On the Utility of Bottleneck Reasoning for Scheduling

Nicola Muscettola
RECOM Technologies
NASA Ames Research Center
AI Research Branch, Mail Stop: 269-2
Moffett Field, CA 94035-1000
e-mail: mus@ptolemy.arc.nasa.gov

Abstract

The design of better schedulers requires a deeper understanding of each component technique and of their interactions. Although widely accepted in practice, bottleneck reasoning for scheduling has not yet been sufficiently validated, either formally or empirically. This paper reports an empirical analysis of the heuristic information used by bottleneck-centered, opportunistic scheduling systems to solve constraint satisfaction scheduling problems. Different configurations of a single scheduling framework are applied to a benchmark set of scheduling problems and compared with respect to number of problems solved and processing time. We show superior performances for schedulers that use bottleneck information. We also show that focusing at the bottleneck might not only provide an effective "most constrained first" heuristic but also, unexpectedly, increase the utility of other heuristic information.

Introduction

Problem solvers often use combinations of several different heuristics and reasoning methods (*e.g.*, constraint propagation, search). Empirical comparisons of performances of different problem solvers can show that one combination of techniques is superior to another. However, to design better problem solvers we need a deeper understanding of the importance of each component technique and of how different techniques interact.

This paper reports an empirical analysis of the performance of heuristic information typically used to solve constraint satisfaction scheduling problems. We will focus on bottleneck-centered, opportunistic schedulers, a class of systems that has shown better performance than other kinds of schedulers (Adams, Balas, & Zawack 1988; Sadeh 1991; Muscettola 1993). We will show that there is strong empirical evidence on the effectiveness of reasoning about bottlenecks. Moreover, we will show evidence of the fact that heuristic information gathered at the bottleneck is "more useful" than average. This allows a bottleneck centered scheduler to make several decisions without having to re-evaluate its heuristic information too often.

Although widely accepted in practice (*e.g.*, manufacturing scheduling), bottleneck reasoning has not yet been sufficiently validated, either formally or empirically. Although formal validation would be most desirable, at present no formal model realistically captures the deep structure of scheduling problems. In its absence, strong evidence of performance can be gathered with empirical studies. We believe that such studies will also be extremely useful to discover new "phenomena" which will guide the search for an appropriate formal model.

The goal of an opportunistic scheduler is to build an assignment of time and resources to a network of activities and a set of resources such as to avoid resource over-subscriptions. The goal is achieved by repeatedly applying the following basic *opportunistic scheduling cycle*:

1. *Analyze*: analyze the current problem solving state;
2. *Focus*: select one or more activities that are expected to participate in a critical interaction among problem constraints;
3. *Decide*: add constraints to reduce negative interactions among critical activities.

Opportunistic schedulers differ on the specific techniques used to implement each phase (Smith *et al.* 1990; Biefeld & Cooper 1991; Sadeh 1991; Adams, Balas, & Zawack 1988; Muscettola 1993) but share several fundamental characteristics. The *Analyze* phase consists of building estimates of demand/supply ratios for the different resources and/or activities. These estimates are usually conducted on relaxed versions of the problem obtained by dropping some of its original temporal and/or resource constraints. During the *Focus* phase, all opportunistic schedulers use bottlenecks as the primary means of selecting critical interactions. While the exact definition of a bottleneck may vary, all opportunistic schedulers agree on relating this concept to a resource and time interval with a high demand/supply ratio. Critical activities are usually defined as those that are likely to request the use of a bottleneck. The constraints posted during the *Decide* phase impose arbitration among conflicting capacity and time requests. This is the phase where opportunistic schedulers differ the most with respect to the type of constraint posted (assigning a value to a variable versus imposing a precedence among activities) and to the granularity of the decision making process (the number of decisions taken at each cycle).

The study in this paper was conducted on Conflict Partition Scheduling (CPS) (Muscettola 1993), a scheduling method that implements the opportunistic scheduling paradigm. We first describe CPS and the stochastic simulation method used to compute heuristic information. Then we analyze two steps of the procedure: bottleneck detection and scheduling decision making. We discuss modifications of these steps and make hypotheses on how these modifications might affect performance. We then verify our hypotheses against the results of an experimental analysis.

We believe that our results are typical of the performance of other opportunistic schedulers. We base our belief on the similarity of each CPS step to other opportunistic schedulers and on the applicability of the performance hypotheses to comparable modifications of other schedulers.

Conflict Partition Scheduling

CPS adopts a *constraint posting* approach to scheduling, *i.e.*, it operates by posting temporal precedences among activities (activity α_i must precede activity α_j). During problem solving, each activity has an associated window of opportunity for its execution; these can be deduced by propagating activity durations and metric temporal constraints (absolute and relative) across the activity network (Dechter, Meiri, & Pearl 1991). Previous empirical studies have shown that constraint posting schedulers perform better than schedulers that proceed by assigning precise values to the start and end time of each activity (Applegate & Cook 1990; Muscettola 1993; Smith & Cheng 1993).

Figure 1 shows how CPS organizes its computation. In relation to the opportunistic cycle described in the introduction, Capacity Analysis corresponds to *Analyze*, Conflict Identification to *Focus*, and Conflict Arbitration to *Decide*. The consistency test is a propagation of the metric temporal constraints in the activity network.

The algorithm described in the diagram follows an iterative sampling approach to search (Minton *et al.* 1992; Langley 1992). If the consistency test fails, the activity network is reset to the initial state and the procedure is re-started. As we will see later, CPS' capacity analysis is stochastic in nature; therefore, each repetition can explore a different path in the problem solving space. If a solution has not been found after a fixed number of repetitions (in our case, 10), CPS terminates with an overall failure. The choice of iterative sampling is consistent with our interest in isolating the information content of the heuristic information generated by the Capacity Analysis. However, it is also possible to use the internal CPS cycle in a systematic search approach. For example, Conflict Arbitration could sprout several alternative ways of adding constraints among conflicting activities. Prioritization of these alternatives could make use of Capacity Analysis information and a backtracking scheme would ensure continuation when reaching a dead end.

The Capacity Analysis computes all the heuristic information used for decision-making by generating estimates of the structure of the remaining search space without engaging in detailed problem solving. Such estimates are statistics computed from a sample of complete time assignments to activity start times. These assignments are consistent with all the temporal constraints explicitly represented in the current activity network[1], but do not usually result in consistent schedules since they do not necessarily satisfy all the constraints of the problem (*i.e.*, those capacity constraints that have not yet been explicitly enforced). CPS uses a stochastic simulation process to generate each complete time assignment. This process differs from other stochastic simulation techniques (Drummond & Bresina 1990; Hanks 1990) used to estimate the possible outcomes of executing a detailed schedule in an uncertain environment. Having to insure executability, these simulations must introduce additional constraints to complete an intermediate problem solving state into a consistent schedule. Therefore, they end up considering many more details than are useful or necessary for an aggregate capacity analysis. Instead, CPS' stochastic simulation (Muscettola & Smith 1987) considers only the constraints that are explicit in the current intermediate state, with very weak assumptions on how it will be extended into a complete schedule.

In the following, $EST\ (\alpha)$ and $LFT\ (\alpha)$ will denote, respectively, the earliest start time and the latest finish time of the activity α, H will denote the overall scheduling horizon, and R will be the set of resources.

CPS' stochastic simulation proceeds by repeating the following cycle. Before the simulation starts, a temporal constraint propagation establishes the range of possible start times for each activity. At each simulation cycle i, an activity α_i is selected according to a given strategy. After the selection, a start time is randomly chosen among α_i's possible start times and assigned to the activity. The random choice follows a given probability distribution, or *selection rule*. The consequences of the start time assignment are then propagated through the network to restrict the range of other activities' start times, and the simulation cycle is repeated. The simulation terminates when all the activities of the network have been assigned a start time.

Different implementations of CPS can choose differ-

[1]Although CPS can deal with activities with flexible durations (*i.e.*, the duration of α must fall in the range $[d_\alpha, D_\alpha]$), we will only consider activities with fixed durations to simplify the presentation.

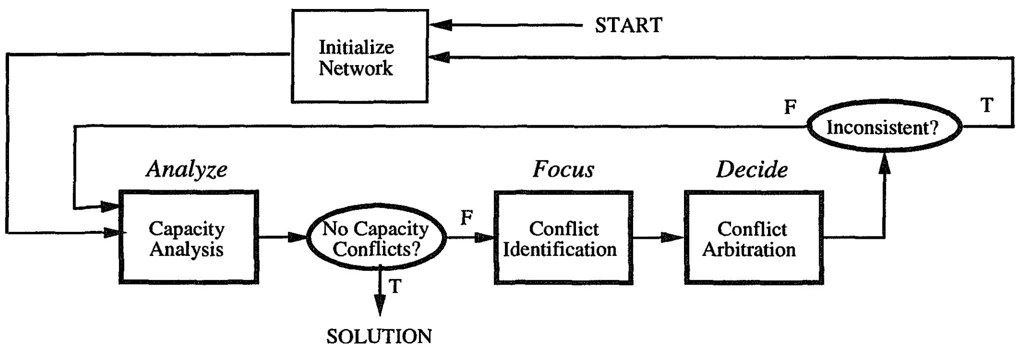

Figure 1: Conflict Partition Scheduling

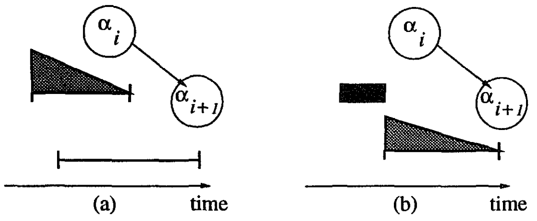

Figure 2: Simulation step: (a) before step i; (b) after step i

ent activity selection strategies and start time selection rules. A typical activity selection strategy is *forward temporal dispatching* which selects α_i among the set of activities whose predecessors have all start times assigned by previous simulation cycles. A possible start time selection rule is a linearly biased distribution (*i.e.*, the weight of the currently available times increases or decreases linearly over the time bound) for each activity in the network. These choices are crucial to the performance of CPS since they determine: (1) the computational cost of each cycle and (2) the probability of generating each of the possible total start time assignments and, therefore, the bias of the sampling base.

Figure 2 illustrates a single simulation cycle using a linear selection rule. In the figure, activity α_i precedes activity α_{i+1} in the activity network. Figure 2 (a) shows the time bounds for each activity; α_i has a linear value selection rule superimposed on its time bound. In Figure 2 (b), a start time has been selected for α_i and time has been reserved for its execution; the reservation is represented by the black rectangle. This causes α_{i+1}'s time bound to shrink. A triangular selection rule is now superimposed on α_{i+1}'s time bound and the simulation cycle can start again.

Repeating the simulation N times yields a sample of N complete time assignments. CPS' Capacity Analysis uses this sample to estimate the following two 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 complete time assignments in the sample for which α is being executed at time t_i.
- **resource contention**: for each resource $\rho \in R$ and for each time $t_j \in H$, the resource contention, $X(\rho, t_j)$, is equal to n_{t_j}/N, where n_{t_j} is the number of complete time assignments in 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 current problem solving state. Activity demand is a measure of preference; it indicates how much the current constraints bias an activity toward being executed at a given time. Resource contention is a measure of potential inconsistency; it indicates how likely it is that the current constraints will generate a congestion of capacity requests on a resource at a given time.

Bottleneck Detection

In problem solving, a widely accepted principle is to focus on the most tightly interacting variables, *i.e.*, those with the smallest set of possible values. For example, in constraint satisfaction search (Haralick & Elliot 1980) the 'most constrained first' heuristic minimizes the expected length of any path in the search tree and, therefore, increases the probability of achieving a solution in less time. In opportunistic scheduling this principle translates into looking for bottleneck resources.

In CPS each problem solving cycle focuses on a set of activities that are potentially in conflict, called the *conflict set*. More precisely, a conflict set is a set of activities that: (1) request the same resource, (2) have overlapping execution time bounds, and (3) are not necessarily totally ordered according to the precedence constraints of the current activity network.

To detect a conflict set CPS first identifies a bottleneck using resource contention:

- **Bottleneck**: Given the set of resource contention functions $\{X(\rho,t)\}$ with $\rho \in R$ and $t \in H$, we define a bottleneck to be 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 $X(\rho,t) > 0$.

The conflict set is then extracted among the activities requesting ρ_b with current time bounds overlapping t_b.

Although focusing on bottlenecks is widely accepted in opportunistic scheduling, there is little quantitative evidence of its effectiveness. One could wonder if the performance of the scheduler would remain unaffected if it focussed on *any* set of activities, either associated with a bottleneck or not. If this were true, one could save the additional cost required to compute resource contention and base all decision making on activity demand alone.

To answer this question we consider two different configurations of CPS' Bottleneck Identification step.

1. **Maximum contention bottleneck (BTL)**: The original method used in CPS; the set of conflicting activities is selected around the bottleneck.
2. **Random (RAND)**: The conflict set is selected around a randomly chosen resource and time.

Conflict Arbitration

At each Conflict Arbitration step, CPS introduces additional precedence constraints between pairs of activities to restrict their mutual position and their time bounds. An important differentiating aspect among schedulers is the granularity of decision making. At one end of the spectrum there are schedulers that make the minimum possible decision at each scheduling cycle; this follows the spirit of micro-opportunistic scheduling (Sadeh 1991). At the other end there are schedulers that make decisions that eliminate any possibility of conflict among all the activities in the conflict set; this follows the spirit of macro-opportunistic approaches (Smith *et al.* 1990; Adams, Balas, & Zawack 1988).

Within CPS we can explore the consequences of different decision making granularities. For example, a micro-opportunistic approach translates into adding a single precedence constraint between two activities in the conflict set. Conversely, a macro-opportunistic approach could be implemented by imposing a total ordering on all activities in the conflict set.

The current implementation of CPS (Muscettola 1993) proposes an intermediate granularity approach by partitioning the conflict set into two subsets, A_{before} and A_{after}, and then constraining every activity in A_{before} to occur before any activity in A_{after}. The bi-partition of the original conflict set relies on a clustering analysis of activity demands. Figure 3 (a) shows four conflicting activities and their demand profiles; figure 3 (b) shows the new precedence constraints added by Conflict Arbitration.

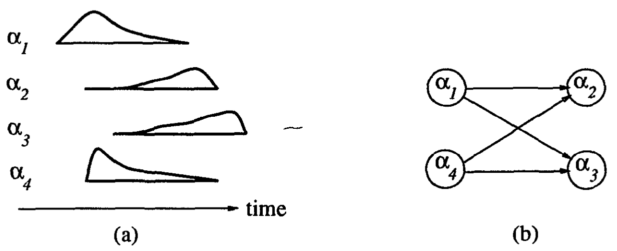

Figure 3: A Conflict Arbitration step

To choose the appropriate decision granularity one needs to evaluate a trade-off. The greater the number of scheduling decisions in a step, the greater the pruning of the search space and, therefore, the faster the scheduler. However, the more decisions that are made, the greater the change in the topology of the activity network after the step. Therefore, an analysis done before the step could give little information on the structure of the destination state. This can increase the likelihood of failure and consequent backtracking, and therefore slow down the scheduler. In summary, an important trade-off involves, the speed of convergence vs. the number of restarts needed during iterative sampling. This trade-off rests on the reasonable assumption that making fewer decisions at each cycle is always at least as accurate as making several; in other words, although possibly slower, a micro-opportunistic scheduler should solve at least as many problems as a larger granularity scheduler (Sadeh 1991).

We therefore consider two distinct Conflict Arbitration rules:

1. **conflict bi-partition (BIP)**: The technique originally used in CPS; the conflict set is separated into two A_{before} and A_{after} subsets.
2. **most separated activities (MS)**: A *micro* arbitration technique; it introduces a single precedence between two activities extracted from the conflict set. To minimize the impact of sampling noise, we select the two activities whose demand profiles are maximally separated.

Experimental Results

The experimental analysis made use of the Constraint Satisfaction Scheduling benchmark proposed in (Sadeh 1991). The benchmark consists of 6 groups of 10 problems, each with 50 activities, 5 resources, and non-relaxable release and due date constraints. The groups vary according to their expected difficulty. Each group is identified by two parameters: (1) the spread of the release and due dates, which can assume the three levels (in increasing order of expected difficulty) *W* for wide, *N* for narrow and *0* for null; (2) the number of expected bottleneck resources, either *1* or *2*. For more details see (Sadeh 1991).

	< BTL, BIP >	< BTL, MS >	< RAND, BIP >	< RAND, MS >
W/1	10.00	10.00	9.95	7.68
W/2	10.00	10.00	9.95	9.10
N/1	10.00	10.00	9.10	8.70
N/2	10.00	10.00	8.20	6.26
0/1	10.00	9.95	8.30	8.60
0/2	9.25	10.00	4.85	4.05
TOT	59.25	59.95	50.35	46.30

Table 1: Experimental results: number of problem solved

	< BTL, BIP >	< BTL, MS >	< RAND, BIP >	< RAND, MS >
W/1	44.94	120.90	61.74	281.49
W/2	44.59	134.00	90.82	476.97
N/1	47.02	127.40	106.67	360.72
N/2	46.03	138.90	142.81	754.39
0/1	50.27	140.40	118.24	405.10
0/2	64.86	161.10	212.96	908.89
AVG	49.62	137.10	122.20	531.26

Table 2: Experimental results: processing time

Tables 1 and 2 report the performance of all possible combinations of the alternative settings described in the previous sections. Table 1 shows the average number of problems solved over 20 independent runs of the procedure. Table 2 reports the corresponding average processing times. In order to factor out the effects of known implementation inefficiencies, processing times are given as the number of opportunistic cycles needed either to find a solution or to fail. The number of iterations was weighed differently depending on the conflict identification method, with each **RAND** cycle taking 85.64% of the time of a **BTL** cycle. The speed-up results from avoiding the computation of resource contention.

The cardinality of the Capacity Analysis sample was $N = 10$. We used *forward temporal dispatching* as the activity selection strategy. The start time selection rule was linearly biased over the time bound, with highest preference to the earliest time and lowest (0) to the latest.

All of the following conclusions have been tested for statistical significance using the methods available in the S statistical package (Chambers & Hastie 1992). For the average number of problems solved we fitted the results as a function of the configuration; this was done through a *logit* generalized linear regression. An analysis of deviance and a Chi-squared test yielded the desired measure of significance (see (Chambers & Hastie 1992), chapter 6). For the processing time we used a standard analysis of variance (see (Chambers & Hastie 1992), chapter 5). Unless otherwise noted, differences in performance are statistically significant at a 1% level.

To test the importance of bottleneck information, let us compare each < **RAND, ?x** > entry with the corresponding < **BTL, ?x** > entry. Differences in performance are always significant except for the number of problems solved for groups *W/1* and *W/2* when using bi-partition for Conflict Arbitration (**BIP**). These are the two problem groups with lowest expected difficulty. In every other case, random focusing performs significantly worse than bottleneck focusing, with an average slowdown of approximately 3.1 times. Therefore, these results show that, all things remaining equal, there is a substantial advantage in focusing problem solving on what CPS characterizes as bottlenecks.

To test the effect of decision making granularity, let us compare each < **BTL, BIP** > configuration with the corresponding < **BTL, MS** > configuration. With respect to the number of problem solved, only for group *0/2* there is a statistically significant advantage in using most-separated-pair for Conflict Arbitration; *0/2* is the the group with the highest expected difficulty. This advantage is due to a single problem that **MS** always solves while **BIP** solves less than 50% of the time. Although small, this advantage is consistent with the expectation of better problem solving accuracy with smaller decision making granularity, especially on difficult problems. However, when comparing processing times we see an average slowdown of approximately 3.2 times going from **BIP** to **MS** which makes the cost of micro-granularity scheduling prohibitive (except for one problem).

Unexpectedly, the trend toward better accuracy with lower decisions granularity is completely reversed when comparing < **RAND, BIP** > to < **RAND, MS** >. In fact, for all problem groups, the average number of problems solved tend to *decrease* when going from bi-partition to most-separated-pair. This trend is statistically significant for groups *W/1* (at a 2% level), *W/2*, and *N/2*. This result contradicts our expectations. After all, a macro decision step can always be seen as a sequence of micro steps without additional intermediate capacity analyses. A macro-granularity approach should be less informed than a micro-granularity approach and therefore more prone to errors.

However, worse performance with lower granularity can be explained by assuming that at each step there is a probability p of selecting a misleading conflict set, *i.e.*, one for which the preferential information leads to a wrong ordering among activities. The overall probability of following a dead-end path is the sum of the probabilities of failing after x cycles, with x less or equal to maximum path length in the search tree. When the decision making granularity decreases, the expected path length increases. Correspondingly, if p does not substantially decrease, the overall probability of failure increases. The experimental results seem to indicate that the decrement of p is adequate only when using the bottleneck information for focusing. In other words, the expected utility of preferential information at the bottleneck is higher than average.

Conclusions

In this paper we experimentally analyzed the role of bottleneck reasoning in opportunistic schedulers. The aim was to go beyond a bulk comparison of systems and to identify important design trade-offs among system components. The results of the study empirically validate the importance of bottlenecks to focus problem solving. The results seem to indicate that preferential information at bottlenecks has a higher expected utility than average. Therefore the utility of bottleneck-focusing goes beyond the classical view of a "most constrained first" heuristic in a constraint satisfaction search. This is an unexpected result that will further focus the search for a plausible formal model of the performance of schedulers.

Acknowledgements

The author thanks the following people for reviewing earlier drafts of the paper: John Allen, John Bresina, Mark Drummond, Keith Swanson. This work was carried out when the author was at the Center for Integrated Manufacturing and Decision Systems, The Robotics Institute, Carnegie Mellon University. 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 # F30602-91-F-0016, and the Robotics Institute.

References

Adams, J.; Balas, E.; and Zawack, D. 1988. The shifting bottleneck procedure for job shop scheduling. *Management Science* 34:391–401.

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

Biefeld, E., and Cooper, L. 1991. Bottleneck identification using process chronologies. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, 218–224. Menlo Park, California: AAAI Press.

Chambers, J., and Hastie, T., eds. 1992. *Statistical Models in S.* Wadsworth and Brooks/Cole.

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

Drummond, M., and Bresina, J. 1990. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proceedings of the 8th National Conference on Artificial Intelligence*, 138–144. AAAI Press.

Hanks, S. 1990. Practical temporal projection. In *Proceedings of the 8th National Conference on Artificial Intelligence*, 158–163. AAAI Press.

Haralick, R., and Elliot, G. 1980. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence* 14(3):263–313.

Langley, P. 1992. Systematic and nonsystematic search strategies. In *Proceedings of the 1st International Conference on Artificial Intelligence Planning Systems*, 145–152. Morgan Kaufmann.

Minton, S.; Drummond, M.; Bresina, J.; and Philips, A. 1992. Total order vs. partial order planning: Factors influencing performance. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, 83–92. Morgan Kaufmann.

Muscettola, N., and Smith, S. 1987. A probabilistic framework for resource-constrained multiagent planning. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, 1063–1066. Menlo Park, California: AAAI Press.

Muscettola, N. 1993. Scheduling by iterative partition of bottleneck conflicts. In *Proceedings of the 9th Conference on Artificial Intelligence for Applications*, 49–55. Los Alamitos, California: IEEE Computer Society Press.

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

Smith, S., and Cheng, C.-C. 1993. Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI 93)*, 139–144. Menlo Park, California: The AAAI Press.

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