according to the number of solutions to the relaxation with which it is compatible (i.e. the number of compatible schedules for the job to which the critical operation belongs) that are expected to survive resource contention. The reservation compatible with the largest number of such schedules is the one selected by the heuristic. The following describes the approximations used by the system to compute the probability that a reservation survives resource contention and the probability that a job schedule survives resource contention. A dynamic programming technique to efficiently count the number of schedules compatible with a given reservation and expected to survive contention is presented in Appendix A. This technique is an adaptation of a procedure developed by Dechter and Pearl for the ABT heuristic [Dechter 88] (see also [Pearl 88]). It is shown that this technique can be further speeded up by taking advantage of the linearity of precedence constraints.

3.5.4.1. Estimating the Probability that a Reservation Survives Contention

Let O_i^l be an unscheduled operation and $\rho = <st_i^l = t, R_{i1}^l = r_{i1k_1}^l, R_{i2}^l = r_{i2k_2}^l, \ldots>$ one of its remaining reservations. The probability that assigning reservation ρ to operation O_i^l will not conflict with the resource requirements of other operations will be referred to as the **survivability** of reservation ρ (for O_i^l). It will be denoted $surv_i^l(\rho)$. The survivability of assigning reservation ρ to O_i^l will be approximated by the product of the probability that each one of the resources required by that reservation will be available between t and $t + du_i^l$:

$$surv_i^l(\rho) = \prod_{r_{ijk}^l \in \{r_{i1k_1}^l, r_{i2k_2}^l, \ldots\}} avail_i^l(r_{ijk}^l, t, t + du_i^l) \tag{3.1}$$

where $avail_i^l(r_{ijk}^l, t, t + du_i^l)$ stands for the probability that resource r_{ijk}^l will not be required by any other operation between t and $t + du_i^l$ (also the probability that assigning this resource to O_i^l will not create backtracking).

Let $r_{ijk}^l = R_p \in RES$. The probability $avail_i^l(r_{ijk}^l, t, t + du_i^l)$ that resource $r_{ijk}^l = R_p$ will not be required by any other operation between t and $t + du_i^l$ can be approximated using the aggregate demand profile of resource R_p (which is already maintained by the system for the ORR variable ordering heuristic) and a vector $n_p(\tau)$, which is also maintained by the system to keep track of the number of (unscheduled) operations competing for R_p as a function of time[28]. At any time $t \le \tau < t + du_i^l$ there are by definition $n_p(\tau) - 1$ unscheduled operations competing with operation O_i^l for resource R_p. The total demand of these other unscheduled operations for R_p at time τ is $D_{R_p}^{aggr}(\tau) - D_i^l(R_p, \tau)$. Assuming that each of these $n_p(\tau)$–1 other operations equally contributes to this demand, the probability that none of these operations requires R_p at time τ is given by:

$$\left(1 - \frac{D_{R_p}^{aggr}(\tau) - D_i^l(R_p, \tau)}{n_p(\tau) - 1}\right)^{n_p(\tau) - 1} \tag{3.2}$$

It is tempting to approximate $avail_i^l(r_{ijk}^l, t, t + du_i^l)$, i.e. the probability that $r_{ijk}^l = R_p$ will be

[28] $D_{R_p}^{aggr}(\tau)$ is the aggregate demand for R_p at time τ whereas $n_p(\tau)$ is the number of operations contributing to that demand.

available to O_i^l between t and $t+du_i^l$, as the product of the probabilities that this resource will be available to O_i^l on each one of the du_i^l time intervals between t and $t+du_i^l$. In general, this approximation is too pessimistic. It assumes that the operations competing with O_i^l have a duration equal to 1, i.e. that any of these operations could require $r_{ijk}^l = R_p$ over time interval $[\tau, \tau+1[$ without requiring it over time interval $[\tau+1, \tau+2[$ or over time interval $[\tau-1, \tau[$. Instead, because operations competing for $r_{ijk}^l = R_p$ generally require several contiguous time intervals, a better approximation consists in subdividing the calendar of that resource into buckets of duration $AVG(du)$, where $AVG(du)$ is the average duration of the operations competing for $r_{ijk}^l = R_p$. $avail_i^l(r_{ijk}^l, t, t+du_i^l)$ is then approximated as the probability that O_i^l will be able to secure the $\frac{du_i^l}{AVG(du)}$ time buckets that it requires to fit on the resource's calendar. Using Equation (3.2), this can be approximated as:

$$avail_i^l(R_p, t, t+du_i^l) = \left(1 - \frac{AVG(D_{R_p}^{aggr}(\tau) - D_i^l(R_p, \tau))}{AVG(n_p(\tau) - 1)}\right)^{AVG(n_p(\tau)-1) \times \frac{du_i^l}{AVG(du)}} \qquad (3.3)$$

where $AVG(D_{R_p}^{aggr}(\tau) - D_i^l(R_p, \tau))$ and $AVG(n_p(\tau) - 1)$ are respectively the average of $D_{R_p}^{aggr}(\tau) - D_i^l(R_p, \tau)$ and the average of $n_p(\tau) - 1$ over time interval $[t, t+du_i^l[$.

Figure 3-9 depicts reservation survivabilities for the three operations in job j_3, the job to which belongs O_3^3, the operation selected to be scheduled in the initial search state. The shape of these survivability curves is easily interpreted by looking at Figures 3-3 and 3-7. Consider operation O_1^3. Figure 3-3 indicates that O_1^3 only competes with one other operation for resource R_3, namely operation O_3^1. Because operation O_1^3 has a duration $du_1^3 = 3$ and because the earliest possible start time of operation O_3^1 is $st_3^1 = 6$, operation O_1^3 will never conflict with operation O_3^1 if it is scheduled at $st_1^3 = 0$, 1, 2, or 3. This is why the survivability of each of these start times is equal to 1. For start times $st_1^3 = 4, 5$, and 6 the probability of conflicting with a reservation assigned to O_3^1 increases, as indicated in Figure 3-7 by the higher aggregate demand for resource R_3 between time 6 and 9 (the only times where a conflict between the two operations is possible). Because the probability of such a conflict remains fairly low (i.e. the conflicts involve only two operations and only a small fraction of the reservations of these two operations conflict with each other), the survivabilities of start times $st_1^3 = 4, 5$, and 6 remain fairly close to 1 (though smaller than 1). Because operations O_2^3 and O_3^3 compete with more operations

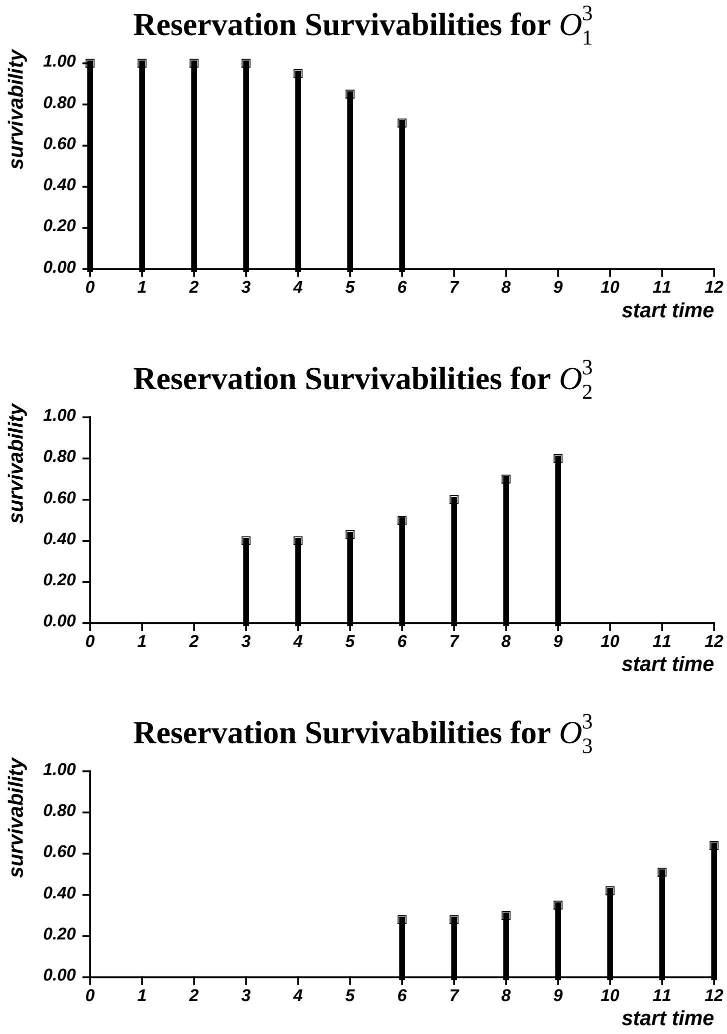


Figure 3-9: Survivability measures for the reservations of operations in job j_3, the job to which belongs O_3^3, the current critical operation.

than O_1^3, their reservation survivabilities are smaller. The shape of the survivability curves of these two operations can be interpreted using similar, though slightly more complex arguments.

3.5.4.2. Estimating the Probability that a Job Schedule Survives Contention

A good reservation is not only one that is likely to survive resource contention locally. A good reservation should also leave enough room to other operations in the same job (i.e. same process routing) so that they too can be allocated good reservations. Such good reservations can be identified by estimating for each remaining reservation (of the operation to schedule) the expected number of compatible job schedules that are likely to survive resource contention (which, in short, will be referred to as the expected number of survivable schedules). When some operations in the job have already been scheduled, rather than looking at the entire job, it is sufficient to look at the relaxation comprised of all unscheduled operations that can be reached from the current (critical) operation via precedence constraints without visiting a scheduled operation. The goodness of a reservation is then determined by the expected number of survivable solutions to this relaxation. Indeed unscheduled operations that are completely separated from the current (critical) operation by already scheduled operations will not be affected by the reservation assigned to the current operation.

In order to proceed, a few notations need to be defined:

- O_i^l: the current critical operation (i.e. the operation selected to be scheduled next);

- ρ: one of O_i^l's remaining reservations;

- $RELAX_i^l \subseteq O^l$: the set of operations making up the relaxation used by the heuristic. This set consists of O_i^l and the unscheduled operations that can be reached from O_i^l via precedence constraints without visiting a scheduled operation;

- $good_i^l(\rho)$: the goodness of assigning ρ to O_i^l, expressed as the expected number of survivable solutions to the relaxation;

- $comp_i^l(\rho)$: the set of solutions to the relaxation that are compatible with the assignment of ρ to O_i^l;

- $sol \in comp_i^l(\rho)$: a solution to the relaxation that is compatible with the assignment of ρ to O_i^l;

- $\rho(O_k^l | sol)$: the reservation assigned to an operation $O_k^l \in RELAX_i^l$ in solution *sol*.

Assuming that the probability that a solution *sol* survives resource contention can be approximated by the product of the probabilities that each reservation $\rho(O_k^l | sol)$ in *sol* survives contention, the goodness of assigning ρ to O_i^l is:

$$good_i^l(\rho) = \sum_{sol \in comp_i^l(\rho)} \prod_{O_k^l \in RELAX_i^l} surv_k^l(\rho(O_k^l | sol)) \tag{3.4}$$

This independence assumption is equivalent to omitting the interactions induced by precedence constraints in other jobs. It generally appears to be sufficient, as suggested by the results presented in the following section. Notice that, as a consequence of this assumption, the only reservation survivabilities that need to be computed in each search state are those of operations in $RELAX_i^l \subseteq O^l$, i.e. a subset of the operations in the job to which the critical operation belongs. Formula (3.4) can be rewritten to separate the survivability of reservation ρ from those of the other operations in $RELAX_i^l$:

$$good_i^l(\rho) = surv_i^l(\rho) \times \sum_{sol \in comp_i^l(\rho)} \prod_{O_k^l \in RELAX_i^l \setminus \{O_i^l\}} surv_k^l(\rho(O_k^l | sol)) \tag{3.5}$$

This can be further rewritten as:

$$good_i^l(\rho) = surv_i^l(\rho) \times compsurv_i^l(\rho) \tag{3.6}$$

where $compsurv_i^l(\rho)$ is the number of solutions compatible with the assignment of ρ to O_i^l that are expected to survive contention at operations in $RELAX_i^l \setminus \{O_i^l\}$;

$$compsurv_i^l(\rho) = \sum_{sol \in comp_i^l(\rho)} \prod_{O_k^l \in RELAX_i^l \setminus \{O_i^l\}} surv_k^l(\rho(O_k^l | sol))$$

$compsurv_i^l(\rho)$ is only a function of the start time st_i^l allocated to O_i^l in reservation ρ.

In tree-like process routings, it is possible to evaluate $compsurv_i^l(\rho)$ for all the possible start times of O_i^l in $O(\nu_l k)$ steps, where $\nu_l \leq n_l$ is the number of operations in relaxation $RELAX_i^l$, and k the maximum number of possible reservations of an operation. This is done using a dynamic programming procedure described in appendix A. This technique is an adaptation of a procedure described in [Dechter 88][29]. For non-tree-like process plans, it should be possible to remove a small number of precedence constraints (e.g. precedence constraints that are not on a critical path) to transform the process routing into a tree-like one, and use the resulting relaxation to compute goodness measures.

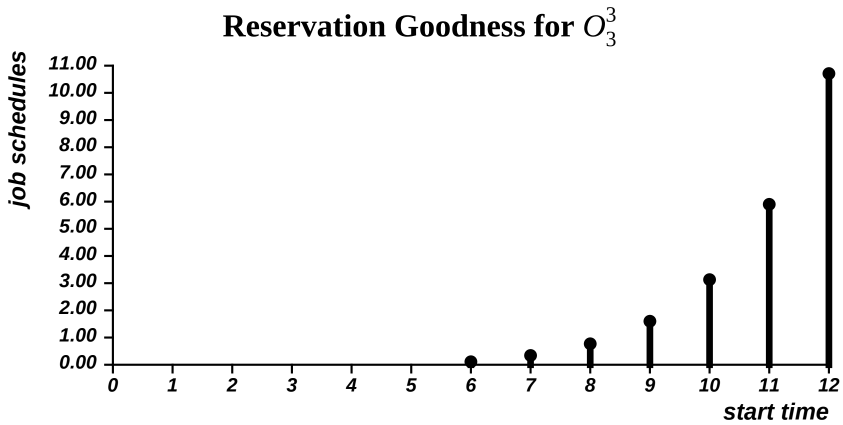


Figure 3-10: Value goodness for O_3^3 expressed as the number of compatible job schedules expected to survive resource contention.

In the example discussed earlier, the critical operation is O_3^3. Since no operation has been scheduled yet, the relaxation used by the heuristic consists of all three operations in job j_3. Figure 3-10 displays the goodness measures computed using (3.6). Start time $st_3^3=6$ for instance is only compatible with one solution to the relaxation, namely a solution in which $st_2^3=3$ and $st_1^3=0$. Therefore, the goodness of this start time is given by: $good(st_3^3{=}6) = surv_1^3(st_1^3{=}0) \times surv_2^3(st_2^3{=}3) \times surv_3^3(st_3^3{=}6)$. On the other hand, start time

[29]The complexity of Dechter's procedure is $O(\nu_l k^2)$ for general tree-like CSPs. Here we have further reduced this complexity to $O(\nu_l k)$ by taking advantage of the linearity of precedence constraints. If the model was to allow for other temporal constraints such as those described in [Allen 83], the complexity of the algorithm would be $O(\nu_l k^2)$.

$st_3^3 = 7$ is compatible with three solutions to the relaxation, one with $st_2^3 = 3$ and $st_1^3 = 0$, one with $st_2^3 = 4$ and $st_1^3 = 0$, and one with $st_2^3 = 4$ and $st_1^3 = 1$. The survivability of this start time was obtained by adding the survivabilities of each of these three solutions.

Start time $st_3^3 = 12$ is the one compatible with the largest number of solutions to the relaxation that are expected to survive contention, hence this is the start time selected by this value ordering heuristic. By assigning this start time to O_3^3, and iteratively applying the variable and value ordering heuristics that were just described, the micro-opportunistic scheduler easily completes the schedule without backtracking. This problem is relatively easy, and is also solved without backtracking by Keng and Yun's heuristic.

No heuristic is perfect. Although our value ordering heuristic recommends the right start time, a careful analysis reveals for instance that its second best choice, namely $st_3^3 = 11$, is actually infeasible. Notice however that, in the absence of backtracking, the scheduler does not need to try the second best value recommended by the heuristic: it is enough for the first value to be good.

3.5.4.3. Further Refinement

For some reservations ρ, $compsurv_i^l(\rho)$ can be very large, and can start having too much influence in (3.6) compared to $surv_i^l(\rho)$. Consider the following two reservations ρ_1 and ρ_2:

- ρ_1: $compsurv_i^l(\rho_1) = 1000$ and $surv_i^l(\rho_1) = 0.5$
- ρ_2: $compsurv_i^l(\rho_2) = 200$ and $surv_i^l(\rho_2) = 1.0$

Ideally, a good value ordering heuristic should recognize that reservation ρ_2 is better than reservation ρ_1, despite the fact that, according to Equation (3.6), $good_i^l(\rho_1) = 500$ is larger than $good_i^l(\rho_2) = 200$. Indeed, in this example, it does not really matter whether $compsurv_i^l(\rho)$ equals 200 or 1000: in either case there will certainly be enough compatible schedules. Instead, the factor that really matters here is the survivability of the reservation itself (i.e. locally). In the experiments reported at the end of this chapter, this problem was handled by filtering the number of survivable solutions compatible with a reservation ρ, $compsurv_i^l(\rho)$. Instead of relying on Equation (3.6), the system computed goodness measures with the following revised formula:

$$good_i^l(\rho) = surv_i^l(\rho) \times MIN(\Phi^{\nu_l - 1}, compsurv_i^l(\rho)) \qquad (3.7)$$

where *MIN* denotes the minimum function and Φ is a parameter of the system that is empirically adjusted. By using a filter of the form $\Phi^{\nu_l - 1}$, the heuristic attempts to ensure that, on the average, each one of the $\nu_l - 1$ other operations in the relaxation has Φ survivable reservations.

The resulting heuristic will be referred to as **FSS**, which stands for "Filtered Survivable Schedules" (value ordering) heuristic[30].

3.6. Overall Complexity

In each search state, the worst-case complexity of the look-ahead analysis is $O(max(Nk, Hm))$, where N is the number of unscheduled operations in the current search state, k the maximum number of reservations left to an operation in that state, H the scheduling horizon, and m the number of resources in the system. In general $O(Nk)$ appears to be the dominant factor. In the absence of backtracking (i.e. the number of search states generated by the system is equal to the number of operations to be scheduled), the overall complexity of the approach is $O(NOP^2 k)$, where *NOP* denotes the total number of operations to be scheduled. Experimentation with problems of different sizes suggests that, in the absence of backtracking, this is the true complexity of the approach. When backtracking occurs the overall complexity of the procedure can be much higher, though it is not often the case.

3.7. Empirical Evaluation

This section reports the results of an experimental study comparing the ORR variable ordering heuristic and FSS value ordering heuristic against the DSR (Dynamic Search Rearrangement) variable heuristic (DSR) [Bitner 75, Haralick 80, Purdom 83], the ABT (Advised Backtracking) value ordering [Dechter 88], and the combination of variable and value ordering heuristics developed by Keng and Yun [Keng 89] (and referred to as SMU). Experiments in which the granularity of the micro-opportunistic approach was increased.

---

[30] A more sophisticated way to filter $compsurv_i^l(\rho)$ would consist in filtering the number of compatible reservations of each operation in the relaxation. This would ensure that each one of the operations in the relaxation has enough compatible reservations. In general, because the critical operation is also the one in the relaxation whose reservations are the least survivable, a single filter for all the other operations in the relaxation seems sufficient.

3.7.1. Design of the Test Data

A set of 60 scheduling problems was randomly generated, each with 5 resources and 10 jobs of 5 operations each (i.e. a total of 50 operations per problem). Each job had a linear process routing specifying a sequence in which the job had to visit each one of the five resources. This sequence was randomly generated for each job, except for bottleneck resources, which were each visited after a fixed number of operations (in order to further increase resource contention).

Two parameters were adjusted to cover different scheduling conditions: a **range parameter**, *RG*, controlled the distribution of job due dates and release dates, and a **bottleneck parameter**, *BK*, controlled the number of major bottleneck resources. Six groups of ten problems were randomly generated, by considering three different values of the range parameter and two different bottleneck configurations. The value of a third parameter, which will be referred to as the **slack parameter**, *S*, had to be adjusted in function of the first two in order to keep demand for bottleneck resource(s) close to 100% over the major part of each problem[31].

The three parameters were set as follows:

- **Range Parameter** (*RG*): this parameter controlled the release date and due date distributions in each problem. Due dates were randomly drawn from a uniform distribution $(1+S)M\,U(1-RG,1)$, where $U(a,b)$ represents a uniform probability distribution between a and b, M is an estimate of the minimum makespan of the problem, and S is the slack parameter, which is defined below in function of *BK* and *RG*. The minimum makespan of the problem was estimated as $M=(n-1)\overline{du}_{R_{btnk}}+\sum_{R=R_1}^{R_m}\overline{du}_R$, where n is the number of jobs, m the number of resources, R_{btnk} the main bottleneck resource (or one of them if there are several) and $\overline{du}_{R_i}$ denotes the average duration of the operations requiring resource R_i. This estimate was first suggested in [Ow 85]. Similarly, release dates were randomly drawn from a uniform distribution of the form: $(1+S)M\,U(0,RG)$. Three values of the

[31] If this parameter had been fixed while the other parameters varied, a large proportion of the problems would have been either infeasible or trivial.

range parameter were used to generate problems[32]: $RG = 0.0, 0.1$, and 0.2.

- **Bottleneck Parameter** (BK): In half of the problems, there was only one major bottleneck ($BK = 1$), while in the other half there were two major bottlenecks ($BK = 2$).

- **Slack Parameter** (S): For problems with 2 bottlenecks or jobs with different release dates and due dates, the length of the problem was increased to $(1+S)M$ so that most problems remained feasible. The slack parameter was empirically set as $S = 0.1 \times (BK - 1) + RG$. At the same time, these values of the slack factor generally maintained demand for the bottlenecks close to 100% over the major part of each problem.

Finally, operation durations were randomly drawn from two different distributions, depending on whether the operation required a bottleneck resource or not. Bottleneck operations had durations randomly drawn from a uniform distribution $U(8, 16)$ whereas non-bottleneck operations had their durations randomly drawn from a uniform distribution $U(3,11)$. As a consequence, operations in these problems had a bit over 100 possible start times (i.e. values) after applying the consistency checking procedure to the initial search state.

3.7.2. Comparison Against Other Heuristics

Five combinations of variable and value ordering heuristics were compared:

- *DSR & ABT*: the Dynamic Search Rearrangement heuristic combined with the Advised BackTracking [Dechter 88] value ordering heuristic. The version of ABT used in these experiments was one based on the same predetermined tree-like relaxation as FSS, namely it used the process routing to which the current operation belonged. This version of ABT was carefully implemented to run in $O(\nu_l k)$ steps in each search state (where ν_l is the

[32]Due to the moderate size of the scheduling problems considered here, larger values of RG quickly tend to produce less resource contention. This is also because the slack parameter S is increased when RG becomes larger, in order to keep from generating infeasible problems.

number of operations in the tree-like relaxation and k the maximum number of remaining start times of an operation after consistency checking). This was done using a procedure similar to the one implemented in FSS[33].

- *DSR & FSS*: the DSR heuristic combined with the Filtered Survivable Schedules (FSS) value ordering heuristic (with $\Phi = 2.5$).

- *ORR & ABT*: the Operation Resource Reliance (ORR) variable ordering heuristic together with the ABT value ordering heuristic.

- *ORR & FSS*: The ORR and FSS heuristics (with $\Phi = 2.5$) advocated in this chapter.

- *SMU*: The variable and value ordering heuristics developed by Keng and Yun at the Southern Methodist University [Keng 89].

All combinations of variable and value ordering heuristics were run in a modular testbed in which all common functions were shared (e.g. consistency enforcing module, backtracking module, etc), and unnecessary functions were bypassed whenever possible (e.g. the construction of demand profiles was bypassed in DSR&ABT). All functions were implemented with equal care.

On each problem, search was stopped if it required more than 1,000 search states. The performance of each combination of variable and value ordering heuristics was compared along 3 dimensions:

1. **Search efficiency**: the ratio of the number of operations to be scheduled over the total number of search states that were explored. In the absence of backtracking, only one search state is generated for each operation, and hence search efficiency is equal to 1.

---

[33] An implementation of ABT using MST relaxations would have been too slow to be competitive. It would have required computing constraint satisfiabilities and identifying an MST relaxation in each search state. Additionally, the time required to count the number of solutions to a general MST relaxation would have been $O(\nu_l k^2)$.

2. **Number of experiments solved in less than 1,000 search states each**.

3. **Average CPU time per operation** (in seconds): this is the average CPU time required to *successfully* schedule an operation on a DECstation 3100 running Knowledge Craft on top of Allegro Common Lisp. This measure was obtained by dividing the total CPU time by the number of operations to be scheduled.

The results in Table 3-1 indicate that DSR is generally not sufficient to solve realistic job shop scheduling problems. Combined with ABT, this heuristic was only able to solve 31 problems out of 60 in less than 1,000 search states each. Even when combined with the FSS value ordering heuristic, DSR only achieved a search efficiency of 56%, and failed to solve 27 problems out of 60 in less than 1,000 search states. These results not only suggest that job shop scheduling requires a dynamic variable ordering heuristic (see also the results in the following subsection). They also indicate that the variable ordering heuristics proposed so far in the CSP literature are often too shallow for problems such as job shop scheduling. After replacing DSR with ORR in combination with ABT, search efficiency went up by 20% and 12 additional problems were solved in less than 1,000 search states each. The SMU heuristic achieved a higher efficiency of 71% and solved 43 problems out of 60 in less than 1,000 states. Even this heuristic had trouble solving many problems. In fact, it was not able to solve more problems than ORR&ABT. ORR&FSS, the variable and value ordering heuristics advocated in this chapter, yielded an impressive 85% search efficiency, and solved 52 problems out of 60 in less than 1,000 search states. Simultaneously this heuristic combination also achieved important speedups over all the other heuristics. Notice that the set of problems that ORR&FSS was not able to solve in less than 1,000 search states includes problems with $RG=0.0, BK=1$ and $S=1$, namely problems that require scheduling all jobs within the expected minimum makespan of the problem. In fact, it is possible that a couple of these problems were infeasible.

On problems with larger numbers of operations, the savings achieved by ORR&FSS appear to become even more important, although the poor performance of the generic CSP heuristics precluded systematic experimentation with such problems. At the current time, ORR&FSS has been successfully applied to several hundred scheduling problems, including a large number of problems with 100 operations, approximately 300 possible

Performance of Five Heuristics						
		DSR &ABT	**DSR &FSS**	**ORR &ABT**	**SMU**	**ORR &FSS**
RG=0.2 BK=1	**Search Efficiency**	**0.70** (0.43)	**0.81** (0.40)	**0.96** (0.06)	**1.00** (0.00)	**0.96** (0.07)
	Nb. exp. solved	**8**	**8**	**10**	**10**	**10**
	CPU seconds	**20.02** (30.17)	**13.22** (19.16)	**2.59** (0.25)	**6.19** (0.52)	**3.12** (0.36)
RG=0.2 BK=2	**Search Efficiency**	**0.49** (0.44)	**0.72** (0.46)	**0.52** (0.42)	**0.78** (0.40)	**0.99** (0.02)
	Nb. exp. solved	**7**	**7**	**6**	**8**	**10**
	CPU seconds	**28.64** (32.22)	**14.11** (15.84)	**25.16** (28.23)	**14.76** (16.54)	**3.31** (0.21)
RG=0.1 BK=1	**Search Efficiency**	**0.59** (0.46)	**0.81** (0.40)	**0.79** (0.37)	**0.62** (0.49)	**0.77** (0.39)
	Nb. exp. solved	**8**	**8**	**9**	**6**	**8**
	CPU seconds	**16.34** (18.97)	**7.62** (7.86)	**9.04** (15.23)	**18.71** (16.15)	**14.37** (23.70)
RG=0.1 BK=2	**Search Efficiency**	**0.18** (0.29)	**0.43** (0.49)	**0.47** (0.47)	**0.70** (0.45)	**0.87** (0.31)
	Nb. exp. solved	**3**	**4**	**6**	**7**	**9**
	CPU seconds	**32.59** (17.91)	**12.11** (8.20)	**31.60** (32.76)	**14.00** (12.44)	**6.04** (9.61)
RG=0.0 BK=1	**Search Efficiency**	**0.24** (0.40)	**0.29** (0.40)	**0.51** (0.46)	**0.43** (0.49)	**0.72** (0.46)
	Nb. exp. solved	**2**	**3**	**7**	**4**	**7**
	CPU seconds	**34.84** (17.51)	**25.44** (16.56)	**25.69** (29.44)	**26.37** (17.71)	**18.49** (25.62)
RG=0.0 BK=2	**Search Efficiency**	**0.27** (0.36)	**0.34** (0.46)	**0.43** (0.42)	**0.74** (0.43)	**0.81** (0.40)
	Nb. exp. solved	**3**	**3**	**5**	**8**	**8**
	CPU seconds	**28.30** (23.52)	**21.29** (14.82)	**28.00** (27.32)	**14.77** (17.33)	**11.00** (16.73)
Overall Performance	**Search Efficiency**	**0.41** (0.43)	**0.56** (0.47)	**0.61** (0.42)	**0.71** (0.43)	**0.85** (0.32)
	Nb. exp. solved	**31**	**33**	**43**	**43**	**52**
	CPU seconds	**26.79** (24.00)	**15.86** (15.03)	**20.34** (26.11)	**15.80** (15.35)	**9.42** (16.65)

Table 3-1: Comparison of 5 heuristics over 6 sets of 10 job shop problems. Standard deviations appear between parentheses.

start times per operation, and bottleneck loads close to 100% over the major part of each problem. The heuristics have also been run on a smaller set of problems with 200 operations. Backtracking remained very low on most of these problems.

3.7.3. Varying the Granularity of the Approach

A second set of experiments was carried out in order to evaluate the impact of the granularity of the micro-opportunistic search procedure on the ability of the system to achieve high search efficiency. Indeed, important computational savings could possibly be realized if it were possible to achieve as high a search efficiency without having to revise the search procedure in each search state.

Two variations of the micro-opportunistic scheduler based on the ORR and FSS heuristics were implemented. These two variations differed from the original version in the number , G, of operations that had to be scheduled before the scheduler was allowed to look for a new critical resource/time interval, i.e. before the scheduler was allowed to revise its current search strategy. For $G > 1$, the G operations with the largest reliance on the current critical resource/time interval were scheduled one by one in decreasing order of their reliance.

Impact of the Granularity on Performance			
	$G=3$	$G=2$	$G=1$
Search Efficiency	**0.70** (0.41)	**0.76** (0.38)	**0.85** (0.32)
Nb. exp. solved	**46**	**48**	**52**
CPU seconds	**14.19** (19.17)	**13.51** (20.74)	**9.42** (16.65)

Table 3-2: Varying the granularity of the approach.
Standard deviations appear between parentheses.

Table 3-2 compares the overall performance of the micro-opportunistic scheduler ($G=1$) against two coarser variations ($G=2$ and $G=3$) over the same set of 60 scheduling problems. The results indicate that the performance of the procedure quickly degrades as its granularity increases, thereby suggesting that all the flexibility of the micro-opportunistic search procedure is actually required to efficiently solve these types of problems. Notice also that, even with $G=3$, ORR&FSS outperforms all other four heuristic combinations studied in the previous subsection.

3.8. Summary and Conclusions

This chapter studied a variation of the job shop scheduling problem in which operations have to be performed within one or several non-relaxable time windows. Examples of such problems include factory scheduling problems in which some operations have to be performed within one or several shifts, spacecraft mission scheduling problems, some factory rescheduling problems, or any other scheduling problem with hard deadlines. These types of scheduling problems cannot be solved with traditional scheduling techniques such as priority dispatch rules or similar one-pass scheduling techniques. Mixed Integer Programming techniques which could potentially deal with these problems have been overwhelmed so far by the combinatorial number of binary variables required to account for limited resource capacities [Nemhauser 88]. Instead, this chapter demonstrated that many instances of these problems can be efficiently solved within the micro-opportunistic search paradigm by combining consistency enforcing techniques and look-ahead techniques to decide which operation to schedule next and which reservation to assign to that operation. Experiments in which the granularity of the micro-opportunistic procedure was increased indicate that the ability of the micro-opportunistic approach to constantly revise its search strategy is instrumental in efficiently solving many of these problems.

Several variable and value ordering heuristics to guide the micro-opportunistic scheduler were successively studied, including both generic heuristics which had been reported to perform particularly well on other CSPs and specialized heuristics developed for similar scheduling problems. The review indicated that generic CSP heuristics are usually not sufficient to solve hard CSPs such as job shop scheduling. This is because these heuristics fail to properly account for the constraint interactions induced by the high connectivity of the constraint graphs typically encountered in job shop scheduling. Instead, a new probabilistic model of the search space was introduced which allows to estimate the reliance of an operation on the availability of a reservation, and the degree of contention among unscheduled operations for the possession of a resource over some time interval. Based on this probabilistic model, new variable and value ordering heuristics were defined:

1. The "Operation Resource Reliance" (ORR) variable ordering heuristic selects the operation that relies most on the most contended resource/time interval, and

2. The "Filtered Survivable Schedules" (FSS) value ordering heuristic assigns to that operation the reservation which is expected to be compatible with the largest number of survivable job schedules, i.e. job schedules that are expected to survive resource contention.

Experimental results show that these two heuristics enable the micro-opportunistic scheduler to *efficiently* solve many job shop scheduling problems that could not be efficiently solved by prior heuristics (both generic CSP heuristics and specialized heuristics designed for similar scheduling problems). The results also indicate that the ORR and FSS heuristics not only yield significant increases in search efficiency but also achieve important reductions in search time.

The estimates of resource contention used in the ORR and FSS heuristics are based on several independence assumptions. More sophisticated versions of these heuristics have also been implemented, which attempt to better account for different dependencies, some using more complex analytical models [Sadeh 88, Sadeh 89a, Sadeh 90] others relying on Monte Carlo simulations [Sadeh 89b]. The increase in search efficiency generally achieved by these more sophisticated versions did not seem to justify their heavier computational requirements.

Our experimental study suggests that there remains a small number of particularly difficult problems that cannot be solved efficiently by the ORR and FSS heuristics (or by any of the other heuristics that were tested). Because job shop scheduling is NP-complete, this is likely to always be the case. Nevertheless, more powerful variable and value ordering heuristics may allow to further reduce the number of problems that cannot be solved efficiently. Alternatively, new more powerful consistency enforcing techniques or more sophisticated deadend recovery schemes could also further improve the efficiency of the micro-opportunistic approach.

The heuristics presented in this chapter are intended for job shop scheduling problems, i.e. scheduling problems with both precedence and capacity constraints. The probabilistic measures of reliance and contention that were described can be used in any resource allocation problem, and more generally any CSP with disequality constraints (i.e. constraints preventing two variables from being assigned the same value), since these problems can be formulated as resource allocation problems (e.g. the N-queens problem can be formulated as a resource allocation problem in which each queen/row is a task and each column is a resource[34]). However, the lessons learned from this work go beyond job shop scheduling and resource allocation problems. Fundamental weaknesses of generic variable and value ordering heuristics often praised in the CSP literature were identified. Variable ordering heuristics like DSR count the number of values left to each variable but do not account for the chances that these values remain available in the future. Variable ordering heuristics like MW or MC count the number of constraints incident on a variable but do not account for the tightness of these constraints. Value ordering heuristics like ABT assume that the problem admits a tight tree-like relaxation. The probabilistic model of the search space used to define the ORR and FSS heuristics allows to overcome these weaknesses by providing a framework in which more sophisticated approximations of variable criticality and value goodness can be defined. For instance, the ORR and FSS heuristics rely on efficient probabilistic approximations of resource contention. These measures of resource contention enable the scheduler to account for entire cliques of capacity constraints rather than rely on advice based on tree-like relaxations of these cliques. Last but not least, this work suggests that benchmark problems often used in the CSP literature are not representative of hard problems such as job shop scheduling. It is hoped that this dissertation will prompt researchers in the field to look for new benchmark problems and new more powerful heuristics for these problems.

The heuristics discussed in this chapter all share a common weakness: they always perform the same analysis independently of the problem that they are presented, and regardless of the difficulty of the current search state. A more flexible approach would allow for different heuristics to be used according to the difficulty of the problem and even according to the difficulty of the current search state. One such mechanism was implemented in an earlier version of the system which relied on Monte Carlo sampling to measure resource contention. Because, the system measured resource contention between feasible job schedules, it was possible to accurately determine if the system had reached a search state in which backtracking could no longer occur. Because, like ORR, the variable ordering heuristic implemented in that version of the system was particularly good at quickly reducing contention, such search states were generally reached after 30 to 60% of the operations had been scheduled. At that point, the system would arbitrarily

[34]Constraints representing the ability of queens to attack each other along diagonals can be represented as constraints further restricting admissible resource assignments.

complete the schedule (i.e. using arbitrary variable and value orderings), which generally resulted in important speedups[35]. This dynamic switch could also be implemented in the current version of the system, though it would be less accurate. Similarly, different consistency enforcing techniques could be applied to different problems, different search states, or even to different parts of a same problem (e.g. enforcing higher consistency levels with respect to capacity constraints at the bottlenecks). Preliminary experimentation with such flexible consistency enforcing techniques have been reported by Collinot and Le Pape [Collinot 90].

The next chapter deals with another variation of the job shop scheduling problem, in which the objective is not only to come up with a feasible schedule as fast as possible but also to produce as good a schedule as possible. The domain of application is factory scheduling.

[35]Consistency enforcement was still carried out in each search state, and was at that point sufficient to guarantee (within the accuracy of the Monte Carlo sampling method) backtrack-free search.

Chapter 4

Factory Scheduling: A Constrained Optimization Problem

4.1. Introduction

This chapter describes **MICRO-BOSS**[36], a micro-opportunistic factory scheduling system. The factory scheduling problem is a particularly difficult Constrained Optimization Problem. Attempts to build optimal solutions to any slightly realistic version of this problem have all failed so far[37]. Like in any other heuristic approach to this problem, the objective in MICRO-BOSS is to efficiently come up with as good a solution as possible. MICRO-BOSS differs however from many earlier approaches in at least two important ways:

1. **A micro-opportunistic approach**: While earlier approaches to factory scheduling such as those embedded in ISIS, OPT, and OPIS have relied on coarse problem decompositions in which large resource subproblems or large job subproblems are scheduled one by one, the search procedure in MICRO-BOSS allows each operation to be considered as an independent decision point. Experimental results presented in Chapter 5 indicate that this extra flexibility in the search procedure yields significantly better schedules.

2. **A more realistic cost model**: Many scheduling techniques have been developed that focus on minimizing makespan (i.e. maximizing resource utilization) without taking care of meeting job due dates, or attempt to meet job due dates without paying attention to inventory costs. MICRO-BOSS

[36]MICRO-BOSS is an acronym for Micro-Bottleneck Scheduling System.

[37]See the review in Chapter 1.

uses a more realistic cost model that directly accounts for the tardiness costs, in-process inventory costs, and finished-goods inventory costs introduced by each job. This model is intended to allow for the production of just-in-time schedules [Voss 87], and was influenced by a similar cost model developed by Morton et al. in the context of the SCHED-STAR system [Morton 88].

The emphasis of this chapter is on look-ahead techniques that enable MICRO-BOSS to continuously track the evolution of highly contended resource/time intervals during the construction of the schedule. Like in the CSP approach described in the previous chapter, these look-ahead techniques help the scheduler decide which operation to schedule next and which reservation to assign to that operation. However, because the objective in this chapter is not simply to come up with a feasible solution as fast as possible, but rather to efficiently come up with a solution that minimizes schedule costs, the notions of critical operation and promising reservation are different. In this chapter, it is argued that a critical variable in a COP is one that participates in an important tradeoff (e.g. a couple of jobs with important tardiness costs that all require the same resource at the same time in order to meet their due dates). A promising value for that variable is one that will optimize this tradeoff. It is shown that the probabilistic model introduced in Chapter 3 for the job shop CSP can be extended to identify operations participating in important tradeoffs by building demand profiles that are biased towards those reservations expected to produce the best schedules.

The next section provides a formal definition of the factory scheduling model hypothesized in this chapter. The following sections describe the look-ahead technique used in MICRO-BOSS to decide which operation to schedule next and which reservation to assign to that operation. Experimental results comparing MICRO-BOSS against several other scheduling techniques as well as against different variations of the scheduler itself are presented in Chapter 5.

4.2. Problem Definition

The job shop scheduling problem[38] requires scheduling a set of jobs $J=\{j_1,...,j_n\}$ on a set of physical resources $RES=\{R_1,...,R_m\}$. Each job j_l consists of a set of operations $O^l=\{O^l_1,...,O^l_{n_l}\}$ to be scheduled according to a process routing that specifies a partial ordering among these operations (e.g. O^l_i BEFORE O^l_j). This chapter assumes factory scheduling problems with *in-tree* process routings, in which operations can have several direct predecessors but at most one direct successor (i.e. assembly-type of process routings). Figure 4-1 displays two examples of in-tree process routings.

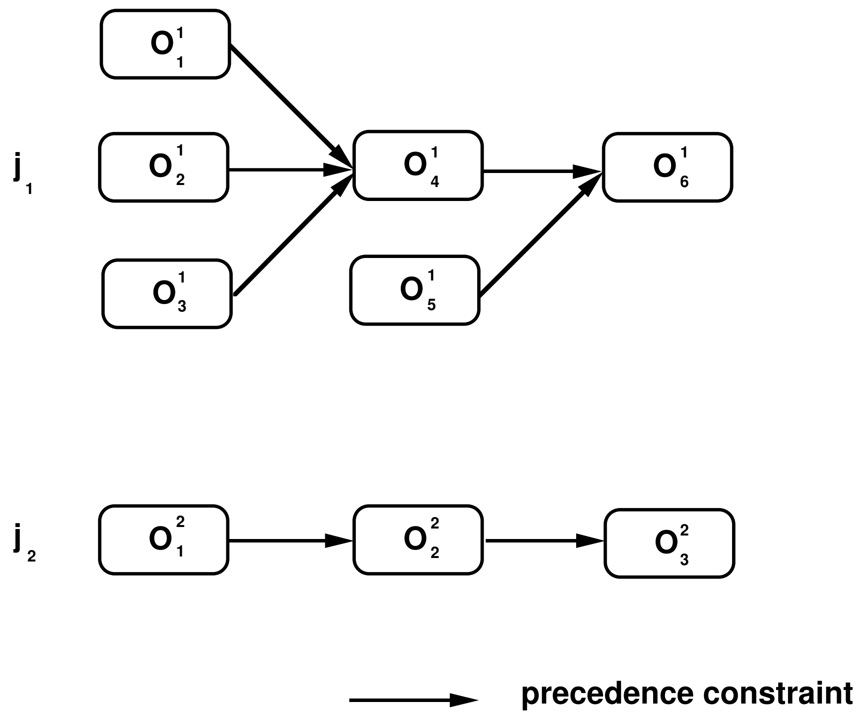


Figure 4-1: Two examples of in-tree process routings.

Additionally a job j_l has an earliest acceptable release date erd_l, a due-date dd_l, and a latest acceptable completion date lcd_l. It is assumed that, for each job j_l, $lcd_l \geq dd_l \geq erd_l$. Each job has to be scheduled between its earliest acceptable release date and latest

---

[38]The scheduling model defined in this section is very close to the one used in the previous chapter. For the sake of completeness, the model is entirely redefined here. Notice however the following differences with the previous chapter: the introduction of costs to be minimized, the restriction to in-tree process routings instead of tree-like process routings, and the restriction to problems where operations have a single resource requirement, for which there is no alternative.

acceptable completion date. The earliest acceptable release date may correspond to the earliest possible arrival date of raw materials or to a rough release date provided by a master scheduling module. It is assumed that the actual release date will be determined by the schedule to be constructed. The latest acceptable completion date may correspond to a date after which the customer will refuse delivery. If such a date does not actually exist, it can always be chosen far enough in the future so that it is no longer a constraint.

Each operation O_i^l has a fixed duration, du_i^l, and a start time, st_i^l (to be determined), whose domain of possible values is delimited by an earliest start time, est_i^l, and a latest start time, lst_i^l (initially derived from the job's earliest acceptable release date erd_l and latest acceptable completion date lcd_l). In order to be successfully executed, each operation O_i^l requires a resource R_i^l (e.g. $R_i^l = R_1$, a milling machine).

More formally, the problem can be defined as follows:

VARIABLES:

The variables of the problem are the operation start times, st_i^l $(1 \le l \le n,\ 1 \le i \le n_l)$.

CONSTRAINTS:

The non-unary constraints of the problem are of two types:

1. The **precedence constraints** defined by the process routings translate into linear inequalities of the form: $st_i^l + du_i^l \le st_j^l$ (i.e. O_i^l BEFORE O_j^l).
2. The **capacity constraints** that restrict the use of each resource to only one operation at a time translate into disjunctive constraints of the form: $R_i^k \ne R_j^l \vee st_i^k + du_i^k \le st_j^l \vee st_j^l + du_j^l \le st_i^k$. These constraints simply express that, unless they use different resources, two operations O_i^k and O_j^l cannot overlap.

As mentioned earlier, each job has to be performed between its earliest acceptable release date and latest acceptable completion date. Time is assumed discrete. Operation start times and end times can only take integer values. The model also allows for any type of unary constraint that further restricts the set of possible start times of an operation. Resources can also be unavailable over some prespecified time intervals (e.g. night shifts, week-ends, time intervals reserved for maintenance, etc.).

COSTS:

Each job j_l has:

• a **marginal tardiness cost**, $tard_l$: the cost incurred for each unit of time that the job is tardy (i.e. completes past its due date). Marginal tardiness costs generally include tardiness penalties, interests on lost profit, loss of customer goodwill, etc[39]. The total tardiness cost of job j_l, in a given schedule, is:

$$TARD_l = tard_l \times Max(0, C_l - dd_l) \tag{4.1}$$

where $C_l = st^l_{n_l} + du^l_{n_l}$ is the completion date of job j_l in that schedule, assuming that $O^l_{n_l}$ is the last operation in job j_l.

• **marginal in-process and finished-goods inventory costs**: Each operation O^l_i can incrementally introduce its own non-negative marginal inventory cost, inv^l_i. Typically, the first operation in a job introduces marginal inventory costs that correspond to interests on the costs of raw materials, interests on processing costs (for that first operation), and marginal holding costs. Downstream operations[40] introduce additional marginal inventory costs in the form of interests on processing costs or interests on the costs of additional raw materials required by these operations. The total inventory cost for a job j_l, in a given schedule, is:

$$INV_l = \sum_{i=1}^{n_l} inv^l_i \times [Max(C_l, dd_l) - st^l_i] \tag{4.2}$$

Notice that, for the sake of simplicity, inventory costs are always counted from the start time of the operation that introduces them[41].

---

[39] In this model, extra inventory costs incurred past the due date are not accounted for in the tardiness costs. Instead they are accounted for in the inventory costs described below.

[40] An operation O^k_i is said to be downstream (upstream) of another operation O^k_j, within the same job, if O^k_i is a direct or indirect successor (predecessor) of O^k_j in that job, as defined by its process routing.

[41] Some costs such as direct holding costs are typically incurred after the operation is completed. However, since each operation is assumed to have a fixed duration, this simplification is equivalent to adding a *fixed* cost to the total schedule cost. In other words, the difference between the costs of two schedules is not affected by this simplification.

Inventory costs can be decomposed into in-process and finished-goods inventory costs:

$$INV_l = \sum_{i=1}^{n_l} [\, inv_i^l \times (C_l - st_i^l) \,] \; + \; (\sum_{i=1}^{n_l} inv_i^l) \times Max(0, dd_l - C_l) \tag{4.3}$$

Sometimes, it is also convenient to rewrite (4.2) as:

$$INV_l = \sum_{i=1}^{n_l - 1} [\, (\sum_{k \in BEF_i^l} inv_k^l) \times (st_{succ(i)}^l - st_i^l)] \tag{4.4}$$

$$+ \; (\sum_{i=1}^{n_l} inv_i^l) \times [\, Max(C_l, dd_l) - st_{n_l}]$$

where $BEF_i^l = \{i\} \cup \{k \,|\, O_k^l \textit{ is upstream of } O_i^l \textit{ in } j_l\}$ and $O_{succ(i)}^l$ is the direct successor of O_i^l $(1 \le i < n_l)$. This last equation emphasizes the cumulative effect of the marginal inventory costs incrementally introduced by each operation preceding O_i^l in j_l. Every unit of time that operation $O_{succ(i)}^l$ is delayed after O_i^l has been completed results in an overhead inventory cost of $\sum_{k \in BEF_i^l} inv_k^l$.

The total cost of a schedule is obtained by summing the cost of each job schedule:

$$\boxed{Schedule\ Cost = \sum_{l=1}^{n} (TARD_l + INV_l)}$$

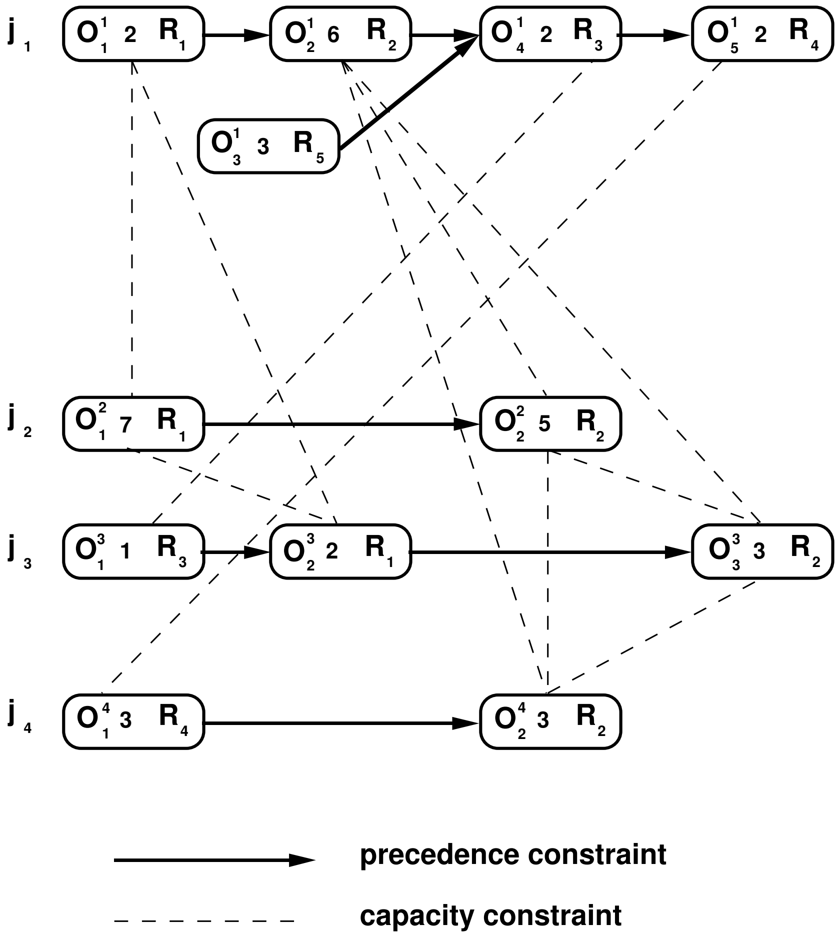
OBJECTIVE:

The goal of the scheduler is to efficiently produce a feasible schedule that reduces as much as possible the total schedule cost.

EXAMPLE:

These costs are now illustrated with the small scheduling problem introduced in Chapter 1, and depicted again in Figure 4-2. This example will also be used throughout this chapter to illustrate the look-ahead mechanisms implemented in MICRO-BOSS.

Each square box in Figure 4-2 represents an operation. Each box is labeled with the name of the operation that it represents (e.g. O_1^1), the duration of that operation (e.g. $du_1^1 = 2$), and its resource requirement (e.g. $R_1^1 = R_1$). The arrows represent precedence constraints. The other arcs in the graph represent capacity constraints. The earliest



Job j_l	erd_l	dd_l	lcd_l	$tard_l$	inv_1^l	inv_2^l	inv_3^l	inv_4^l	inv_5^l
j_1	0	12	20	20	2	1	2	0	0
j_2	0	14	20	20	5	0	-	-	-
j_3	0	9	20	5	1	0	0	-	-
j_4	0	18	20	10	1	0	-	-	-

Figure 4-2: A simple job shop problem with 4 jobs. Each node is labeled by the operation that it represents, its duration, and the resource that it requires. The earliest acceptable release date, due date, and latest acceptable completion date of each job is provided in the table along with marginal tardiness and inventory costs.

acceptable release date, due date, and latest acceptable completion date of each job is provided in the table along with marginal tardiness and inventory costs. Once again, notice that R_2 is the only resource required by four operations (one from each job). Notice also that, in three out of four jobs (namely j_1 ,j_3 , and j_4), the operation requiring R_2 is one of the job's longest operations. Consequently resource R_2 can be expected to be the main bottleneck of the problem. To some extent, resource R_1 constitutes a secondary bottleneck.

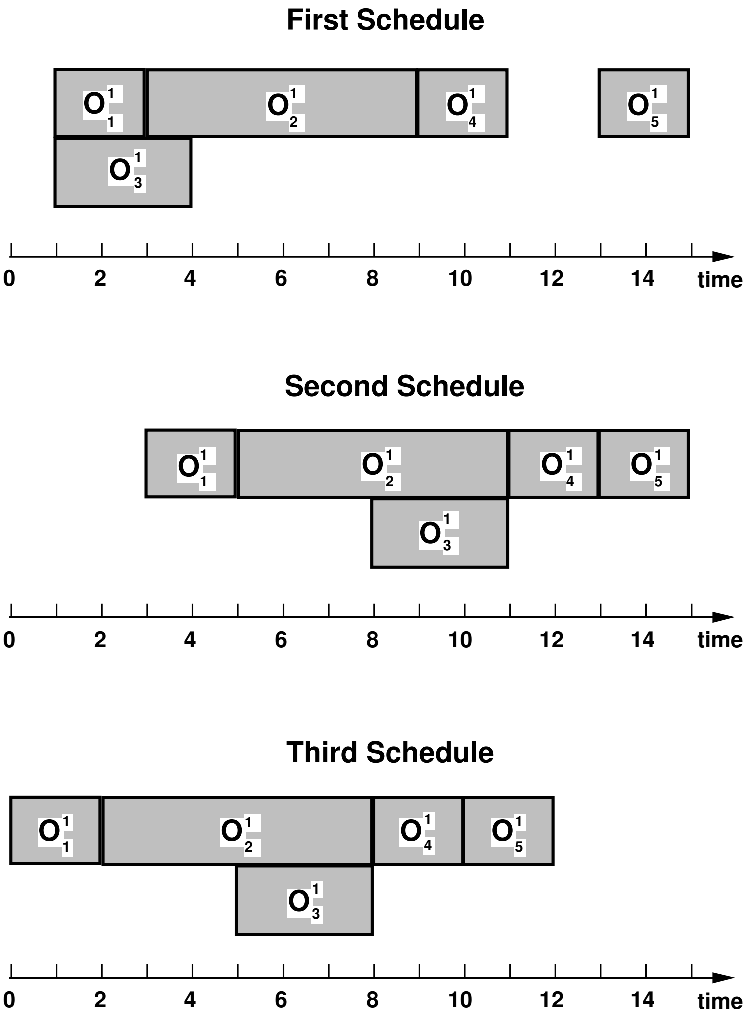


Figure 4-3: Three possible schedules for j_1.

Figure 4-3 depicts three possible schedules for job j_1. In the first schedule, job j_1 is completed at time 15, i.e. 3 time units past its due date. According to Equation (4.1), the tardiness cost incurred by job j_1 in that schedule is $TARD_1 = 20 \times 3 = 60$, and the inventory cost, using Equation (4.2), is $INV_1 = 2 \times 14 + 1 \times 12 + 2 \times 14 = 68$. The total cost incurred by j_1 would therefore be 60 + 68 = 128. In the second schedule, job j_1 completes at the same time. Its tardiness cost is therefore the same as in the first schedule. Its inventory cost is lower, since all the operations preceding O_5^1 have been compactly scheduled before that operation. In this second example, $INV_1 = 2 \times 12 + 1 \times 10 + 2 \times 7 = 48$. Finally, the third schedule displayed in Figure 4-3 is a just-in-time schedule. This is the ideal schedule: job j_1 is compactly scheduled to finish exactly on its due date. In this case, $TARD_1 = 0$ and, like in the second schedule, $INV_1 = 2 \times 12 + 1 \times 10 + 2 \times 7 = 48$. Therefore, the total cost incurred by job j_1 in the third schedule is $0 + 48 = 48$.

4.3. Look-ahead Analysis

4.3.1. Overview

The top-level search procedure implemented in MICRO-BOSS is the micro-opportunistic search procedure described in Chapter 2. It is exactly the same top-level procedure as the one used for the job shop CSP in the previous chapter. The notions of critical variable and promising values are however different, and hence require the modification of the look-ahead analysis developed for the job shop CSP. Notice also that the search procedure implemented in the current version of MICRO-BOSS stops as soon as a first solution has been found. Clearly, if additional computational time is available, the procedure can be readily transformed into a branch-and-bound procedure [Nemhauser 88] or a beam search procedure [Lowerre 76, Fox 83][42].

4.3.1.1. General Considerations

An ideal value ordering heuristic (for a COP) would be one that would always return an optimal assignment (i.e. an assignment that participates in one of the best solutions compatible with the current partial solution) and could be evaluated in no time. Like for CSPs, if such an ideal value ordering heuristic existed, the order in which variables are instantiated would not matter: backtracking would never occur and the system would always obtain optimal solutions. Given an imperfect value ordering heuristic, the order in which variables are instantiated will generally affect search efficiency, search time, and the quality of the solution. Ideally, given an imperfect value ordering heuristic, a perfect variable ordering heuristic would simultaneously minimize search time and maximize the quality of the solution. Unfortunately, these two objectives are not necessarily synonymous: some variable ordering heuristics may be better at reducing backtracking while others tend to lead to better solutions. Rigorously, the merit of a variable ordering heuristic depends on the importance of reducing search time relative to the importance of producing a good solution. It is not the purpose of this study to get into such considerations. Instead our aim is to come up with heuristics that will provide as good a solution as possible to realistic job shop COPs (with several hundred, possibly

[42]The look-ahead analysis would remain the same as the one described in this chapter since the goal of the procedure would still be to explore those branches of the search tree that are expected to lead to the best (feasible) solutions. In a branch-and-bound procedure, the best solution already obtained by the search procedure would provide an upper-bound that would be used to prune the search tree from alternatives that are provably more expensive than the current best solution.

several thousand operations) in a reasonable amount of time (typically under an hour of CPU time on a state-of-the-art workstation). To this end, efficient heuristics are developed that attempt to produce as good a solution as possible while maintaining backtracking at a low level.

4.3.1.2. Optimizing Critical Conflicts First

If all jobs could be scheduled optimally (i.e. just-in-time), there would be no scheduling problem. Generally this is not the case. Jobs typically have conflicting resource requirements. The look-ahead analysis carried out by MICRO-BOSS in each search state is meant to allow the scheduler to focus its effort on those conflicts that currently appear most critical. In job shop COPs, a critical conflict is one that will require an important tradeoff, i.e. a tradeoff that will significantly impact the quality of the *entire* schedule. By first focusing on critical conflicts, MICRO-BOSS ensures that it has as many options as possible to optimize these conflicts. As illustrated by a trace provided at the end of this chapter, once critical tradeoffs have been worked out, the remaining unscheduled operations tend to become more decoupled and hence easier to optimize. As contention subsides, so does the chance of backtracking. In other words, by constantly redirecting search towards those tradeoffs that appear most critical, MICRO-BOSS is expected to produce better schedules and simultaneously reduce its chances of backtracking. Experimental results presented in Chapter 5 suggest that this is indeed the case.

4.3.1.3. The Look-ahead Procedure

In general, the importance of a conflict (and the criticality of the operations participating in that conflict) depends on the number of jobs that are competing for the same resource, the amount of temporal overlap between the requirements of these jobs, the number of alternative reservations (i.e. start times) still available to the conflicting operations and the differences in cost between these alternative reservations. In order to identify critical conflicts, MICRO-BOSS performs a two-step look-ahead analysis:

Step 1: Within each job, MICRO-BOSS identifies the best remaining reservations available to each unscheduled operation in the job, and the marginal costs that would be incurred by the job if alternative reservations were instead assigned to these operations. These marginal costs are indicators of the reliance of each operation on the availability of its remaining reservations. Operations with many possible reservations left and low marginal costs do not heavily rely on anyone of their remaining reservations. On the other hand, operations with only a small number of good reservations rely more on the availability of these reservations.

Step 2: MICRO-BOSS estimates resource contention over time by building probabilistic demand profiles that are biased towards those reservations that are more heavily relied upon, i.e. towards those reservations that are expected to produce the best schedules. Highly contended resource/time intervals correspond to important conflicts/tradeoffs on which the scheduler should work first.

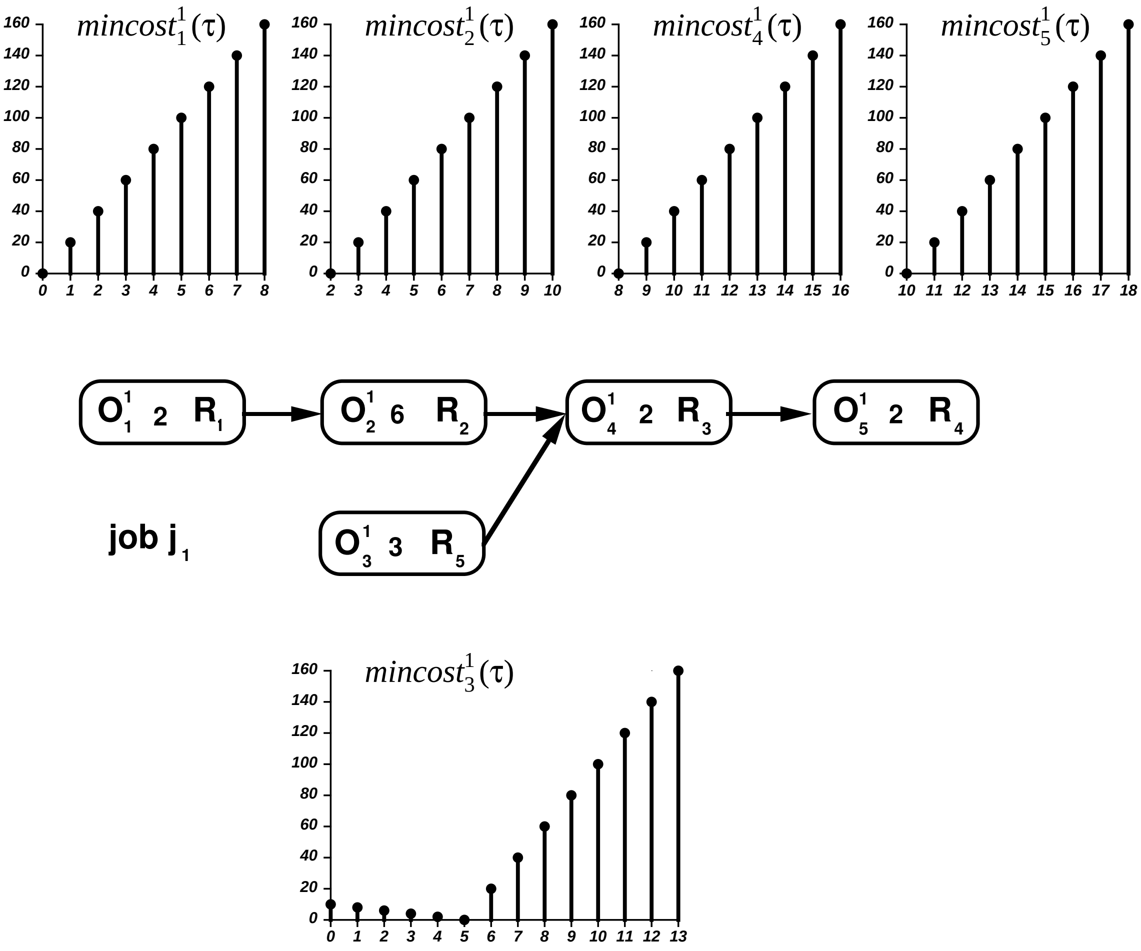
Critical operations are identified as operations whose good reservations conflict most with the good reservations of other operations, namely operations that heavily rely on the most contended resource/time intervals. Promising reservations for these operations are those reservations that minimize the costs of the conflicting jobs.

The balance of this section provides further details on the two-step look-ahead analysis. The following two sections describe the operation and reservation ordering heuristics based on this look-ahead analysis.

4.3.2. Step 1: Reservation Optimization Within a Job

In order to detect critical conflicts between the resource requirements of unscheduled operations, MICRO-BOSS keeps track of the best start times that remain available to each unscheduled operation within its job, and the marginal costs that would be incurred by the job if the operation was allocated another start time. More specifically, for each remaining possible start time τ of each unscheduled operation O_i^k, MICRO-BOSS (implicitly) keeps track of the minimum additional costs, $mincost_i^k(\tau)$, that would be incurred by the job j_k to which O_i^k belongs, if O_i^k was to start at $st_i^k = \tau$ rather than at one of its best remaining possible start times. By definition, if $st_i^k = \tau$ is one of the best start times that remain available to O_i^k within its job, $mincost_i^k(\tau) = 0$. This subsection first illustrates how these costs change from one search state to another, then describes efficient procedures to keep track of these changes.

Figure 4-4 displays the $mincost_i^1$ curves for each of the five operations required by job j_1



$erd_1 = 0$ $dd_1 = 12$ $lcd_1 = 20$ $tard_1 = 20$

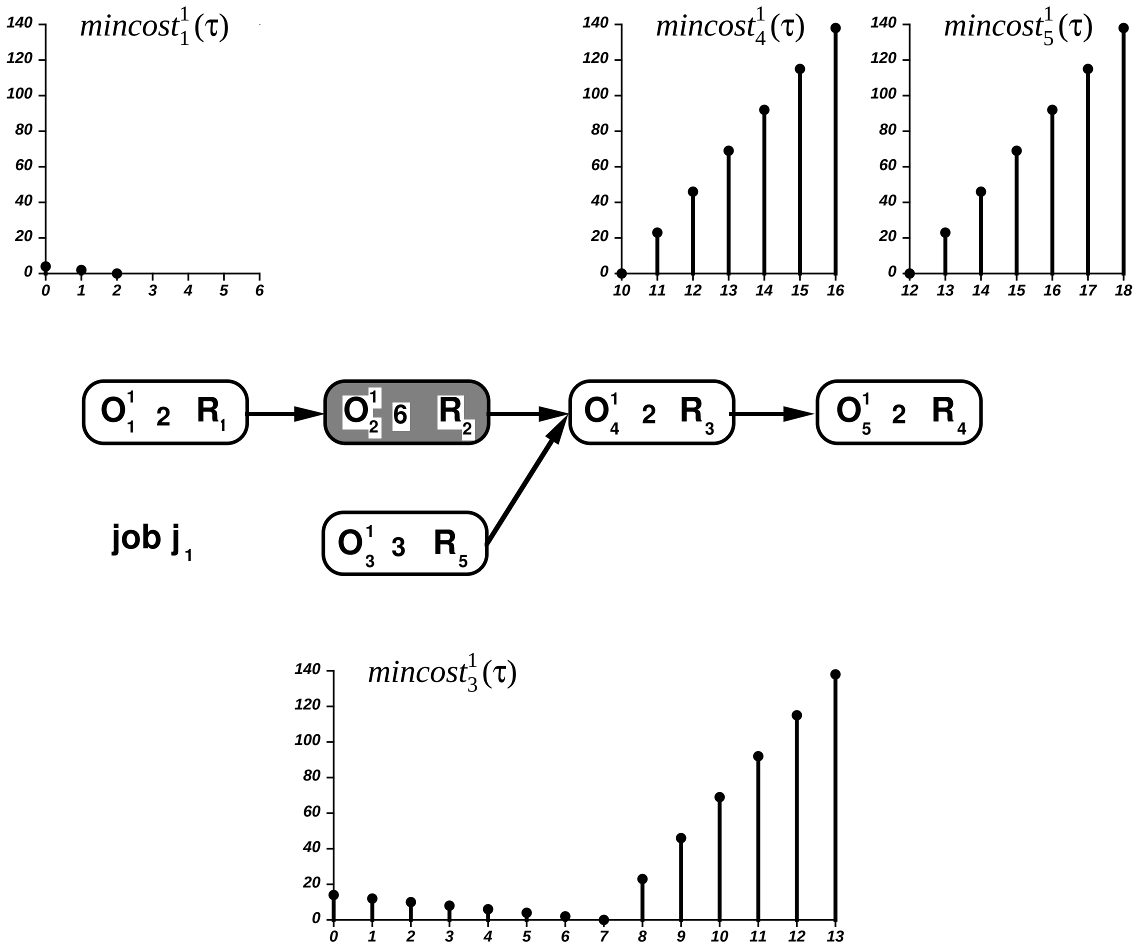
$inv_1^1 = 2$ $inv_2^1 = 1$ $inv_3^1 = 2$ $inv_4^1 = 0$ $inv_5^1 = 0$

Figure 4-4: Assessing the merits of alternative scheduling decisions in the initial search state.

in the example introduced in the previous section. These curves were computed in the initial search state. In that state, all operations still have to be scheduled, and all resources are entirely available. Consider operation O_1^1. After consistency checking, the earliest possible start time of that operation is $est_1^1=0$ and its latest possible start time $lst_1^1=8$: O_1^1 has 9 possible start times. In order for job j_1 to meet its due date (at time 12), operation O_1^1 would have to start at $st_1^1=0$, and the other operations in job j_1 would have to be compactly scheduled right after it. The optimal schedule for this job is the just-in-time schedule displayed in Figure 4-3. Any delay in starting O_1^1 will translate into tardiness costs. For instance, starting O_1^1 at $st_1^1=1$ would result in tardiness costs of at least 20×1=20 (since, in the best case, job j_1 would be late by 1 time unit and $tard_1=20$). Ideally, the operations in $jobj_1$ would still be compactly scheduled, and inventory costs would not be larger than those in the just-in-time schedule. Hence, in this initial state, the minimum additional costs incurred by job j_1, if $st_1^1=1$, is $mincost_1^1(1)=20$. Similarly, $mincost_1^1(2)=40$, $mincost_1^1(2)=60$, etc. Similar considerations apply to operations O_2^1, O_4^1, and O_5^1. For operation O_3^1, things are slightly different. Indeed, if that operation is started before $st_3^1=5$ (and hence completes before time 8), tardiness costs no longer decrease. Instead, additional inventory costs will be incurred by job j_1. For instance, if $st_3^1=4$, the best compatible schedule for j_1 will complete on time but will incur an overhead inventory cost of 2×1=2 (since $inv_3^1=2$), hence $mincost_3^1(4)=2$. Similarly, $mincost_3^1(3)=4$, $mincost_3^1(2)=6$, etc.

Suppose that the scheduler creates a new search state and schedules[43] O_2^1 to start at $st_2^1=4$ (see Figure 4-5). In this search state (and all its possible descendants, i.e. all the possible schedules that can be built by completing this partial schedule), job j_1 can no longer be completed on time. Job j_1 will be at least 2 time units behind its due date. This affects the optimal start time of O_3^1 which shifts from $st_3^1=5$ to $st_3^1=7$. Starting O_3^1 earlier than at $st_3^1=7$ no longer reduces tardiness costs. Instead, it would only increase inventory costs (e.g. $mincost_3^1(6)=2$ and $mincost_3^1(0)=14$). Notice that the penalty incurred by job j_1 for every time unit that O_3^1 starts after its best start time (i.e. $st_3^1=7$ in the current state, $st_3^1=5$ in the initial state) has increased compared to what it used to be in the initial state. Indeed, in the current search state, if O_3^1 is scheduled to start past its best start time, say at $st_3^1=8$ for instance (i.e. 1 time unit past its best start time), job j_1 will not only incur a tardiness cost of $1\times tard_1=20$, but it will also see its inventory costs augment by at least $(inv_1^1+inv_2^1)\times 1=3$. This is because the start time of O_2^1 has already been fixed. If O_3^1 were to start at $st_3^1=8$, O_1^1 and O_2^1 would no longer be compactly scheduled right in front

[43]This scheduling decision as well as the other ones assumed later in this example are not good decisions. They are only made to illustrate interesting changes in the *mincost* functions.



$erd_1 = 0 \qquad dd_1 = 12 \qquad lcd_1 = 20 \qquad tard_1 = 20$

$inv_1^1 = 2 \qquad inv_2^1 = 1 \qquad inv_3^1 = 2 \qquad inv_4^1 = 0 \qquad inv_5^1 = 0$

Figure 4-5: Assessing the merits of alternative scheduling decisions in a search state where O_2^1 has been scheduled to start at $st_2^1 = 4$.

of O_4^1 : in the best case there would be a gap of 1 time unit between O_2^1 and O_4^1. In other words, the apparent marginal tardiness cost at operation O_3^1 has changed from 20 to 23 ! For the same reason, the apparent marginal tardiness costs at O_4^1 and O_5^1 are 23 also. The best remaining possible start times for these two operations are respectively 10 and 12. For instance, the minimum overhead cost for scheduling O_5^1 at $st_5^1 = 18$ (instead of its best remaining start time $st_5^1 = 12$) is $mincost_5^1(18) = 6 \times 23 = 138$ (while it was 160 in the initial search state).

The *mincost* curve of O_1^1 has changed more dramatically. Whichever start time is assigned to O_1^1, this start time no longer affects the tardiness costs incurred by the job. While, in the initial search state, scheduling O_1^1 early was reducing the minimum tardiness costs incurred by the job, now it only increases the minimum inventory costs of that job (e.g. $mincost_1^1(0) = 4$, since $inv_1^1 = 2$).

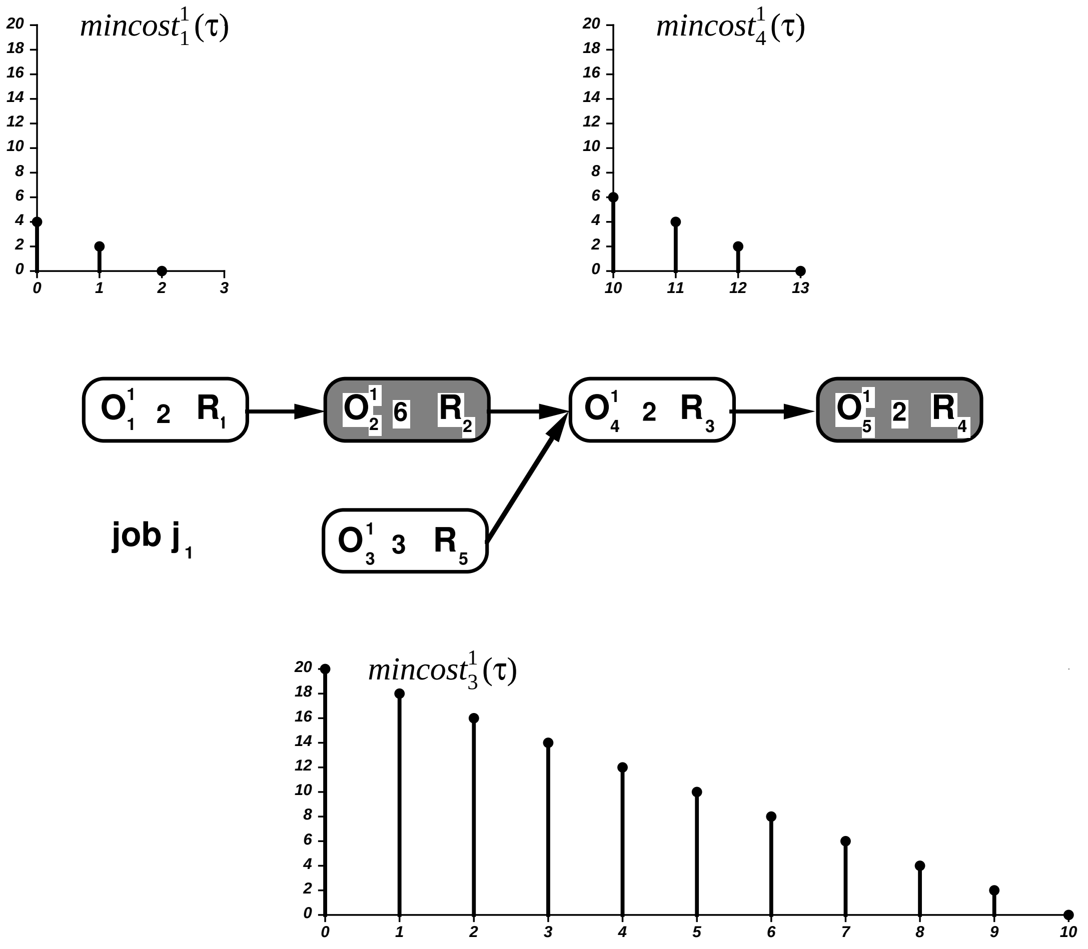
As the schedule is constructed, the merits of different possible start times can dramatically change. It is critical for the scheduler to constantly keep track of these changes in order to:

1. focus on the scheduling of operations that participate in important tradeoffs, and

2. identify promising reservations for these operations, namely reservations that provide a good compromise between the cost incurred by these critical operations and the other operations with which they compete.

Suppose that the system further schedules O_5^1 to start at $st_5^1 = 15$, thereby creating a new search state (Figure 4-6). Now the selection of a start time for O_3^1 or O_4^1 no longer affects the tardiness costs of the job. However that selection still affects the inventory costs introduced by operation O_3^1 via the $inv_3^1 \times [Max(C_1, dd_1) - st_3^1]$ term in Equation (4.2). For instance, starting O_4^1 at $st_4^1 = 10$ would force O_3^1 to start earlier, and would increase the inventory costs by $3 \times 2 = 6$ (compared to the best remaining start time of O_4^1, $st_4^1 = 13$). In other words, in this new search state, $mincost_4^1(10) = 6$. Similarly, the best remaining start time for operation O_3^1 is $st_3^1 = 10$. All earlier possible start times introduce additional inventory costs (e.g. $mincost_3^1(0) = 20$.). Notice that the selection of a start time for O_4^1 no longer affects the inventory costs introduced by O_1^1 and O_2^1.

Finally, if O_3^1 is scheduled to start at $st_3^1 = 7$, as displayed in Figure 4-7, all four remaining possible start times of operation O_4^1 become equally good (within the job), since they no longer have any impact on the tardiness costs or inventory costs incurred by the job.

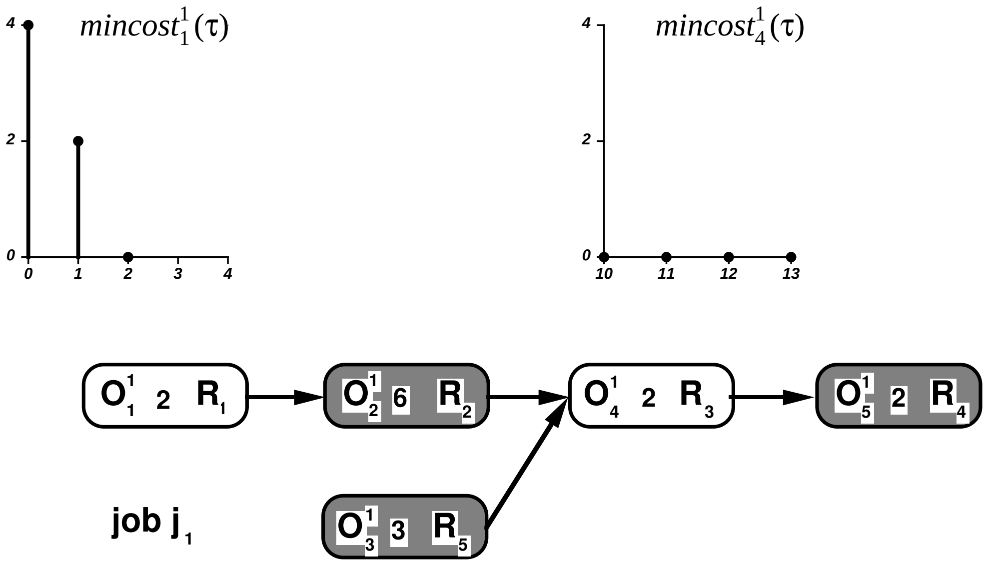


$erd_1 = 0$ $dd_1 = 12$ $lcd_1 = 20$ $tard_1 = 20$

$inv_1^1 = 2$ $inv_2^1 = 1$ $inv_3^1 = 2$ $inv_4^1 = 0$ $inv_5^1 = 0$

Figure 4-6: Assessing the merits of alternative scheduling decisions in a search state where O_2^1 has been scheduled to start at $st_2^1 = 4$ and O_5^1 at $st_5^1 = 15$.

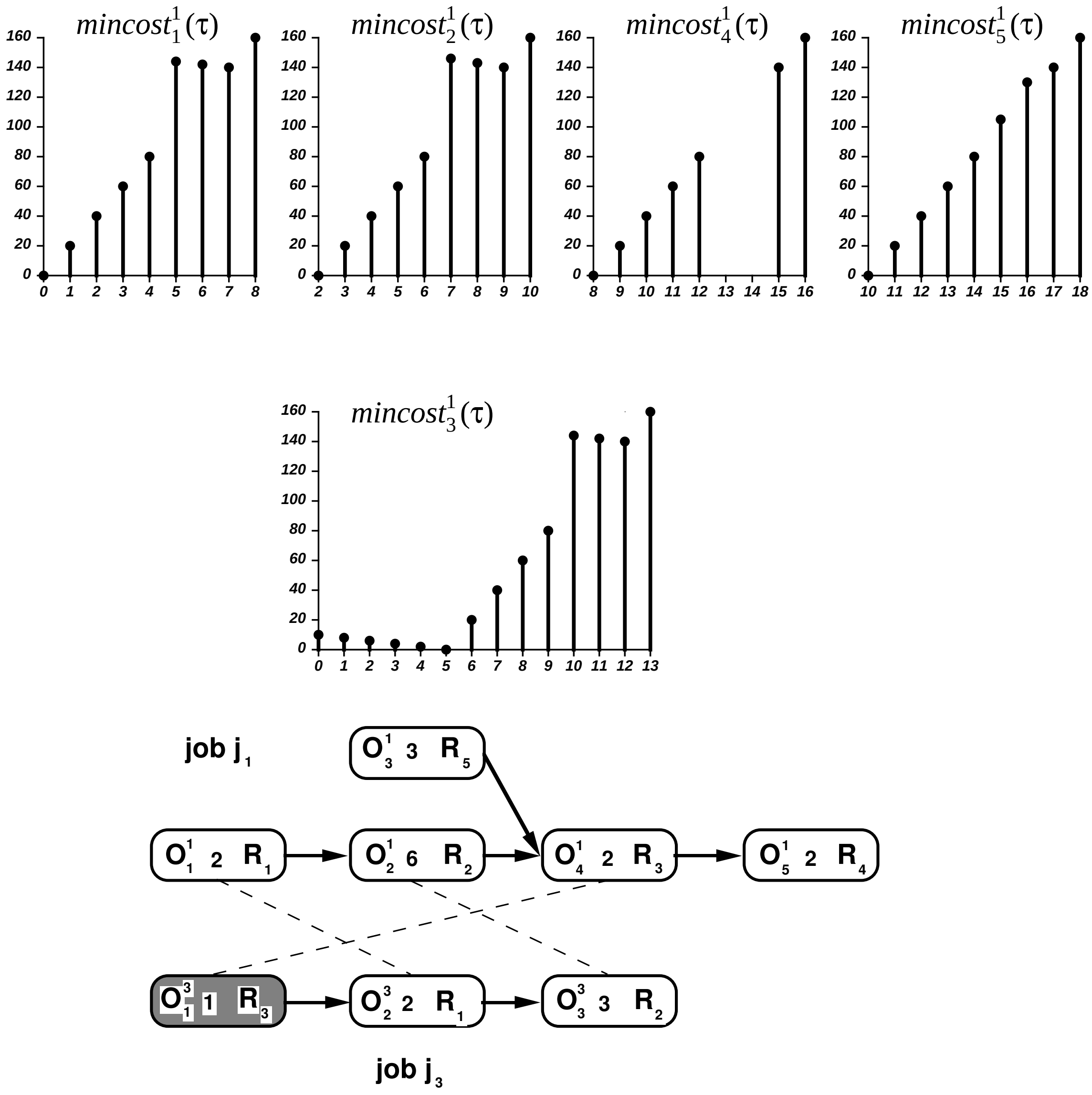
The examples presented so far only involved operations with a single contiguous interval of possible start times. In general, as operations in other jobs are scheduled, the possible start times left to an operation (as determined by the consistency checking mechanism described in Chapter 2) may form several disjoint intervals. When this happens, *mincost* functions can become more complex.



$erd_1 = 0$	$dd_1 = 12$	$lcd_1 = 20$	$tard_1 = 20$	
$inv_1^1 = 2$	$inv_2^1 = 1$	$inv_3^1 = 2$	$inv_4^1 = 0$	$inv_5^1 = 0$

Figure 4-7: Assessing the merits of alternative scheduling decisions in a search state where O_2^1 has been scheduled to start at $st_2^1 = 4$, O_5^1 at $st_5^1 = 15$, and O_3^1 at $st_3^1 = 7$.

Consider, for instance, a search state in which all operations are still unscheduled, except operation O_1^3 (in job j_3), which has been scheduled on resource R_3 between time 14 and 15 (i.e. $st_1^3 = 14$). Figure 4-8 displays the *mincost* functions of the operations in job j_1. Due to the allocation of resource R_3 to O_1^3 (in job j_3) between 14 and 15, O_4^1 (in job j_1) can no longer start at $st_4^1 = 13$ or $st_4^1 = 14$. The set of possible start times of this operation now consists of two disjoint time intervals. This in turn affects the shape of the *mincost* functions of other operations in j_1. While starting O_1^1 at $st_1^1 = 4$ would force job j_1 to be late by at least 4 time units (i.e. $mincost_1^1(4) = 80$), starting this same operation 1 time unit later, namely at $st_1^1 = 5$, would force the job to be late by at least 7 time units. This is because the earliest compatible start time for O_4^1 would then be $st_4^1 = 15$. Consequently,



$erd_1 = 0$ $dd_1 = 12$ $lcd_1 = 20$ $tard_1 = 20$

$inv_1^1 = 2$ $inv_2^1 = 1$ $inv_3^1 = 2$ $inv_4^1 = 0$ $inv_5^1 = 0$

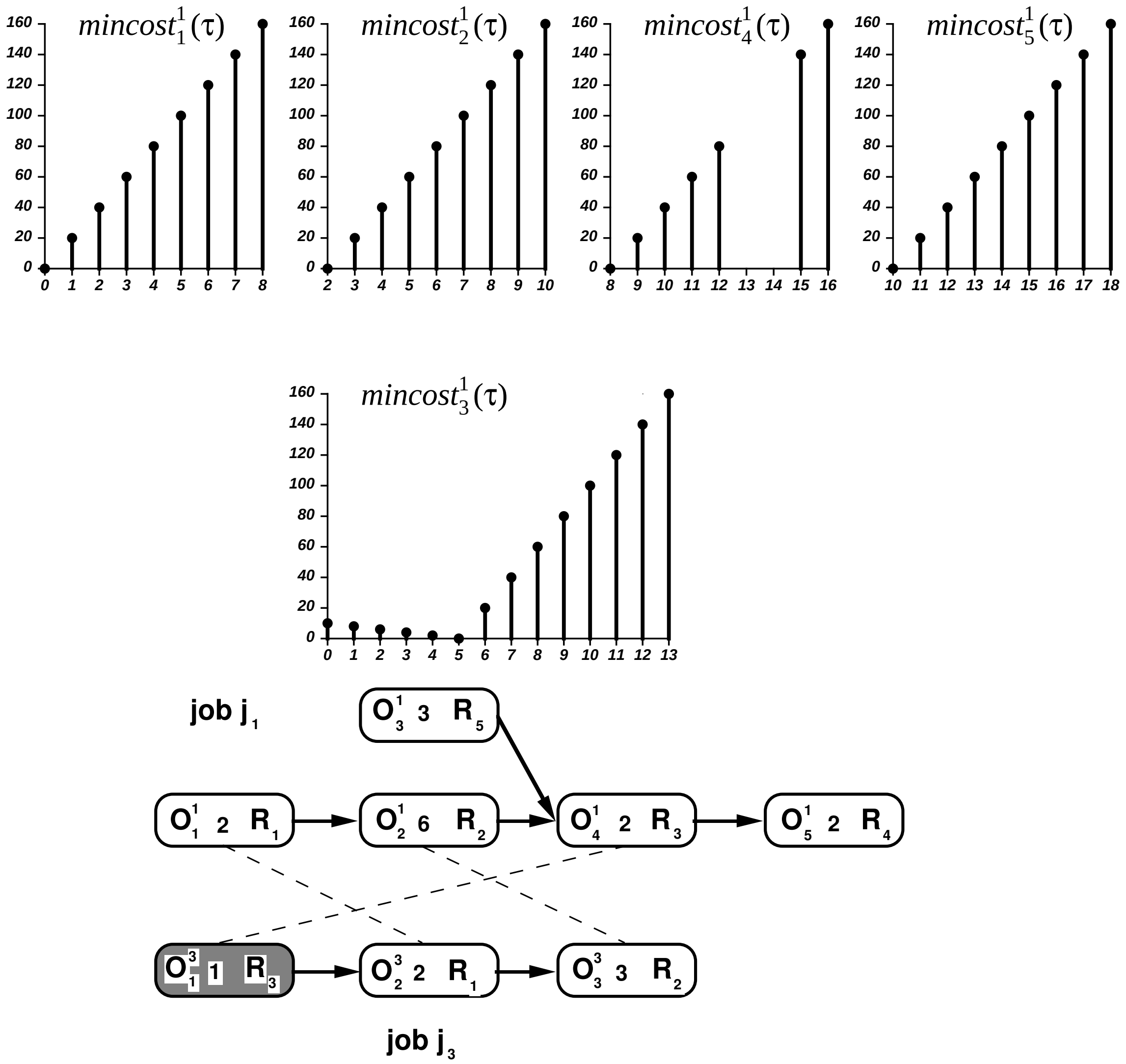
Figure 4-8: Assessing the merits of alternative scheduling decisions in job j_1, given a search state where O_1^3 has been scheduled to start at $st_1^3 = 14$.

starting O_1^1 at $st_1^1 = 5$ would result in an overhead tardiness cost of at least 140 (compared to the best start time $st_1^1 = 0$). Additionally, job j_1 would also incur an extra inventory cost of at least $inv_1^1 \times 2 = 4$. The total overhead cost for scheduling O_1^1 at $st_1^1 = 5$ is $mincost_1^1(5) = 140 + 4 = 144$. Similarly, $mincost_1^1(6) = 142$ and $mincost_1^1(7) = 140$. Similar computations can be performed to see that $mincost_2^1(7) = 146$, $mincost_2^1(8) = 143$, $mincost_5^1(15) = 105$, $mincost_5^1(16) = 130$, etc.

For the sake of efficiency, the current implementation of MICRO-BOSS computes *mincost* functions *as if* the set of possible start times of each unscheduled operation consisted of a single contiguous time interval. Under this simplifying assumption, it is not necessary to explicitly maintain *mincost* values for each possible start time of each unscheduled operation. Instead, for each unscheduled operation O_i^k, MICRO-BOSS maintains an apparent marginal tardiness cost, $ap\text{–}tard_i^k$, an apparent marginal inventory cost, $ap\text{–}inv_i^k$, and keeps track of the best possible start time(s) that are currently available to that operation. *The apparent marginal tardiness cost, $ap\text{–}tard_i^k$ (apparent marginal inventory cost, $ap\text{–}inv_i^k$), of operation O_i^k, in a given search state, is the minimum cost increase incurred by job j_k for every unit of time that O_i^k is scheduled past (before) its best possible start time(s), under the simplifying assumption of single contiguous intervals of possible start times*.

Notice that accounting for disjoint intervals of possible start times can only increase the value of the *mincost* functions, either because a job will be forced to finish later than anticipated or because it will incur larger inventory costs. Assuming contiguous intervals of possible start times in the computation of the *mincost* functions is an optimistic simplification. Figure 4-9 displays the simplified *mincost* functions corresponding to the search state studied in Figure 4-8.

When moving from one search state to the other, apparent marginal inventory costs and apparent marginal tardiness costs only change in the job in which an operation was just scheduled. Best remaining start times of unscheduled operations may however change both within that job and in other jobs. Updating apparent costs in the job in which an operation was just scheduled can be done in $O(n_l)$ time, where n_l is the number of operations in that job. Altogether, updating apparent marginal costs within the job where an operation was just scheduled, and the best remaining start times of all unscheduled operations, takes at most $O(n)$ time, where n is the number of unscheduled operations.



$erd_1 = 0$ $dd_1 = 12$ $lcd_1 = 20$ $tard_1 = 20$

$inv_1^1 = 2$ $inv_2^1 = 1$ $inv_3^1 = 2$ $inv_4^1 = 0$ $inv_5^1 = 0$

Figure 4-9: Simplified mincost functions that do not account for the two disjoint intervals that make up the set of remaining possible start times of operation O_4^1.

Below an efficient procedure is outlined that updates the apparent marginal tardiness costs, apparent marginal inventory costs, and best remaining possible start times of operations within the same job as the operation that was last scheduled. This procedure is based on the following observations[44]:

- **Apparent Tardiness Costs:** In a job j_k, in which no operation has been scheduled, the apparent marginal tardiness cost $ap\text{–}tard_i^k$ of an operation O_i^k is equal to $tard_k$, the marginal tardiness cost of the job. For instance, in the initial state displayed in Figure 4-4, all the operations in job j_1 have an apparent marginal tardiness cost equal to $tard_1 = 20$. As soon as an operation downstream of an operation O_i^k is scheduled, the apparent marginal tardiness cost $ap\text{–}tard_i^k$ of that operation becomes zero (e.g. operation O_1^1 in Figure 4-5 and operations O_1^1, O_3^1, and O_4^1 in Figure 4-6). When an operation O_a^b still subject to apparent tardiness costs (i.e. an operation with no operation scheduled downstream of it) is scheduled, the marginal apparent inventory costs of that operation and all the operations upstream of it (namely $\sum_{k \in BEF_a^b} inv_k^b$) has to be added to the apparent marginal tardiness costs of all the operations in the same job that are still subject to apparent tardiness costs. For instance, in Figure 4-5 (i.e. after operation O_2^1 was scheduled), the apparent marginal costs of operations O_4^1, O_5^1 and O_3^1 became $inv_1^1 + inv_2^1 + tard_1 = 23$.

- **Apparent Inventory Costs:** In a job j_k in which no operation has been scheduled, the apparent marginal inventory cost $ap\text{–}inv_i^k$ of an operation O_i^k is equal to the sum of the marginal inventory costs introduced by that operation and all the operations upstream of it: $\sum_{k \in BEF_i^l} inv_k^l$ (see Equation (4.4)). When

---

[44]These observations suppose apparent marginal costs computed under the assumption of contiguous intervals of possible start times. They also suppose in-tree process routings, they assume that both tardiness and inventory costs are non-negative and linear, as defined by Equations (4.1) and (4.2). Finally they assume finished-goods inventory costs (see $Max(C_l,dd_l)$ in Equation (4.2)). Different assumptions would entail different updating procedures.

an operation O_j^k upstream of O_i^k is scheduled, the marginal inventory costs of O_j^k and of all the operations upstream of O_j^k have to be subtracted from the apparent marginal inventory cost of O_i^k. For instance, in Figure 4-6, the apparent marginal inventory cost of O_4^1 is $ap\text{–}inv_4^1 = inv_3^1 + inv_4^1 = 2 + 0 = 2$ (since operation O_2^1 has already been scheduled[45]). When operation O_3^1 is scheduled (see Figure 4-7), the apparent marginal inventory cost of O_4^1 becomes zero (i.e. $ap\text{–}inv_4^1 = 0$) because both of its predecessors have been scheduled, and O_4^1 itself does not introduce any inventory cost (i.e. $inv_4^1 = 0$).

- **Best Start Time(s)**: When both the apparent marginal tardiness and inventory costs of an operation are positive, that operation has a unique best start time. This best possible start time can be obtained by subtracting the duration of the operation itself and the durations of all downstream operations from the job due date (or the earliest possible completion date, if it is larger than the due date). For instance, in the initial search state depicted in Figure 4-4, the best possible start time of operation O_1^1 is $st_1^1 = dd_1 - du_5^1 - du_4^1 - du_2^1 - du_1^1 = 12 - 2 - 2 - 6 - 2 = 0$. Operations whose apparent marginal tardiness costs are zero while their apparent marginal inventory costs are positive also have a unique best start time. This best start time is the operation's latest possible start time (e.g. operation O_1^1 in Figure 4-5). When the apparent marginal inventory cost of an operation is zero while its apparent marginal tardiness cost is positive, all start times allowing the job to complete before its due date (or before its current earliest possible completion date, if this date is past the due date) are optimal (within the job). Finally, if both the apparent marginal tardiness and inventory costs of an operation are zero, all remaining possible start times of that operation are equally good (e.g. operation O_4^1 in Figure 4-7).

[45]In the initial state the apparent marginal inventory cost of O_4^1 is $ap\text{–}inv_4^1 = inv_1^1 + inv_2^1 + inv_3^1 + inv_4^1 = 2 + 1 + 2 + 0 = 5$. This does not appear in Figure 4-4, because, in that state, the earliest possible start time of O_4^1 is also its best possible start time.

Accordingly, when an operation O_i^l is scheduled in job j_l, the apparent marginal tardiness costs, apparent marginal inventory costs, and best possible start times of the operations that remain unscheduled within that job can be updated by successively running each of the following procedures[46]:

- **Upstream Propagation (starting from O_i^l)**: Change the apparent marginal tardiness costs of all upstream operations (i.e. operations upstream of O_i^l) to zero. For each operation O_j^l upstream of O_i^l, if O_j^l has a positive apparent marginal inventory cost, change its best remaining start time to its new latest possible start time. Otherwise all its remaining possible start times are equally good. These computations can be performed by successively visiting each operation upstream of O_i^l, starting from O_i^l.
- **Downstream Propagation**: If $ap\text{–}inv_i^l > 0$ (i.e. in the previous search state, just before O_i^l was scheduled), then subtract that apparent marginal inventory cost from the apparent marginal inventory cost of all the operations downstream of O_i^l.
- **Upstream Propagation (starting from $O_{n_l}^l$)**: An upstream propagation starting from the last operation in the job (i.e. $O_{n_l}^l$) takes place when either of the following two conditions are met:

  condition 1: If O_i^l still had a positive apparent marginal tardiness cost when it was scheduled, and if it was scheduled past its best possible start time(s), the earliest possible completion date of the job needs to be updated. This new earliest possible completion date is then used to update the best start time(s) of all the operations in the job that still have a positive apparent marginal tardiness cost. This is done using a CPM-type of propagation that starts from the last operation in the job, $O_{n_l}^l$, and updates the best possible start time(s) of that operation and of all operations upstream of it. This best start time propagation moves upstream along each branch (and sub-branch) in the process routing until a scheduled operation is reached in that branch (or sub-branch).

  condition 2: If O_i^l still had a positive apparent marginal inventory cost when it was scheduled, that apparent marginal inventory cost needs to be

---

[46]The exact order in which these procedures are applied does not matter.

> added to the apparent marginal tardiness costs of all operations in the job with a positive apparent marginal tardiness cost. These unscheduled operations are precisely those visited in the upstream propagation described in **condition 1**.

A similar procedure can be designed to update apparent marginal costs and sets of best remaining possible start times, when the scheduler backtracks.

4.3.3. Step 2: Building Biased Demand Profiles to Identify Highly Contended Resource/Time Intervals

Once it has identified the best remaining reservations (i.e. start times) of each unscheduled operation (within its job), and the marginal costs incurred by each job for not selecting one of these reservations, MICRO-BOSS looks for operations whose good reservations (within their jobs) conflict with the good reservations of other operations. The scheduler and focuses on optimizing these conflicts first.

A simple way to identify resource/time intervals that are highly contended for would be to assume that all operations will be simultaneously scheduled at their best start times. Resource/time intervals that would need to be reserved more than once would indicate the need for a tradeoff. In search states, where some operations have more than one best start time, this approach would not work. Because, in general, it is impossible to schedule all operations at one of their best start times, this approach may also fail to identify conflicts that will arise later, as some operations are scheduled at start times that appear to be less than optimal in the current search state. Last but not least, demand profiles obtained by simultaneously scheduling each operation at one of its best start times would not provide any information on the *reliance* of each operation on the availability of a given start time. In general, this reliance is a function of the number of best start times available to the operation, and the marginal costs incurred by the job for selecting a suboptimal start time. A conflict involving a small number of operations, all with high marginal costs, might turn out to be more critical than a conflict with a larger number of operations whose marginal costs are lower.

For these reasons, a probabilistic framework was adopted, similar to the one used for the job shop CSP. This time however, the scheduler does not assume uniform probability distributions. Instead, start time distributions are now biased towards those start times that are expected to produce better schedules. Each start time τ that remains possible for an unscheduled operation O_i^l is assigned a *subjective probability* $\sigma_i^l(\tau)$ to be selected for that operation. Possible start times with lower *mincost* values are simply assigned a larger probability, thereby reflecting our expectation that they will allow for the production of better schedules. Like in the previous chapter, using these start time distributions, MICRO-BOSS builds for each unscheduled operation O_i^l, an *individual demand profile* $D_i^l(t)$, that indicates the subjective probability that the operation will be requiring its resource as a function of time. By aggregating the individual demand profiles of all unscheduled operations requiring a given resource, R_k, the scheduler obtains an *aggregate demand profile,* $D_{R_k}^{aggr}(t)$, that indicates contention between unscheduled operations for that resource as a function of time. Equivalently, a stochastic mechanism can be assumed that completes the current partial schedule by randomly assigning a start time to each unscheduled operation O_i^l, according to its σ_i^l distribution. $D_{R_k}^{aggr}(t)$ is then the expected number of reservations that would be made for resource R_k at time t.

In the current implementation, the start time probability distribution of a typical unscheduled operation O_i^l is of the form:

$$\sigma_i^l(\tau) = N_i^l \times [1 - B \times \frac{mincost_i^l(\tau)}{Maxcost}] \tag{4.5}$$

where:

- N_i^l is a normalization factor that ensures that $\sum_\tau \sigma_i^l(\tau) = 1$, since O_i^l will be assigned exactly one start time ;

- *Maxcost* is the maximum value of the *mincost* functions taken over all the remaining possible start times of all remaining unscheduled operations;

- B is a system parameter that controls the degree to which start time distributions are biased towards good start times ($0 \le B < 1$). In particular, if $B = 0$, all possible start times are considered equally probable (uniform start time distributions like for the job shop CSP). Both in the example presented below and in the experiments reported in Chapter 5, the value of this parameter was set at $B = 0.9$.

Because the micro-opportunistic scheduler optimizes the most important conflicts first, the costs that remain to be optimized tend to decrease during the construction of the schedule. By updating the value of *Maxcost* in each search state and using it to measure cost differences in Equation (4.5), MICRO-BOSS constantly adjusts its sensitivity to the largest differences in costs that are still possible in the current state[47].

Given these start time probability distributions, the probability that an unscheduled operation O_i^l uses its resource at time t, which is referred to as the *individual demand* of O_i^l for R_i^l, is:

$$D_i^l(t) = \sum_{t - du_i^l < \tau \leq t} \sigma_i^l(\tau) \tag{4.6}$$

Notice that the normalization factor N_i^l ensures that the total demand of each unscheduled operation O_i^l is equal to the duration of that operation, du_i^l (i.e. $\sum_{\tau} D_i^l(\tau) = du_i^l$). This total demand has simply been spread over the different start times that remain available to that operation in the current search state, according to the distribution defined in Equation (4.5). Operations with a lot of good start times (i.e. operations that have a lot of start times with a low *mincost* value, compared to the value of *Maxcost* in the current search state) will have individual demand profiles that are almost uniformly spread over long time intervals. Their individual demands will be fairly low at any given moment in time. In other words, because these operations have a large number of good start times still available, they do not rely heavily on anyone of them. Instead, operations with a small number of good start times will have more compact individual demands. Their demand will be high over relatively short intervals, thereby reflecting a higher reliance on the availability of these intervals.

For each resource R_k, the total demand of all unscheduled operations for that resource is given by:

---

[47]When dealing with jobs that have a lot of slack (e.g. a job whose latest acceptable completion date is far beyond both its due date and its earliest possible completion date), it is important to only include meaningful cost differences in the computation of *Maxcost*. To this end, the current implementation uses a slightly modified definition of *Maxcost*, which is computed as the maximum value of the *mincost* functions for start times that allow their job to start after $dd_l - 5\sum_{i=1}^{n_l} du_i^l$ and complete by $dd_l + \sum_{i=1}^{n_l} du_i^l$. The probability assigned to earlier (later) start times is obtained by interpolating the start time distribution so that $est_i^l - 1$ ($lst_i^l + 1$) has a probability of zero.

$$D_{R_k}^{aggr}(t) = \sum_{R_i^l = R_k} D_i^l(t) \tag{4.7}$$

where the summation is carried over all unscheduled operations that need resource R_k.

Observe also that the consistency checking technique used by MICRO-BOSS ensures that all start times conflicting with reservations made earlier have been pruned from the set of remaining possible start times of all unscheduled operations (see Chapter 2). For this reason, the aggregate demands never overlap with reservations that have already been made.

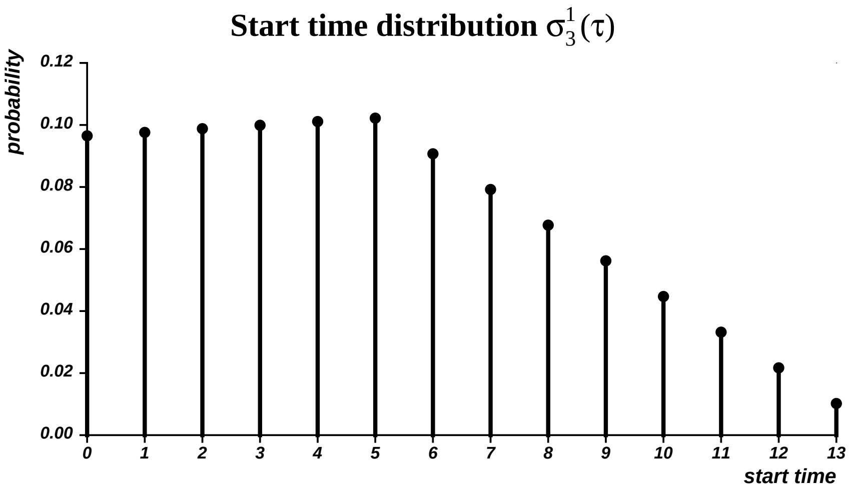


Figure 4-10: Start time distribution $\sigma_3^1(\tau)$ for operation O_3^1 in the initial search state depicted in Figure 4-4.

Figure 4-10 represents $\sigma_3^1(\tau)$, the start time distribution of operation O_3^1, in the initial search state depicted in Figure 4-4. From the *mincost* curves represented in Figure 4-4 and those of the operations in the three other jobs, it can be seen that the largest value taken by a *mincost* function in the initial state is 160, i.e. $Maxcost = 160$. In other words, the largest difference in cost between the worst and the best possible start time of any operation is never larger than 160 in the initial state. The best start time of O_3^1 is $st_3^1 = 5$, i.e. $mincost_3^1(5) = 0$. Hence, Equation (4.5) indicates that $\sigma_3^1(5) = N_3^1$. The next best start time for that operation is $st_3^1 = 4$ with $mincost_3^1(4) = 2$, hence $\sigma_3^1(4) = N_3^1 \times (1 - 0.9 \times \frac{4}{160}) = N_3^1 \times 0.94375$. Similar computations for the other possible start times of O_3^1 indicate that $N_3^1 = \frac{1}{9.78125}$. Consequently $\sigma_3^1(4) = 0.1011$, $\sigma_3^1(5) = 0.1022$, $\sigma_3^1(6) = 0.0907$, etc. Similar computations can be performed for the other unscheduled operations.

Based on these start time distributions, MICRO-BOSS builds individual demand profiles according to Equation (4.6). These demand profiles are in turn aggregated according to Equation (4.7). This process is illustrated in Figure 4-11 for resource R_2. There are four operations requiring resource R_2: O_2^1, O_2^2 O_3^3, and O_2^4. Figure 4-11 displays the individual demand profile of each of these four operations, and the aggregate demand for R_2 obtained by summing the four individual demands. As expected, the individual demands of operations O_3^3 and O_2^4 are quite uniform since these two operations have relatively low apparent marginal costs. In contrast, the individual demands of operations O_2^1 and O_2^2, which have larger apparent marginal costs, are more concentrated around the good reservations of these operations.

The aggregate demand for each of the five resources are shown in Figure 4-12. As anticipated, R_2 is the resource that is the most contended for. The aggregate demand for that resource is well above 1.0 over a large time interval, with a peak at 1.49. Resource R_1 appears to be a potential bottleneck at the beginning of the problem, with a demand peaking at 1.20. Whether R_1 will actually be an auxiliary bottleneck or not cannot be directly determined from the curves displayed in Figure 4-12. Instead, the scheduler needs to update these curves in each search state to account for earlier decisions. It could be the case that, as operations requiring R_2 are scheduled, the aggregate demand for R_1 becomes smoother. A trace provided at the end of this chapter indicates that this is not the case in this example. On the contrary, after only a fraction of the operations requiring resource R_2 have been scheduled, MICRO-BOSS will abandon scheduling operations on R_2 and temporarily shift to resource R_1.

Updating the start time distributions, the individual and aggregate demand profiles, requires at most $O(nk)$ time in each search state, where n is the number of unscheduled operations and k an upper-bound on the number of possible start times left to an unscheduled operation.

The idea of building biased demand profiles is not new. [Muscettola 87] describes a technique in which a random schedule generator is used to detect potential resource congestion, given a predefined scheduling strategy. The random generator is biased to

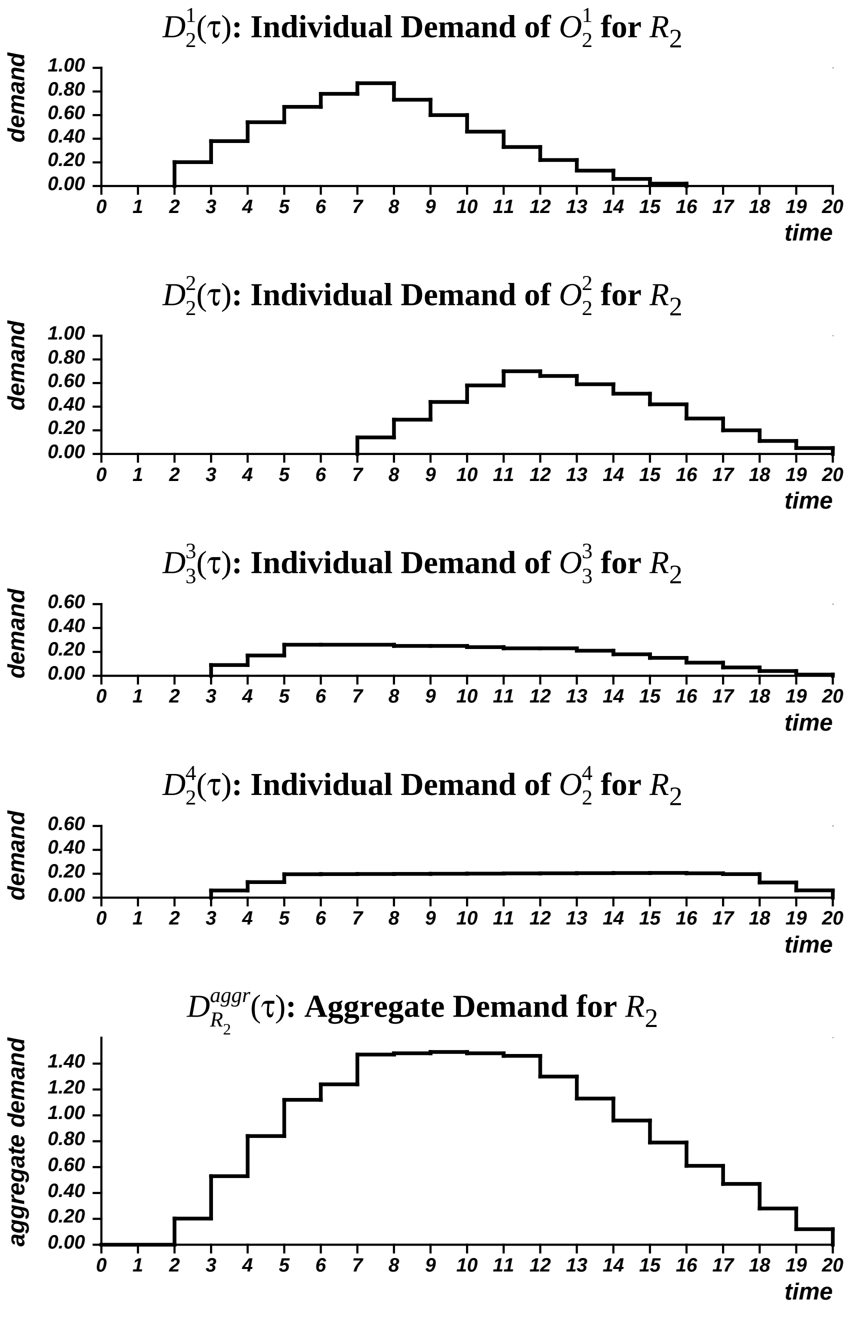


Figure 4-11: Building R_2's aggregate demand profile in the initial search state.

select more often those reservations that are expected to produce better schedules. Our technique is different, as it does not assume a predefined scheduling strategy, but instead uses the resulting demand profiles to dynamically revise the current strategy.

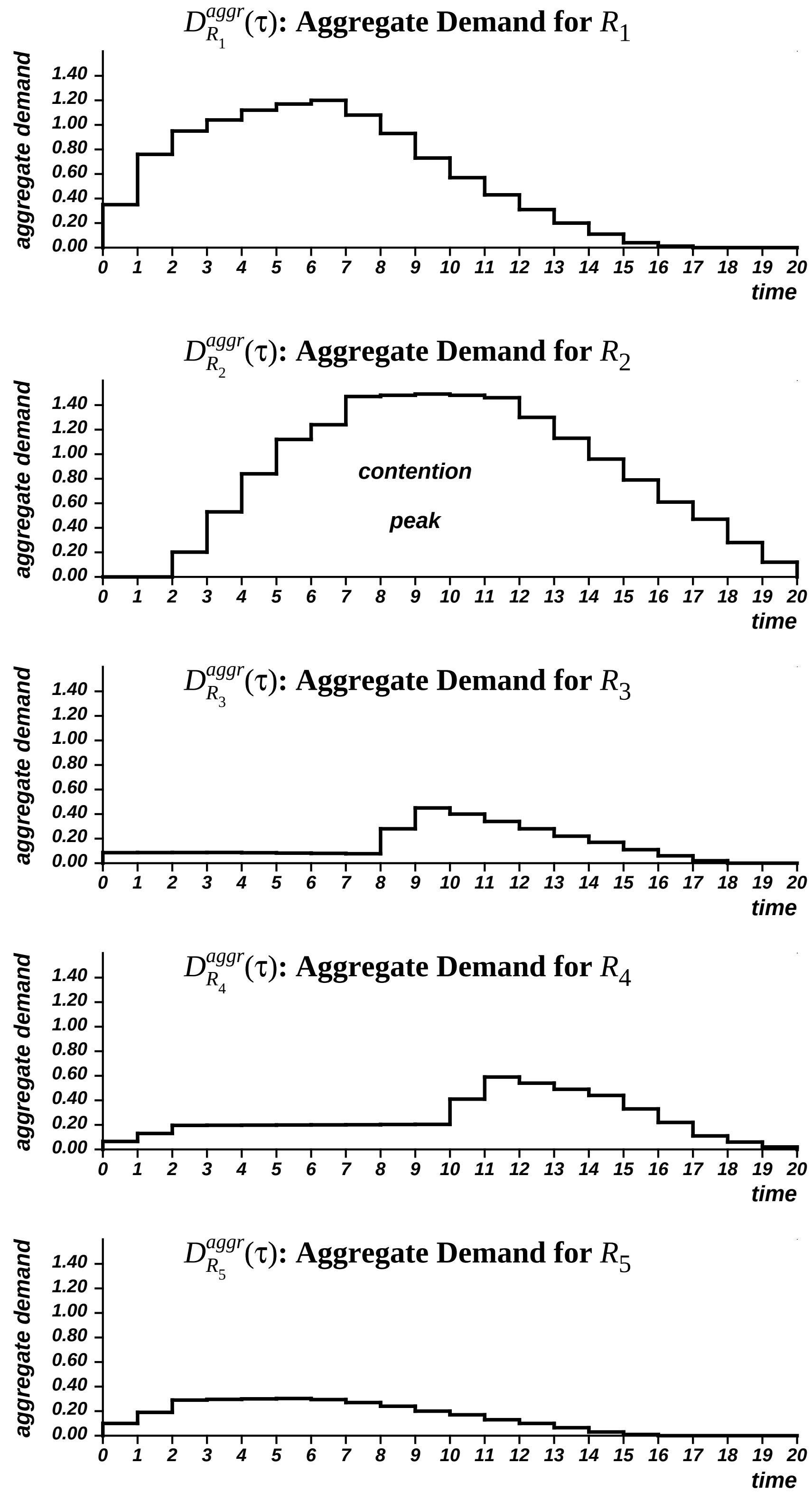


Figure 4-12: Aggregate demands in the initial search state for each of the five resources.

4.4. Operation Selection

In MICRO-BOSS, critical operations are identified as operations whose good reservations conflict with the good reservations of other operations. The largest peak in the aggregate demand profiles determines the next conflict (or micro-bottleneck) to be optimized, and the operation with the largest reliance on the availability of that peak is selected to be scheduled next. Intuitively the operation that relies most on the availability of the most contended resource/time interval is also the one whose good start times are the most likely to become unavailable if other operations contending for that resource/time interval were scheduled first.

To identify critical operations, MICRO-BOSS uses the exact same "Operation Resource Reliance" (ORR) heuristic introduced in Chapter 3 for the job shop CSP. MICRO-BOSS decomposes the demand profile of each resource into time intervals of length equal to the average duration of the operations requiring that resource. The time interval with the largest *average* aggregate demand is identified as the one for which contention is the highest, and the operation with the largest individual demand for that resource/time interval is selected to be scheduled next[48]

In the example introduced earlier, the most contended demand peak is that for resource R_2 over interval [7,12[. Figure 4-13 displays the aggregate demand for resource R_2 together with the individual demands of the four operations contributing to that demand. The operation with the largest contribution to the demand peak is O_2^1. Therefore, this operation is selected to be scheduled next. This is no real surprise: O_2^1 belongs to one of the two jobs in the problem that have a high marginal tardiness cost ($tard_1 = 20$). While any delay in starting job j_1 will cause j_1 to complete late, job j_2 (i.e. the other job with a high marginal tardiness cost) can tolerate a small amount of delay without ending up late.

[48]Multiple variations of of this heuristic have been tried, including variations using different interval lengths and different methods for selecting the critical operation, once the demand peak has been identified. The variation described here appears to be best both with respect to schedule quality and search efficiency.

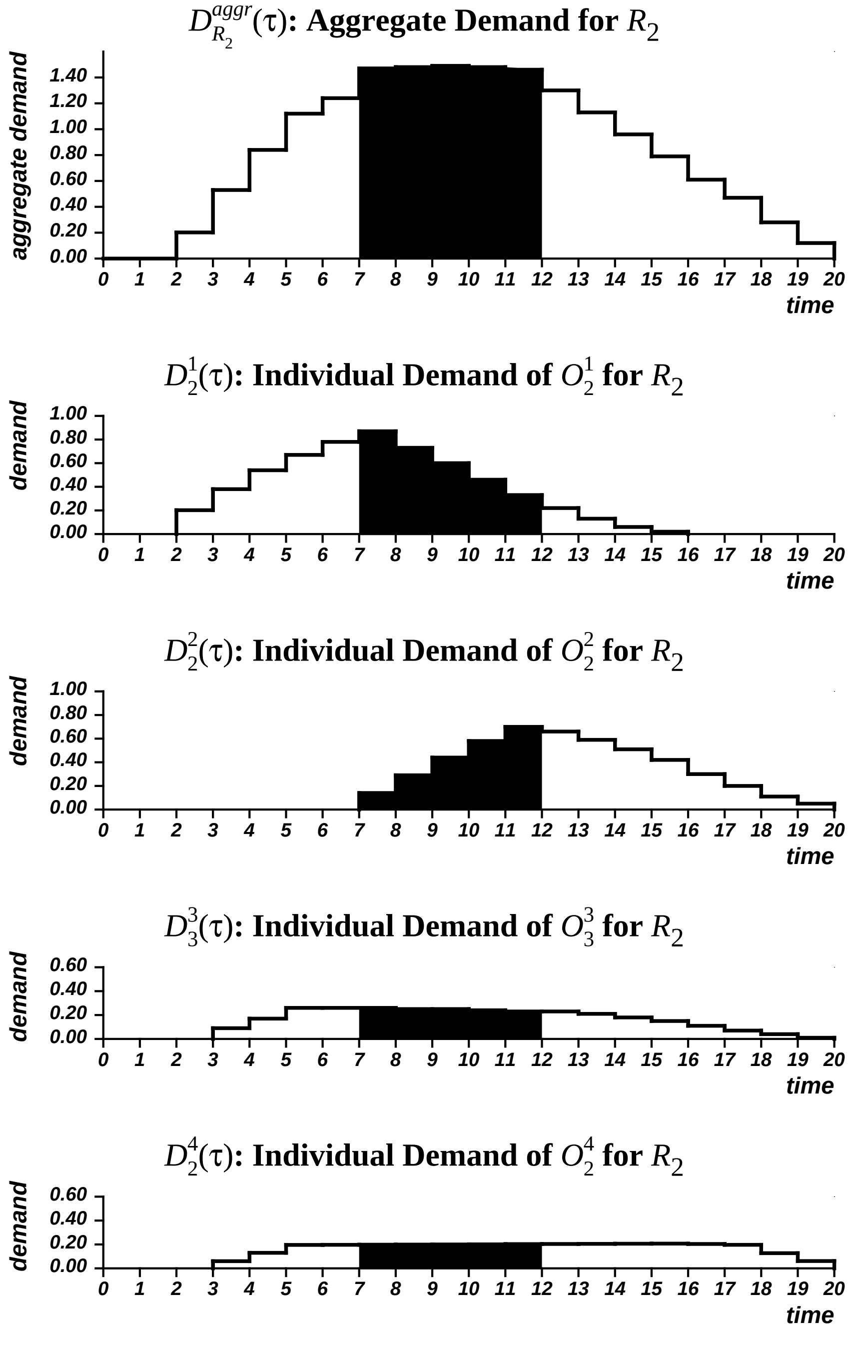


Figure 4-13: Operation selection in the initial search state.

4.5. Reservation Selection

Once it has selected an operation, MICRO-BOSS attempts to identify a reservation for that operation that will minimize as much as possible the costs incurred by the job to which that operation belongs and by other competing jobs. This is equivalent to solving a one-machine early/tardy problem in which operations scheduled past their best start times incur penalties determined by their apparent marginal tardiness costs, while operations scheduled before their best start times incur earliness penalties determined by their apparent marginal inventory costs.

The one-machine early/tardy problem is an NP-complete problem in itself [Garey 88]. Because of the presence of earliness costs, solving this problem to optimality requires, in general, the insertion of idle-time in the schedule. It can be shown that, given a fixed operation sequence, idle time can be optimally inserted in that sequence in $O(N \log N)$ time, where N is the number of operations to be scheduled on the resource [Garey 88]. Hence the problem is reduced to that of finding an optimal sequence, namely a sequence that will minimize the early/tardy costs, once idle time has been optimally inserted in it. Most of the procedures proposed so far to deal with this problem rely on branch-and-bound enumeration procedures [Baker 90]. [Fry 87] describes a dominance criterion that helps speed up this approach. Nevertheless the computational requirements of this procedure remain quite prohibitive even for small problems [Fry 87][49]. On the other hand, Ow and Morton have developed a family of much faster procedures that are pretty good at minimizing early/tardy costs under the simplifying assumption that no idle time will be inserted [Ow 89]. A delayed release policy derived from the priority dispatch version of the procedure has also been proposed that attempts to further reduce early costs by allowing for the insertion of idle time in front of each job [Morton 88].

For these reasons, MICRO-BOSS uses a hybrid reservation ordering heuristic that adapts to the amount of contention for the critical resource/time interval. When contention is high, MICRO-BOSS successively runs several variations of the early/tardy procedure developed by Ow and Morton, including variations with an immediate release policy and variations using the delayed release policy. The one-machine schedule that reduces most the costs of the operations competing for the critical resource determines the reservation that is assigned to the critical operation. When contention is lower,

[49]Close to 10 minutes of CPU time on a VAX 780 for a 15-operation problem.

MICRO-BOSS dynamically switches to a greedy reservation ordering heuristic, in which reservations are simply rated according to their apparent costs (i.e. according to their *mincost* values). Indeed, in situations where contention is not too high, a sizable proportion of the good start times of non critical operations tend to still be available after more critical operations have been scheduled. When this is the case, a greedy reservation ordering tends to produce high quality solutions. In particular, it inserts idle time as required by the operation it is scheduling.

The early/tardy procedure used in the current system is based on the simplest version of the procedure developed by Ow and Morton. It consists of a parametric dispatch rule, in which each operation O_i^l is assigned a priority $\pi_i^l(slack_i^l)$ given by:

$$\pi_i^l(slack_i^l) = \begin{cases} W_i^l & \text{if } slack_i^l \leq 0 \\ W_i^l - \dfrac{slack_i^l}{\kappa D_i^l}(W_i^l + H_i^l) & \text{if } 0 \leq slack_i^l \leq \kappa D_i^l \\ -H_i^l & \text{otherwise} \end{cases}$$

where:

- $W_i^l = \dfrac{ap\text{–}tard_i^l}{du_i^l}$
- $H_i^l = \dfrac{ap\text{–}inv_i^l}{du_i^l}$
- $slack_i^l$ is O_i^l 's slack measured with respect to the latest of O_i^l's best remaining end times (which plays the role of an apparent due date)
- κ is the look-ahead parameter of the priority rule
- $D_i^l = \sum_R \overline{du}_R$, where $\overline{du}_R$ is the average duration of the operations requiring resource R. The summation is over the critical resource and the resources required by the unscheduled operations downstream of O_i^l (i.e. the summation stops as soon as a scheduled operation downstream of O_i^l is encountered[50]).

---

[50]This differs from the early/tardy rule described in [Ow 89], which deals with a one-machine scheduling problem. Here, instead, the slack at operation O_i^l is shared between O_i^l and all the unscheduled operations downstream of it. This slack determines whether these operations can meet the job due date or complete without bumping into a downstream operation that has already been scheduled.

This priority rule can be viewed as a simple way to interpolate between two special cases of the early/tardy problem without inserted idle time [Ow 89]. The first case is one in which a Weighted Shortest Processing Time (WSPT) ordering of the operations competing for the critical resource produces a schedule in which each operation is scheduled past its apparent due date. Under this condition, a WSPT ordering of the operations is optimal. The second case is the converse of the first one. It corresponds to a situation in which a Weighted Longest Processing Time (WLPT) ordering produces a schedule in which each operation is scheduled before its apparent due date. In this case, the WLPT sequence is optimal (assuming that idle time cannot be inserted). The first case typically corresponds to situations in which all operations have little or no slack, whereas the second case corresponds to situations in which each operation has an ample amount of slack. Clearly, the amount of slack required to be in one situation or the other depends on the amount of contention between jobs, which is itself a function of the number of competing jobs and their due date distributions. Ow and Morton account for this by introducing a look-ahead parameter that is empirically adjusted to control the interpolation between the two extreme cases. Small values of the parameter are generally best for problems where contention is low, while larger values need to be used when contention is higher [Ow 89].

In the current implementation, two variations of the early/tardy rule are successively tried by MICRO-BOSS. The first variation is a straightforward implementation of the dispatch rule, while the second variation schedules the critical operation at the latest of its best start times and completes the one-machine schedule using the dispatch rule. The use of this second variation of the procedure was motivated by the fact that the costs of the critical operation are often more important than those of all the other operations with which it competes. As a consequence, just scheduling that operation at one of its best start times can be optimal. These two variations of the early/tardy rule are coupled to two release policies: an immediate release policy which releases each operation at its earliest possible start time, and a delayed release policy which releases each operation only after its priority has become positive [Morton 88]. All four combinations of the two variations of the early/tardy rule and the two release policies are successively tried by MICRO-BOSS. If during the construction of a one-machine schedule, an operation is scheduled past its latest possible start time, that schedule is immediately discarded. For each legal one-machine schedule, MICRO-BOSS remembers the start time assigned to the critical operation in that schedule and the total apparent cost of the schedule, as determined by the sum of the apparent costs of all operations competing for the critical resource (including the critical operation itself). At the end, the start times allocated to the critical operation in these one-machine schedules are ranked according to the cheapest schedule in which they participate (with the best start time corresponding to the cheapest schedule). Start times that are still possible for the critical operation but have not been selected in any of the one-machine schedules, are kept as secondary alternatives (in case the scheduler backtracks), and are ranked according to their local apparent costs (*mincost* values). The worst-case complexity of this reservation ordering heuristic is $O(max\{N^2, k \log k\})$, where N is the number of jobs competing for the critical resource and k the number of start times that remain possible for the critical operation[51].

Table 4-1 displays the values of the different parameters used by the early/tardy procedure to select a start time for operation O_2^1 in the example introduced earlier. ist_i^l represents the latest of the best remaining start times of O_i^l. For instance, for O_2^1, we have $ist_2^1 = 2$ (see Figure 4-4). We also have $W_2^1 = \frac{20}{6} = 3.33$, $H_2^1 = \frac{3}{6} = 0.5$, and $D_2^1 = \frac{17}{4} + \frac{3}{2} + \frac{5}{2} = 8.25$. With the immediate release policy, each operation is released at its earliest possible start time. With the delayed release policy, each operation O_i^l is released at $del\text{–}rel_i^l$, the earliest possible start time at which the priority π_i^l of that operation becomes positive:

$$del\text{–}rel_i^l = Max\{est_i^l, ist_i^l - \kappa D_i^l \frac{W_i^l}{W_i^l + H_i^l}\}$$

In this simple example, both release policies happen to produce the same result. O_2^1 is the only operation available to be scheduled at time 2. Consequently, all the one-machine schedules produced by the early/tardy procedure suggest to schedule this operation at time 2.

---

[51] $O(N^2)$ corresponds to the complexity of the dispatch rule, and $O(k \log k)$ is the time required to sort the possible start times of the critical operation. When MICRO-BOSS switches to its greedy reservation ordering heuristic, the complexity of the reservation ordering heuristic simply becomes $O(k \log k)$. In practice, given that, on the average, there is very little backtracking, it is more efficient to just look for the best start time. Later, if MICRO-BOSS needs to backtrack, it looks for the next best start time, and so on. In the absence of backtracking, this requires only $O(k)$ steps, instead of $O(k \log k)$.

Reservation Selection for O_2^1								
O_i^l	est_i^l	lst_i^l	ist_i^l	du_i^l	W_i^l	H_i^l	D_i^l	$del\text{–}rel_i^l$ (κ=3)
O_2^1	2	10	2	6	3.33	0.5	8.25	2
O_2^2	7	15	9	5	4	1	4.25	7
O_3^3	3	17	6	3	1.66	0.33	4.25	3
O_2^4	3	17	15	3	3.33	0.33	4.25	4

Table 4-1: Reservation Selection.

4.6. A Small Example

The current version of MICRO-BOSS has been implemented in Knowledge Craft, a frame-based language that runs on top of Common Lisp. The program runs on a DECstation 3100 under Mach UNIX. The small example used throughout this chapter requires a little over 2 seconds of CPU time in the current implementation. An edited trace of that example appears in Figure 4-14.

Observe that, rather than entirely scheduling the main bottleneck resource, namely resource R_2, MICRO-BOSS shifted to resource R_1 only after two out of the four operations requiring R_2 had been scheduled. The average expected demand displayed in each search state is the average demand for the critical demand peak, and the average contribution is that of the critical operation for the demand over that peak. The decoupling effect of the operation ordering heuristic is very clear in this example. In particular, the average demand over the critical peak consistently decreases from one search state to the next, thereby indicating a regular decrease in contention as the schedule is constructed (remember that the demand peak corresponds to the interval of highest contention in the current search state). This observation is correlated by the average contribution of the critical operation to the demand for the peak in each search state. As the schedule is constructed, the contribution of the critical operation to the peak becomes a larger proportion of the total demand for that peak. This indicates that there are fewer and fewer operations contending with each other. After half of the operations have been scheduled (depth 7), contention has totally disappeared: the critical operation is the only one to contribute to the demand for the peak. In other words, the problem has

MON NOV 12 1990 --- 17:04:49 EST

>> Depth: 0, Number of states visited: 0
Critical demand peak:
R_2 **between 7 and 12, Avg. expected demand: 1.48**
Critical Operation: O_2^1**, Avg. contrib.: 0.60**
Using early/tardy reservation ordering heuristic:
O_2^1 **scheduled between 2 and 8 on** R_2

>> Depth: 1, Number of states visited: 1
Critical demand peak:
R_2 **between 10 and 15, Avg. expected demand: 1.33**
Critical Operation: O_2^2**, Avg. contrib.: 0.64**
Using early/tardy reservation ordering heuristic:
O_2^2 **scheduled between 9 and 14 on** R_2

>> Depth: 2, Number of states visited: 2
Critical demand peak:
R_1 **between 0 and 4, Avg. expected demand: 1.35**
Critical Operation: O_1^2**, Avg. contrib.: 0.75**
Using early/tardy reservation ordering heuristic:
O_1^2 **scheduled between 2 and 9 on** R_1

>> Depth: 3, Number of states visited: 3
Critical demand peak:
R_2 **between 14 and 19, Avg. expected demand: 1.13**
Critical Operation: O_3^3**, Avg. contrib.: 0.58**
Using early/tardy reservation ordering heuristic:
O_3^3 **scheduled between 17 and 20 on** R_2

>> Depth: 4, Number of states visited: 4
Critical demand peak:
R_2 **between 14 and 19, Avg. expected demand: 0.60**
Critical Operation: O_2^4**, Avg. contrib.: 0.60**
Using greedy reservation ordering heuristic:
O_2^4 **scheduled between 14 and 17 on** R_2

Figure 4-14: An edited trace

```
>> Depth: 5, Number of states visited: 5
   Critical demand peak:
   R4 between 10 and 13, Avg. expected demand: 0.57
   Critical Operation: O5^1, Avg. contrib.: 0.34
   Using greedy reservation ordering heuristic:
   O5^1 scheduled between 10 and 12 on R4

>> Depth: 6, Number of states visited: 6
   Critical demand peak:
   R3 between 8 and 10, Avg. expected demand: 1.08
   Critical Operation: O4^1, Avg. contrib.: 1.0
   Using greedy reservation ordering heuristic:
   O4^1 scheduled between 8 and 10 on R3

>> Depth: 7, Number of states visited: 7
   Critical demand peak:
   R5 between 4 and 7, Avg. expected demand: 0.55
   Critical Operation: O3^1, Avg. contrib.: 0.55
   Using greedy reservation ordering heuristic:
   O3^1 scheduled between 5 and 8 on R5

>> Depth: 8, Number of states visited: 8
   Critical demand peak:
   R1 between 0 and 4, Avg. expected demand: 0.50
   Critical Operation: O1^1, Avg. contrib.: 0.50
   Using greedy reservation ordering heuristic:
   O1^1 scheduled between 0 and 2 on R1

>> Depth: 9, Number of states visited: 9
   Critical demand peak:
   R4 between 5 and 8, Avg. expected demand: 0.44
   Critical Operation: O1^4, Avg. contrib.: 0.44
   Using greedy reservation ordering heuristic:
   O1^4 scheduled between 7 and 10 on R4
```

Figure 4-14, continued

>> Depth: 10, Number of states visited: 10
Critical demand peak:
R_1 between 12 and 16, Avg. expected demand: 0.31
Critical Operation: O_2^3, Avg. contrib.: 0.31
Using greedy reservation ordering heuristic:
O_2^3 scheduled between 15 and 17 on R_1

>> Depth: 11, Number of states visited: 11
Critical demand peak:
R_3 between 13 and 15, Avg. expected demand: 0.14
Critical Operation: O_1^3, Avg. contrib.: 0.14
Using greedy reservation ordering heuristic:
O_1^3 scheduled between 14 and 15 on R_3

>> Depth: 12, Number of states visited: 12
Schedule Completed

Figure 4-14, concluded

been totally decoupled. The resource requirements of the operations that still need to be scheduled no longer interact with each other. This phenomenon is not specific to this example, but can be observed in all the problems that we have run. This suggests that the ORR operation ordering heuristic is indeed very good at constantly redirecting search towards the most important conflicts.

Notice also that no backtracking was necessary to produce the schedule. The schedule produced by MICRO-BOSS is displayed in Figure 4-15.

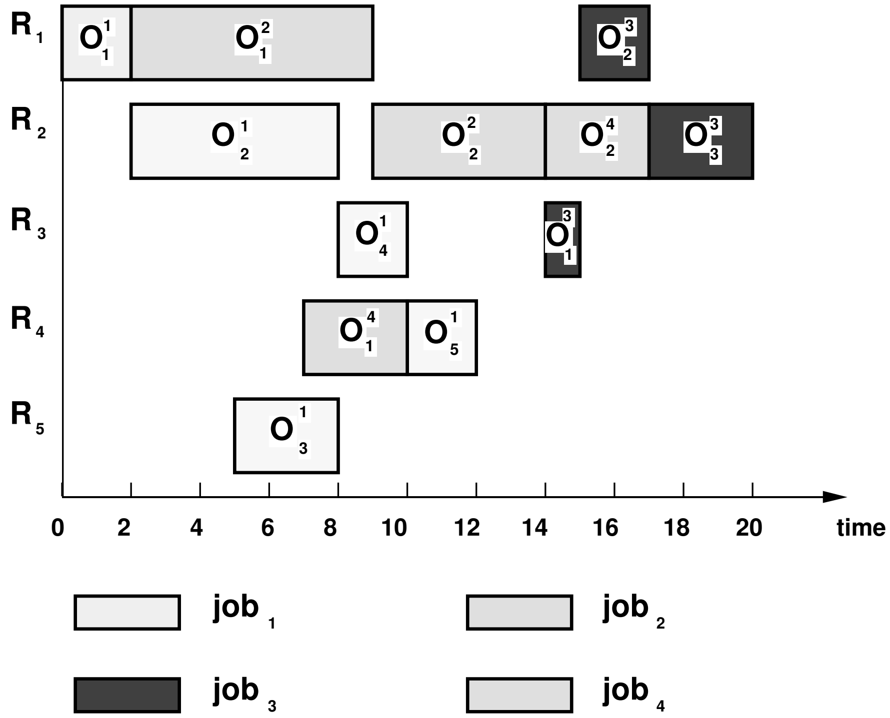


Figure 4-15: The final schedule produced by the micro-opportunistic scheduler.

The cost of this schedule is 180. In comparison, the Earliest Due Date priority rule produces a schedule with a cost of 208, the Weighted Shortest Processing Time rule and the Weighted COVERT priority rule [Vepsalainen 87]produce a schedule with a cost of 255, and the Slack per Remaining Processing Time rule produces a schedule with a cost of 300. In this example the savings allowed by the micro-opportunistic approach are the results of reductions both in tardiness and inventory costs. Extensive experimental studies of the micro-opportunistic approach to scheduling are presented in the next chapter.

Chapter 5
Performance Evaluation of MICRO-BOSS

This chapter reports several experimental studies that were carried out in order to evaluate the performance of MICRO-BOSS. Section 5.1 describes how scheduling problems were randomly generated to cover a wide range of scheduling conditions. It also describes the metrics used to evaluate performance. Section 5.2 describes a study comparing MICRO-BOSS with six priority dispatch rules. Section 5.3 compares MICRO-BOSS with a macro-opportunistic scheduler that dynamically switches between a resource-centered perspective and a job-centered perspective. The next section analyzes the impact of using biased demand profiles to guide the micro-opportunistic scheduler. Section 5.4 attempts to answer the question of how much flexibility/opportunism is really required to produce good schedules. Finally Section 5.5 studies the sensitivity of MICRO-BOSS to changes in the cost function.

5.1. Design of the Test Data

A set of 80 scheduling problems was randomly generated to cover a wide range of scheduling conditions. Scheduling conditions were varied by adjusting a set of three parameters[52]: a parameter controlling the average due date of the jobs to be scheduled (**tardy factor**), a parameter controlling the variance of the job due dates (**due date range**), and a parameter controlling the **number of major bottleneck machines**.

These three scheduling parameters were set as follows:

- **Tardy Factor** (τ): this factor controlled the average tightness of job due dates in the experiments. Given an estimate of the expected makespan of the problem, say M, the average job due date was set to $(1-\tau)M$, where τ is the

[52]These parameters or similar ones are commonly used in the scheduling literature to study different shop floor situations [Srinivasan 71, Fisher 76, Ow 85, Morton 88].

tardy factor[53]. If τ=0 it might be possible to complete all jobs on time. As τ increases, the proportion of jobs that can still be completed on time decreases. In the scheduling problems that were generated, two values of the tardy factor were used: $\tau = 0.2$ (loose average due date) and $\tau = 0.4$ (tight average due date).

- **Due Date Range** (R): job due dates were randomly drawn from a uniform probability distribution $(1-\tau)M\,U(1-R/2\,,1+R/2)$, where U(a,b) represents a uniform distribution between a and b, and R is the due date range. Two due date ranges were used: 1.2 (wide due date range) and 0.6 (tight due date range).

- **Number of Bottlenecks**: All problems involved five resources. In half of the problems, one out of the five resources was selected to be a major bottleneck, while the other half of the problems had two major bottlenecks.

A set of 10 scheduling problems was randomly generated for each parameter combination (see Table 5-1), resulting in a total of 80 scheduling problems (10 problems x 2 tardy factors x 2 due date ranges x 2 bottleneck configurations). All experiments involved 20 jobs and 5 resources, for a total of 100 operations. Each job had a linear process routing, and had to go through each machine exactly once. The order in which a job would go through the machines was randomly generated for each job, except for the bottleneck machines, which were always visited after a fixed number of operations (in order to further increase resource contention). In the case with one bottleneck resource, the bottleneck operation corresponded to the 4th operation (out of 5); in the case with two bottlenecks, the bottlenecks corresponded to the 2d and 5th operations of each job.

Job sizes (S_l): The size S_l of job j_l (i.e. the number of parts to be produced) was randomly drawn from a uniform distribution $U(1,7)$

Operation durations (du_i^l): Operation durations were correlated with job sizes. The duration of an operation O_i^l on a non-bottleneck resource was randomly drawn from a uniform distribution $S_l\,U(0.5\,,1.5)$. In problems with

---

[53] $M = (n-1)\,\overline{du}_{R_{btnk}} + \sum_{R=R_1}^{R_m} \overline{du}_R$, where n is the number of jobs, m the number of resources, R_{btnk} the main bottleneck resource and $\overline{du}_{R_i}$ denotes the average duration of the operations requiring resource R_i. This estimate was first proposed in [Ow 85].

Problem Sets			
Number of Bottlenecks	τ	R	Problem Set
1	0.2	1.2	**1**
1	0.2	0.6	**2**
1	0.4	1.2	**3**
1	0.4	0.6	**4**
2	0.2	1.2	**5**
2	0.2	0.6	**6**
2	0.4	1.2	**7**
2	0.4	0.6	**8**

Table 5-1: Parameter settings for each of the eight problem sets.

a single bottleneck, the duration of an operation O_i^l requiring that bottleneck was set to $du_i^l = 2S_l$. In problems with two bottlenecks, the duration of the operation requiring the first bottleneck was set to $1.8S_l$ and that of the operation requiring the second bottleneck was set to $2S_l$.

Tardiness costs ($tard_l$): the marginal tardiness cost of job j_l, $tard_l$, was randomly drawn from a uniform distribution $5U(1, 2S_l)$.

Inventory costs (inv_i^l): Inventory costs were only introduced by the first operation in each job, namely operation O_1^l. inv_1^l can be interpreted as the sum of the marginal holding costs and the interests on raw material costs for job j_l. These costs are typically proportional to the size of the job. In these experiments, inv_1^l was simply set to S_l. Given that the mean of the marginal tardiness cost distribution is slightly larger than $5S_l$, this corresponds to a ratio of marginal tardiness cost over marginal inventory cost with a mean slightly larger than 5.

Earliest Acceptable Release Date/Latest Acceptable Completion Date (erd_l, lcd_l): In order to increase contention, all jobs were given the same earliest acceptable release date, $erd_l = 0$. Since these experiments were designed to study the quality of the schedules produced by the micro-opportunistic approach, all jobs were given a non-constraining latest acceptable completion date[54], which was set to $lcd_l = 2M$ (where M is the makespan estimate described earlier). Studies of problems with more constraining latest acceptable completion dates can be found in [Sadeh 90].

The objective in all these experiments was to minimize total schedule cost, as defined in Section 4.2. Additionally, performance with respect to the following criteria was also measured:

- **Average weighted tardiness**:

$$\frac{\sum_l tard_l \times Max(0, C_l - dd_l)}{\sum_l tard_l}$$

- **Average weighted flowtime**:

$$\frac{\sum_l inv_1^l \times (C_l - st_1^l)}{\sum_l inv_1^l}$$

  This measure corresponds to the average in-process inventory of the schedule (or work-in-process).

- **Average weighted in-system time**:

$$\frac{\sum_l inv_1^l \times [Max(dd_l, C_l) - st_1^l]}{\sum_l inv_1^l}$$

  This measure accounts for both job flowtime (i.e. in-process inventory) and job earliness (i.e. finished-goods inventory). It corresponds to the average weighted time spent by a job in the system (i.e. it is a measure of the total inventory in the system).

- **Search efficiency**: the number of operations to be scheduled (100) divided by the number of search states generated. Search efficiency is only reported for the opportunistic schedulers, since priority dispatch rules are one-pass procedures that never backtrack.

[54]Another reason for choosing non-constraining latest acceptable completion dates was that we wanted to compare the micro-opportunistic approach against some priority dispatch rules. These rules are unable to account for such constraints.

5.2. Comparison Against Six Priority Dispatch Rules

A first set of 560 experiments was performed to compare MICRO-BOSS against six priority dispatch rules. These rules included two variations of the early/tardy rule developed by Ow and Morton [Ow 89, Morton 88]: a linear variation, LIN-ET, similar to the rule used in the hybrid reservation ordering heuristic implemented in MICRO-BOSS, and a (more sophisticated) exponential version of the rule, referred to as EXP-ET. The other four priority dispatch rules selected for the study were the Weighted Shortest Processing Time (WSPT) rule, the Earliest Due Date (EDD) rule, the Slack per Remaining Processing Time (S/RPT) rule and the Weighted Cost OVER Time (WCOVERT) rule. These four rules have been reported to be particularly good at reducing tardiness under different scheduling conditions [Vepsalainen 87].

Both variations of the early/tardy rule were run together with release policies that allowed jobs to be released only after their priorities had become positive, as suggested in [Morton 88]. The other four dispatch rules were run in combination with the Average Queue Time release policy (AQT) described in [Morton 88]. AQT is a parametric release policy that estimates queuing time as a multiple of the average job duration. The release of a job is determined by offsetting its due date by the sum of the total job duration and the estimated queueing time.

This section successively presents the results of the comparison with the WSPT, EDD, WCOVERT and S/RPT dispatch rules and the results of the comparison with the LIN-ET and EXP-ET dispatch rules.

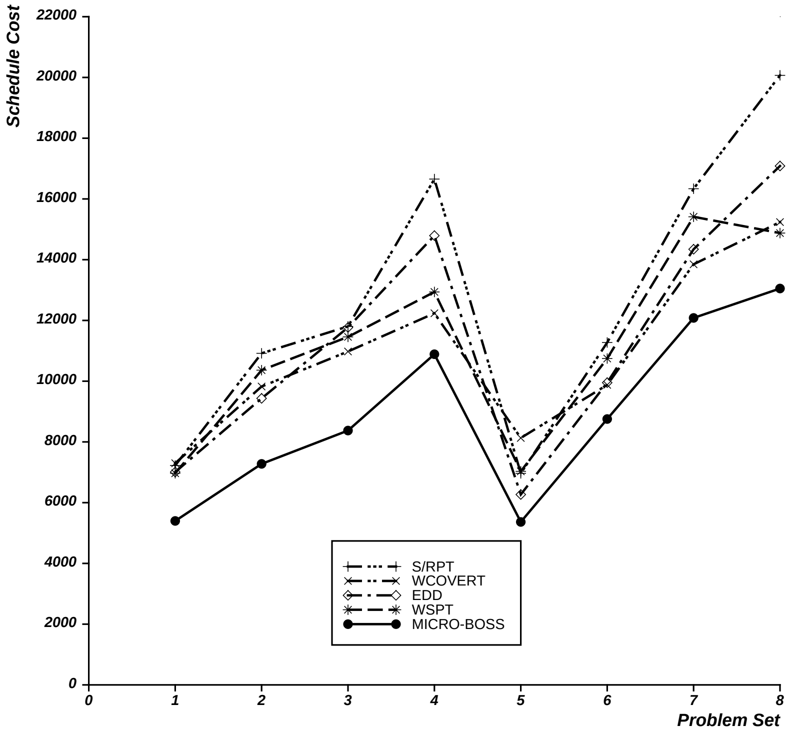


Figure 5-1: Comparison of the cost of the schedules produced by MICRO-BOSS and the WSPT, EDD, S/RPT and WCOVERT dispatch rules under 8 different scheduling conditions.

Figure 5-1 displays the average costs of the schedules produced by MICRO-BOSS and the WSPT, EDD, S/RPT and WCOVERT dispatch rules. Figures 5-2, 5-3 and 5-4 summarize performance with respect to weighted tardiness, weighted flowtime and weighted in-system time.

Remarkably enough, MICRO-BOSS consistently outperformed all four dispatch rules under all eight conditions of the study. Figures 5-2, 5-3 and 5-4 indicate that, while performing at a level comparable to the dispatch rules with respect to tardiness, MICRO-BOSS yielded significant reductions in inventory (between 15 and 50 percent depending on the scheduling situation). The most important reductions in inventory were observed on the most difficult problems, namely those with tight average due dates and narrow due date ranges. Overall, MICRO-BOSS reduced the average schedule cost by 18% compared to its closest competitor, WCOVERT.

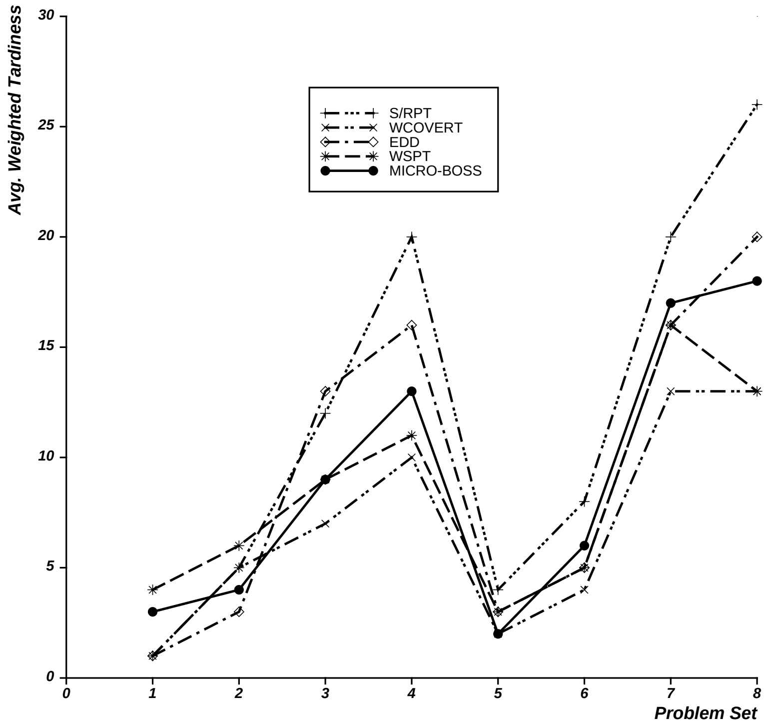


Figure 5-2: Weighted tardiness performance of MICRO-BOSS and the WSPT, EDD, S/RPT and WCOVERT dispatch rules under 8 different scheduling conditions.

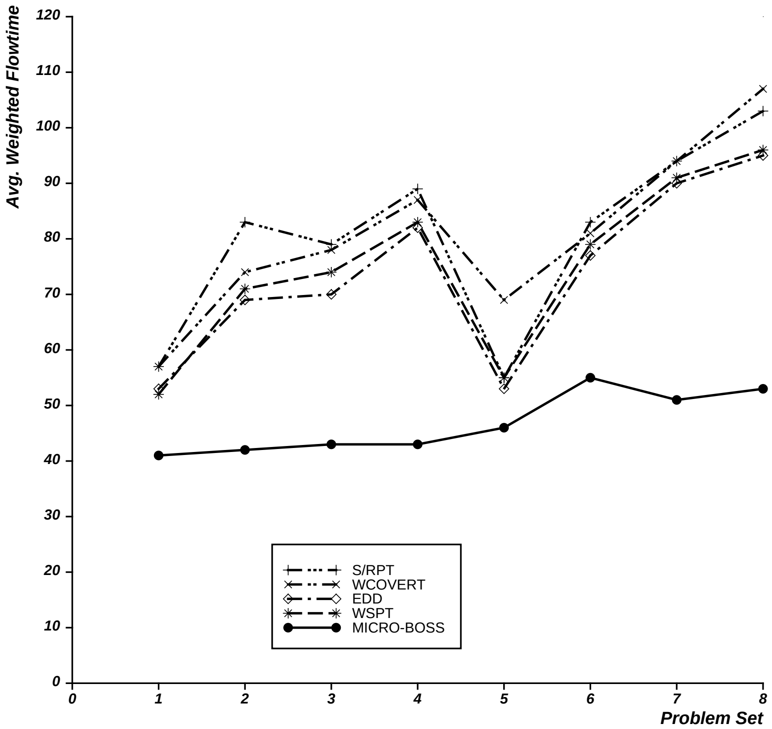


Figure 5-3: Weighted flowtime performance of MICRO-BOSS and the WSPT, EDD, S/RPT and WCOVERT dispatch rules under 8 different scheduling conditions.

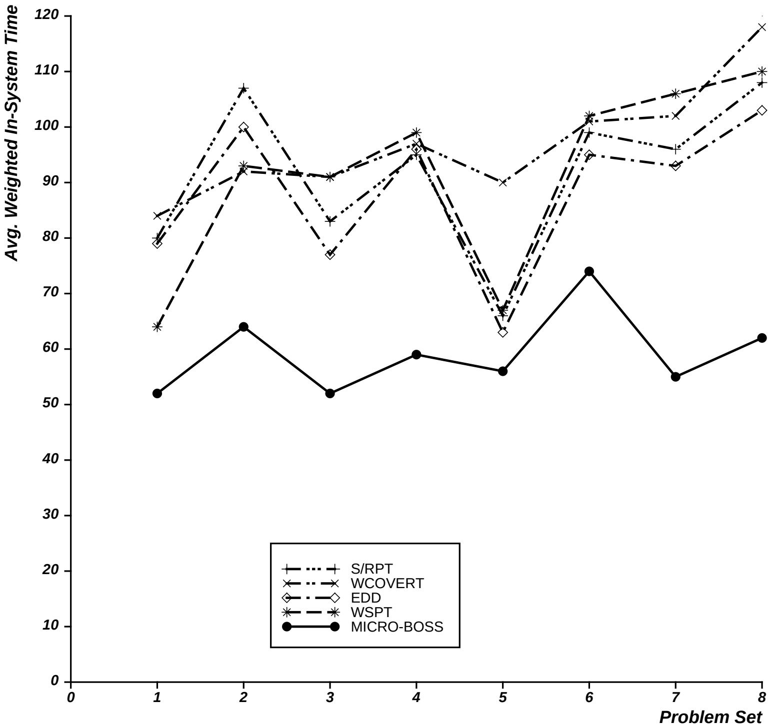


Figure 5-4: Weighted in-system time performance of MICRO-BOSS and the WSPT, EDD, S/RPT and WCOVERT dispatch rules under 8 different scheduling conditions.

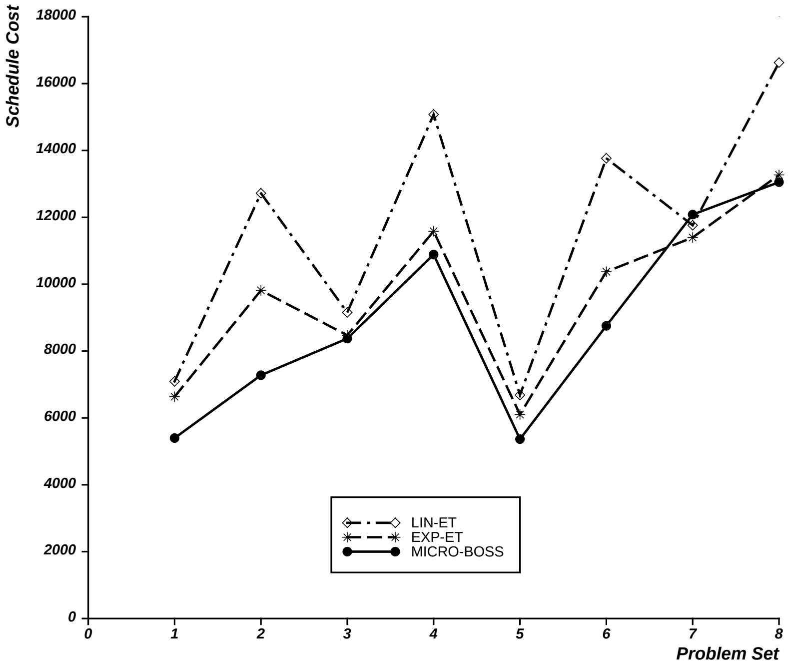


Figure 5-5: Comparison of the cost of the schedules produced by MICRO-BOSS and the LIN-ET and EXP-ET dispatch rules under 8 different scheduling conditions.

Figure 5-5 displays the average costs of the schedules produced by MICRO-BOSS, LIN-ET and EXP-ET. Figures 5-6, 5-7 and 5-8 respectively summarize tardiness, flowtime and in-system time performance.

MICRO-BOSS outperformed these two priority dispatch rules on seven problem sets out of eight, while being slightly outperformed on problem set 7. With respect to tardiness, MICRO-BOSS performed at the same level as EXP-ET (see Figure 5-6), while clearly outperforming LIN-ET (especially on problems with a narrow due date range). In general, LIN-ET and EXP-ET performed better with respect to inventory than the other four dispatch rules (EDD, WSPT, WCOVERT and S/RPT). This is not really a surprise since LIN-ET and EXP-ET explicitly account for these costs. Nevertheless, MICRO-BOSS still yielded very significant reductions in inventory compared to these rules. In particular, compared to EXP-ET, the best of the two dispatch rules, MICRO-BOSS allowed for reductions of 15 to 35 percent in average weighted flowtime, and 10 to 30 percent in average weighted in-system time[55]. Overall, MICRO-BOSS saved more than 8 percent in schedule costs compared to EXP-ET.

[55]The relatively good performance of LIN-ET with respect to in-system time is in part due to the poor performance of this rule with respect to tardiness.

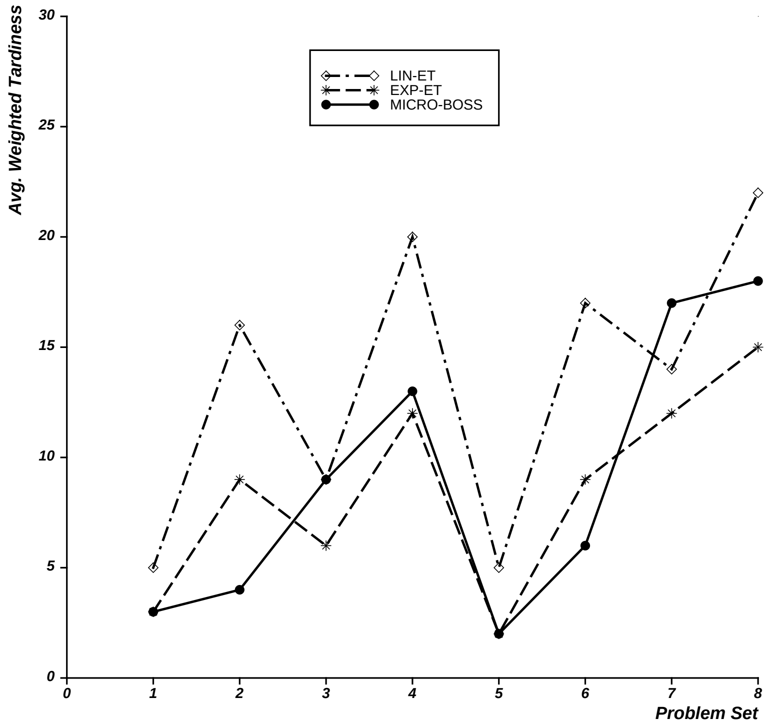


Figure 5-6: Weighted tardiness performance of MICRO-BOSS and the LIN-ET and EXP-ET dispatch rules under 8 different scheduling conditions.

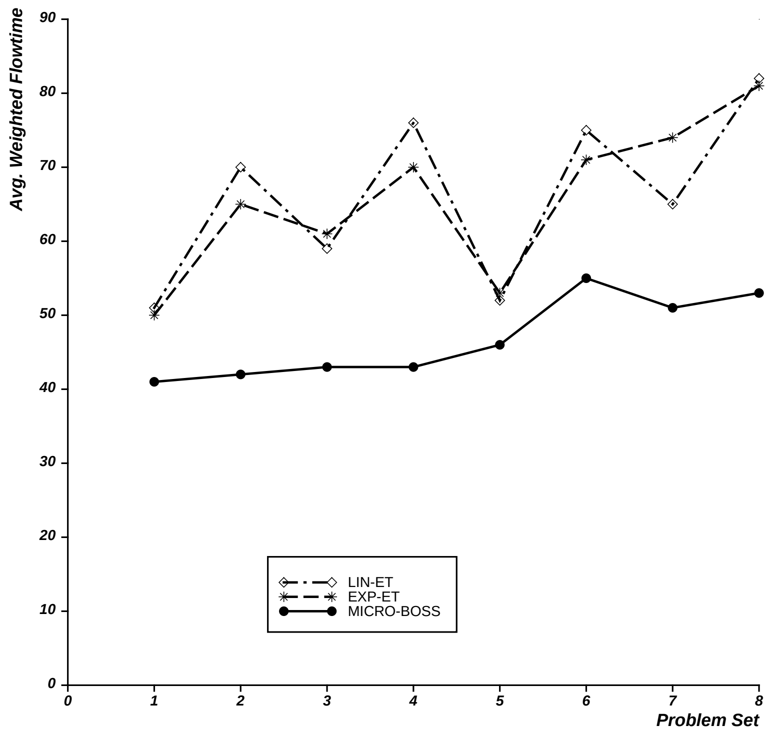


Figure 5-7: Weighted flowtime performance of MICRO-BOSS and the LIN-ET and EXP-ET dispatch rules under 8 different scheduling conditions.

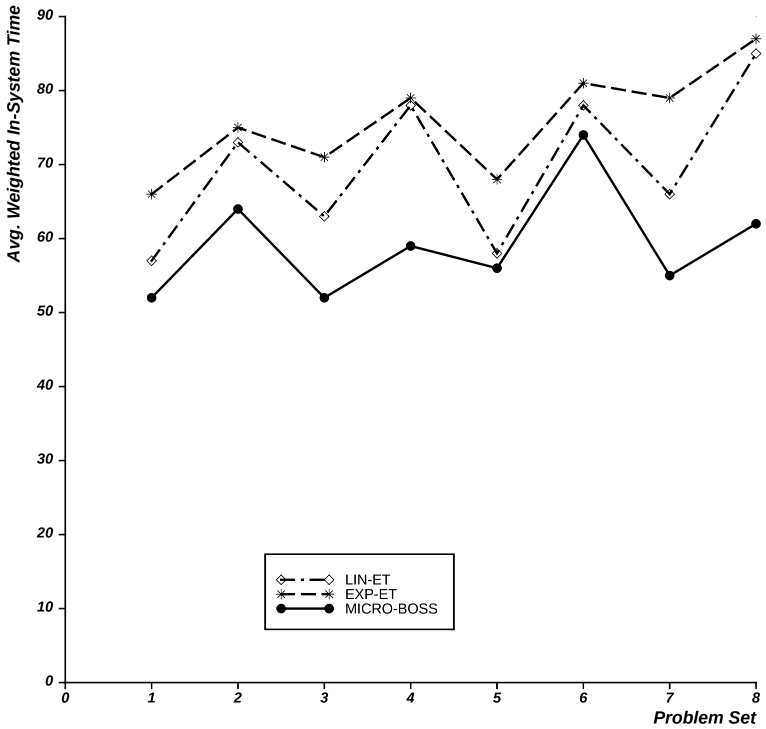


Figure 5-8: Weighted in-system time performance of MICRO-BOSS and the LIN-ET and EXP-ET dispatch rules under 8 different scheduling conditions.

In most problems, MICRO-BOSS achieved a search efficiency of 100%, which was computed as the ratio of the number of operations to be scheduled over the number of search states that were visited. Problem set 7 (two bottlenecks, a tight average due date, and a wide due date range) turned out to be the most difficult. Even on this problem set, MICRO-BOSS was still able to achieve a search efficiency of 90%. In the set of 80 scheduling problems, the worst problem required to generate 168 search states (for 100 operations). On a DECstation 3100, MICRO-BOSS required slightly over 20 minutes of CPU time for each problem[56].

[56]Preliminary experiments in C seem to indicate that this time could be reduced to 1 or 2 minutes, on the same machine.

5.3. Comparison Against A Macro-opportunistic Scheduler

A macro-opportunistic scheduler was implemented that dynamically combined both a resource-centered perspective and a job-centered perspective, like in the OPIS scheduling system [Smith 86a, Ow 87, Ow 88a]. However, while OPIS relies on a set of repair heuristics to recover from inconsistencies [Ow 88b], the macro-opportunistic scheduler of this study was built to use the same consistency enforcing techniques and the same backtracking scheme as MICRO-BOSS[57]. The macro-opportunistic scheduler also used the same demand profiles as MICRO-BOSS. When average demand for the most critical resource/time interval was above some threshold level (a parameter of the system that was empirically adjusted), the macro-opportunistic scheduler focused on scheduling the operations requiring that resource/time interval, otherwise it used a job-centered perspective to identify a critical job and schedule some or all the operations in that job. Each time a resource/time interval or a portion of a job was scheduled, new demand profiles were computed to decide which scheduling perspective to use next.

In each scheduling perspective the macro-opportunistic scheduler respectively relied on the following heuristics:

- **Resource-centered perspective**: In its resource-centered perspective, the scheduler used the same early/tardy procedure as MICRO-BOSS. As in the micro-opportunistic scheduler, the one-machine schedules produced by this procedure were ranked according to how well they minimized the apparent costs of the jobs competing for the critical resource. The best schedule was tried first. Operations requiring the critical resource/time interval were scheduled starting with those that relied most on that resource/time interval. Finally, in order to maintain backtracking at an acceptable level, it was found necessary to limit to four the number of operations that could be scheduled at once within this perspective (i.e. once the four operations that relied most on the critical resource/time interval had been scheduled, the macro-opportunistic scheduler updated its demand profiles in order to decide which perspective to adopt next).

[57]An alternative would have been to implement a variation of MICRO-BOSS using the same repair heuristics as OPIS. Besides being quite time-consuming to implement, such a comparison would have been affected by the quality of the specific repair heuristics currently implemented in the OPIS scheduler.

- **Job-centered perspective**: In this scheduling perspective, each job was partitioned into (one or several) disjoint subproblems, each corresponding to a contiguous set of unscheduled operations in the job's process routing[58]. In this perspective, the operation ordering heuristic implemented in MICRO-BOSS was used to identify a critical operation. The subproblem to which that operation belonged was selected to be scheduled next. If the job to which the critical subproblem belonged had an operation scheduled downstream of the subproblem, operations in the subproblem would be compactly scheduled next to that operation, starting with the last operation in the subproblem (backward scheduling), to reduce inventory. Otherwise operations would be scheduled in a forward fashion so as to minimize both tardiness and inventory. This was simply accomplished by scheduling these operations one by one at their best remaining start times.

[58]For instance, in Figure 4-6, job j_1 would be partitioned into two subproblems: a subproblem with operation O_1^1 and one with operations O_3^1 and O_4^1.

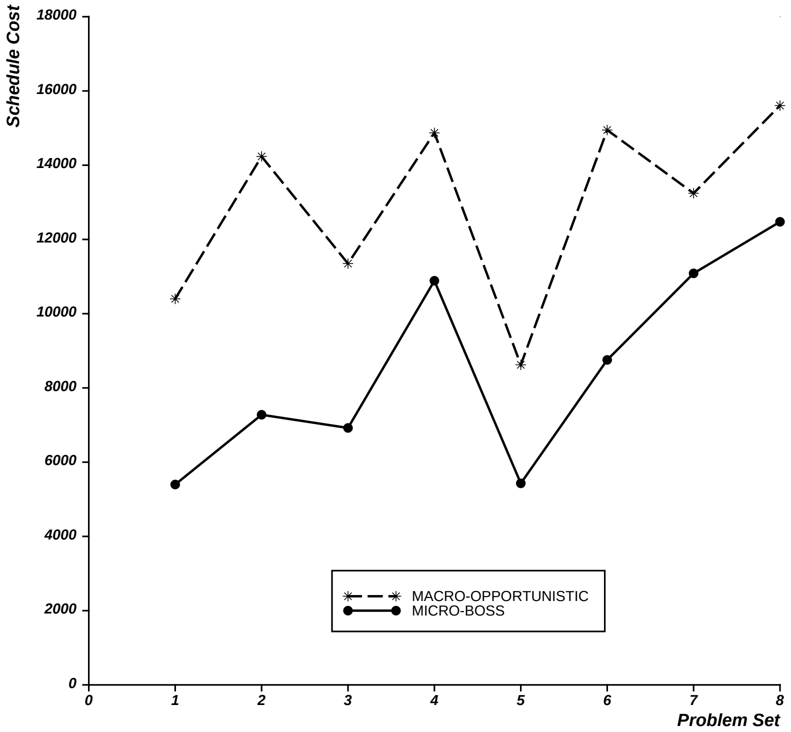


Figure 5-9: Comparison of the cost of the schedules produced by MICRO-BOSS and a macro-opportunistic scheduler under 8 different scheduling conditions.

Figure 5-9 summarizes the results of the comparison between MICRO-BOSS and the macro-opportunistic scheduler[59]. The macro-opportunistic scheduler was clearly outperformed by MICRO-BOSS under all eight scheduling conditions.

Figures 5-10, 5-11 and 5-12 indicate that the micro-opportunistic scheduler produced important savings both in tardiness and inventory over the macro-opportunistic scheduler. Although the macro-opportunistic approach built schedules with less inventory than the dispatch rules, it was unable to achieve inventory reductions comparable to those obtained by the micro-opportunistic scheduler. Interestingly enough, the macro-opportunistic scheduler seemed to have problems meeting due dates. For this reason, even though it allowed for important reductions in inventory, the macro-opportunistic scheduler was often outperformed by some of the dispatch rules. Although a more sophisticated macro-opportunistic scheduler might have produced better results, it seems unlikely that it would have been able to outperform the micro-opportunistic approach.

[59]The results presented in this section correspond to the 69 experiments (out of 80) that were each solved in less than 1,000 search states by the macro-opportunistic scheduler.

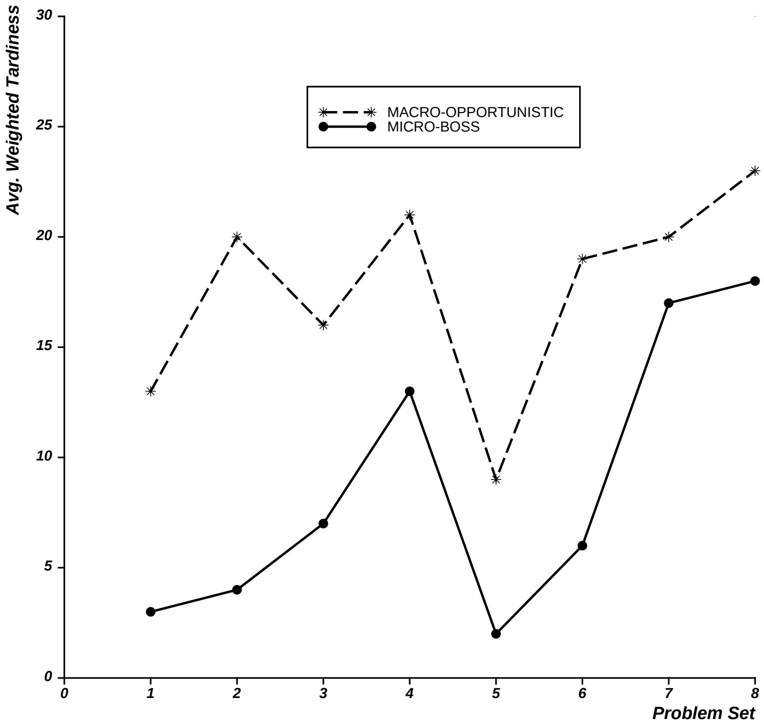


Figure 5-10: Weighted tardiness performance of MICRO-BOSS and a macro-opportunistic scheduler under 8 different scheduling conditions.

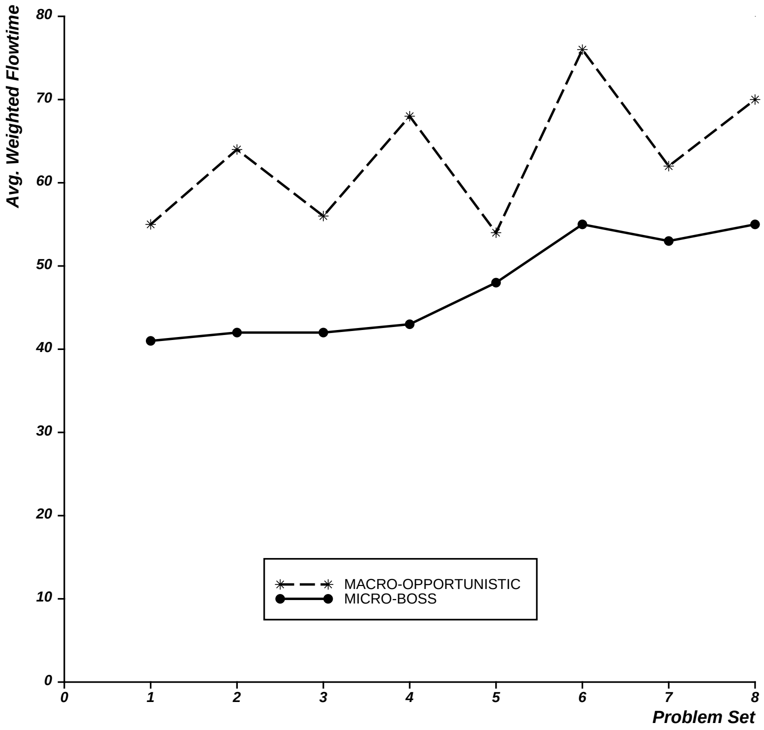


Figure 5-11: Weighted flowtime performance of MICRO-BOSS and a macro-opportunistic scheduler under 8 different scheduling conditions.

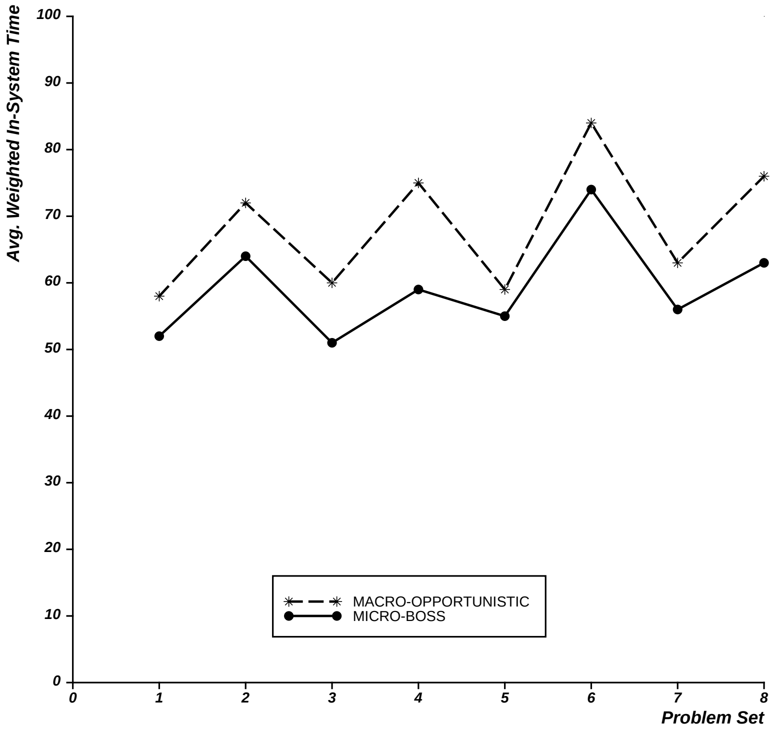


Figure 5-12: Weighted in-system time performance of MICRO-BOSS and a macro-opportunistic scheduler under 8 different scheduling conditions.

These experimental results complement earlier experimental studies of macro-opportunistic schedulers, in which sequence-dependent setups played a major role [Ow 88a]. In these earlier experiments, the macro-opportunistic scheduler outperformed the competing dispatch rule (WCOVERT) by using a heuristic that greatly reduced setups at the bottleneck. This enabled the macro-opportunistic scheduler to produce schedules with less tardiness than those produced by the dispatch rule, which relied on a myopic (i.e. local) decision rule to reduce setups (i.e. the dispatch rule did not know which machine was the bottleneck and used decision rules to minimize setups that were only looking at the jobs currently queuing in front of each machine). After using its resource-centered perspective to schedule the main bottleneck, the job-centered perspective was called upon to complete the schedule while minimizing inventory. Although our experiments confirm the ability of a macro-opportunistic scheduler to reduce inventory compared to dispatch rules, they seem to indicate that macro-opportunistic schedulers have more problems meeting due dates when dealing with problems in which sequence-dependent setups are not a major factor.

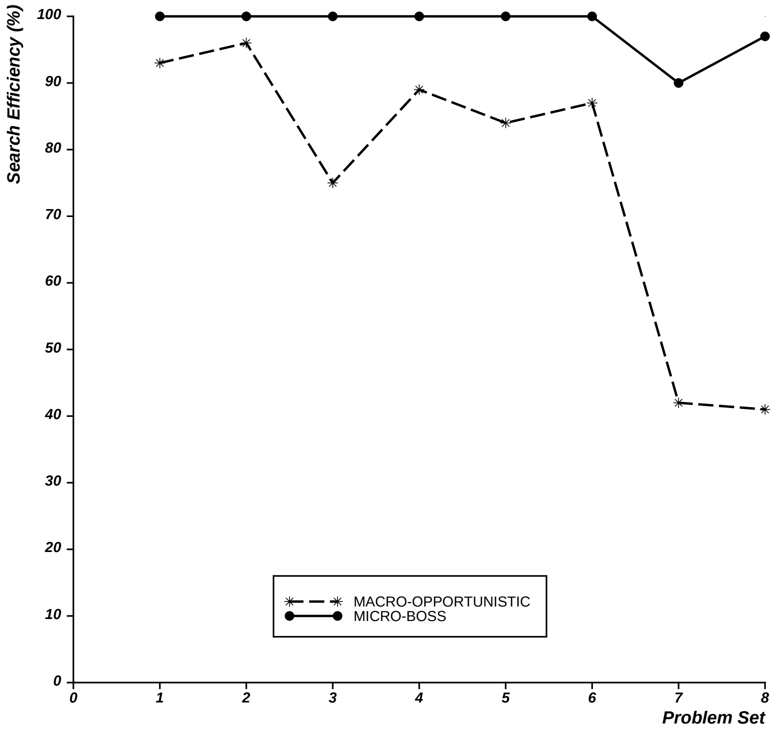


Figure 5-13: Search efficiency of MICRO-BOSS and a macro-opportunistic scheduler under 8 different scheduling conditions.

Finally Figure 5-13 indicates that the macro-opportunistic approach produced a lot more backtracking than the micro-opportunistic approach, especially on the more difficult problems with 2 bottlenecks and tight average due dates. The search efficiency of the macro-opportunistic scheduler dropped below 50% on problem sets 7 and 8.

Because of several hacks in the implementation of the macro-opportunistic scheduler, a direct comparison of actual search times for both the macro- and micro-opportunistic schedulers would not be fair to the macro-opportunistic scheduler. In the absence of backtracking, the macro-opportunistic scheduler could have been up to four times faster than the micro-opportunistic scheduler in its resource-centered perspective (since it was allowed to schedule up to four operations in that perspective), and up to five times faster in its job-centered perspective (since job subproblems could be as large as entire jobs, which, in these experiments, had five operations). In practice, backtracking quickly reduces this advantage. In particular, on the 11 problems (out of 80) that it could not solve in less than 1,000 states, the macro-opportunistic scheduler had to generate at least 10 times as many search states as the micro-opportunistic scheduler. Macro-opportunistic schedulers like OPIS do not have this problem as they rely on efficient repair heuristics to recover from inconsistencies. It appears however that, if used in combination with repair heuristics, a micro-opportunistic scheduler would not have to call upon these heuristics as often as a macro-opportunistic scheduler. pIf, like in many schedulers, the only repair heuristic is one that relaxes due dates in order to resolve inconsistencies, each call to that heuristic is likely to degrade the quality of the schedule.

5.4. Evaluating the Impact of the Biased Demand Profiles

A third set of experiments was performed to test the effect of using biased demand profiles to guide the micro-opportunistic scheduler. To this end, a variation of the micro-opportunistic scheduler using unbiased demand profiles ($B=0$ in Equation (4.5)) was run on the same set of 80 scheduling problems.

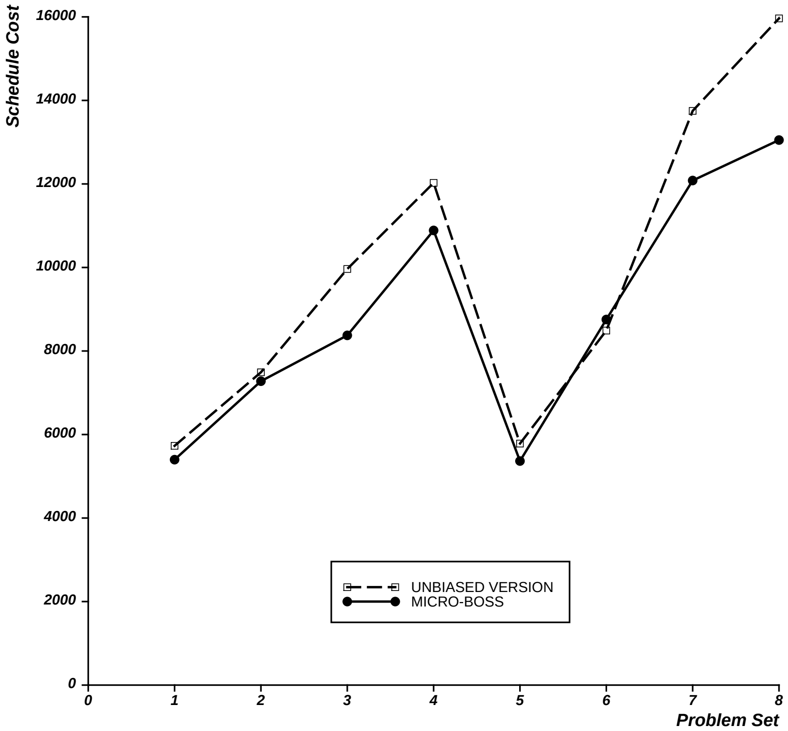


Figure 5-14: Comparison of the cost of the schedules produced with $B=0$ (unbiased version) and $B=0.9$ (biased version) under eight different scheduling conditions.

Figure 5-14 compares the average schedule costs obtained by both variations of the micro-opportunistic scheduler. Figures 5-15 and 5-16 respectively report tardiness and global inventory performance. In seven out of the eight scheduling situations of the study, simply biasing the demand profiles allowed for savings ranging from 3% to 18%, including an impressive 18% in the most difficult scheduling situation (2 bottlenecks, $\tau=0.4$, and $R=0.6$). In the one case (out of eight) where the unbiased version produced better schedules, the biased version was only outperformed by a small 3%. Overall, the biased version of MICRO-BOSS performed 23% better with respect to tardiness (Figure 5-15) while incurring a slight increase of 2% in inventory (Figure 5-16). Altogether, biasing the demand profiles reduced schedule costs by 10%. These results validate both the idea of building biased demand profiles and the particular technique described in this

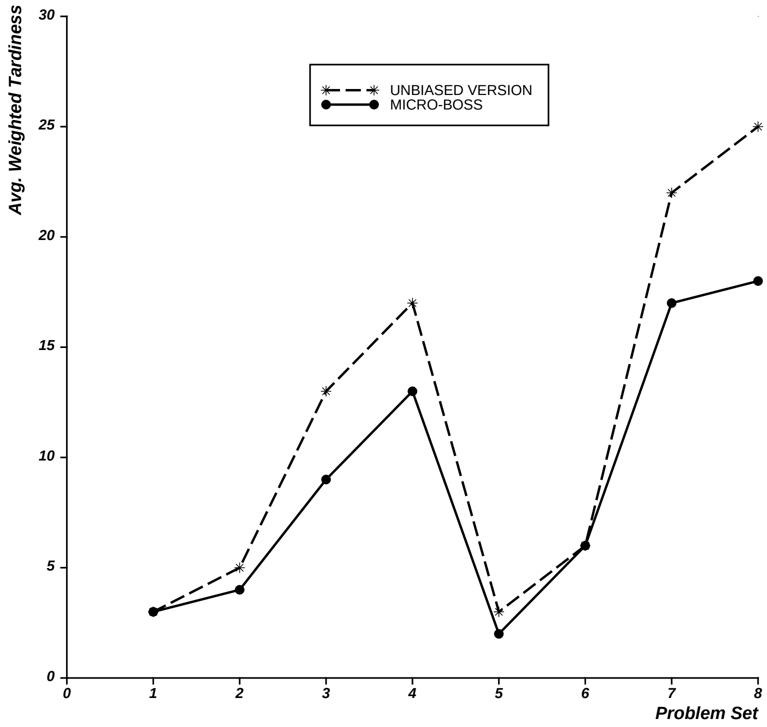


Figure 5-15: Weighted tardiness performance with $B = 0$ (unbiased version) and $B = 0.9$ (biased version) under eight different scheduling conditions.

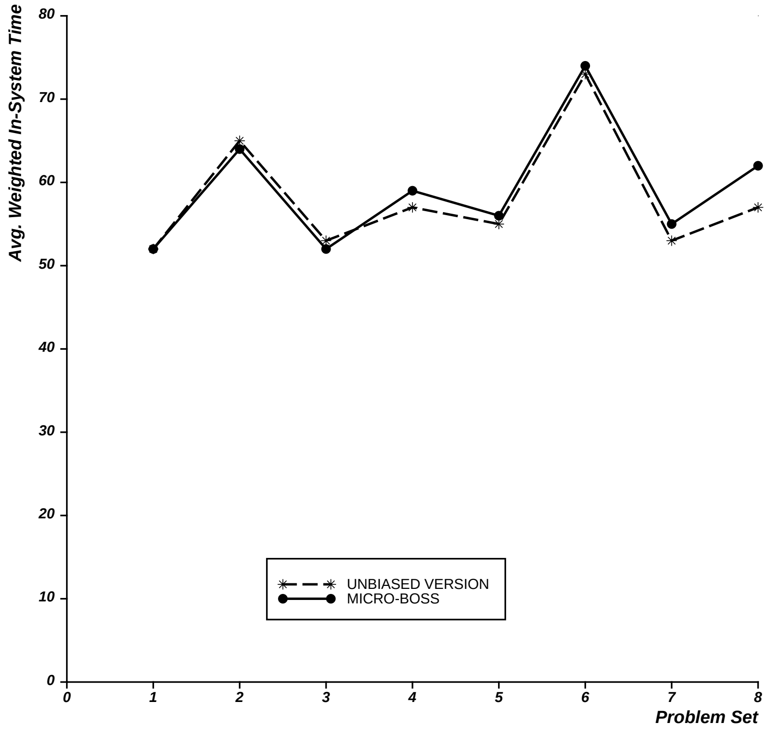


Figure 5-16: Weighted in-system time performance with $B = 0$ (unbiased version) and $B = 0.9$ (biased version) under eight different scheduling conditions.

dissertation to operationalize this idea (namely the use of the *mincost* functions). They also validate the simplifying assumption of contiguous time intervals, which was made to speedup the computation of the *mincost* functions. In general, it should be possible to obtain even better results by varying the bias in function of different problem characteristics. One might even consider adjusting the bias during the construction of the schedule. Other methods for biasing the demand profiles should be investigated as well.

5.5. Varying the Granularity of the Micro-opportunistic Approach

The above experiments indicate that a micro-opportunistic approach to scheduling indeed allows for the production of good schedules while maintaining search efficiency at a very high level. Important savings in computation time could however be achieved if it was not necessary to update the demand profiles in each search state. This section describes experiments performed in an attempt to determine how often it is really necessary to update the demand profiles and redirect the current search strategy in order to produce good schedules.

Like in Chapter 3, variations of the micro-opportunistic scheduler were built, which differed in the number, G, of operations that had to be scheduled before the scheduler was allowed to look for a new critical resource/time interval. The version with $G=1$ is the MICRO-BOSS scheduler studied throughout this chapter. For $G>1$, the G operations with the largest reliance on the current critical resource/time interval were scheduled one by one in decreasing order of their reliance. For instance, in the example displayed in Figure 4-13, the version with a granularity $G=3$ would first schedule O_2^1, then O_2^2, and finally O_3^3. The scheduler would then update the demand profiles and look for a new critical resource/time interval[60].

Overall, cost performance of MICRO-BOSS and two coarser variations of the scheduler with $G=2$ and $G=3$ is summarized in Figure 5-17[61]. These results indicate that the quality of the schedules consistently degrades as the granularity of the micro-opportunistic approach increases. These results corroborate those of Section 5.3. They suggest that all the flexibility of the micro-opportunistic scheduling approach is necessary to obtain the results reported earlier in this chapter. In other words critical resource/time intervals are highly dynamic, and it is critical to constantly follow their evolution in order to produce quality schedules. Additional results provided in Figures 5-18, 5-19, 5-20 and 5-21 indicate that the degradation observed for larger values of G is due to poorer performance both with respect to tardiness and inventory. As in Chapter 3, the results also indicate that search efficiency degrades as the granularity of the scheduler increases (especially on problems with tight average due dates and two bottlenecks).

[60]Apparent costs were still updated in each search state since they are needed for the reservation ordering heuristic.

[61]The results reported in this section were obtained on the 78 experiments (out of 80) solved in less than 1,000 search states by the scheduler with granularity $G=3$.

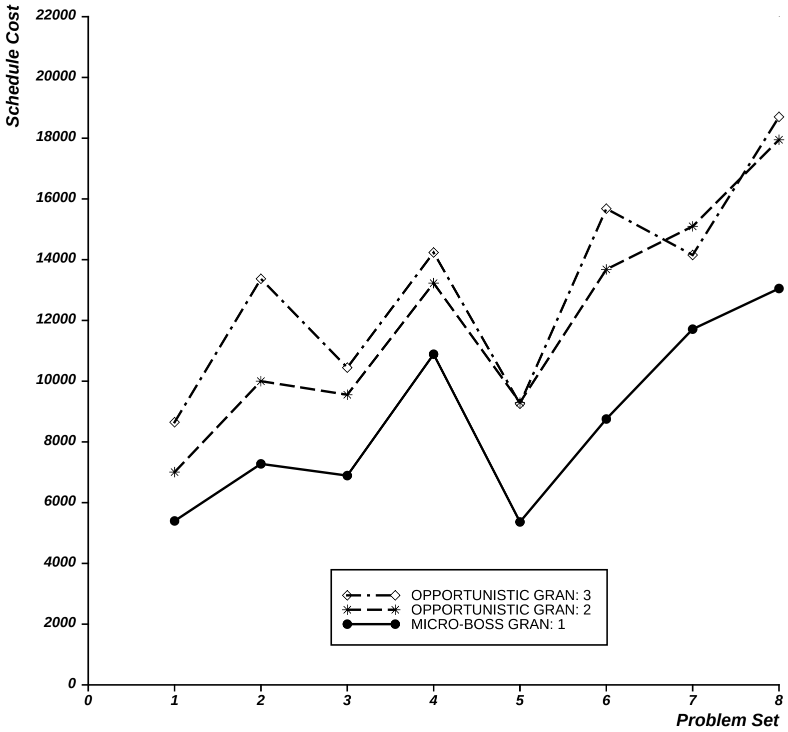


Figure 5-17: Comparison of the cost of the schedules produced by MICRO-BOSS and two coarser opportunistic schedulers under 8 different scheduling conditions.

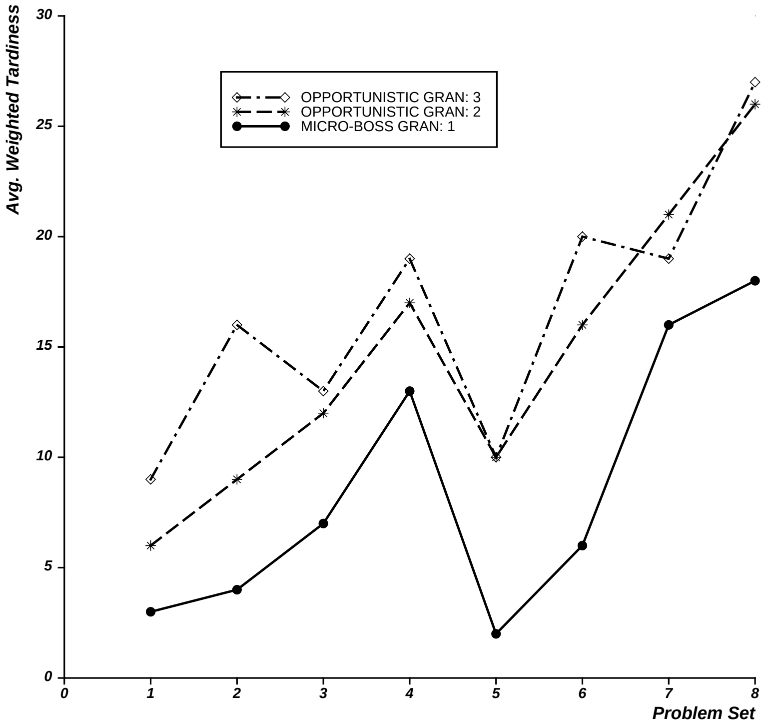


Figure 5-18: Weighted tardiness performance of MICRO-BOSS and two coarser opportunistic schedulers under 8 different scheduling conditions.

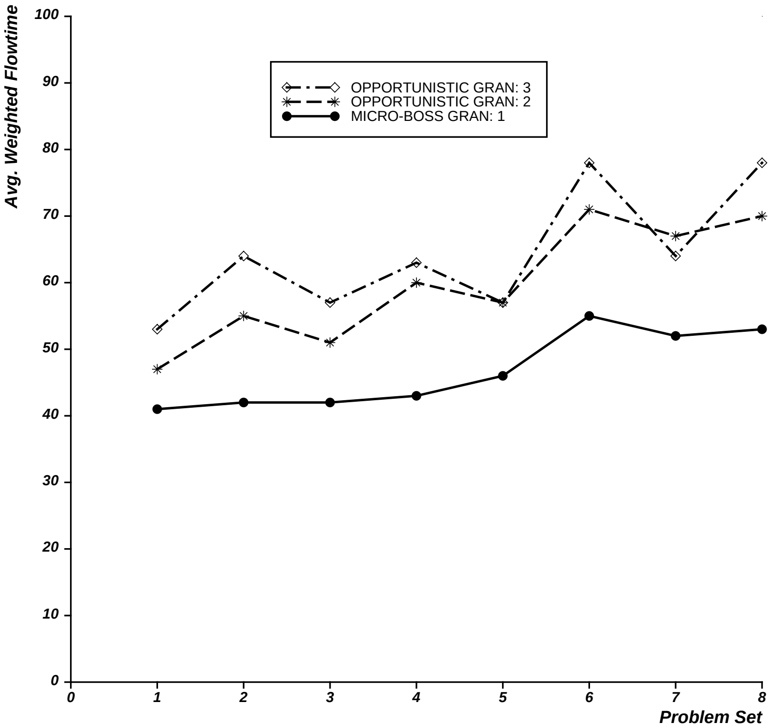


Figure 5-19: Weighted flowtime performance of MICRO-BOSS and two coarser opportunistic schedulers under 8 different scheduling conditions.

Nevertheless, it might however be possible to adapt the granularity of the scheduler in function of the difficulty of the problem or even in function of the difficulty of the current search state. In particular one could imagine a scheduler whose granularity increases as contention subsides during the construction of the schedule. Further experimentation is needed to determine whether such a scheduler can actually be constructed without incurring performance degradation.

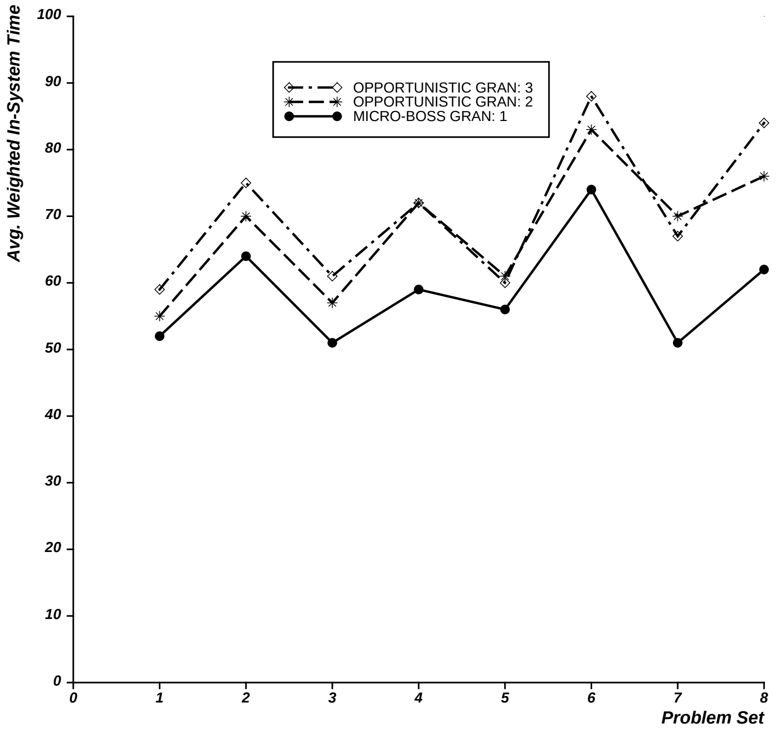


Figure 5-20: Weighted in-system time performance of MICRO-BOSS and two coarser opportunistic schedulers under 8 different scheduling conditions.

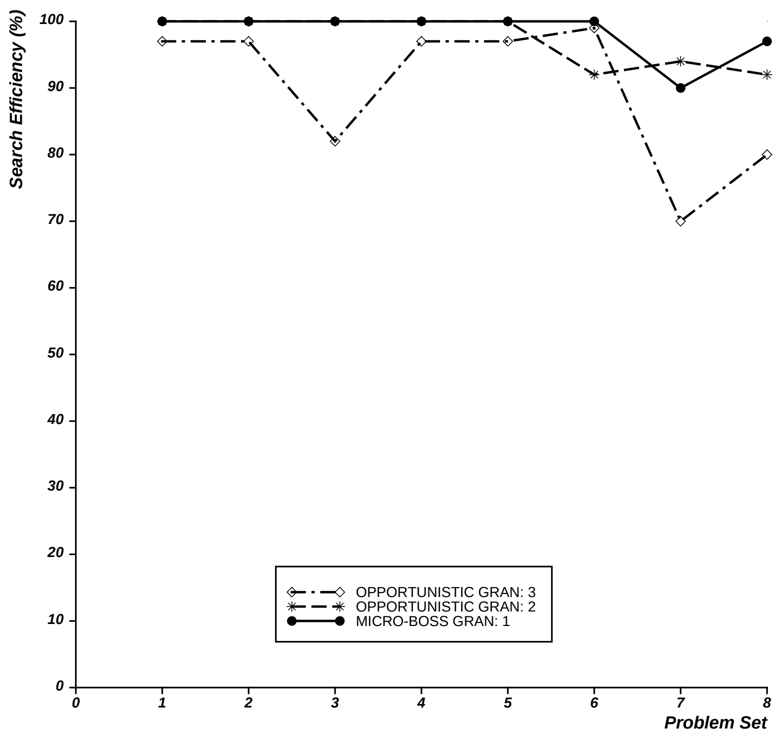


Figure 5-21: Weighted in-system time performance of MICRO-BOSS and two coarser opportunistic schedulers under 8 different scheduling conditions.

5.6. Sensitivity to Changes in the Cost Function

The experiments reported earlier in this chapter were performed on scheduling problems in which the average value r of the ratio $\frac{tard_l}{inv_l}$ was equal to 5, namely experiments in which tardiness was given more importance than inventory, as it is often the case in practice. This section reports additional experimental results obtained on a set of problems in which the average value of this ratio was set to 1 (i.e. problems in which tardiness and inventory performance were treated equally). The set of problems with $r=1$ was generated to be identical to the set with $r=5$, except for tardiness costs which were randomly drawn from a uniform distribution $U(1,2S_l)$ (in order to have $r=1$).

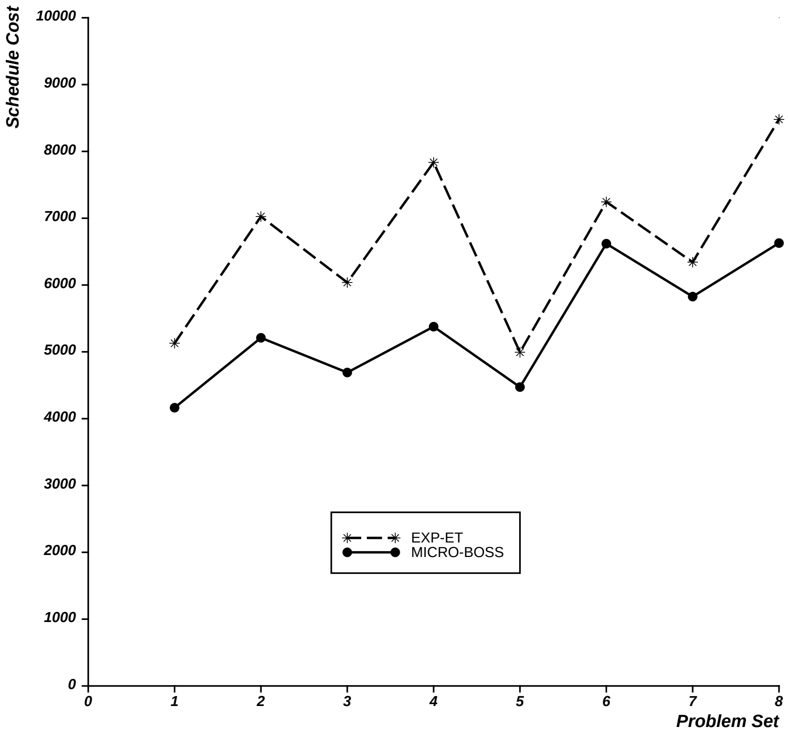


Figure 5-22: Comparison of MICRO-BOSS and EXP-ET on problems with $r=1$ (i.e. problems with low tardiness costs).

In these experiments, MICRO-BOSS was compared against the EXP-ET dispatch rule, the rule that had performed best on problems with $r=5$. Figure 5-22 compares the cost of the schedules produced by MICRO-BOSS and EXP-ET for $r=1$. It appears that MICRO-BOSS outperformed EXP-ET even more clearly than on the set with $r=5$.

Figures 5-23, 5-24, and 5-25 respectively compare tardiness, flowtime, and in-system time performance. Both MICRO-BOSS and EXP-ET produced schedules with more

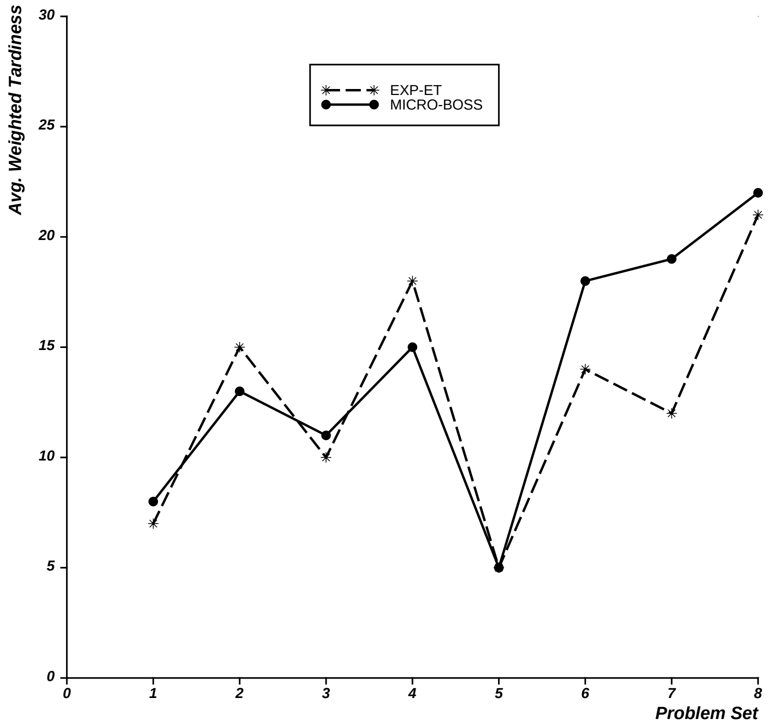


Figure 5-23: Weighted tardiness performance of MICRO-BOSS and EXP-ET on problems with $r = 1$ (i.e. problems with low tardiness costs).

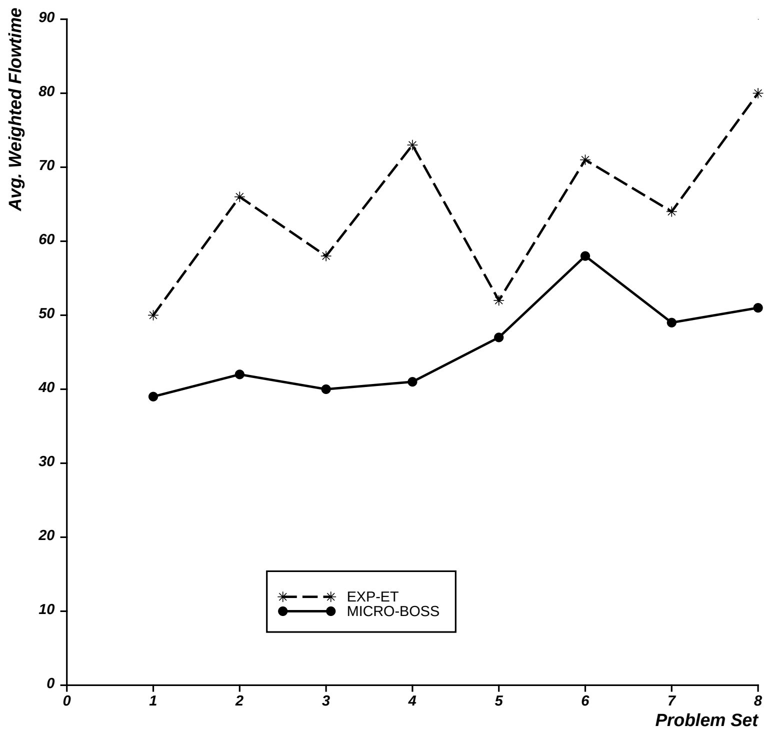


Figure 5-24: Weighted flowtime performance of MICRO-BOSS and EXP-ET on problems with $r = 1$ (i.e. problems with low tardiness costs).

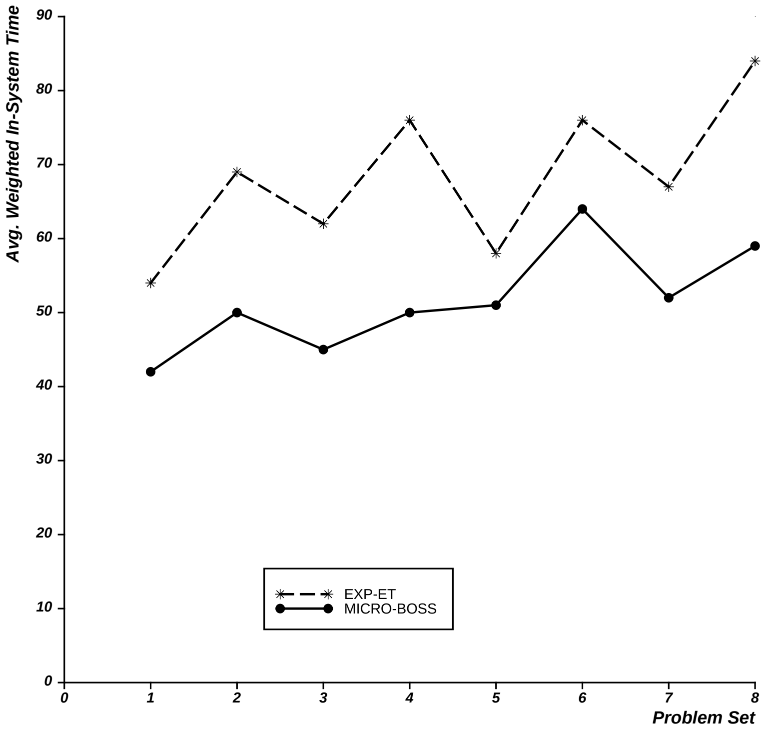


Figure 5-25: Weighted in-system time performance of MICRO-BOSS and EXP-ET on problems with $r = 1$ (i.e. problems with low tardiness costs).

tardiness than for the problem set with $r = 5$ (i.e. the problem set where tardiness costs were more important than inventory costs). It appears however that MICRO-BOSS was a lot more effective at reducing inventory than EXP-ET.

Figures 5-26 and 5-27 illustrate the adaptation of MICRO-BOSS to changes in the cost function by comparing the tardiness and inventory of the schedules produced for $r = 1$ and for $r = 5$. The adaptation to changes in the cost function is very clear: for $r = 5$ (i.e. problems with high tardiness costs), MICRO-BOSS produced schedules with less tardiness and more inventory than for $r = 1$ (i.e. problems with low tardiness costs).

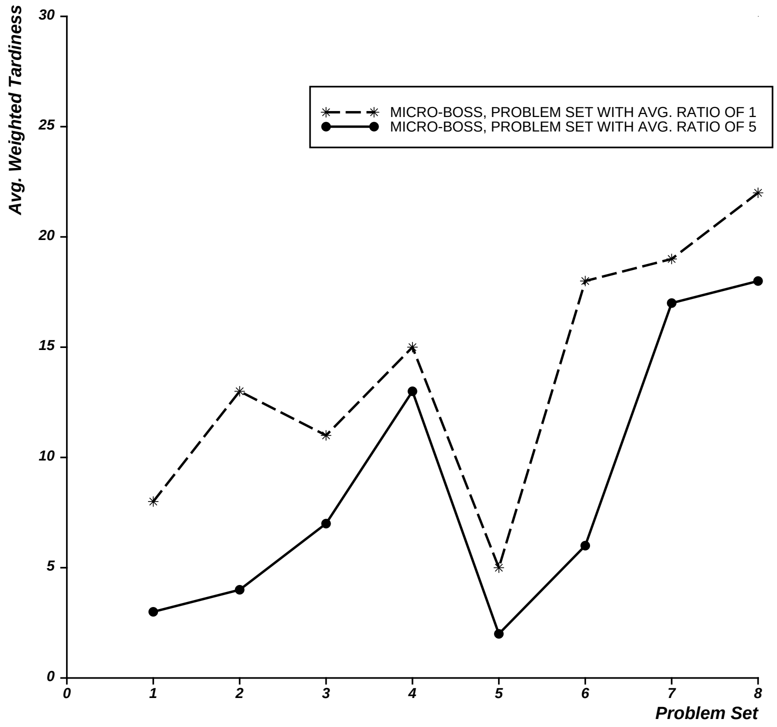


Figure 5-26: Weighted tardiness performance of MICRO-BOSS on 80 problems with $r = 5$ (i.e. high tardiness costs) and 80 problems with $r = 1$ (i.e. low tardiness costs).

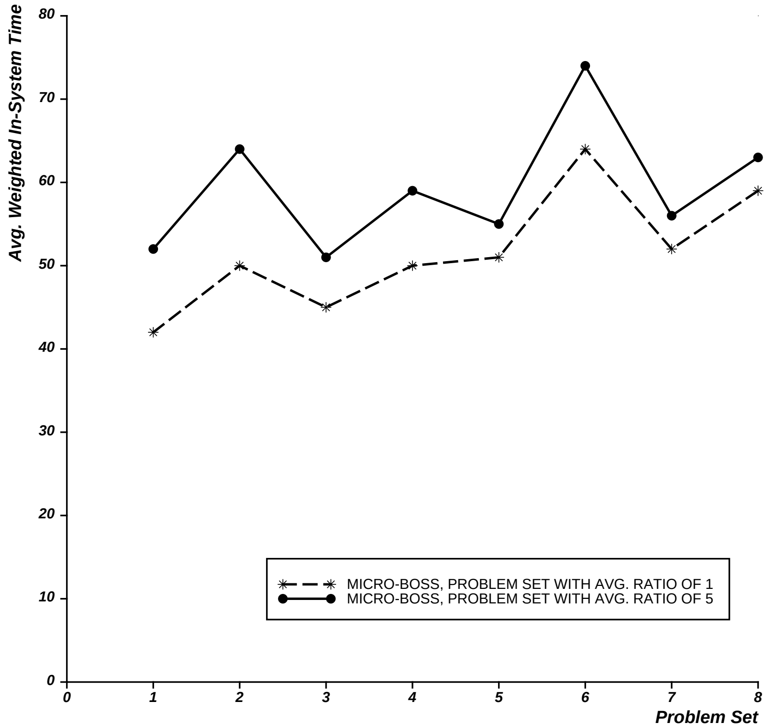


Figure 5-27: Weighted in-system time performance of MICRO-BOSS on 80 problems with $r = 5$ (i.e. high tardiness costs) and 80 problems with $r = 1$ (i.e. low tardiness costs).

Chapter 6
Summary and Concluding Remarks

This chapter summarizes the main contributions of this work. To conclude, a discussion of future possible research directions is provided.

6.1. Contributions

The main contributions of this dissertation are as follows:

- *A Micro-opportunistic Approach to Job Shop Scheduling*: While earlier schedulers such as ISIS, OPT and OPIS have relied on coarse problem decompositions, this dissertation presented a micro-opportunistic approach to job shop scheduling that allows for much finer subproblems. Rather than scheduling large resource subproblems or large job subproblems one by one, a micro-opportunistic scheduler can view each operation as an independent decision point. In this dissertation, it was demonstrated that the extra flexibility of this approach can be exploited to constantly redirect the scheduling effort towards those bottleneck operations that appear to be most critical. Experimental results demonstrate that the flexibility of this approach is instrumental in efficiently solving problems in which some operations have to be performed within non-relaxable time windows (e.g. non-relaxable release and due dates). Experimentation in the factory scheduling domain also indicates that the ability of the micro-opportunistic approach to constantly revise its search procedure yields important improvements in schedule quality.

- *Application of the CSP Paradigm to Job Shop Scheduling*: The CSP paradigm embedded in the micro-opportunistic approach aims at solving

Constraint Satisfaction Problems by interleaving backtrack search with the application of consistency enforcing techniques and look-ahead techniques to decide which variable to instantiate next and which value to assign to that variable. At the time this research started, a good deal of experimental results reported in the CSP literature had been obtained on toy problems such as N-queens. This dissertation demonstrates that the CSP paradigm scales up to larger and harder problems such as job shop scheduling. It also shows that generic CSP heuristics that have been proposed in the literature are not sufficient to solve these problems. This is because these heuristics fail to properly account for the constraint interactions induced by the high connectivity of the constraint graphs generally encountered in job shop scheduling. Instead, new heuristics were developed that were shown to outperform both generic CSP heuristics and specialized scheduling heuristics.

This work also suggests that benchmark problems often used in the CSP literature are not representative of hard problems such as job shop scheduling. It is hoped that this dissertation will prompt researchers in the field to look for new benchmark problems and new more powerful heuristics for these problems.

- *New Variable and Value Ordering Heuristics based on a Probabilistic Model of the Search Space*: A probabilistic model of the search space was introduced in which, for CSPs, critical operations are identified as those likely to create backtracking and promising values as those expected to be compatible with a large number of solutions. Within this model, measures were developed to estimate the reliance of an operation on the availability of a reservation, and the degree of contention among unscheduled operations for the possession of a resource over some time interval. Based on these measures, a pair of new variable and value ordering heuristics was defined:
    1. The *Operation Resource Reliance* (ORR) variable ordering heuristic selects the operation that relies most on the most contended resource/time interval, and

2. The *Filtered Survivable Schedules* (FSS) value ordering heuristic assigns to that operation the reservation which is expected to be compatible with the largest number of survivable job schedules, i.e. the largest number of job schedules expected to survive resource contention.

Experimental results show that these two heuristics enable the micro-opportunistic scheduler to *efficiently* solve many job shop scheduling problems with non-relaxable time windows that could not be efficiently solved by prior heuristics (both generic CSP heuristics and specialized heuristics designed for similar scheduling problems). The results also indicate that the ORR and FSS heuristics not only allow for significant increases in search efficiency but also allow for important reductions in search time.

- *Extension of the CSP Paradigm to deal with COPs*: This dissertation also extends the CSP paradigm to deal with job shop scheduling as an optimization problem. While least constraining value ordering heuristics used to solve CSPs are particularly good at reducing backtracking, they typically fail to provide good solutions. In order to produce quality solutions, more constraining value ordering heuristics are required. This generally prompts the use of stronger consistency enforcing mechanisms and more accurate variable ordering heuristics in order to maintain backtracking at a low level.

This dissertation suggests that a critical variable in a COP is one involved in an important tradeoff, and a promising value one that is expected to optimize this tradeoff. An important tradeoff is a subproblem, whose solution will critically impact the quality of the entire solution. By first optimizing the most important tradeoffs in a problem, the system can later use the solutions to these tradeoffs to help solve the remainder of the problem. Once critical tradeoffs have been worked out, the remainder of the problem tends to become more decoupled, and hence easier to optimize. Chances of backtracking tend to simultaneously subside. A system, that does not attempt to work out critical tradeoffs first, runs the risk of overconstraining its set of alternatives before having worked on the subproblems that will impact most the quality of the entire solution.

It was shown that the probabilistic model introduced for the job shop CSP can be generalized to identify tradeoff operations. This is done by constructing biased demand profiles that reflect contention between the good reservations of unscheduled operations. The ORR variable ordering heuristic is then used to identify operations whose good reservations are expected to conflict with the good reservations of other unscheduled operations. Experimental results comparing versions of the MICRO-BOSS scheduler using both biased and unbiased demand profiles validate the idea of working on tradeoff operations first, and validate the probabilistic model implemented in MICRO-BOSS to identify these tradeoff operations via the construction of biased demand profiles.

- *The MICRO-BOSS Factory Scheduling System*: One of the most tangible contribution of this thesis is certainly the MICRO-BOSS factory scheduling system itself, which is able to deal explicitly with tardiness costs, in-process inventory costs, and finished-goods inventory costs. This system has outperformed a variety of competing scheduling techniques under a wide range of scheduling conditions. MICRO-BOSS introduces the notion of bottleneck operation, which is directly formalized in terms of tardiness and inventory costs and accounts for earlier scheduling decisions. An early version of MICRO-BOSS is currently used as the scheduling module of the CORTES decentralized production control system [Sycara 91].

6.2. Future Research

There are a number of ways in which this research can be extended, some of which are briefly described below.

6.2.1. Dynamically Adapting the Search Procedure

Many aspects of the micro-opportunistic search procedure could benefit from being tuned to the problem at hand, to the state currently being explored, or even to the subproblem currently under consideration. Some specific suggestions follow.

- **Slowly Increasing the Granularity of the Search Procedure**: Experimental results reported in Chapter 3 and 5 indicate that bottleneck operations can shift very fast from one part of the search space to another during the construction of the schedule. In the current version of the micro-opportunistic search procedure demand profiles are updated in each search state to track the evolution of bottleneck operations. While this can be done relatively fast, important speedups could be achieved if the demand profiles did not have to be updated so often. It is possible that, as contention subsides, the granularity of the search procedure could be gradually increased without affecting the performance of the scheduler.

- **Dynamically Adjusting the Bias in the Demand Profiles**: This dissertation indicates that important reductions in schedule costs can be achieved by biasing the demand profiles in order to identify operations involved in important tradeoffs. Exactly how much bias should be used is still very much an open issue. It seems in fact that the optimal bias is a function of the difficulty of the problem at hand. The optimal bias is also expected to vary from one search state to another, and even within one search state from one part of the problem to another. Too strong a bias will generally result into extra backtracking on problems where contention is high, whereas too little bias may produce poorer solutions. A mechanism that progressively increases the bias in the demand profiles as contention subsides might further improve schedule quality without reducing search efficiency.

- **Dynamic Variable and Value Ordering Heuristics/Dynamic Consistency Enforcing Mechanisms/Dynamic Deadend Recovery Schemes**: At the end of Chapter 3, a switch mechanism was briefly described that would enable the CSP version of the micro-opportunistic scheduler to automatically switch to simpler variable and value ordering heuristics, once contention has dropped below a threshold level. One such mechanism was actually implemented in an earlier version of the micro-opportunistic scheduler, which relied on Monte Carlo sampling to measure resource contention. This switch allowed for important speedups. A similar switch is currently embedded in the reservation ordering heuristic of MICRO-BOSS: when contention decreases, the system automatically switches to a simpler greedy reservation ordering heuristic. More generally, dynamic variable and value ordering heuristics, dynamic consistency enforcing mechanisms, and dynamic deadend recovery schemes could allow for important speedups by always performing just enough work to allow the system to make the best possible decision in each search state. This is in the line of some of the work performed within the context of the OPIS scheduling system, especially in the area of schedule revision [Ow 88a]. Collinot and Lepape have pushed this idea one step further and studied a consistency enforcing mechanism that works differently on different parts of a same problem [Collinot 90].

Learning techniques [Michalski 83] might prove very useful to properly tune all these dynamic mechanisms [Shaw 90].

6.2.2. Breaking Away from the Depth-first Backtrack Search Paradigm

The micro-opportunistic approach to job shop scheduling described in this dissertation is based on a depth-first backtrack search procedure. One advantage of this procedure is that it requires keeping track of only one partial solution at a time. It is also relatively quick at finding a first solution, since it does not waste time simultaneously exploring several paths. Another important advantage of this search procedure is that it guarantees search completeness: even if there is only one solution, the procedure is bound to find it; if the procedure does not find a solution, one can infer that the problem is infeasible. In order to guarantee search completeness, backtrack search requires that, within each branch of the search tree, assignments are undone only if they are found to violate a constraint. This requirement as well as other assumptions of depth-first backtrack search may actually be worth tampering with.

- **Iterative Improvement**: The search procedure studied in this dissertation can readily be transformed into a branch-and-bound search procedure [Nemhauser 88] or a beam search procedure [Lowerre 76, Fox 83], but such brute force search generalizations may not be very cost-effective. Instead, as the schedule is constructed, one could attempt to improve earlier scheduling decisions based on more recent ones. One such reoptimization technique has been developed by Adams, Balas, and Zawack in the framework of the Shifting Bottleneck Procedure (SBP) [Adams 88]. Rather than reoptimizing entire resources, a micro-opportunistic version of this procedure would only reoptimize those operations that had already been scheduled on these resources.

- **Undoing Decisions that are not Provably Wrong**: In the case of the job shop CSP, partial solutions that are not detected as deadends by the consistency enforcing mechanism might still be worth abandoning simply because they do not appear promising enough. This could be the case if a search state is reached where contention for a resource is above some threshold level over a long time interval. Rather than wasting time exploring a portion of the search space that is unlikely to contain many solutions, the system might decide to abandon the exploration of that portion of the search space. If the procedure keeps track of search states abandoned in this fashion, search can still be complete. Alternatively, the procedure could decide to forget altogether about these states, hoping that other branches in the search tree are rich enough in solutions.

- **Repair Heuristics**: Deadend recovery schemes such as chronological backtracking, backjumping [Gaschnig 79, Dechter 89b] or more sophisticated forms of dependency-directed backtracking [Stallman 77] can waste a lot of time in trial-and-error mode attempting to undo the smallest possible number of decisions in order to recover from a deadend. As mentioned above, it may be more efficient to use heuristics that sometimes undo more decisions than really necessary. Rather than moving back along the current search branch and then down an alternative branch in the search tree, an even more radical approach consists in directly patching the current partial schedule. Several repair heuristics to patch inconsistent schedules are described in [Ow 88b] that could readily be combined with a micro-opportunistic scheduler. The simplest of these heuristics is a right-shifting heuristic, which simply pushes conflicting operations forward in time until operations no longer have conflicting resource requirements. In problems with relaxable due dates and non constraining latest acceptable completion dates, this simple repair procedure would ensure a polynomial time worst-case complexity to the whole approach (whereas using backtracking the worst-case time complexity will always be exponential). As pointed out at the beginning of the dissertation, scheduling systems that dynamically relax due dates by pushing conflicting operations forward in time also tend to produce poorer schedules. An intermediate approach would be to combine a backtrack recovery scheme with a right shifting mechanism. The system would first attempt to recover from deadends using its backtrack mechanism in order to preserve schedule quality. However, if search efficiency falls below a threshold level, the system could resort to a right-shifting procedure that efficiently patches the schedule. Clearly this approach is only applicable to problems with relaxable due dates. Scheduling problems with operations that need to be performed within non-relaxable time windows will always have an exponential worst-case complexity. Finally, notice that, while right-shifting is always guaranteed to terminate in polynomial time, more complex repair heuristics may not share that property. Some of these heuristics actually require very careful implementations to ensure that they do not loop forever. Nevertheless, the development of new repair heuristics appears a worthy endeavor, especially because they can also help patch the schedule as it gets invalidated by unexpected events (see also next point).

- **Progressively Lowering the Accuracy of the Procedure**: Real life scheduling problems are often fraught with contingencies. Raw materials arrive late, machines break down, operation durations vary, machinists call in sick, etc. Often within hours, schedules get invalidated by unexpected events. Under such conditions, the need for building crisp schedules over long term horizons seems questionable. While it certainly makes sense to ensure that capacity constraints are precisely met over the next few hours or the next few days, the same amount of accuracy is not necessarily justified further along in the future. Large savings in computational time could possibly be achieved by building a scheduling system whose level of accuracy progressively decreases in time. In such a system, backtracking would only take place for constraints that are violated beyond the local level of accuracy. The system would still have to ensure that the schedule appears feasible from a global perspective (i.e. that it does not violate global capacity constraints) even if it contains some local inconsistencies. Kerr and Walker have reported initial research in that direction using fuzzy arithmetic to represent constraints that have to be satisfied up to a certain level of accuracy [Kerr 89].

6.2.3. Refining the Scheduling Model

Additional research needs to be carried out in order to enable the micro-opportunistic procedure to deal with sequence-dependent setups, machines with non-unary capacities, and more complex types of process routings (e.g. more complex temporal relations between operations or alternative process routings).

6.2.4. Adding Reactive Capabilities

Future work on MICRO-BOSS is also expected to include the addition of a reactive component that will enable the system to revise the current schedule in the presence of unexpected events rather than having to rebuild a new schedule from scratch. Rather than relying on a predefined set of repair heuristics, our research will attempt to come up with reactive scheduling strategies that are dynamically formulated according to the time available to patch the schedule and the amount of disruption in the schedule. This work will be performed within a decentralized setting in which MICRO-BOSS interacts with several dispatch modules. The dispatch modules are each responsible for the execution of the schedule on a group of machines. These modules attempt to contain disruptions locally based on a set of simple heuristics, such as heuristics that switch the order of two jobs on a machine or switch jobs from one machine to another similar machine. When a dispatch module can no longer contain disruptions without radically departing from the original schedule, it will call upon the reactive component of MICRO-BOSS to perform some global reoptimization.

6.2.5. Infusing Parallelism

Research on parallel implementations of the micro-opportunistic approach to job shop scheduling has already begun[62] [Sycara 91]. A first distributed scheduling system was built in which several micro-opportunistic scheduling agents are each responsible for scheduling different groups of jobs which may all require a small group of *shared resources*. Monitoring agents ensure that no two agents reserve the same resource at the same time, and broadcast each valid resource reservation to all agents that could require the reserved resource/time interval. Scheduling agents account for each others' resource requirements and coordinate their scheduling activities by regularly communicating their aggregate demand profiles to the monitoring agents. These profiles are further aggregated by the monitoring agents to measure system-wide resource contention, and the resulting system-wide aggregate demand profiles are communicated back to the agents. Based on these system-wide aggregate demand profiles, each agent focuses its local scheduling effort on globally critical decision points (variable ordering). The system-wide profiles also provide predictive measures of the impact of an agent's local scheduling decisions on the ability of other agents to build good schedules (value ordering).

Fine-grained parallel implementations of the procedures that build demand profiles may be worth investigating as well.

---

[62]This work was performed jointly with Katia Sycara, Steve Roth, and Mark Fox.

Appendix A
Counting the Number of Survivable Schedules

This appendix describes a dynamic programming procedure that allows to efficiently count the number of survivable job schedules (or more generally the number of survivable solutions to the relaxation defined in Chapter 3 for the FSS value ordering heuristic) that are compatible with the assignment of a reservation ρ to the current critical operation O_i^l. This number was referred to as $compsurv_i^l(\rho)$ in Chapter 3. However, because it only depends on the start time t allocated to O_i^l in reservation ρ, it will now be written $compsurv_i^l(t)$. The procedure presented here is a variation of a similar method developed by Dechter and Pearl for the ABT value ordering heuristic [Dechter 88] (see also [Pearl 88]). While a direct generalization of Dechter and Pearl's procedure would have an $O(\nu_l k^2)$ complexity (where ν_l is the number of operations in the relaxation used by the FSS value ordering heuristic, and k the maximum number of possible reservations of an operation in that relaxation), the procedure described here takes advantage of the linearity of precedence constraints to reduce this complexity to $O(\nu_l k)$.

Figure A-1 represents a prototypical tree-like process routing, which has been reorganized with the current critical operation as the root of the tree. The arrows represent precedence constraints between operations in the process routing. The children of the critical operation O_i^l in the tree are those operations that are directly connected to O_i^l by a precedence constraint, the grandchildren the operations directly connected to these operations by precedence constraints, etc.

All the computations presented in this appendix refer to a single search state, in which consistency checking has already been performed. The notations are those used in Chapter 3. A few extra notations need to be defined:

- $surv_p^l(t) = \sum_{\rho \in G} surv_p^l(\rho)$, where G is the set of remaining reservations of O_p^l with $st_p^l = t$;
- α_p^l: the direct children of O_p^l that are after O_p^l in the process routing;

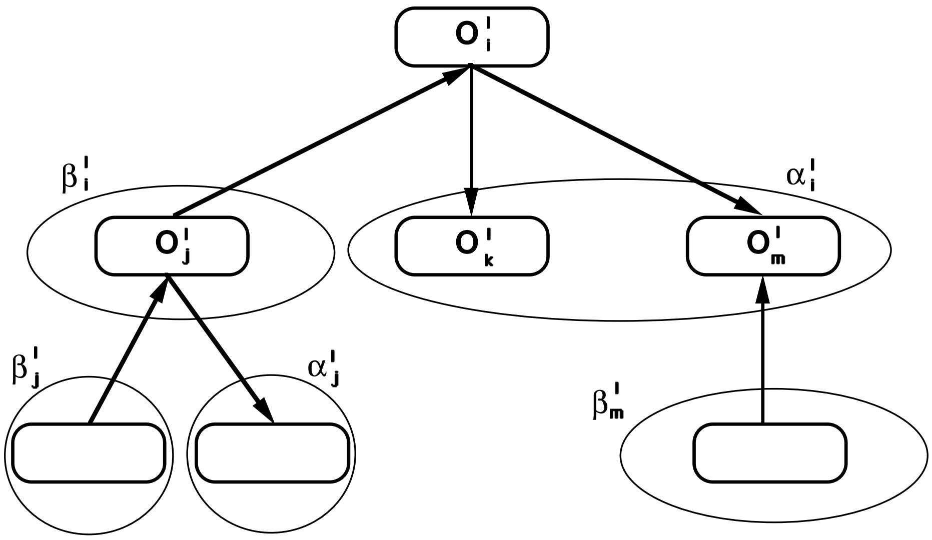


Figure A-1: A tree-like process routing, organized with the current critical operation as its root. Arrows represent precedence constraints.

- β_p^l: the direct children of O_p^l that are before O_p^l in the process routing;
- Δ: the time granularity of the problem. In Chapter 3, it was assumed that $\Delta = 1$ (i.e. that all start times and end times have to be integers). For the sake of clarity, the formulas presented in this appendix account explicitly for Δ.

In tree-like process routings, each operation O_p^l is the unique link between otherwise disjoint sets of operations, that each correspond to one of its children. Each of these sets contains exactly one child of operation O_p^l and defines a subproblem that only interacts with the other subproblems via operation O_p^l. Accordingly,

- For each $O_j^l \in \beta_p^l$, we define $BEF_{p,j}^l(t)$ as the number of survivable solutions to the subproblem defined by operation O_j^l and its descendants that are compatible with the assignment of $st_p^l = t$ to O_p^l;
- For each $O_k^l \in \alpha_p^l$, we define $AFT_{i,k}^l(t)$ as the number of survivable solutions to the subproblem defined by operation O_k^l and its descendants that are compatible with the assignment of $st_p^l = t$ to O_p^l;

Given that operation O_i^l is the only link between the subproblems defined by each one of its children, we have:

$$compsurv_i^l(t) = \prod_{j \in \beta_i^l} BEF_{i,j}^l(t) \times \prod_{k \in \alpha_i^l} AFT_{i,k}^l(t)$$

Notice that this formula also relies on an independence assumption made in Chapter 3: the probability that a solution to the relaxation survives contention is assumed to be given by the product of the probabilities that each one of the reservation assignments in that solution survives contention.

$BEF^l_{i,j}(t)$ is obtained by adding all the subproblem solutions compatible with the precedence constraint $st^l_j + du^l_j \le t$:

$$BEF^l_{i,j}(t) = \sum_{\tau \le t - du^l_j} [surv^l_j(\tau) \times \prod_{p \in \beta^l_j} BEF^l_{j,p}(\tau) \times \prod_{q \in \alpha^l_j} AFT^l_{j,q}(\tau)]$$

Similarly for $AFT^l_{i,k}(t)$, we have:

$$AFT^l_{i,k}(t) = \sum_{\tau \ge t + du^l_i} [surv^l_k(\tau) \times \prod_{s \in \beta^l_k} BEF^l_{k,s}(\tau) \times \prod_{u \in \alpha^l_k} AFT^l_{k,u}(\tau)]$$

We can speed up the computation of this recurrence using partial sums:

$$BEF^l_{i,j}(t) = BEF^l_{i,j}(t-\Delta) + [surv^l_j(t-du^l_j) \times \prod_{p \in \beta^l_j} BEF^l_{j,p}(t-du^l_j) \times \prod_{q \in \alpha^l_j} AFT^l_{j,q}(t-du^l_j)]$$

$$AFT^l_{i,k}(t) = AFT^l_{i,k}(t+\Delta) + [surv^l_k(t+du^l_i) \times \prod_{s \in \beta^l_k} BEF^l_{k,s}(t+du^l_i) \times \prod_{u \in \alpha^l_k} AFT^l_{k,u}(t+du^l_i)]$$

The recurrence is initialized with:

$$BEF^l_{j,p}(est^l_j - \Delta) = 0$$
$$AFT^l_{k,s}(lst^l_k + \Delta) = 0$$

and uses the convention:

$$\prod_{\varnothing} = 1$$

In order to compute $compsurv^l_i(t)$ for all remaining start times of the critical operation O^l_i, the system starts by computing all $BEF^l_{j,p}(t)$ or all $AFT^l_{j,p}(t)$ at the leaf operations in the tree depicted in Figure A-1. The procedure then moves up in the tree by combining at each level the $BEF^l_{j,p}(t)$ and $AFT^l_{j,p}(t)$ computed at the previous level. At each operation O^l_p in the tree, the procedure computes at most λ $BEF^l_{j,p}(t)$ expressions if O^l_p is before O^l_j, its parent operation, or λ $AFT^l_{j,p}(t)$ expressions, if O^l_p is after O^l_j (where λ is the maximum number of possible start times of an operation). Each such computation involves 2 multiplications and 1 addition. Hence, if ν_l is the number of operations in the relaxation used by the FSS value ordering heuristic, computing all $compsurv_i^l(t)$ can be done in $O(\nu_l \lambda)$ elementary computations. Computing $surv_p^l(t) = \sum_{\rho \in G} surv_p^l(\rho)$ for all the possible start times of all the operations in the relaxation requires however $O(\nu_l k)$ steps where k is the maximum number of reservations left to an operation[63]. Hence the overall complexity of the procedure is also $O(\nu_l k)$.

[63]The real complexity is actually $O(\nu_l k\, du)$, where du is the duration of the longest operation in the relaxation. This duration is assumed to be bounded by a constant.

References

[Abramson 89] B. Abramson and M. Yung.
Divide and Conquer under Global Constraints: A Solution to the N-Queens Problem.
Journal of Parallel and Distributed Computing 6(3):649-662, June, 1989.

[Adams 88] J. Adams, E. Balas, and D. Zawack.
The Shifting Bottleneck Procedure for Job Shop Scheduling.
Management Science 34(3):391-401, 1988.

[Allen 83] J.F.Allen.
Maintaining Knowledge about Temporal Intervals.
Communications of the ACM 26(11):832-843, 1983.

[Badie 90] C. Badie, G. Bel, E. Bensana, and G. Verfaillie.
Operations Research and Artificial Intelligence Cooperation to Solve Scheduling Problems: the OPAL and OSCAR Systems.
Technical Report, Centre d'Etude et de Recherches de Toulouse, Toulouse, France, 1990.
Presented at the First International Conference on Expert Planning Systems held in Brighton, United Kingdom.

[Baker 74] K.R. Baker.
Introduction to Sequencing and Scheduling.
Wiley, 1974.

[Baker 90] Kenneth R. Baker and Gary D. Scudder.
Sequencing with Earliness and Tardiness Penalties: A Review.
Operations Research 38(1):22-36, January-February, 1990.

[Bitner 75] J.R. Bitner and E.M. Reingold.
Backtrack Programming Techniques.
Communications of the ACM 18(11):651-655, 1975.

[Burke 89] Peter Burke and Patrick Prosser.
A Distributed Asynchronous System for Predictive and Reactive Scheduling.
Technical Report AISL-42, Department of Computer Science, University of Strathclyde, 26 Richmond Street, Glasgow, GI IXH, United Kingdom, October, 1989.

[Chow 68] C.K. Chow and C.N. Liu.
Approximating Discrete Probability Distributions with Dependence Trees.
IEEE Transactions on Information Theory IT-14(3):462-467, 1968.

[Collinot 88] A. Collinot, C. Le Pape and G. Pinoteau.
SONIA: a Knowledge-based Scheduling System.
International Journal of Artificial Intelligence in Engineering 2(4):86-94, 1988.

[Collinot 90] A. Collinot and C. Le Pape.
Adapting the Behavior of a Job-Shop Scheduling System.
International Journal of Decision Support Systems , 1990.
To appear.

[Dauzere-Peres 90]
S. Dauzere-Peres and J.B. Lasserre.
A Modified Shifting Bottleneck Procedure for Job Shop Scheduling.
Technical Report LAAS 90106, Laboratoire d'Automatique et d'Analyse des Systemes, 7, Av. du Colonel Roche, 31077 Toulouse Cedex, France, 1990.

[Davis 87] Ernest Davis.
Constraint Propagation with Interval Labels.
Artificial Intelligence 32:281-331, 1987.

[Dechter 88] Rina Dechter and Judea Pearl.
Network-Based Heuristics for Constraint Satisfaction Problems.
Artificial Intelligence 34(1):1-38, 1988.

[Dechter 89a] Rina Dechter and Itay Meiri.
Experimental Evaluation of Preprocessing Techniques in Constraint Satisfaction Problems.
In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 271-277. 1989.

[Dechter 89b] Rina Dechter.
Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition.
Artificial Intelligence 41:273-312, 1989.

[Dechter 89c] Rina Dechter and Judea Pearl.
Tree Clustering for Constraint Networks.
Artificial Intelligence 38:353-365, 1989.
Research Note.

[Dechter 89d] Rina Dechter, Itay Meiri, and Judea Pearl.
Temporal Constraint Networks.
In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*. 1989.

[Dechter 90] Rina Dechter, Avi Dechter, and Judea Pearl.
Optimization in Constraint Networks.
Influence Diagrams, Belief Nets, and Decision Analysis.
In R.M. Oliver & J.Q. Smith,
John Wiley and Sons, Ltd., West Sussex, England, 1990.

[Dechter 91] Rina Dechter.
Constraint Networks: A Survey.
Technical Report, Department of Information and Computer Science, University of California at Irvine, Irvine, CA 92717, 1991.
To appear in the Encyclopedia of Artificial Intelligence.

[Dhar 90] Vasant Dhar and Nicky Ranganathan.
Integer Programming vs. Expert Systems: An Experimental Comparison.
Communications of the ACM 33(3):323-336, 1990.

[Dincbas 88] M. Dincbas, H. Simonis, and P. Van Hentenryck.
Solving the Car-Sequencing Problem in Constraint Logic Programming.
In *Proceedings of the 1988 European Conference on Artificial Intelligence*, pages 290-295. 1988.

[Doyle 79] John Doyle.
A Truth Maintenance System.
Artificial Intelligence 12(3):231-272, 1979.

[Elleby 89] P. Elleby, H.E. Fargher and T.R. Addis.
A Constraint-based Scheduling System for VLSI Wafer Fabrication.
Knowledge Based Production Management Systems.
In J. Browne,
Elsevier Science Publishers B.V. (North Holland), 1989.

[Erman 80] Lee D. Erman, Frederick Hayes -Roth, Victor R. Lesser and D. Raj Reddy.
The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty.
Computing Surveys 12(2):213-253, June, 1980.

[Feldman 89] Ronen Feldman and Martin Charles Golumbic.
Constraint Satisfiability Algorithms for Interactive Student Scheduling.
In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence,* pages 1010-1016. 1989.

[Fisher 76] M.L. Fisher.
A Dual Algorithm for the One Machine Sequencing Problem.
Mathematical Programming 11:229-251, 1976.

[Fox 83] Mark S. Fox.
Constraint-Directed Search: A Case Study of Job-Shop Scheduling.
PhD thesis, Department of Computer Science, Carnegie-Mellon University, 1983.

[Fox 86] Mark Fox.
Observations on the Role of Constraints In Problem Solving.
In *Proceedings of the Annual Conference of the Canadian Society for The Computational Studies of Intelligence*, pages 172-187. Montreal, Canada, 1986.

[Fox 87] R.E. Fox.
OPT: Leapfrogging the Japanese.
Just-in-time Manufacture.
In C.A. Voss,
IFS Ltd, Springer Verlag, 1987.

[Fox 89] Mark S. Fox, Norman Sadeh, and Can Baykan.
Constrained Heuristic Search.
In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 309-315. 1989.

[French 82] S. French.
Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop.
Wiley, 1982.

[Freuder 82] E.C. Freuder.
A Sufficient Condition for Backtrack-free Search.
Journal of the ACM 29(1):24-32, 1982.

[Fry 87] Timothy Fry, G. Keong Leong, and Terry R. Rakes.
Single Machine Scheduling: A Comparison of Two Solution Procedures.
OMEGA International Journal of Management Science 15(4):277-282, 1987.

[Fukumori 80] K. Fukumori.
Fundamental Scheme for Train Scheduling.
Technical Report MIT AI Memo No. 596, Massachusetts Institute of Technology, Cambridge, MA, 1980.

[Garey 79] M.R. Garey and D.S. Johnson.
Computers and Intractability: A Guide to the Theory of NP-Completeness.
Freeman and Co., 1979.

[Garey 88] Michael R. Garey, Robert E. Tarjan, and Gordon T. Wilfgong.
One-Processor Scheduling with Symmetric Earliness and Tardiness Penalties.
Mathematics of Operations Research 13(2):330-348, May, 1988.

[Gaschnig 79] John Gaschnig.
Performance Measurement and Analysis of Certain Search Algorithms.
Technical Report CMU-CS-79-124, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, 1979.

[Ginsberg 90] Matthew L. Ginsberg, Michael Frank, Michael P. Halpin, and Mark C. Torrance.
Search Lessons Learned from Crossword Puzzle.
In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 210-215. 1990.

[Goldratt 80] Eliyahu M. Goldratt.
Optimized Production Timetable: Beyond MRP: Something Better is finally Here.
October, 1980
Speech to APICS National Conference.

[Goldratt 86] E. M. Goldratt and J. Cox.
The Goal: A Process of Ongoing Improvement, Revised Edition.
North River Press, Inc, 1986.

[Goldstein 75] Ira Goldstein.
Bargaining Between Goals.
In *Proceedings of the Fourthth International Conference on Artificial Intelligence*, pages 175-180. 1975.

[Golomb 65] Solomon W. Golomb and Leonard D. Baumert.
Backtrack Programming.
Journal of the Association for Computing Machinery 12(4):516-524, 1965.

[Graves 81] Graves, S.C.
A Review of Production Scheduling.
Operations Research 29(4):646-675, July-August, 1981.

[Haralick 80] Robert M. Haralick and Gordon L. Elliott.
Increasing Tree Search Efficiency for Constraint Satisfaction Problems.
Artificial Intelligence 14(3):263-313, 1980.

[Hax 84] Arnoldo C. Hax and Dan Candea.
Production and Inventory Management.
Prentice-Hall, 1984.

[HayesRoth 79] Barbara Hayes-Roth and Frederick Hayes-Roth.
A Cognitive Model of Planning.
Cognitive Science 3:275-310, 1979.

[Jacobs 84] F. Robert Jacobs.
OPT Uncovered: Many Production Planning And Scheduling Concepts Can Be Applied With Or Without The Software.
Industrial Engineering 16(10):32-41, October, 1984.

[Johnson 74] L.A. Johnson and D.C. Montgomery.
Operations Research in Production Planning, Scheduling, and Inventory Control.
Wiley, 1974.

[Johnston 90] Johnston, M.D.
SPIKE: AI Scheduling for NASA's Hubble Space Telescope.
In *Proceedings of the Sixth IEEE Conference on AI Applications, Santa Barbara, California,* pages 184-190. 1990.

[Keng 89] Naiping Keng and David Y.Y. Yun.
A Planning/Scheduling Methodology for the Constrained Resource Problem.
In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence,* pages 998-1003. 1989.

[Kerr 89] Kerr R.M. and Walker R.N.
A Job Shop Scheduling System Based on Fuzzy Arithmetic.
In *Proceedings of the Third International Conference on Expert Systems and the Leading Edge in Production and Operations Management,* pages 433-450. 1989.

[Knoblock 91] Craig A. Knoblock.
Automatically Generating Abstractions for Problem Solving.
PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, 1991.

[Kraitchick 42] M. Kraitchick.
Mathematical Recreations.
Norton, 1942.

[Lawler 82] E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan.
Recent Developments in Deterministic Sequencing and Scheduling: A Survey.
Deterministic and Stochastic Scheduling.
In M.A.H. Dempster, J.K. Lenstra, and A.H.G. Rinnooy Kan,
Reidel, 1982.

[Lawrence 84] Kenneth D. Lawrence and Stelios H. Zanakis.
Production Planning and Scheduling: Mathematical Programming Applications.
Industrial Engineering and Management Press, Institute of Industrial Engineers, Norcross, GA, 1984.

[LePape 87] Claude Le Pape and Stephen F. Smith.
Management of Temporal Constraints for Factory Scheduling.
Technical Report, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1987.
also appeared in Proc. Working Conference on Temporal Aspects in Information Systems, Sponsored by AFCET and IFIP Technical Committee TC8, North Holland Publishers, Paris, France, May 1987.

[LePape 88] Claude Le Pape.
Des Systemes d'Ordonnancement Flexibles et Opportunistes.
PhD thesis, Universite de Paris-Sud, Centre d'Orsay, 1988.

[Lowerre 76] B.T. Lowerre.
The HARPY Speech Recognition System.
PhD thesis, Department of Computer Science, Carnegie-Mellon University, 1976.

[Mackworth 77] A.K. Mackworth.
Consistency in Networks of Relations.
Artificial Intelligence 8(1):99-118, 1977.

[Mackworth 85] A.K. Mackworth and E.C. Freuder.
The Complexity of some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems.
Artificial Intelligence 25(1):65-74, 1985.

[McGregor 79] J.J. McGregor.
Relational Consistency Algorithms and their Applications in Finding Subgraph and Graph Isomorphisms.
Information Sciences 19(3):229-250, 1979.

[Michalski 83] Ryszard Michalski, Jaime G. Carbonell, and Tom M. Mitchell.
Machine Learning: An Artificial Intelligence Approach.
Tioga, Palo Alto, California, 1983.

[Minton 90] S. Minton, M.D. Johnston, A.B. Philips, P. Laird.
Solving Large-Scale Constraint Satisfaction and Scheduling Problems Using a Heuristic Repair Method.
In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 17-24. 1990.

[Montanari 71] Ugo Montanari.
Networks of Constraints: Fundamental Properties and Applications to Picture Processing.
Technical Report, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, 1971.
Also appears in Information Science, 1974, vol. 7,pp. 95-132.

[Morton 88] T.E. Morton, S.R. Lawrence, S. Rajagopolan, S Kekre.
SCHED-STAR: A Price-Based Shop Scheduling Module.
Journal of Manufacturing and Operations Management :131-181, 1988.

[Muscettola 87] Nicola Muscettola, and Stephen Smith.
A Probabilistic Framework for Resource-Constrained Muti-Agent Planning.
In *Proceedings of the Tenth International Conference on Artificial Intelligence*, pages 1063-1066. 1987.

[Muscettola 89] N. Muscettola, S. F. Smith, G. Amiri, and D. Pathak.
Generating Space Telescope Observation Schedules.
Technical Report, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1989.

[Muth 63] J.F. Muth and G.L. Thompson.
Industrial Scheduling.
Prentice-Hall, 1963.

[Nadel 86a] B.A. Nadel.
The General Consistent Labeling (or Constraint Satisfaction) Problem.
Technical Report DCS-TR-170, Department of Computer Science, Laboratory for Computer Research, Rutgers University, New Brunswick, NJ 08903, 1986.

[Nadel 86b] B.A. Nadel.
Three Constraint Satisfaction Algorithms and Their Complexities: Search-Order Dependent and Effectively Instance-specific Results.
Technical Report DCS-TR-171, Department of Computer Science, Laboratory for Computer Research, Rutgers University, New Brunswick, NJ 08903, 1986.

[Nadel 86c] B.A. Nadel.
Theory-based Search-order Selection for Constraint Satisfaction Problems.
Technical Report DCS-TR-183, Department of Computer Science, Laboratory for Computer Research, Rutgers University, New Brunswick, NJ 08903, 1986.

[Nadel 88] Bernard Nadel.
Tree Search and Arc Consistency in Constraint Satisfaction Algorithms.
Search in Articial Intelligence.
In L. Kanal and V. Kumar,
Springer-Verlag, 1988.

[Nemhauser 88] G.L. Nemhauser and L.A. Wolsey.
Integer and Combinatorial Optimization.
Wiley, 1988.

[Nudel 83] Bernard Nudel.
Consistent-Labeling Problems and their Algorithms: Expected-Complexities and Theory-Based Heuristics.
Artificial Intelligence 21:135-178, 1983.

[Ow 85] Peng Si Ow.
Focused Scheduling in Proportionate Flowshops.
Management Science 31(7):852-869, 1985.

[Ow 87] Peng Si Ow and Stephen F. Smith.
Two Design Principles for Knowledge-based Systems.
Decision Sciences 18(3):430-447, 1987.

[Ow 88a] Peng Si Ow and Stephen F. Smith.
Viewing Scheduling as an Opportunistic Problem-Solving Process.
Annals of Operations Research 12:85-108, 1988.

[Ow 88b] P.S. Ow, S.F. Smith, and A. Thiriez.
Reactive Plan Revision.
In *Proceedings of the Seventh National Conference on Artificial Intelligence,* pages 77-82. 1988.

[Ow 89] Peng Si Ow and Thomas Morton.
The Single Machine Early/Tardy Problem.
Management Science 35(2):177-191, 1989.

[Panwalkar 77] S.S. Panwalkar and Wafik Iskander.
A Survey of Scheduling Rules.
Operations Research 25(1):45-61, January-February, 1977.

[Papadimitriou 82]
C.H. Papadimitriou and K. Stieglitz.
Combinatorial Optimization: Algorithms and Complexity.
Prentice-Hall, 1982.

[Pearl 84] Judea Pearl.
Heuristics: Intelligent Search Strategies for Computer Problem Solving.
Addison-Wesley, 1984.

[Pearl 88] Judea Pearl.
Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.
Morgan Kaufmann, 1988.

[Peterson 85] James L. Peterson and Abraham Silberschatz.
Operating System Concepts.
Addison Wesley, 1985.

[Prosser 89] Patrick Prosser.
A Reactive Scheduling Agent.
In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence,* pages 1004-1009. 1989.

[Purdom 83] Paul W. Purdom, Jr.
Search Rearrangement Backtracking and Polynomial Average Time.
Artificial Intelligence 21:117-133, 1983.

[Rinnooy Kan 76] A.H.G. Rinnooy Kan.
Machine Scheduling Problems: Classification, complexity, and computations.
PhD thesis, University of Amsterdam, 1976.

[Rodammer 88] Frederick A. Rodammer, and K. Preston White.
A Recent Survey of Production Scheduling.
IEEE Transactions on Systems, Man, and Cybernetics SMC-18(6):841-851, 1988.

[Sacerdoti 74] E.D. Sacerdoti.
Planning in a Hierarchy of Abstraction Spaces.
Artificial Intelligence 5(2):111-135, 1974.

[Sadeh 88] N. Sadeh and M.S. Fox.
Preference Propagation in Temporal/Capacity Constraint Graphs.
Technical Report CMU-CS-88-193, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, 1988.
Also appears as Robotics Institute technical report CMU-RI-TR-89-2.

[Sadeh 89a] N. Sadeh and M.S. Fox.
Focus of Attention in an Activity-based Scheduler.
In *Proceedings of the NASA Conference on Space Telerobotics*. January, 1989.

[Sadeh 89b] N. Sadeh and M.S. Fox.
CORTES: An Exploration into Micro-Opportunistic Job-Shop Scheduling.
In *Workshop on Manufacturing Production Scheduling*. IJCAI89 - Detroit, 1989.

[Sadeh 89c] Norman Sadeh.
Look-ahead Techniques for Activity-based Job-shop Scheduling.
1989
Thesis Proposal.

[Sadeh 90] Norman Sadeh, and Mark S. Fox.
Variable and Value Ordering Heuristics for Activity-based Job-shop Scheduling.
In *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management, Hilton Head Island, S.C.*, pages 134-144. 1990.

[Serafini 88] P. Serafini, W. Ukovich, H. Kirchner, F. Giardina, and F. Tiozzo.
Job-shop scheduling: a case study.
Operations Research Models in FMS.
In F. Archetti, M. Lucertini, and P. Serafini,
Springer, Vienna, 1988.

[Shaw 90] M.J. Shaw, S.C. Park, and N. Raman.
Intelligent Scheduling with Machine Learning Capabilities: The Induction of Scheduling Knowledge.
Technical Report, Beckman Institute for Advanced Science and Technology, University of Illinois at Urbana-Champaign, Urbana, IL 61801, February, 1990.

[Silver 85] Edward E. Silver and Rein Peterson.
Decision Systems for Inventory Management and Production Planning.
Wiley, 1985.

[Smith 83] Stephen F. Smith.
Exploiting Temporal Knowledge to Organize Constraints.
Technical Report, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1983.

[Smith 86a] Stephen F. Smith, Peng Si Ow, Claude Lepape, Bruce McLaren, Nicola Muscettola.
Integrating Multiple Scheduling Perspectives to Generate Detailed Production Plans.
In *Proceedings 1986 SME Conference on AI in Manufacturing*, pages 123-137. 1986.

[Smith 86b] S. Smith, M. Fox, and P.S. Ow.
Constructing and Maintaining Detailed Production Plans: Investigations into the Development of Knowledge-Based Factory Scheduling Systems.
AI Magazine 7(4):45-61, Fall, 1986.

[Srinivasan 71] V. Srinivasan.
A Hybrid Algorithm for the One Machine Sequencing Problem to Minimize Total Tardiness.
Naval Res. Logist. Quart. 18:317-327, 1971.

[Stallman 77] R. Stallman and G. Sussman.
Forward Reasoning and Dependency-directed Backtracking in a Sysem for Computer-aided Circuit Analysis.
Artificial Intelligence 9:135-196, 1977.

[Stefik 81a] Mark Stefik.
Planning and Meta-Planning (MOLGEN: Part 2).
Artificial Intelligence 16():141-169, 1981.

[Stefik 81b] Mark Stefik.
Planning with Constraints (MOLGEN: Part 1).
Artificial Intelligence 16():111-140, 1981.

[Stone 86] Harold S. Stone and Paolo Sipala.
The average complexity of depth-first search with backtracking and cutoff.
IBM Journal of Research and Development 30(3):242-258, 1986.

[Sussman 80] G.J. Sussman and G.L. Steele Jr.
CONSTRAINTS: A language for expressing almost-hierarchical descriptions.
Artificial Intelligence 14(1):1-39, 1980.

[Sycara 91] Katia Sycara, Stephen Roth, Norman Sadeh, and Mark S. Fox.
Coordinating Resource Allocation in Distributed Factory Scheduling: A Constrained Heuristic Search Approach.
IEEE EXPERT 6(1):29-40, February, 1991.

[Tarjan 83a] Robert Endre Tarjan.
Shortest Paths.
In *CBMS-NSF Regional Conference Series in Applied Mathematics*. Number 44: *Data Structures and Network Algorithms*, chapter 7. Society for Industrial and Applied Mathematics, 1983.

[Tarjan 83b] Robert Endre Tarjan.
Minimum Spanning Trees.
In *CBMS-NSF Regional Conference Series in Applied Mathematics*. Number 44: *Data Structures and Network Algorithms*, chapter 6. Society for Industrial and Applied Mathematics, 1983.

[Vepsalainen 87] Ari P.J. Vepsalainen and Thomas E. Morton.
Priority Rules for Job Shops with Weighted Tardiness Costs.
Management Science 33(8):1035-1047, 1987.

[Voss 87] C.A. Voss.
Just-in-time Manufacture.
IFS Ltd, Springer Verlag, 1987.

[Walker 60] R.J. Walker.
An Enumerative Technique for a Class of Combinatorial Problems.
Combinatorial Analysis, Proc. Sympos. Appl. Math.
In R. Bellman and M. Hall,
American Mathematical Society, Rhode Island, 1960, pages 91-94, Chapter 7.

[Waltz 75] D.L. Waltz.
Understanding Line Drawings of Scenes with Shadows.
The Psychology of Computer Vision.
In P.H. Winston,
McGraw-Hill, New York, 1975.

[Yaglom 64] A.M. Yaglom and I.M. Yaglom.
Challenging Mathematical Problems with Elementary Solutions.
Holden-Day, San Francisco, 1964.

[Zabih 88] Ramin Zabih and David McAllester.
A Rearrangement Search Strategy for Determining Propositional Satisfiability.
In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 155-160. 1988.

Table of Contents

List of Figures

List of Tables