VARIABLE AND VALUE ORDERING HEURISTICS FOR HARD CONSTRAINT SATISFACTION PROBLEMS: AN APPLICATION TO JOB SHOP SCHEDULING

Norman M. Sadeh and Mark S. Fox

CMU-RI-TR-91-23

Center for Integrated Manufacturing Decision Systems
The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

November 1991

This research was supported, in part, by the Defense Advance Research Projects Agency under contract #F30602-88-C-0001, and in part by grants from McDonnell Aircraft Company and Digital Equipment Corporation

Table of Contents

List of Figures

List of Tables

Abstract

Hard Constraint Satisfaction Problems (HCSPs) are Constraint Satisfaction Problems (CSPs) with very large search spaces and very few solutions. Real-life problems such as design or factory scheduling are examples of HCSPs. These problems typically involve several hundred (or even several thousand) variables, each with up to several hundred possible values, only a very tiny fraction of which ultimately allows for a satisfying solution. This paper addresses the issue of how to generate advice to decide which variable to instantiate next (i.e. *variable ordering heuristics*), and which value to assign to that variable (i.e. *value ordering heuristics*) in order to reduce search for a solution. Our investigation is conducted in the domain of job shop scheduling. It is shown that, in this domain, generic CSP heuristics are usually not sufficient to guide the search for a feasible solution. This is because these heuristics fail to properly account for the *tightness* of constraints and/or the connectivity of the constraint graph. Instead, a probabilistic model of the search space is used to define new heuristics, which better account for these problem characteristics. Experimental results indicate that these new heuristics yield important improvements in both search efficiency and search time.

1 Introduction

Hard Constraint Satisfaction Problems (HCSPs) are Constraint Satisfaction Problems (CSPs) with very large search spaces and very few solutions. Real-life problems such as design [26] or factory scheduling [9, 38] are examples of HCSPs. These problems typically involve hundreds of variables, each with up to several hundred possible values, only a very tiny fraction of which ultimately allows for a satisfying solution. This paper addresses the issue of how to generate advice to decide which variable to instantiate next (i.e. *variable ordering heuristics*), and which value to assign to that variable (i.e. *value ordering heuristics*) in order to reduce search for a solution. Our investigation is conducted in the domain of job shop scheduling.

More specifically, we study 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 study requires finding a feasible schedule as fast as possible. An adaptation of the techniques presented in this paper to a Constrained Optimization version of the problem can be found in [38, 39].

The job shop CSP is a well-known NP-complete problem [12]. Accordingly, the worst-case complexity of any procedure to solve this problem is expected to be exponential. CSP techniques that interleave search with consistency enforcing techniques and variable/value ordering heuristics have been reported to yield important increases in search efficiency when applied to other CSPs [15, 11, 32, 20, 6, 25, 46, 7, 10]. One of the aims of this study is to determine if similar savings can be obtained in the case of the job shop CSP, and, more generally, if, *on the average*, the CSP paradigm is sufficient to efficiently solve HCSPs like job shop scheduling. In order to address this difficult question, we first review generic variable and value ordering heuristics that have been reported to perform particularly well on other CSPs. The review suggests that these heuristics are often too weak to solve HCSPs like job shop scheduling. This is because these heuristics fail to properly account for the tightness of constraints and/or for the interactions induced by the high connectivity of the constraint graphs often encountered in job shop scheduling problems[1]. The second part of this paper introduces a probabilistic framework, within which new variable and value ordering heuristics are defined that better account for these interactions. Our study suggests that a key to defining these more powerful heuristics lies in the ability of the probabilistic framework to provide estimates of the *reliance* of a variable on the availability of one of its remaining values (e.g., in job shop scheduling, the reliance of an operation on the availability of a reservation), and measures of *contention* between variables for the allocation of incompatible values (e.g., in job shop scheduling, measures of *resource*

---

[1]Constraint graphs are graphical representations of binary CSPs (i.e. CSPs with binary constraints) in which each variable is represented by a node, and binary constraints are represented by arcs between two nodes.

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 job shop CSPs. This work also shows that, despite its exponential worst-case complexity, the job shop CSP admits many instances that can be solved efficiently. There remain however some particularly difficult problems that require larger amounts of search.

Last but not least, this study strongly suggests that benchmark problems often used in the CSP literature are not representative of HCSPs like job shop scheduling. It is hoped that this work will prompt researchers in the field to look for new benchmark problems and new more powerful heuristics for these problems.

Section 2 of this paper provides a formal definition of the job shop scheduling CSP. Section 3 details the backtrack search procedure used in our study. Shortcomings of popular variable and value ordering heuristics are respectively discussed in Sections 4 and 5. Section 6 describes new variable and value ordering heuristics based on a probabilistic model of the search space. The complexity of these heuristics as well as that of the overall approach are discussed in Section 7. Experimental results comparing our new heuristics with other heuristics discussed in this paper are presented in Section 8. Section 9 summarizes the paper, and further discusses the implications of our study.

Earlier variations of the techniques presented in this paper are discussed in [33, 34, 35, 36, 10, 37].

2 The Job Shop Constraint Satisfaction Problem

The job shop CSP 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 study assumes job shop CSPs with *tree-like* process routings. A tree-like process routing is one whose graph of precedence constraints forms a tree[2]. Two examples of tree-like process routings are depicted in Figure 1.

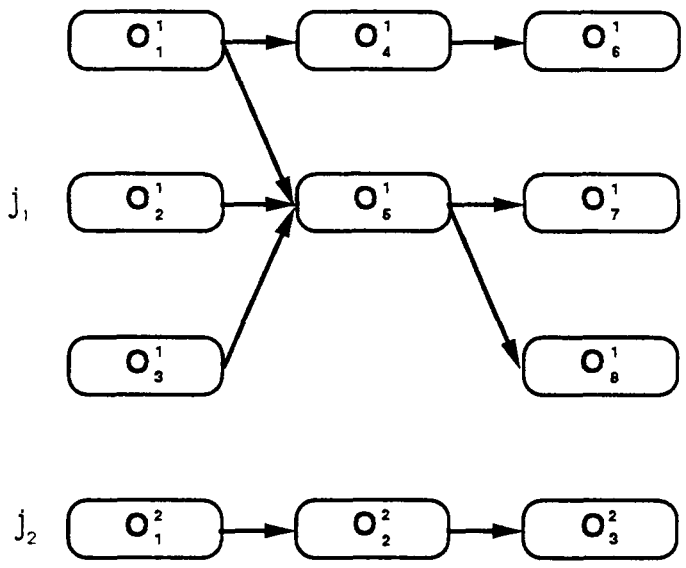

Figure 1: Examples of tree-like process routings.

In the job shop CSP studied in this paper, 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 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, $\Omega^l_{ij}=\{r^l_{ij1},...,r^l_{ijq_{ij}}\}$, 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:

---

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

VARIABLES:

The variables of the problem are:

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

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

In our search procedure, each operation is considered an aggregate variable (or vector) consisting of the start time of the operation and each one of its resource requirements.

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: $st_i^l + du_i^l \le st_j^l$ (i.e. O_i^l BEFORE O_j^l);

2. **Capacity constraints** that restrict the use of each resource to only one operation at a time translate into disjunctive constraints of the form: $(\forall p \forall q\ R_{ip}^k \ne R_{jq}^l) \vee st_i^k + du_i^k \le st_j^l \vee st_j^l + du_j^l \le st_i^k$. These constraints simply express that, unless they use different resources, two operations O_i^k and O_j^l cannot overlap[3].

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_{ij}^l has to be selected from a set of resource alternatives, $\Omega_{ij}^l \subseteq RES$.

OBJECTIVE:

In the job shop CSP studied in this paper, 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 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 example, each operation has a single resource requirement with a single possible value. Operation start times are the only variables. For the

---

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

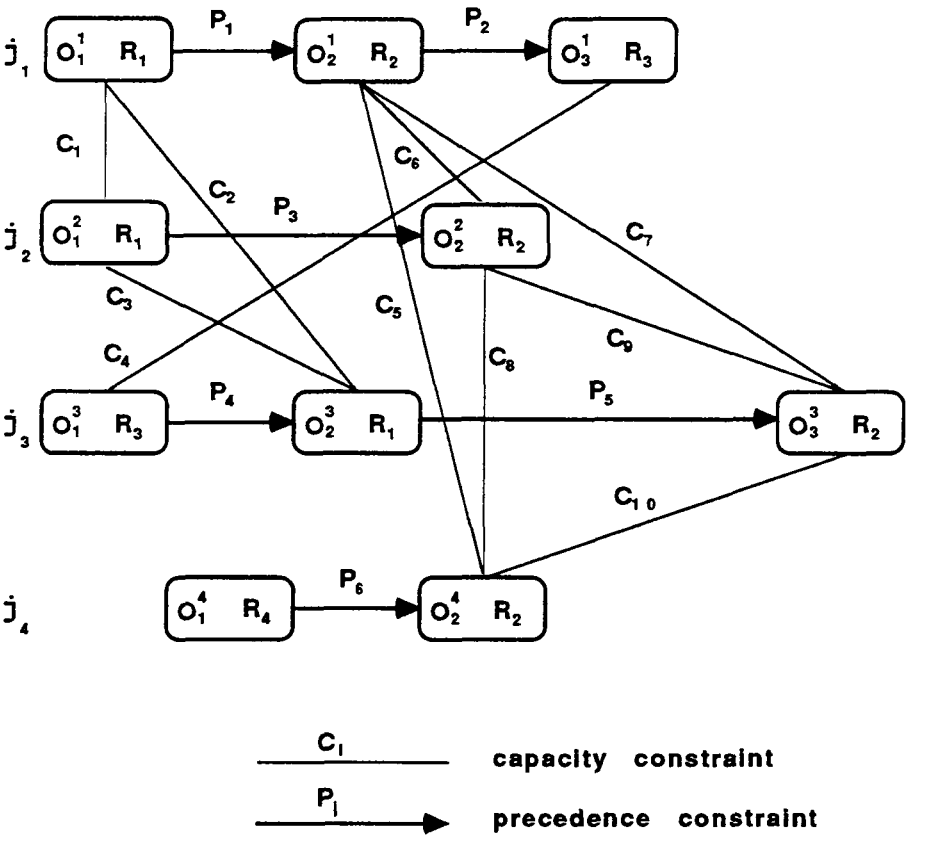

Figure 2: A simple job shop problem with 4 jobs. Each node is labeled by the operation that it represents and the resource required by this operation.

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[4]). 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 each of 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 is

[4]The makespan of a schedule is the length of the time interval that spans from the earliest operation start time to the latest operation end time [2].

expected to be a small bottleneck[5].

[5]Informally, a bottleneck is a resource or group of resources whose utilization is expected to be close to or larger than its available capacity.

3 The Search Procedure

A general paradigm for solving CSPs relies on the use of depth-first backtrack search [43, 14, 3, 30]. Within this context, variables or groups of variables (i.e. subproblems) are successively instantiated (i.e. assigned a value). Each time a new variable (or group of variables) is instantiated, a new search state is created that corresponds to a new, more complete, partial solution. This process goes on until either a complete solution is obtained or until a so-called deadend state is reached that cannot be completed without violating one or several problem constraints. In the latter case, the system needs to undo one or several assignments and try alternative ones, if there are any left (otherwise the problem is infeasible). This process of undoing earlier assignments is known as *backtracking*. It results in lower search efficiency, and, hence, is undesirable.

In the *worst case*, for NP-complete CSPs such as the job shop scheduling CSP, exponential amounts of backtracking may be necessary to come up with a feasible solution (schedule). In practice, as demonstrated by the experimental study presented in this paper, it is generally possible to maintain the *average complexity* of the procedure at a very low level. This is achieved by interleaving search with the application of consistency enforcing techniques and variable/value ordering heuristics:

- *Consistency Enforcing (Checking) Techniques*: These techniques are meant to prune the search space by eliminating local inconsistencies that cannot participate in a global solution [20]. 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.

- *Variable/Value Ordering Heuristics*: These heuristics are concerned with the order in which variables are instantiated and values assigned to each variable. As discussed in the remainder of this paper, these heuristics can have a great impact on search efficiency.

Concretely, for job shop scheduling CSPs, the search procedure starts in a search state in which no operation has been scheduled yet, and proceeds by scheduling one operation (i.e. aggregate variable) at a time:

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. Select the next operation to be scheduled (so-called **variable ordering** heuristic);

5. Select a promising reservation for that operation (so-called **value ordering** heuristic)

6. Create a **new search state** by adding the new reservation assignment to the current partial schedule. Go back to 1.

The results reported in this study were obtained using a simple chronological backtracking scheme.

Clearly there is a tradeoff between the time spent enforcing consistency in each search state and the actual savings achieved in search time. As with other CSPs, on the average, it is not a good idea to seek very high levels of consistency in a search state [22, 15, 28, 20, 6, 7, 25] . However, while for many CSPs simply achieving arc-consistency [44, 19] in a *forward checking* [15, 25] fashion appears to be optimal, job shop scheduling generally entails slightly higher levels of consistency with respect to precedence constraints[6]. Indeed, 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 [41]. As in PERT/CPM [16], 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[7]. It turns out that this method actually guarantees decomposability [8] [5, 8]. 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 [18]. In other words,

---

[6]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 [21, 15]. 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.

[7]See also [40] for an incremental version of this procedure, as new operations are scheduled.

[8]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 [8]. Decomposability is sufficient to ensure backtrack-free search.

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. Additionally, in order to rapidly catch some capacity constraint violations, it was found useful to add a set of redundant capacity constraints to the problem formulation. Short of enforcing full arc-consistency, these constraints, which are described in Appendix A, efficiently enforce a higher level of arc-consistency than simple forward checking.

Because,it is only possible to efficiently enforce partial consistency with respect to capacity constraints, backtracking will sometimes occur. In other words, the scheduling procedure will sometimes reach 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. Notice, however, that because consistency enforcement with respect to precedence constraints is sufficient to guarantee decomposability (with respect to these constraints), backtracking can only occur as the result of capacity constraint violations.

Because it is impossible to efficiently guarantee backtrack-free search for job shop CSPs, variable and value ordering heuristics are generally critical in determining the actual complexity of the search procedure. The next two sections examine popular variable and value ordering heuristics developed for generic CSPs as well as more specialized heuristics and study their applicability to HCSPs such as job shop scheduling.

4 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.

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. [7]). 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 [32]. For these more difficult problems, it has often been suggested that a simple heuristic known as the **Dynamic Search Rearrangement** heuristic (DSR) would generally be sufficient [3, 32, 7, 13]. 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 paper 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 depicts the problem after the application of the consistency enforcing procedure described in section 3.

According to DSR, there are six operations that are equally good candidates to be assigned a

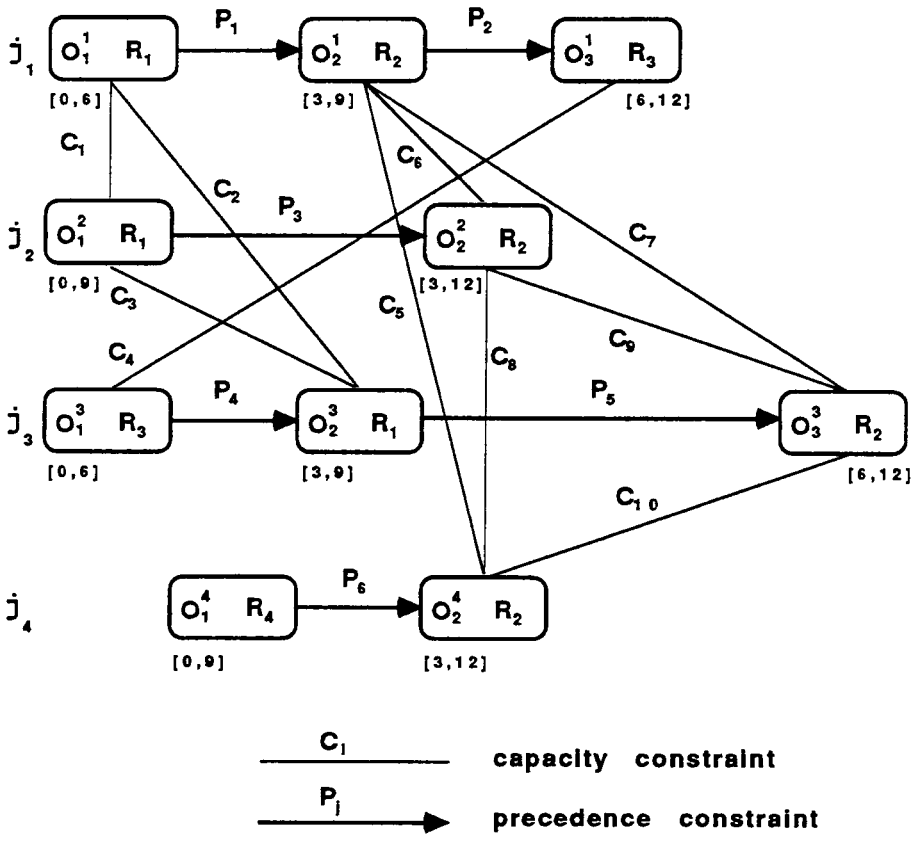

Figure 3: The same job shop CSP after consistency labeling. Start time labels are represented as intervals. For instance, [0,6] represents all start times between time 0 and time 6, as allowed by the time granularity, namely {0,1,2,3,4,5,6}.

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 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 [11, 7]. MW "*orders the variables from last to first by selecting, at each stage, a node in the constraint graph which has a minimal degree*[9] *in the graph remaining after deleting from the graph all nodes which have been selected already*" [7]. A variation of this heuristic known as the **Minimum Degree** (MD) heuristic simply ranks variables according to their degree in the initial constraint graph [7]. In the example depicted in Figure 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 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. We now have:

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

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

The two cliques of capacity constraints corresponding to R_1 and R_3 are now subsumed by a larger clique of capacity constraints[10] involving five operations: O_1^1, O_3^1, O_1^2, O_1^3, and O_2^3 (Figure 4). Due to the additional capacity constraints resulting from the introduction of R_5, there are now MW orderings and MC orderings starting with some of these five operations. In fact the addition of R_5 has significantly loosened the capacity constraints participating in the new clique, and the operations connected by these constraints are even easier to schedule than before. Failure of MW and MC to identify that these operations are actually easy to schedule is due to the inability of these heuristics to account for **constraint tightness**, namely the difficulty of

---

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

[10]Capacity constraints between operations belonging to the same job are subsumed by precedence constraints in that job. For instance, a capacity constraint between O_1^3 and O_2^3, which would require that either O_1^3 precede O_2^3 *or* O_2^3 precede O_1^3, if both operations use R_5. This constraint is subsumed by the precedence constraint between the two operations, which requires that O_1^3 always precede O_2^3.

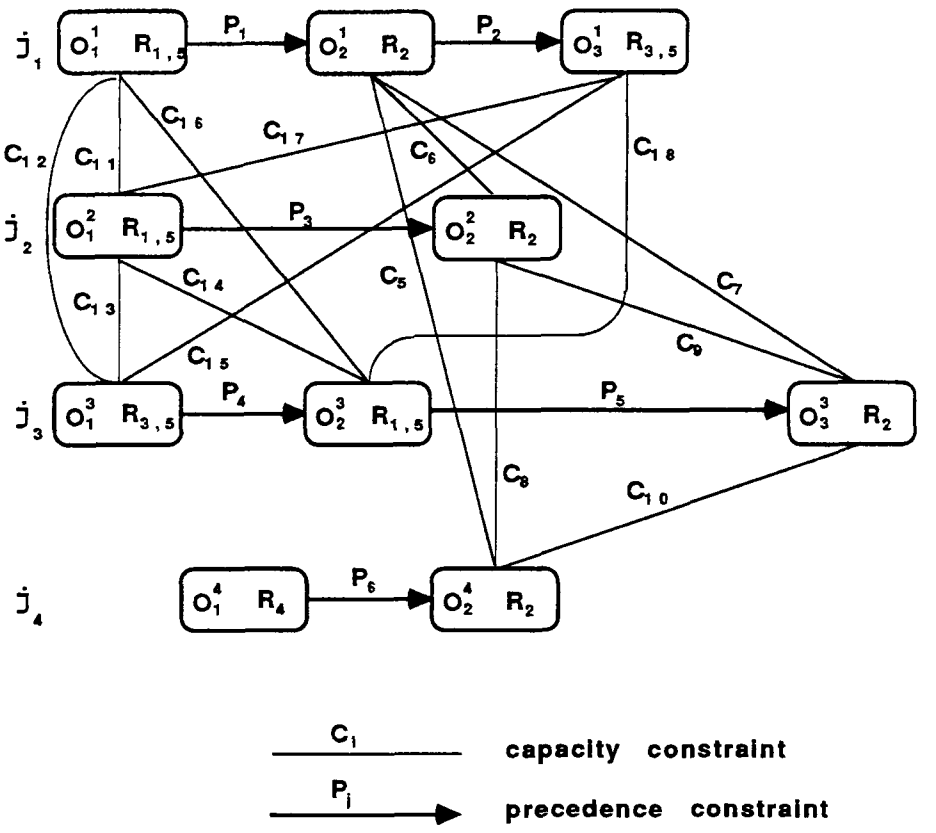

Figure 4: A new resource R_5 is added to the problem. $R_{1,5}$ stands for R_1 or R_5. $R_{3,5}$ stands for R_3 or R_5.

satisfying a specific constraint [24, 10].[11].

Another problem with variable ordering heuristics described in the CSP literature comes from the fact that they treat all problem constraints uniformly. In many real-life CSPs, different types of constraints entail different levels of consistency checking. This in turn determines the types of conflicts that can arise during search, and affects the effectiveness of different variable ordering heuristics. This is the case in job shop scheduling, where, as explained in section 3, consistency enforcing techniques take advantage of particular algebraic properties of precedence constraints to efficiently ensure that backtracking only occurs as a result of capacity constraint violations. Consequently, 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. This observation can be exploited to design more effective variable ordering heuristics.

---

[11]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 [21, 7]. This heuristic can also be seen as a fixed variation of DSR.

A specialized variable ordering heuristic that takes advantage of this observation is the one developed by Keng and Yun [17], 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 paper, 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[12]. In scheduling problems with several hundred operations, each with several 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, and selecting the operation 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.

---

[12]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 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 to it.

5 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**[13], 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.

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 [42] (hence a total of $O(n^3)$ elementary computations). The experimental results presented at the end of this paper 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[14]. 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

---

[13]ABT stands for Advised Backtracking.

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

will provide sufficiently good advice to guide search[15]. This is most likely to be the case with job shop scheduling problems, as explained below with an example.
Consider constraint P_1 in the scheduling problem depicted in Figure 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 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[16]. Using the MST relaxation represented in Figure 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 resource

---

[15]The experiments reported in [6] 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 were obtained on rather small problems with a relatively high density of solutions.

[16]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.

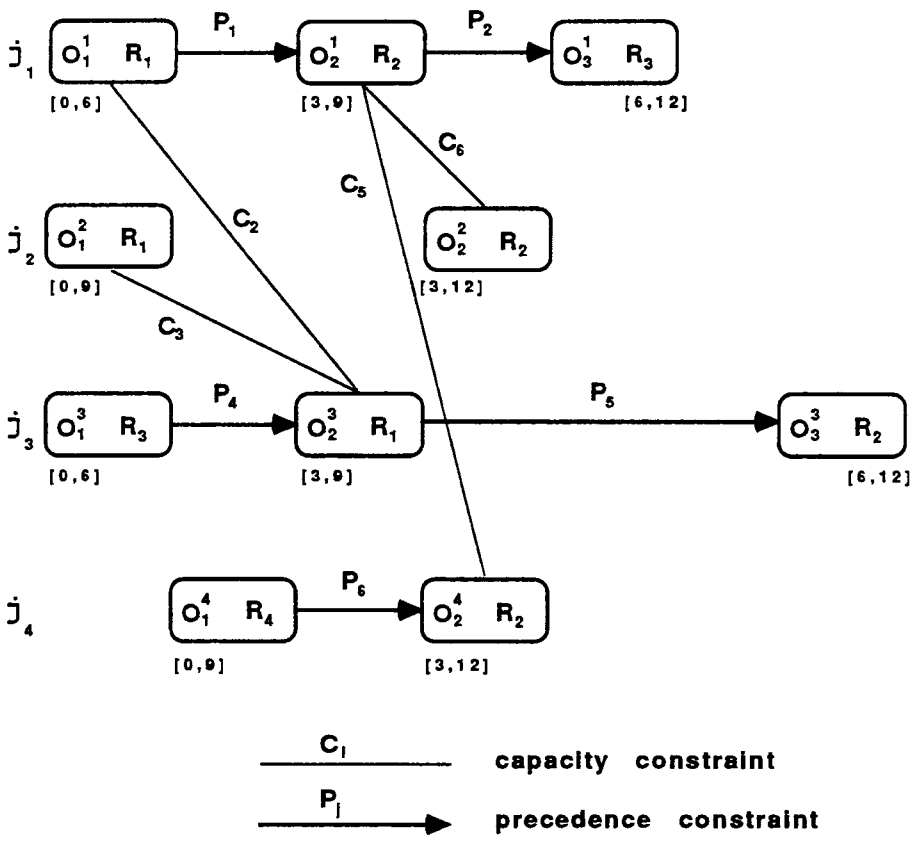

Figure 5: An MST relaxation of the scheduling problem.

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 [17]. 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 paper 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 only accounts for the capacity constraints incident to 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.

6 New Variable and Value Ordering Heuristics

6.1 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.* Assuming that 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 [15][17]. 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.

---

[17]See [15] 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.

6.2 A Probabilistic Model of the Search Space

Critical variables are those expected to cause backtracking. In other words, a critical variable is one whose values are expected to conflict with the values of other variables. In order to approximate variable criticality, a probabilistic framework is described which accounts for the chances that a given value will be assigned to a variable (also a measure of the reliance of a variable on the availability of this value), and the chances that values assigned to different variables conflict with each other (so called value contention measures). By only accounting for those conflicts that will not be prevented by the consistency enforcing procedure, highly effective measures of value contention can be obtained for job shop scheduling CSPs. These measures are then used to determine which variable (i.e. operation) to instantiate next and which value (i.e. reservation) to try first for that variable.

The job shop scheduling consistency enforcing procedure described in section 3 ensures that, in any given search state, the only conflicts that can still occur are capacity constraint violations between currently unscheduled operations. Accordingly, a critical operation in this search procedure, is one whose resource requirements are likely to conflict with the resource requirements of other unscheduled operations. In general, the chances that an operation's resource requirements conflict with those of other unscheduled operations is determined by the number of operations competing for the same reservations and the reliance of each operation on the availability of each one of its possible reservations (i.e. values). Typically, operations with few possible reservations left will heavily rely on the availability of any one of their remaining reservations, whereas operations with a handful of remaining reservations will rely much less on any one of these reservations.

Accordingly, a probabilistic model 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:

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

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 $D_i^l(R_k,\tau)$. It 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 then 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 [17]. The heuristics that will be presented differ from those of Keng and Yun in the way they exploit these demand profiles[18]. Earlier, Muscettola and Smith also proposed techniques to build probabilistic demand profiles, based on a *predefined variable ordering* [23].

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

Consider operation O_2^1 in the initial search state depicted in Figure 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 time 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 $st_2^1=4$. 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 6 shows the individual

---

[18]The work presented here was performed concurrently to that of Keng and Yun [33, 34, 10]. Notice that the interpretation given by Keng and Yun for their demand profiles is not a probabilistic one.

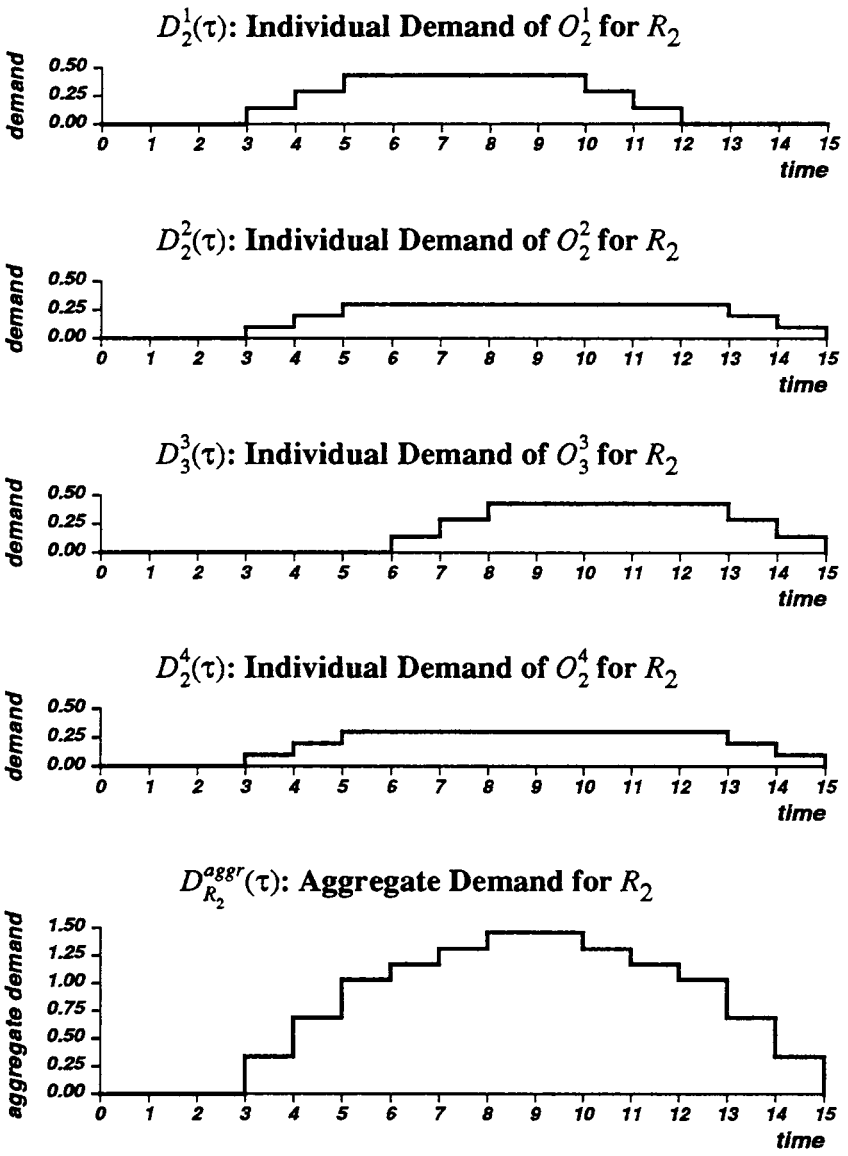

Figure 6: Building R_2's aggregate demand profile in the initial search state.

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.

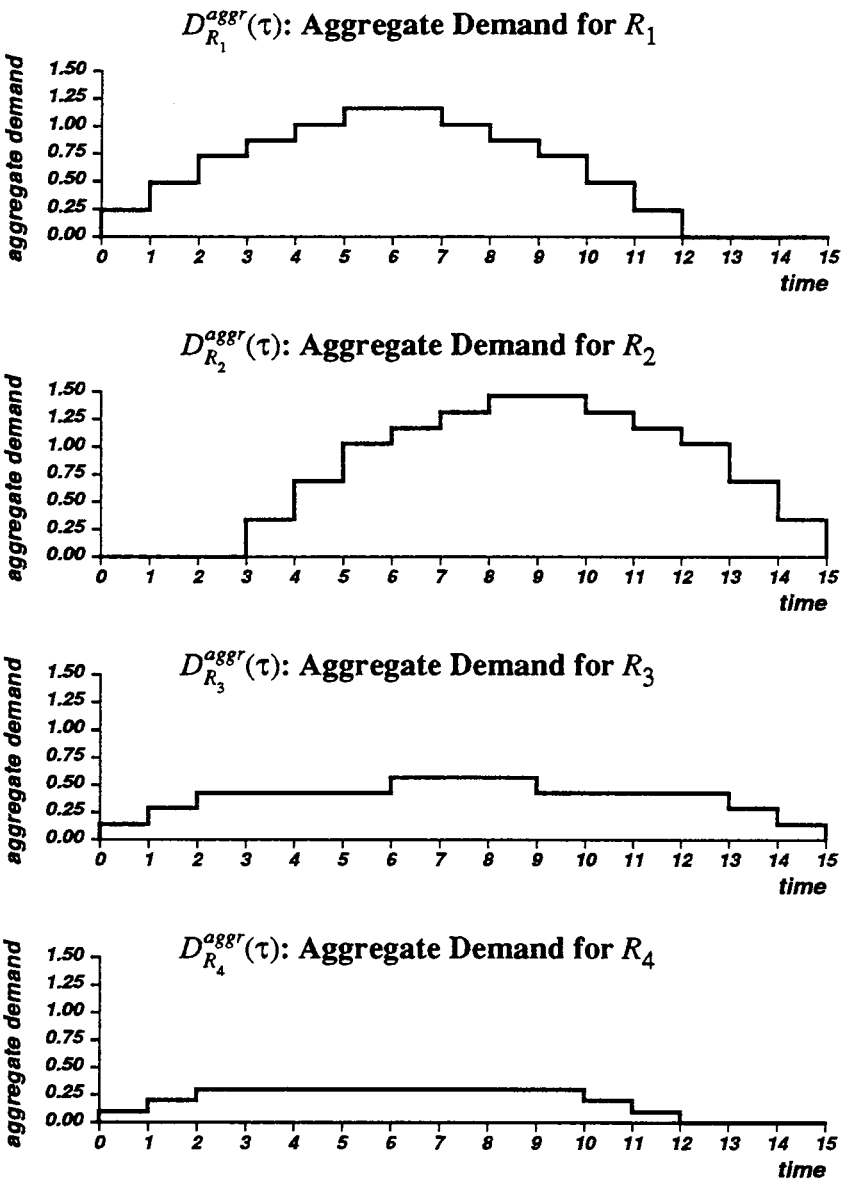

Figure 7: Aggregate demands in the initial search state for each of the four resources.

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

In general, building aggregate demand profiles requires looking at each remaining reservation of each unscheduled operation. Hence, in each search state, the worst-case complexity of the

procedure is $O(Nk)$, 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 demand profiles, and the new individual demands are added instead. It is possible to perform similar updates when the system backtracks.

6.3 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[19]. Accordingly, *the operation with the highest contribution to the demand for the most contended-for 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 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 paper. It will be referred to as **ORR**, which stands for "Operation Resource Reliance" heuristic.

Figure 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 8 indicates that the operation with the largest contribution to that demand is O_3^3. This is no coincidence: O_3^3 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 would have selected O_2^1 as the operation to be scheduled next. In fact, O_3^3 and O_2^2 appear equally critical in this example.

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

---

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

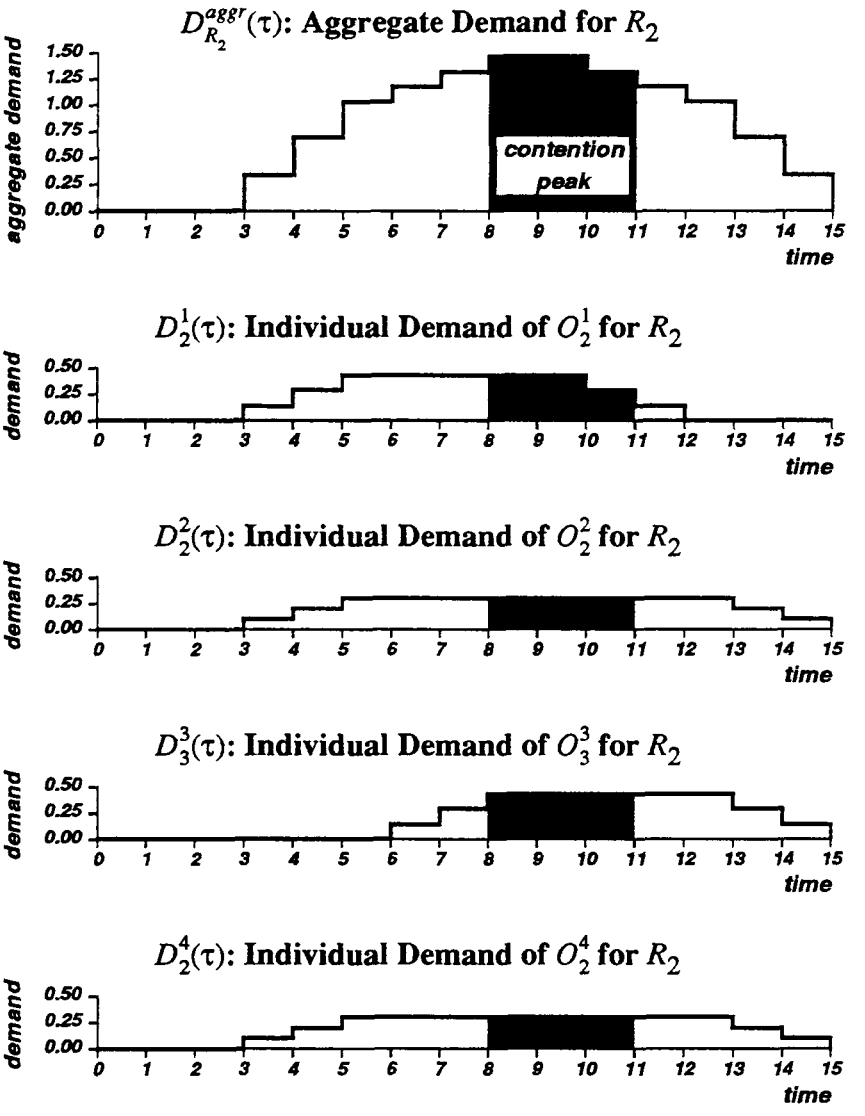

Figure 8: ORR Heuristic: the most critical operation is the one that relies most on the most contended resource/time interval.

6.4 A Value Ordering Heuristic Avoiding Resource Contention

In Section 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 predetermined tree-like relaxations are

comprised of some or all unscheduled operations in the job to which the current critical operation (i.e. the operation to be scheduled next) belongs, along with all precedence constraints between these operations. However, rather than simply counting the number of solutions to the relaxation in which a given reservation assignment participates, *the value ordering heuristic also accounts for the probability that a solution to the relaxation satisfies the cliques of capacity constraints*, thereby making up for the typical lack of information contained in the predetermined relaxation. The probability that a solution (to the relaxation) 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 satisfy capacity constraints (i.e. "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 value ordering heuristic 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 B. This technique is an adaptation of a procedure developed by Dechter and Pearl for the ABT heuristic [6] (see also [31]). It is shown that this technique can be further speeded up by taking advantage of the linearity of precedence constraints.

6.4.1 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 is approximated by the product of the probability that each one of the resources required by that reservation will be available between t and $t+du_i^l$ (independence assumption):

$$surv_i^l(\rho) = \prod_{r_{ijk}^l \in \{r_{i1k_1}^l, r_{i2k_2}^l, \ldots\}} avail_i^l(r_{ijk}^l, t, t+du_i^l) \qquad (1)$$

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 $r^l_{ijk}=R_p \in RES$. The probability $avail^l_i(r^l_{ijk},t,t+du^l_i)$ that resource $r^l_{ijk}=R_p$ will not be required by any other operation between t and $t+du^l_i$ 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[20]. At any time $t \le \tau < t+du^l_i$, there are by definition $n_p(\tau)-1$ unscheduled operations competing with operation O^l_i for resource R_p. The total demand of these other unscheduled operations for R_p at time τ is $D^{aggr}_{R_p}(\tau)-D^l_i(R_p,\tau)$. 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:

$$(1-\frac{D^{aggr}_{R_p}(\tau)-D^l_i(R_p,\tau)}{n_p(\tau)-1})^{n_p(\tau)-1} \tag{2}$$

It is tempting to approximate $avail^l_i(r^l_{ijk},t,t+du^l_i)$, i.e. the probability that $r^l_{ijk}=R_p$ will be available to O^l_i between t and $t+du^l_i$, as the product of the probabilities that R_p will be available to O^l_i on each one of the du^l_i time intervals between t and $t+du^l_i$. In general, this approximation is too pessimistic. It assumes that the operations competing with O^l_i have a duration equal to 1, i.e. that any of these operations could require 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_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 $r^l_{ijk}=R_p$. $avail^l_i(r^l_{ijk},t,t+du^l_i)$ is then approximated as the probability that O^l_i will be able to secure the $\frac{du^l_i}{AVG(du)}$ time buckets that it requires to fit on the resource's calendar. Using Equation (2), this can be approximated as:

$$avail^l_i(R_p,t,t+du^l_i)=(1-\frac{AVG(D^{aggr}_{R_p}(\tau)-D^l_i(R_p,\tau))}{AVG(n_p(\tau)-1)})^{AVG(n_p(\tau)-1)\times\frac{du^l_i}{AVG(du)}} \tag{3}$$

where $AVG(D^{aggr}_{R_p}(\tau)-D^l_i(R_p,\tau))$ and $AVG(n_p(\tau)-1)$ are respectively the average of $D^{aggr}_{R_p}(\tau)-D^l_i(R_p,\tau)$ and the average of $n_p(\tau)-1$ over time interval $[t,t+du^l_i[$.

Figure 9 depicts reservation survivabilities for the three operations in job j_3, the job to which O^3_3 belongs (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 and 7. Consider operation O^3_1.

---

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

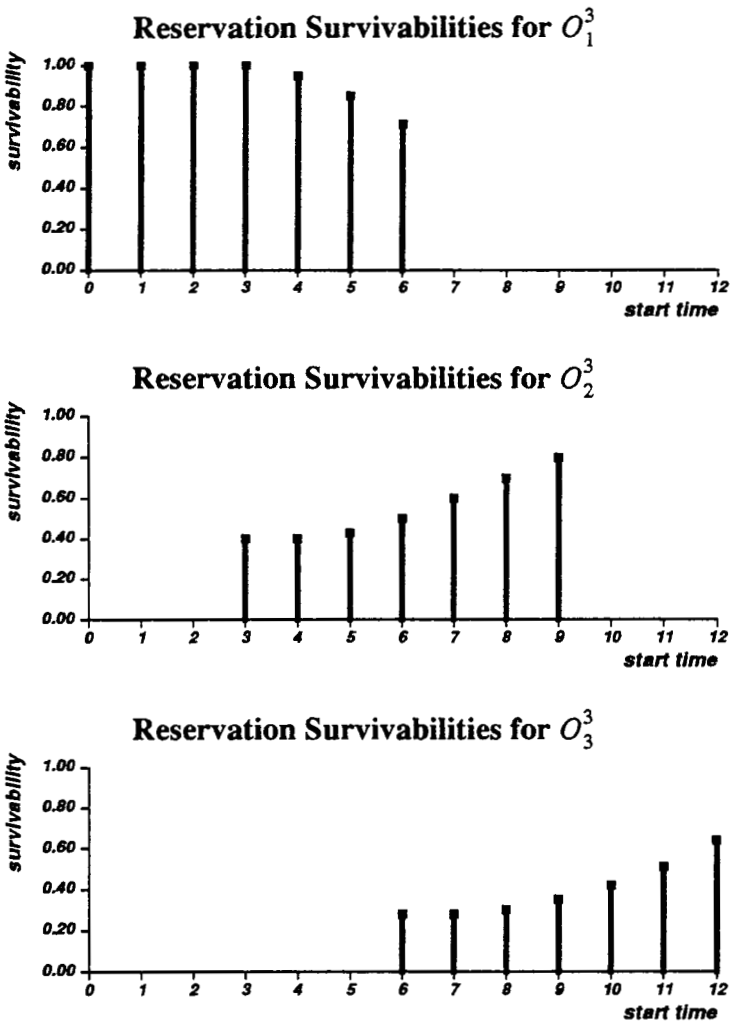

Figure 9: Survivability measures for the reservations of operations in job j_3, the job to which belongs O_3^3, the current critical operation.

Figure 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 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). Since 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).

Operations O_2^3 and O_3^3 compete with more operations than O_1^3. Accordingly, 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.

6.4.2 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 (in short, 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:

$$good_i^l(\rho) = \sum_{sol \in comp_i^l(\rho)} \prod_{O_k^l \in RELAX_i^l} surv_k^l(\rho(O_k^l|sol)) \tag{4}$$

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. Expression (4) can be rewritten to separate the survivability of reservation ρ from that of other operations in $RELAX_i^l$:

$$good_i^l(\rho) = surv_i^l(\rho) \times \sum_{sol \in comp_i^l(\rho)} \prod_{O_k^l \in RELAX_i^l \setminus \{O_i^l\}} surv_k^l(\rho(O_k^l|sol)) \tag{5}$$

This can be further rewritten as:

$$good_i^l(\rho) = surv_i^l(\rho) \times compsurv_i^l(\rho) \tag{6}$$

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:

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

Note that, in fact, $compsurv_i^l(\rho)$ is only a function of the start time st_i^l allocated to O_i^l in reservation ρ an can therefore be written as $compsurv_i^l(t)$

In tree-like process routings, it is possible to evaluate $compsurv_i^l(t)$ for all the possible start times t 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 B. This technique is an adaptation of a procedure described in [6][21]. For non-tree-like process plans, it should be possible to remove a small

[21]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 [1], the complexity of the algorithm would be $O(\nu_l k^2)$.

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.

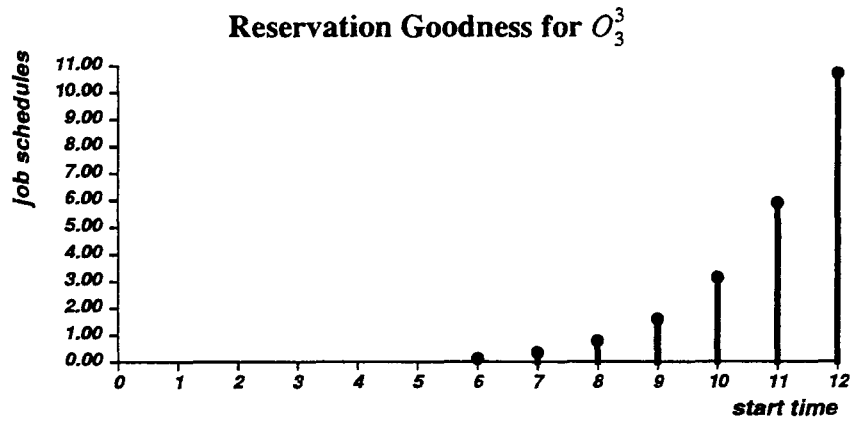

Figure 10: Value goodness for O_3^3 expressed as the number of compatible job schedules expected to survive resource contention.

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 10 displays the goodness measures computed using (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 $st_3^3=7$ 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 survivable solutions to the relaxation. Hence this is the start time selected by the value ordering heuristic. By assigning this start time to O_3^3, and iteratively using the variable and value ordering heuristics that were just described, the search procedure 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.

6.4.3 Further Refinement

For some reservations ρ, $compsurv_i^l(\rho)$ can become very large and have too much influence in (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 (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 paper, 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 (6), our value ordering heuristic measures reservation goodness according to the following revised formula:

$$good_i^l(\rho) = surv_i^l(\rho) \times MIN(\Phi^{\nu_l - 1}, compsurv_i^l(\rho)) \qquad (7)$$

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[22].

---

[22]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.

7 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 $O(NOP^2k)$, 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.

8 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) [3, 15, 32], the ABT (Advised Backtracking) value ordering [6], and the combination of variable and value ordering heuristics developed by Keng and Yun [17] (and referred to as SMU).

8.1 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[23].

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 $(1+S)MU(1-RG,1)$, 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 [29]. Similarly, release dates were randomly drawn from a uniform distribution of the form: $(1+S)M\,U(0,RG)$. Three values of the

[23] 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[24]: $RG = 0.0, 0.1$, 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.

8.2 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 [6] 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 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[25].

---

[24]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.

[25]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 $O(\nu_l k^2)$.

- *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 paper.

- *SMU*: The variable and value ordering heuristics developed by Keng and Yun at the Southern Methodist University [17].

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 500 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.

2. **Number of experiments solved in less than 500 search states each.**

3. **Average CPU time per operation** (in seconds): this is the average CPU time required to *successfully* schedule an operation. If a solution was found, this time was obtained by dividing the total CPU time required to solve the problem by the number of operations to be scheduled. Otherwise, this time was approximated by dividing the CPU time spent exploring 500 search states by the number of operations. All CPU times were obtained on a DECstation 5000 running Knowledge Craft on top of Allegro Common Lisp. Experimentation with an earlier version of the system written in C suggests that the search procedure should run 10 to 20 times faster in that language.

The results are summarized in Table 1. They indicate that DSR is generally not sufficient to solve realistic job shop scheduling problems. Combined with ABT, this heuristic was only able

Performance of Five Heuristics						
		DSR &ABT	DSR &FSS	ORR &ABT	SMU	ORR &FSS
RG=0.2 BK=1	Search Efficiency	**0.72** (0.42)	**0.82** (0.38)	**0.96** (0.06)	**1.00** (0.00)	**0.96** (0.07)
	Nb. exp. solved	**8**	**8**	**10**	**10**	**10**
	CPU seconds	**10.48** (13.91)	**7.60** (10.30)	**1.57** (0.21)	**3.76** (0.28)	**1.77** (0.26)
RG=0.2 BK=2	Search Efficiency	**0.49** (0.40)	**0.73** (0.43)	**0.54** (0.39)	**0.79** (0.38)	**0.99** (0.02)
	Nb. exp. solved	**7**	**7**	**6**	**8**	**10**
	CPU seconds	**17.73** (16.38)	**9.13** (9.78)	**11.33** (11.83)	**7.69** (7.59)	**1.86** (0.15)
RG=0.1 BK=1	Search Efficiency	**0.60** (0.44)	**0.82** (0.38)	**0.79** (0.36)	**0.64** (0.46)	**0.78** (0.36)
	Nb. exp. solved	**7**	**8**	**9**	**6**	**8**
	CPU seconds	**9.47** (9.73)	**5.32** (4.98)	**5.80** (8.32)	**9.29** (7.81)	**6.63** (10.07)
RG=0.1 BK=2	Search Efficiency	**0.22** (0.27)	**0.46** (0.46)	**0.31** (0.37)	**0.71** (0.42)	**0.87** (0.29)
	Nb. exp. solved	**2**	**4**	**4**	**7**	**9**
	CPU seconds	**18.50** (9.20)	**9.66** (6.48)	**18.36** (11.50)	**7.10** (6.03)	**3.68** (5.62)
RG=0.0 BK=1	Search Efficiency	**0.28** (0.38)	**0.32** (0.38)	**0.53** (0.44)	**0.46** (0.46)	**0.73** (0.43)
	Nb. exp. solved	**2**	**3**	**6**	**4**	**7**
	CPU seconds	**17.14** (8.22)	**13.18** (7.58)	**16.64** (16.34)	**12.52** (7.99)	**9.50** (12.81)
RG=0.0 BK=2	Search Efficiency	**0.31** (0.33)	**0.37** (0.43)	**0.46** (0.40)	**0.75** (0.41)	**0.82** (0.38)
	Nb. exp. solved	**3**	**3**	**5**	**8**	**8**
	CPU seconds	**13.59** (10.28)	**12.30** (8.40)	**18.14** (16.60)	**7.67** (8.30)	**6.01** (8.88)
Overall Performance	Search Efficiency	**0.44** (0.41)	**0.58** (0.45)	**0.60** (0.41)	**0.72** (0.41)	**0.86** (0.31)
	Nb. exp. solved	**29**	**33**	**40**	**43**	**52**
	CPU seconds	**14.49** (11.71)	**9.53** (8.23)	**11.97** (13.31)	**8.00** (7.13)	**4.91** (8.07)

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

to solve 29 problems out of 60 in less than 500 search states each. Even when combined with

the FSS value ordering heuristic, DSR only achieved a search efficiency of 58%, and failed to solve 27 problems out of 60 in less than 500 search states. These results not only suggest that job shop scheduling requires a dynamic variable ordering heuristic[26]. 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 16% and 11 additional problems were solved in less than 500 search states each. The SMU heuristic achieved a higher efficiency of 72% and solved 43 problems out of 60 in less than 500 states. Even this heuristic had trouble solving many problems. In fact, it could hardly solve more problems than ORR&ABT. ORR&FSS, the variable and value ordering heuristics advocated in this paper, yielded an impressive 86% search efficiency, and solved 52 problems out of 60 in less than 500 search states[27]. For the 52 experiments that it was able to solve, ORR&FSS never generated more than 78 search states and never took over 150 CPU seconds to solve a problem. This heuristic combination also achieved important speedups over all the other heuristics.

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 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.

---

[26]Experiments reported in [38] were also performed to compare the performance of the ORR heuristic, as it is described here, with less dynamic variations of this heuristic, in which several critical operations were identified at once. These experiments showed that the performance of the variable ordering heuristic quickly degrades as it becomes less dynamic.

[27]In a more recent study, a combination of the ORR and FSS heuristics with two new backtracking schemes solved an even larger number of experiments. A selective dependency-directed backtracking scheme solved 55 problems [45] and a backtracking heuristic that undoes decisions that are not always provably wrong was able to solve all 60 problems.

9 Summary and Conclusions

This paper 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. Traditional 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 [27]. Instead, this study demonstrated that many instances of these problems can be efficiently solved by combining consistency enforcing techniques and look-ahead techniques to decide which operation to schedule next and which reservation to assign to that operation.

Several variable and value ordering heuristics to guide the search procedure 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 a variable (e.g. operation) on the availability of a value (e.g. reservation), and the degree of contention among uninstantiated variables for the assignment of conflicting values (e.g. contention among unscheduled operations for the allocation 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 can *efficiently* solve many job shop scheduling problems that could not be efficiently solved by prior CSP heuristics (both generic CSP heuristics and specialized heuristics designed for similar scheduling problems). The results indicate that the ORR and FSS heuristics not only yield significant increases in search efficiency but also achieve important reductions in search time. Finally, the measures of value reliance and contention presented in this paper have also been adapted to solve optimization version of the job shop scheduling problem [38, 39].

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 [33, 34, 37] others relying on Monte Carlo simulations [35]. The increase in search efficiency generally achieved by these more sophisticated versions did not seem to justify their heavier computational requirements.

Because the job shop scheduling CSP is NP-complete, no heuristic can be expected to efficiently solve all instances of this problem. 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 search procedure[28].

The heuristics presented in this paper 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[29]). 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 to 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 study will prompt researchers in the field to look for new benchmark problems and new more powerful

---

[28]More recent experiments in which the ORR and FSS heuristics were combined with a heuristic backtracking scheme that undoes decisions that are not always provably wrong were in fact able to efficiently solve all 60 problems of this study. See also [45] for experiments with another backtracking scheme.

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

heuristics for these problems.

The heuristics discussed in this paper all share a common weakness: they always perform the same analysis independently of the problem that they are presented with, 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 search had reached a state in which backtracking could no longer occur. Like ORR, the variable ordering heuristic implemented in that version of the system was particularly good at quickly reducing contention. As a consequence, search states where backtracking could no longer occur were generally reached after 50% to 80% of the operations had been scheduled. At that point, the system would arbitrarily complete the schedule (i.e. using arbitrary variable and value orderings), which generally resulted in important speedups[30]. A similar dynamic switch could also be implemented in the current version of the system. 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 has been reported by Collinot and Lepape [4] and Xiong, Sadeh, and Sycara [45].

---

[30]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.

Appendix A. Redundant Capacity Constraints

In order to rapidly detect the violation of capacity constraints, it was found useful to add a set of redundant capacity 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 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 I-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 I-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[31].

[31]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, they would further reduce total processing time.

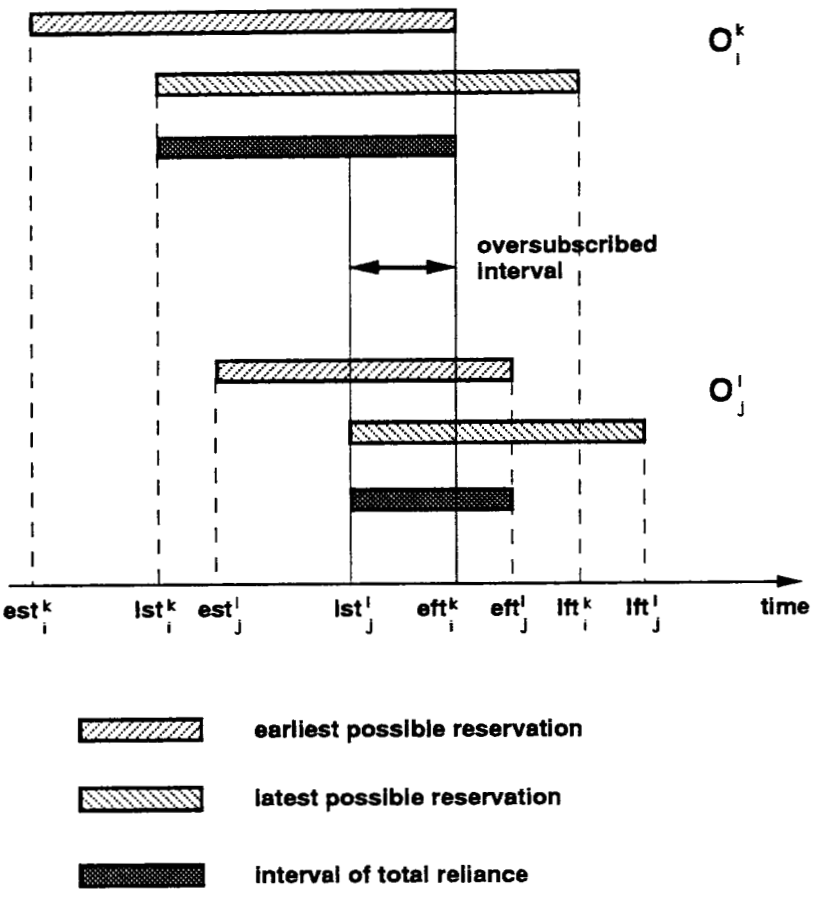

Figure I-1: A situation with an oversubscribed resource that can easily be detected.

Appendix B. Counting the Number of Survivable Schedules

This appendix describes a dynamic programming procedure that efficiently counts the number of survivable job schedules (or more generally the number of survivable solutions to the relaxation defined in Section 6.4 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 $compsurv_i^l(t)$, where t is the start time allocated to O_i^l in reservation ρ, The procedure presented here is a variation of a similar method developed by Dechter and Pearl for the ABT value ordering heuristic [6] (see also [31]). 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 $O(\nu_l k)$.

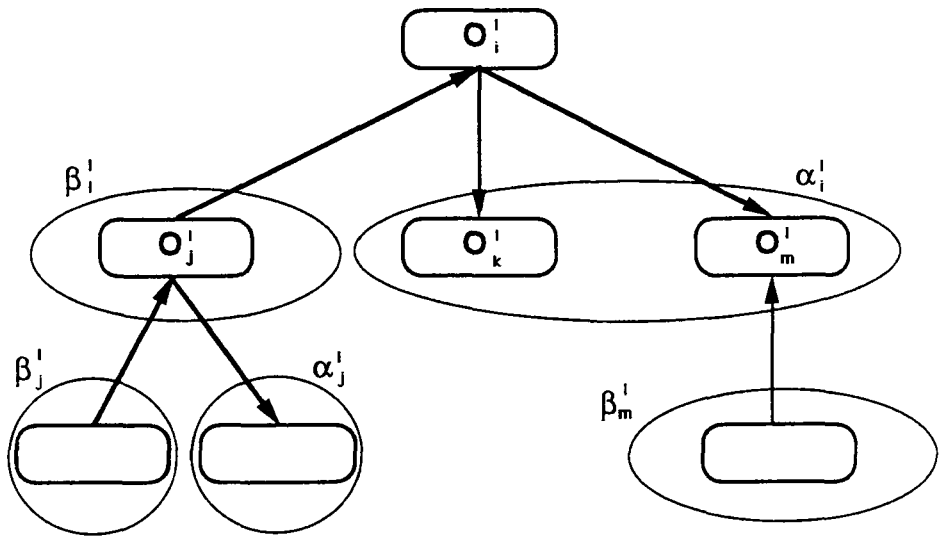

Figure II-1: A tree-like process routing, organized with the current critical operation as its root. Arrows represent precedence constraints.

Figure II-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 Section 6.4. 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;

- 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 Section 6.4, 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 Section 6.4: 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.

$BEF_{i,j}^l(t)$ is obtained by adding all the subproblem solutions compatible with the precedence constraint $st_j^l+du_j^l\le t$:

$$BEF_{i,j}^l(t)=\sum_{\tau\le t-du_j^l}[surv_j^l(\tau)\times\prod_{p\in\beta_j^l} BEF_{j,p}^l(\tau)\times\prod_{q\in\alpha_j^l} AFT_{j,q}^l(\tau)]$$

Similarly for $AFT_{i,k}^l(t)$, we have:

$$AFT_{i,k}^l(t)=\sum_{\tau\ge t+du_i^l}[surv_k^l(\tau)\times\prod_{s\in\beta_k^l} BEF_{k,s}^l(\tau)\times\prod_{u\in\alpha_k^l} AFT_{k,u}^l(\tau)]$$

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 II-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 $compsurv^l_i(t)$ can be done in $O(\nu_l \lambda)$ elementary computations. Computing $surv^l_p(t) = \sum_{\rho \in G} surv^l_p(\rho)$ 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[32]. Hence the overall complexity of the procedure is also $O(\nu_l k)$.

[32]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

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

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

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

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

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

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

[7] 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.

[8] 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.

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

[10] 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.

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

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

[13] 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.

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

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

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

[17] 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.

[18] 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.

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

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

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

[26] D. Navinchandra.
Exploration and Innovation in Design.
Springer Verlag, 1990.

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

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

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

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

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

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

[33] 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.

[34] 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.

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

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

[37] 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.

[38] Norman Sadeh.
Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling.
PhD thesis, School of Computer Science, Carnegie Mellon University, March, 1991.

[39] Norman Sadeh.
MICRO-BOSS: A Micro-opportunistic Factory Scheduler.
Expert Systems with Applications: An International Journal , 1991.
To Appear in Special Issue on Scheduling Expert Systems and their Performances. Also published as Carnegie Mellon University technical report CMU-RI-TR-91-22.

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

[41] 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.

[42] 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.

[43] 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.

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

[45] Yalin Xiong, Norman Sadeh, and Katia Sycara.
Intelligent Backtracking Techniques for Job Shop Scheduling.
Technical Report, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1992.
Working Paper.

[46] 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.