Scheduling by Iterative Partition of Bottleneck Conflicts

Nicola Muscettola
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
ARPA net: mus@cs.cmu.edu

Abstract

In this paper we describe Conflict Partition Scheduling (CPS), a novel methodology that constructs solutions to scheduling problems by repeatedly identifying bottleneck conflicts and posting constraints to resolve them. The identification of bottleneck conflicts is based on a capacity analysis using a stochastic simulation methodology. Once a conflict is identified, CPS does not attempt to resolve it completely; instead it introduces constraints that merely decrease its criticality. By reducing the amount by which each scheduling decision prunes the search space, CPS tries to minimize the chance of getting lost in blind alleys. The effectiveness of CPS is demonstrated by an experimental analysis that compares CPS to two current scheduling methods: micro-opportunistic constraint-directed search and min-conflict iterative repair. CPS is shown to perform better than both on a standard benchmark of constraint satisfaction scheduling problems.[1]

- **AI topic**: temporal reasoning, constraint satisfaction problems, search space metrics
- **Domain area**: scheduling
- **Language/Tool**: Common Lisp, CRL
- **Status**: prototype fully operational
- **Effort**: 1 person/year
- **Impact**: 15% to 160% increase in the number of scheduling problems solved with respect to competing scheduling methods

[1]This work was sponsored in part by the National Aeronautics and Space Administration under contract # NCC 2-707, the Defense Advanced Research Projects under contract #F30602-88-C-001, and the Robotics Institute.

1 Introduction

Scheduling is one of the combinatorial problems that most stubbornly resists effective computational solutions. In a nutshell, building a schedule means assigning time values to a set of variables that represent the start and end times of a given set of activities. A value assignment is a schedule only if it is consistent with a given set of temporal constraints (e.g., release and due dates), a given set of activity precedences (e.g., activity α_i must end before activity α_j can start) and a given set of resource capacity constraints (e.g., if α_i and α_j require the use of the same resource, they cannot be in process at the same time). Even without the additional requirement to optimize given performance criteria, scheduling problems are known to be very hard.

In this paper we describe Conflict Partition Scheduling (CPS), a novel methodology for solving scheduling problems. CPS repeatedly identifies sets of activities that are potentially in competition for the use of a resource, and adds constraints to insure that no conflict will actually arise. CPS draws its power from the combination of precedence constraint posting, the primary problem solving operator, and the detection of conflicting activities on the basis of a stochastic simulation method. We demonstrate the viability of the approach by reporting the results of an experimental analysis comparing CPS with two other schedulers using alternative methodologies: micro-opportunistic search [12] and min-conflict iterative repair [7]. CPS performs better with respect to both number of problems solved and running time.

In CPS, the initial problem, any intermediate state, and the final schedule are each represented as a network of metric temporal constraints [4]. Posting activity precedences (e.g., α_i must precede α_j) is the primary means to enforce resource capacity constraints. The range of variability for each variable assignments can be easily deduced from the constraint network.

Other scheduling approaches used to solve a simple version of the job-shop scheduling problem [1] [2] use a similar representation; however, CPS is applicable to a wider range of network scheduling problems that include preferences and absolute temporal constraints. An alternative approach to scheduling consists of strictly operating in the space of value assignments for the problem variables. Here, the primary problem solving operator is the assignment of a single value to one of the variables (e.g., α_i must start at time t_0). Currently, this is the most widely used approach to scheduling, including simulation-based dispatching scheduling [11], search-based and opportunistic heuristic scheduling [5] [13] [12], and iterative repair scheduling.[7] [14] [3]. The temporal flexibility of the precedence constraint posting approach gives potential advantages during the generation of a schedule. Each time the value of a problem variable is fixed, the search space (i.e., the set of all the possible assignments of start and end times for the remaining unscheduled operations) loses one dimension. Alternatively, posting a constraint only restricts the range of the problem variables without necessarily a decrease of dimensionality, and therefore potentially leaves open a greater number of variable assignments. This suggests a lower risk for the scheduling algorithm to get lost in blind alleys.

Using a constraint posting approach, however, does not decrease the exponential order of complexity of the search space; we still need heuristics to direct the introduction of new constraints where they are most likely to be effective. Ideally one would like to estimate the distribution of solutions over the search space in order to move toward areas with high solution density. Obviously this cannot be done by actually generating solutions. The method used in CPS follows a reverse approach; first it identifies areas where there is high likelihood of not finding a solution and then it posts constraints to move away from them. As in most of the current bottleneck-guided scheduling approaches [1] [13] [12] [3], CPS computes its focusing metric on a relaxed version of the original problem space. The relaxation hypothesis ignores the limitations of resource capacity but takes into account all the job precedences, absolute time constraints, and preferences of the initial problem formulation, and all the additional constraints introduced so far by the procedure. Given the lack of disjunctive capacity constraints, generating consistent value assignments is very fast. CPS stochastically generates a sample of value assignments out of which it then compiles statistics which give heuristic measures of the current problem state.

In the rest of the paper, we give the details of the schedule representation and of the problem solving procedure used by CPS (section 2), we briefly describe the two competing methods and give the results of the experimental analysis (section 3), and, finally, we report conclusions and future directions (section 4).

2 The CPS Methodology

CPS is implemented in the HSTS planning and scheduling framework [10] which has already been applied to the solution of complex planning and scheduling problems [10]. A typical intermediate problem solving state, stored in HSTS's temporal data base, will contain a set of activities, each of which has a duration constraint and a required resource, and a set of precedence constraints of the type α_i *before* α_j, indicating that the start time of α_j has to occur after the end time of α_i. Some *before* constraints were present in the original problem formulation while others have been added by previous iterations of CPS. Additional absolute temporal constraints (e.g., release and due dates) can restrict the range of variability for the start and end times of the activities. In the following we will identify with A and R respectively the sets of all the activities and of all the resources in the problem. Underlying the activity network there is a graph $< V_t, C_t >$ of time variables V_t (i.e., the start and end times of all the activities) and metric interval distances C_t, the **time points network**. Time propagation allows deduction of the range of possible time values for each activity start and end [4]. We will identify with $EST(\alpha)$ the lower bound of the possible values of α's start time and with $LFT(\alpha)$ the upper bound of α's end time. A given activity α is allowed to be in process on a resource at any time $EST(\alpha) \leq t < LFT(\alpha)$. In the following, we will assume discrete time and identify with H the problem horizon, i.e., the segment of time within which the occurrence of any activity in A must fall.

The outline of the basic CPS procedure is the following:

1. **Capacity Analysis**: estimate activity demand and resource contention.

2. **Termination Test**: If the resource contention is zero for each resource over the entire scheduling horizon, then exit. The current activity network is the solution.

3. **Bottleneck Detection**: Identify resource and time with the highest contention;
4. **Conflict Identification**: Select the activities that are most likely to contribute to the bottleneck contention.
5. **Conflict Partition**: Sort the set of conflicting activities by inserting appropriate temporal constraints according to the activity demand.
6. **Constraint Propagation**: Update the time bounds in the time point network as a consequence of the introduction of the new temporal constraints.
7. **Consistency Test**: If the time point network is inconsistent, exit signaling the inconsistency.
8. Go to 1.

The procedure is strictly monotonic. If it generates an inconsistency, the activity network is reset to the initial state and the procedure is repeated. This is feasible for the stochastic nature of the capacity analysis step. If after a fixed number of repetitions a solution has not been found, CPS terminates with an overall failure. The general structure of the problem solving cycle is similar to other heuristic scheduling approaches. [1] [13] [12] The following subsections discuss the individual steps of the procedure in detail.

2.1 Stochastic Capacity Analysis

The Capacity Analysis extends and generalizes the one first proposed in [9]. The logic behind the method is quite simple. While completing an intermediate state into a consistent schedule is difficult for the presence of unresolved disjunctive capacity constraints, it is straightforward to generate total value assignments that satisfy the temporal constraints already in the network. [4] In the process, it is also easy to attend to additional preference criteria (e.g., select times as soon as possible). For each of such assignments it is possible to identify where the unresolved capacity constraints have been violated (i.e., more than one activity requiring the same resource occurs at the same time).

The capacity analysis generates a set of N total assignments by running N times an iterative **stochastic simulation** process. Given the time points network $< V_t, C_t >$ the following steps are repeatedly applied until all variables in V_t have a value:

1. select a variable $v_t \in V_t$ according to a predefined *variable selection strategy*;
2. select a value for v_t within the current range of its possible values, according to a *stochastic value selection rule*;
3. assign the value to v_t and propagate the consequences throughout the time point network; the results are new ranges of possible values for the variables in V_t;
4. delete t from V_t.

At the end of each stochastic simulation for each activity we record the time interval of occurrence, and for each resource and each instant of time within the scheduling horizon H we detect if the number of activities that require the use of the resource exceed its available capacity at that time. On the basis of the sample constituted by the N total value assignments we compute two different problem space metrics:

- **activity demand**: for each activity α and for each time $EST(\alpha) \leq t_i < LFT(\alpha)$, the activity demand $\Delta(\alpha, t_i)$ is equal to n_{t_i}/N, where n_{t_i} is the number of elements in the sample for which the interval of occurrence of α overlaps t_i.
- **resource contention**: for each resource $\rho \in R$ and for each time t_j within the scheduling horizon H, the resource contention $X(\rho, t_j)$ is equal to n_{t_j}/N, where n_{t_j} is the number of elements of the sample for which ρ is requested by more than one activity at time t_j.

Activity demand and resource contention represent two different aspects of the structure of the current problem solving state. The level of activity demand is a measure of how much current constraints and preferences bias an activity toward being in execution at a given time. The level of resource contention, instead, represents a measure of how likely it is that current constraints and preferences will generate congestion of capacity requests (and therefore potential inconsistency) at a given time.

The stochastic simulation allows several alternative variable selection strategies and value selection rules. A very simple variable selection strategy is *forward temporal dispatching*. A set of time variables, or open set, is initialized to the sources of the time point network, i.e., the time points with no incoming temporal constraints. Variable selection proceeds first by extracting one variable from the open set and then by updating the open set for the next iteration. During the update, all the time points that follow the selected time point in a temporal constraint are considered for

introduction in the open set; a candidate time point is actually inserted in the open set, if all its other incoming temporal constraints originate from time points that have already been assigned a value. Analogously, it is possible to define a *backward temporal dispatching* strategy. More sophisticated strategies may involve the selection of variables according to predefined or dynamically evaluated priorities. The stochastic value selection rule can change, depending on the preference that we might want to emphasize during the stochastic simulation. For example, no preference can be reflected by a random value selection according to a uniform distribution. Preference toward finishing as early as possible (as late as possible) can be introduced by executing the random selection with respect to a monotonically decreasing (increasing) distribution.

Measures of activity demand and resource contention based on an underlying probabilistic model have been successfully used in [12]; here activity demand and resource contention are computed independently and according to different relaxation assumptions, often stronger than those made by CPS. This allows fast computability of the metrics using deterministic methods, but weakens their reliability.

2.2 The focusing heuristics

CPS selects the set of critical decision variables at every problem solving cycle according to the following general principle: the higher the possibility of capacity conflicts for a given set of activities, the more urgent the need to arbitrate start and end time assignments among them.

The first phase of the focusing process, **Bottleneck Detection**, identifies the portion of the activity network in which there is the highest expectation of capacity conflicts; this is directly obtained from the resource contention profiles computed during Capacity Analysis. A **bottleneck** is formally defined as follows:

- **Bottleneck**: Given the set of resource contention functions $\{X(\rho,t)\}$ with $\rho \in R$ and $t \in H$, we call bottleneck a pair $< \rho_b, t_b >$ such that $X(\rho_b, t_b) = \max\{X(\rho,t)\}$ for any $\rho \in R$ and $t \in H$ such that $X(\rho,t) > 0$.

Conflict Identification returns the set A_{conf} of activities that can participate in a conflict. The activities in A_{conf} are selected from those that insist on the bottleneck, i.e., those associated to ρ_b and with time bounds satisfying the constraint $EST(\alpha) \leq t_b < LFT(\alpha)$. A refinement process avoids the selection of activities that are already mutually sequenced and therefore are guaranteed not to be in conflict. Activities whose likelihood of requiring the bottleneck resource at the bottleneck time is too low for them to be a significant contribution to the conflict are also excluded. The details of the Conflict Identification procedure are described in [8].

2.3 Conflict Arbitration

The task of the **Conflict Arbitration** phase is to add new sequencing constraints among the potentially conflicting activities in order to sort their capacity requests. CPS does not make strong assumptions about the "granularity" of the scheduling decisions, i.e., the number of precedence constraints that are posted during Conflict Arbitration, allowing a wide range of possibilities. The finest possible granularity requires the selection of a pair of activities from A_{conf} and the addition of a single precedence constraint among them. This philosophy is similar to the micro-opportunistic approach proposed by [12]. Since more than two activities are typically involved in a conflict, the rest of A_{conf} is likely to still cause a high level of contention at the bottleneck; therefore this strategy increases schedule consistency to a minimum extent and requires a greater number of problem solving cycles to reach the solution. However, the introduction of a single constraint modifies only minimally the topology of the activity network; therefore a fine granularity allows a closer monitoring of the variations of the problem space metrics, and gives a lower chance to generate inconsistent networks and to backtrack. The coarsest possible granularity requires the generation of a total ordering of the activities in A_{conf}. This approach is similar in spirit to macro-opportunistic scheduling [13] [1]. The activities in A_{conf} are now guaranteed not to be in conflict any more and therefore the procedure can converge faster than in the micro approach; however a large variation of the network topology without adequate guidance from the problem space metrics might increase the possibility of backtracking. Most probably, the most effective Conflict Arbitration strategy lies at some intermediate level of granularity.

The Conflict Arbitration strategy that we used in the following experimental analysis (section 3) is intermediate between the micro and the macro opportunistic approaches. The conflict set A_{conf} is partitioned into two subsets, A_{before} and A_{after} by performing a clustering analysis on the preferred times for each activity in A_{conf} according to the activity demand profiles. Appropriate *before* constraints are then introduced in order to force every activity in A_{before} to end before the start of any activity in A_{after}. The ra-

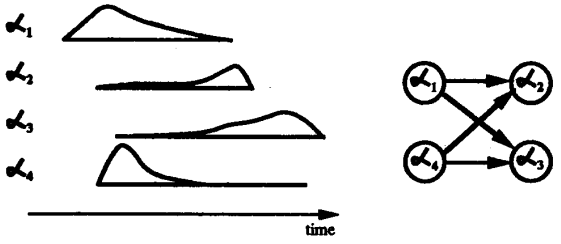

Figure 1: A Conflict Arbitration step

tionale behind the partition is to try to satisfy the preferences indicated by the activity demand profiles even after the introduction of the sequencing constraints. Figure 1 shows the arbitration of a conflict set A_{conf} consisting of four activities. Figure 1(a) shows the time bound and the demand profile of each activity before the arbitration step. Since the demand clustering analysis finds a great separation among the preferred times for α_1 and α_4 and the preferred times for α_2 and α_3, the following partition is enforced: $A_{before} := \{\alpha_1, \alpha_4\}$ and $A_{after} := \{\alpha_2, \alpha_3\}$. Figure 1(b) shows the additional precedence constraints introduced to resolve the conflict. The details of the clustering analysis are described in [8].

Figures 2 and 3 illustrate the effects of a Conflict Arbitration step on resource contention and activity time bounds. The top graph of figure 2 shows the resource contention of the bottleneck resource, while the bottom part shows the time bounds associated with each of the activities requesting the resource. The solid black segment at the far right of each time bound represents the activity duration; notice that each activity has a high degree of slack. The activities have been partitioned in A_{before} and A_{after} sets as a result of previous processing: in particular, the 5 time bounds at the bottom constitute the A_{before} set while the 5 at the top belong to A_{after}. Bottleneck detection on the next cycle identifies $t_b = 140$ as the new bottleneck time and all the 5 top operations are selected to be part of the new A_{conf}. Figure 3 shows the result of the partition of the new A_{conf}; the level of contention around the bottleneck time has been lowered and the resource contention function presents now three peaks, one for each of the new time bound clusters identifiable in the figure. Notice that the slack associated with each time bound is only slightly reduced by this partitioning.

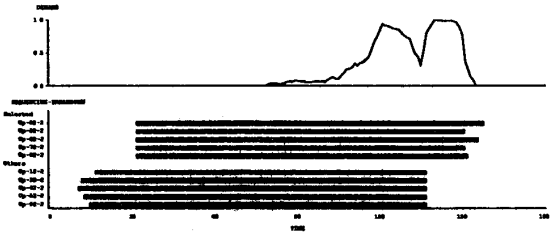

Figure 2: Initial bottleneck resource status

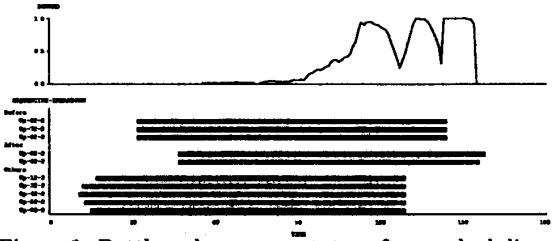

Figure 3: Bottleneck resource status after a scheduling cycle

3 Solving Constraint Satisfaction Scheduling Problems

To demonstrate the viability of CPS, we compared its performance with respect to two state-of-the-art heuristic scheduling methods: micro-opportunistic scheduling [12] and min-conflict iterative repair scheduling [7]. Both methods rely on the value commitment approach to generate schedules. The experimental analysis was conducted on the Constraint Satisfaction Scheduling benchmark proposed in [12]. The problems in the benchmark require the execution of all the activities within given temporal constraints (release and due dates) with no additional cost optimization; the problems bare a strict similarity to situations of practical interest where one has to guaranty the satisfaction of hard deadlines in the presence of limited execution resources.

Micro-opportunistic scheduling [12] proposes a constrained heuristic search framework. The basic operator applied at each step of the search, consists of two subsequent decisions: (1) select which start time variable should be assigned a value; (2) select which value should be assigned to the variable. The method builds a solution by iteratively extending consistent partial value assignments. The search chronologically backtracks when no consistent extension of a partial assignment is possible, i.e., the range of the possible values of some time point not yet committed to has

become empty.

Min-conflict iterative repair [7] is a member of a family of scheduling methods based on the repeated modification of inconsistent total value assignments. The value assigned to each problem variable has an associated measure of conflict; this is obtained by considering all the absolute time, job precedence, and capacity constraints that involve the variable and counting one inconsistency factor for each violated constraint. At each iteration, a variable is selected randomly among those with values having positive conflict measure. The conflict measure is evaluated for all the possible values within the range of the selected variable. Then, a new value is randomly selected within the set of values with minimum level of conflicts and assigned to the selected variable. Min-conflict is very sensitive to the choice of a 'good' initial total value assignment; in our implementation the initial state was generated through a stochastic forward simulation with uniform selection rule, as described in section 2.1. The resulting initial assignment satisfies all the initial problem constraints except for resource capacity.

The benchmark consists of 6 groups of 10 problems, each problem including 10 jobs and 5 resources. Each job is a linear sequence of 5 activities, each requiring a different resource. Two parameters identify each group of problems and give an indication of the expected difficulty for the set. The first parameter is the spread of the release and due dates among jobs; it can assume the three levels (in order of increasing expected difficulty) wide (w), narrow (n) and null (0). The second parameter is the number of a priory bottlenecks, which can be either 1 or 2.[2] Comparative performance was evaluated with respect to the number of problems solved in the benchmark and the average CPU time to process a problem (either finding a solution or failing).

We run Conflict Partition Scheduling (CPS) on a DEC 5000/200. The capacity analysis was conducted on the basis of $N = 20$ runs of the stochastic simulation. The stochastic simulation used a forward temporal dispatching variable selection strategy and a linearly biased value selection rule. The linear bias gives the highest preference to the earliest time and the lowest (0) preference to the latest time. We also ran our implementation of a Min-Conflict Iterative Repair scheduler (MIN CONF). Since the method could cycle indefinitely, we set 25000 as an upper limit on the number of repair cycles, after which the run was considered unsuccessful. The limit was chosen so that

[2]For a more detailed description of the benchmark see [12]

	CPS	MIN CONF (50, 500)	MIN CONF (5, 5000)	MICRO OPP
w/1	10	10	10	10
w/2	10	4	3	10
n/1	10	6	5	8
n/2	10	0	1	9
0/1	10	4	4	7
0/2	10	0	0	8
	60	24	23	52

(a)

	CPS	MICRO OPP
w/1	66.23	78.00
w/2	68.55	82.75
n/1	68.48	359.25
n/2	75.77	151.00
0/1	96.18	462.25
0/2	124.16	275.00

(b)

Figure 4: Performance results

an efficient implementation of MIN CONF would yield run times comparable to successful runs of CPS. If after a fraction of the total number of cycles MIN CONF did not find a solution, its state was reset by generating a new initial guess and the repair process restarted from there. MIN CONF was run in two different configurations. In the first, identified with (50, 500), the state was reset every 500 cycles; in the second, identified with (5, 5000), the state was reset every 5000 cycles. The performance data concerning the constraint satisfaction method of micro-opportunistic scheduling (MICRO OPP) are taken from [12].

The tables in figure 4 reports the comparative performance results. Table (a) reports the number of problems solved by the different methods for each problem group. The rows are labeled with the spread and bottleneck parameters of the problems set in the benchmark; for example label w/2 refers to problems with wide spread and two *a priori* bottlenecks. The last row reports the total number of problems solved by the different methods.

Table (b) reports a comparison of the average CPU times (in seconds) for a run on each problem among CPS and MICRO OPP. These are obtained by adding the processing times for all the instances in the set (successful and unsuccessful) and dividing it for the number of problems in the set. The run times of MICRO OPP have been obtained from those reported in [12] by assuming a speed-up factor of 2 for moving from a DEC 3100 workstation to a DEC 5000/200.

With respect to the number of problems solved, the performance of MIN CONF was significantly inferior to both CPS and MICRO OPP[3]. Our implementation of MIN CONF was effective on the problem categories with a low level of expected interaction, i.e., wide spread and single bottleneck; in those cases, the convergence was very quick. However, the per-

[3]In fact, they also are inferior to any of the heuristic search methods to which MICRO OPP is compared in [12].

formance significantly degraded with the increase of the expected difficulty; in particular, the worse performance was displayed on the set of problems including two bottlenecks. The presence of several bottlenecks in a scheduling problem introduces complex interactions that are traditionally difficult to deal with by scheduling algorithms. The local nature of the measure of conflict used in MIN CONF seems to makes it unable to detect such interactions. Other experimental results reported by [6] indicate that an increase in performance of MIN CONF is highly dependent on the method used to generate the initial total value assignment.

Overall, the results show that CPS has a more stable level of performance that MIN CONF and MICRO OPP over a wide range of problem categories and that its computational cost is competitive with the other approaches.

4 Conclusions

This paper describes the Conflict Partition Scheduling methodology and gives support to the claim that CPS is effective at solving difficult scheduling problems. The framework presented here is based on a strictly monotonic problem solving approach. To overcome unavoidable imprecisions of the capacity analysis, we need to explore the possibility of flexibly increasing and decreasing the number of runs of the stochastic simulation during problem solving, and of introducing backtracking and (local) search. Future work will also involve extensions of the technique toward the solution of optimization scheduling problems.

References

[1] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34, 1988.

[2] D. Applegate and W. Cook. A computational study of job-shop scheduling. Technical Report CMU-CS-90-145, School of Computer Science, Carnegie Mellon University, 1990.

[3] E. Biefeld and L. Cooper. Bottleneck identification using process chronologies. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, 1991.

[4] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, May 1991.

[5] M.S. Fox and S.F. Smith. Isis: A knowledge-based system for factory scheduling. *Expert Systems*, 1(1):25–49, 1984.

[6] M.D. Johnston. Spike: Ai scheduling for hubble space telescope after 23 months of orbital operations. In *presentation at the AAAI Spring Symposium on Practical Approaches to Scheduling and Planning*, May 1992.

[7] S. Minton, M.D. Johnston, A.B. Philips, and P. Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the 8th National Conference on Artificial Intelligence*, 1990.

[8] N. Muscettola. Scheduling by iterative partition of bottleneck conflicts. Technical Report CMU-RI-TR-92-05, The Robotics Institute, Carnegie Mellon University, February 1992.

[9] N. Muscettola and S.F. Smith. A probabilistic framework for resource-constrained multi-agent planning. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 1063–1066. Morgan Kaufmann, 1987.

[10] N. Muscettola, S.F. Smith, A. Cesta, and D. D'Aloisi. Coordinating space telescope operations in an integrated planning and scheduling architecture. *IEEE Control Systems Magazine*, 12(2), 1992.

[11] S.S. Panwalker and W. Iskander. A survey of scheduling rules. *Operations Research*, 25:45–61, 1977.

[12] N. Sadeh. Look-ahead techniques for micro-opportunistic job shop scheduling. Technical Report CMU-CS-91-102, School of Computer Science, Carnegie Mellon University, 1991.

[13] S.F. Smith, P.S. Ow, J.Y. Potvin, N. Muscettola, , and D. Matthys. An integrated framework for generating and revising factory schedules. *Journal of the Operational Research Society*, 41(6):539–552, 1990.

[14] M. Zweben, M. Deale, and R. Gargan. Anytime rescheduling. In *Proceeding of the DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, 1990.