From: AAAI Technical Report SS-92-01. Compilation copyright © 1992, AAAI (www.aaai.org). All rights reserved.

Real-time Scheduling Using Minimin Search

Prasad Tadepalli and **Varad Joshi**
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331-3202

Abstract

In this paper we consider a simple model of real-time scheduling. We present a real-time scheduling system called RTS which is based on Korf's Minimin algorithm. Experimental results show that the schedule quality initially improves with the amount of look-ahead search and tapers off quickly. So it appears that reasonably good schedules can be produced with a relatively shallow search.

1 Introduction

Job shop scheduling is one of the most computationally intensive parts of flexible manufacturing systems. Scheduling in the real world is complicated by several factors including the resource contention, unpredictability of events, multiple agents with mutually conflicting goals, and the sheer combinatorial explosiveness of the task. In this paper, we simplify the real world scheduling problem to a great extent and focus exclusively on one aspect of the problem, namely its real-time character.

This paper looks at detailed job shop scheduling at the level of individual machine operations. The scheduling problem is treated as assigning the job-steps to individual machines and ordering them so that (a) the precedence and resource constraints are satisfied, and (b) the schedule is "good" in some measurable objective sense.

Most approaches to scheduling are static in that the scheduling is done all at once and not during the production process. Static scheduling has several obvious drawbacks: First, optimal static scheduling is computationally prohibitive in any realistic manufacturing system, which involves hundreds of jobs and machine operations. Second, since the static scheduler has to make decisions based on predicted information, it has no way of recovering from incorrect predictions even after they were proved wrong. Thus, it is unable to readjust to or recover from changes in the production environment, including machine failures, new jobs, or machine delays.

Real-time scheduling prevents the above two pitfalls of static scheduling by requiring that after every constant time, some real world action is taken. This not only prevents the system from losing itself in a combinatorially explosive search space, but also makes it possible to continually readjust to the changing environment.

In this paper we present a system called RTS (Real-time Scheduler) which uses the Minimin algorithm of Korf [Korf, 1990] to do real-time scheduling. Minimin is similar to the Minimax algorithm extensively used in games. We view scheduling as a state space search where states represent partial schedules. Minimin performs a fixed depth look-ahead search from the initial state, and applies a heuristic evaluation function to the partial schedules at the leaves of the search tree to estimate the cost of the schedule. This value is backed up to the root of the tree and the system takes the most promising scheduling action, i.e., it assigns a job-step to a machine which leads to a schedule with the best estimated cost.

Since RTS relies on heuristic estimates, the schedules the system produces are not guaranteed to be optimal. However, our experimental results show that the schedule quality initially improves with the amount of look-ahead search and tapers off quickly. So it appears that reasonably good schedules can be produced with a relatively shallow search. We conclude that our approach to real-time scheduling based on Minimin is promising and can be extended in several directions, including learning better evaluation functions, and doing variable depth search.

2 Previous Work

One approach to scheduling is based on expert systems [Fox and Smith, 1984]. However, expert systems approach to scheduling seems inadequate because of the dynamic nature of the scheduling problem, which is due to changes to job loads, availability of machines and labor, introduction of new machines and manufacturing processes, changes in the inventory space, etc. For this reason, there are no experts in this domain, and even if there were, they would be quickly outdated [Kempf et al., 1991].

Many AI-approaches to scheduling are constraint-based [Fox, 1987, Sadeh, 1991, Smith et al., 1986, Zweben and Eskey, 1989]. Here scheduling is viewed as finding a schedule (assignment of machines to various job-steps) which satisfies a set of constraints, including precedence relationships between job-steps and global resource constraints. However, most of these approaches

assume a static scheduling problem, and are not easily adaptable to real-time scheduling.

Traditionally, the "dynamics" of the manufacturing process is handled by local greedy dispatch rules [Vollmann et al., 1988]. One dispatch rule, for example, recommends to schedule the job with Least Processing Time (LPT) first, while another rule uses Earliest Due Date (EDD) to prioritize jobs. While computationally cheap, such local dispatch rules are too short-sighted, and do not guarantee efficient schedules except in very special cases [Kempf et al., 1991].

In summary, static optimal scheduling is computationally prohibitive and is not sufficiently responsive to change. On the other hand, local dispatch rules are too short-sighted to be generally effective. The expert systems approach is plagued by the dynamics of the scheduling problem and paucity of experts. In this paper, we propose an approach based on real-time search which attempts to address each of the above problems.

3 Problem Description

The problem we address can be characterized as scheduling the job-steps in a set of jobs on various machines in real time. We make the following assumptions.

1. Each job consists of a sequence of job-steps that must be performed serially.
2. There may be several machines of each machine type.
3. Each job-step requires a machine of a particular type to perform it.
4. Each machine can only process one operation at a time.
5. Each job may require the same machine (or machine type) more than once. In other words, we have a "job shop" situation rather than a "flow shop" situation [Vollmann et al., 1988].
6. The machine type required for each job-step and the time for each job-step is known in advance.
7. The real-time constraint means that the time for deciding which job-step to schedule next is "small," and should not depend on the number of jobs and job-steps.

For example, each job in Figure 1 consists of a sequence of job-steps. The task of the scheduler is to incrementally add new job-steps to the current machine queues. As the machine queues are filled from the back by the scheduler, they are emptied from the front by the machines executing the job-steps. In addition, the job-step must wait until its predecessor job-step in its job is executed. For example, in Figure 1, job-steps S-11, S-22, and S-42 are in the queue for machine M1 in that order. In addition, S-11, S-12, S-13, and S-14 must also be processed sequentially, because they are all part of a single job.

Since scheduling is done while the jobs are getting executed, the scheduler has only a limited time to decide what job-step to schedule next, and on what machine.

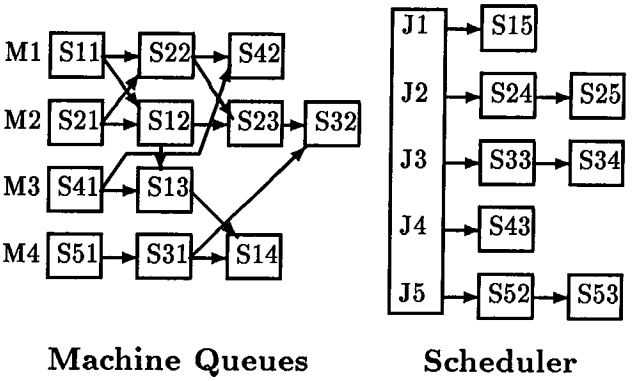

Figure 1: Scheduler assigns job-steps to machine queues.

4 Scheduling as State Space Search

We formulate the scheduling problem as a state space search problem. States in the scheduling task correspond to partial schedules represented as queues of job-steps for the machines. The search problem is characterized by an initial state, where there are no jobs scheduled, and a final state, where all the jobs are scheduled. In any state, there are several alternative assignments of the job-steps to machine queues. A job-step is "ready" when all its precedent job-steps have completed. Scheduling operators or "moves" assign job-steps to one of the machines of the required machine type. In other words, they can be placed on any one of the possible queues of the appropriate machine type. Each such placement creates a new state. The scheduling problem is to find a best assignment of job-steps to machine queues according to some measure of goodness (objective function). For example, we may use the total time for the schedule or the sum of the inventory and shortage costs as an objective function.

The static scheduling problem corresponds to finding the best path in the state space from the initial state to a final state. However, static scheduling suffers from the combinatorial explosion due to deep searches and is not sufficiently responsive to the dynamics of the manufacturing domain. In the following, we describe our approach to scheduling that addresses these problems.

4.1 Minimin search

Our approach to scheduling consists of a real time search method called "Minimin search" [Korf, 1990]. Minimin is similar to minimax search in two-person games, except that instead of alternating Min and Max nodes, the search tree only contains Min nodes.

Minimin works by a fixed depth look-ahead search followed by a real-time action. The search terminates after a small depth called "search horizon," after which the leaves of the tree are evaluated using a heuristic evaluation function. The evaluation function applied at the leaves estimates the minimum total cost of any solution that begins with a partial path ending with that leaf. It is backed up to the root using the Min function. In other words, the value of any node is the minimum of all the values of its children, and the move that results in

that value is the "best move." After searching for a fixed look-ahead depth, Minimin chooses the first best move, executes it, updates the state and once again starts look-ahead search from that point.

The "knowledge" of the Minimin algorithm lies in its heuristic evaluation function f. The more closely it follows the real cost of the solution, the more optimal the algorithm's current decision is going to be. An evaluation function is "admissible" if it never overestimates the real cost of a solution. An evaluation function is monotonic, if its value is monotonically non-decreasing along any single path of the search tree.

When the evaluation function of the Minimin search is monotonic, it is amenable to an effective branch and bound technique called α-pruning. α-pruning works by pruning the branches whose estimated cost is more than the current best estimated cost. Like α-β pruning, α-pruning is guaranteed to preserve the outcome of the look-ahead search.

Each time the Minimin algorithm is called it returns the best next state and its estimated evaluation. The main program then takes the corresponding action in the "real world" and updates its current state to this new state. After this, the program repeats its cycle again by calling the Minimin algorithm.

4.2 Real-time Scheduling

We noted that in Scheduling the states correspond to partial schedules and operators correspond to scheduling actions. In order to complete the mapping of the real-time scheduling problem to Minimin search, we need to specify how a schedule is evaluated.

Several optimality criteria might be used to evaluate the schedules. One of the criteria is the sum of the shortage and the inventory costs. Another criterion is the total length of the schedule from the beginning to the end, also called "make-span." In our system, we currently use the make-span criterion to evaluate schedules. The smaller the make-span, the better the schedule. In Minimin search, the cost of the schedule must be estimated after only a small number of steps are scheduled, i.e., much before the full schedule is known. To do this effectively, we should necessarily rely on heuristic estimates of the schedule cost. A good heuristic evaluation function must approximate the optimality criterion as closely as possible.

As discussed earlier, there is an implicit precedence relationship between the job-steps in the same machine queue, and between the job-steps that belong to the same job. For any job-step s, let $PRE(s)$ be the set of job-steps which are immediate predecessors of s, in that they need to be performed before s is done. In Figure 1, PRE(S-12) = {S-11, S-21}.

Our estimate of the make-span is done as follows: first, we compute the time T_i by which each machine M_i finishes its current queue. Assuming that the expected time $ET(s)$ for each job-step s is known in advance, this can be calculated exactly. Let the expected start time and the expected finish time of a job-step s be denoted by $ES(s)$ and $EF(s)$ respectively. The expected start and finish times of any job-step can then be calculated using

```
Minimin(CurrentState, depth, α)
   If depth = SearchHorizon return (f(CurrentState));
      %Alpha Pruning
   If f(CurrentState) ≥ α return (α + 1)
   S := job-steps which are "ready";
   M := {m | ∃s ∈ S that needs a machine of m's type };
   Pick m ∈ M s.t. its current queue finishes earliest.
   For each job-step s ∈ S which matches m's type, Do
      Begin
      NewState := Assign(s,m);
      Val := Minimin(NewState, depth + 1, α);
      If Val < α
         Begin
         α := Val;
         BestNextState := NewState;
         End;
      End;
   Return(α, BestNextState);
End Minimin;
```

Table 1: Minimin Applied to Scheduling

the following recurrence relations.

$$ES(s) = Max_{r \in PRE(s)}\{EF(r)\}$$
$$EF(s) = ES(s) + ET(s)$$

Let T_i be the time by which machine M_i finishes the last job-step in its current queue. The goal of the Minimin search is to find the best next job-step to add to the current queues by doing a look-ahead search of fixed depth in the space of partial schedules (machine queues).

A job-step is considered "ready" if all its predecessors are either already executed or present in one or the other of the machine queues. At any given state, RTS first filters its machines by discarding those machines which do not have any ready job-steps waiting for their machine type. It then chooses the machine M_i which is expected to finish its queue the earliest, i.e., with a minimum T_i, and considers scheduling various job-steps on it. Each "ready" job-step s whose type matches that of machine M_i is a possible choice. For each such possible choice, Minimin creates a new state by assigning s to M_i, and updates the expected finish time of M_i's current queue using the above recurrence relations. RTS proceeds in depth first search in this manner until it reaches the search horizon.

At the leaves of the look-ahead search tree, the total time required to complete the remaining schedule must be estimated. Since none of the job-steps in the remaining schedule is assigned to a machine yet, their expected finish time cannot be exactly estimated. It is here that we rely on a heuristic lower bound.

Let T_K be the maximum of T_i of all machines M_i of type K. Let W_K be the total work remaining on machines of type K, i.e., the total expected time of all job-steps that need a machine of type K. Assume also that there are N_K machines of type K. Ignoring all the precedence constraints between the job-steps, the work remaining on machines of type K can be distributed as

follows. First fill each machine of type K until they reach the level T_K. This does not increase the make-span because it anyway takes that long to wait for the current queue to finish. This reduces the remaining work on machine of type K to $W_K - \Sigma_i\{T_K - T_i\}$, which may be distributed evenly among all the machines of type K in the best possible case. Hence, we observe that the time for completing the schedule must at least be as high as the following two bounds.

1. $Max_{K \in Machine-Types}\ (T_K + \frac{W_K - \Sigma\{T_K - T_i\}}{N_K})$
2. $Max_{J \in Jobs}(\Sigma_{s \in J} ET(s) + Min(T_i))$

The second lower bound above is obtained by noting that the job-steps in a single job should be executed sequentially. To the total time needed to execute any job, the minimum expected finish time of all machine queues is added. The finish time of the schedule is estimated to be the maximum of the above two bounds.

The above evaluation function is both admissible (never overestimates the true cost) and monotonic (monotonically non-decreasing along any path). This follows because, adding job-steps to the machine queues can only increase but never decrease the delays, by introducing more constraints. Since each step in the search adds a new job-step to the queues, the expected completion time is monotonically non-decreasing. The monotonicity is exploited by RTS by maintaining the current estimate α of the best schedule and evaluating f at internal nodes even before the search horizon is reached. Because f is monotonically non-decreasing, any path whose current estimate of the schedule cost exceeds the current value of α is guaranteed to yield only a worse solution and hence need not be pursued further. In other words, α-pruning would not sacrifice solution quality.

The estimated time for completion is backed up to the internal nodes from the leaves and finally to the root of the look-ahead search tree. The path that promises the lowest make-span is considered the best. An assignment of the first job-step in this path is made as suggested by this path. After this assignment, which corresponds to an action in the "real world," RTS takes a fresh look at its environment and starts a new cycle all over again.

4.3 Experimental Results

The problem specification is a 5-tuple. It consists of the number of jobs, the number of machines, the number of types of machines (this has to be less than the number of machines) and two numbers which specify the upper bounds on the number of steps for any job and the processing time for any step. Number of steps for each job is generated randomly, bound by the upper bound given in the problem specification. Each job-step is randomly assigned a machine type. Each job-step is also assigned some processing time randomly, bound from above as given in the problem specification.

We tested RTS on a sample of 39 randomly generated problems. Each problem had about 4-6 machines divided into 3-4 types, and 4-6 jobs each of which had about 5 steps, each step taking up to 6 units of time. We then ran the system with different look-ahead depths, and measured the total time to execute the whole schedule

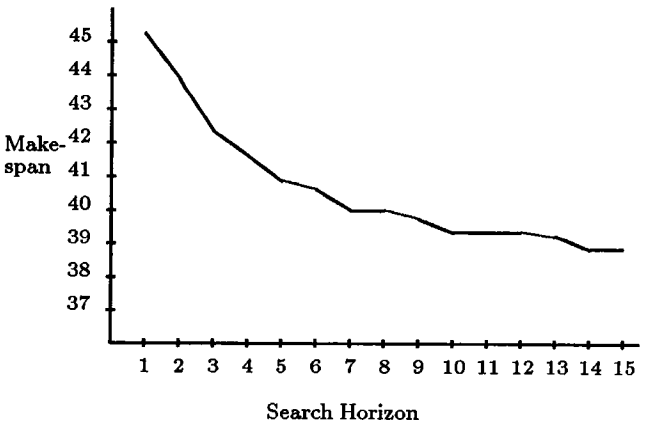

Figure 2: Solution quality improves with search horizon.

(make-span). We plotted the search horizon on the X-axis and the average make-span on the Y-axis.

The results show that the solution quality generally improves with search horizon, as expected. This tradeoff of search for solution quality was very favorable in the beginning, and tapered off toward the end. Although deeper searches resulted in better solutions on the whole, they also required exponentially larger number of nodes, taking exponentially longer time. In our context, the results indicate that a search horizen of 7 to 10 would achieve reasonably good schedules without extravagant search.

In general, it appears that a shallow look-ahead search would suffice to improve solution quality in this domain, which means that deep expensive searches may not be needed.

5 Future Work

The work reported here is preliminary and a lot remains to be done to make the ideas more practical and applicable in a real-world setting. A few of the promising directions to pursue are listed below.

Reactivity: One of the major reasons for building "real-time" systems is that they are more responsive to changes in their environment. This is especially crucial in the manufacturing domain, where unexpected events such as machine break-downs and tool failures are common. We believe that our system would respond better to such changes than a static scheduler. Indeed, it is possible to completely change the machine and job configuration before every cycle of the Minimin algorithm. The system should still be able to make locally optimal decisions with respect to its changed configuration. However, we expect that the system's behavior degrades gradually as the dynamics in the system configuration increases. It might also be expected that the usefulness of the look-ahead search decreases with increased dynamism. These hypotheses need to be experimentally verified.

Variable Depth Search: We assumed that the search horizon is fixed. However, this need not be the

case, and it is possible to change the search horizon across problems and even within the same problem. For example, a method called "Singular Extensions" proved very effective in game domains by focusing the search along narrow paths which appear significantly more promising than their nearest competitors [Anantharaman et al., 1990]. It seems possible to adapt this technique to real-time scheduling and search deeper at places in the search tree which appear promising. We can also add the iterative-deepening capability to Minimin, so that more time can be spent searching for a better schedule if time is available [Korf, 1985]. This also makes it an *anytime* algorithm in the sense of [Dean and Boddy, 1988], in that it can be interrupted at any time during its computation and asked to schedule the next job-step. The utility of the system's decisions is expected to increase with the time available to make the decision.

Learning: The performance of the system at a given search horizon depends mostly on the goodness of the evaluation function used to estimate the optimality of the schedule. Although our current evaluation function performed fairly well on the problems that we tested it on, it does not take into account factors such as bottleneck resources, which are crucial for a good scheduler. However, it is time-consuming and laborious to encode sophisticated evaluation functions. Besides, good evaluation functions are sensitive to the scheduler's environment, and hence may not be generally effective. Hence we plan to apply machine learning to learn effective evaluation functions [Lee and Mahajan, 1988]. There have already been some machine learning methods applied to scheduling domains [Kim, 1990, Shaw et al., 1990]. We think that significant improvements beyond current scheduling techniques can be achieved using machine learning.

6 Summary

In this paper we described a real-time scheduling system based on the Minimin algorithm and showed that it is effective and capable of producing good schedules with reasonably small effort. In particular, we showed that the schedule quality improves with increased look-ahead, confirming some of the results of Korf on Real-time Search in the scheduling domain. The future work includes evaluation function learning, variable depth searches, and demonstration of the reactivity of the system. Although much remains to be done, the preliminary results reported in this paper appear promising.

Acknowledgments

We thank Logen Logendran, Toshi Minoura, and Vijay Srinivasan for many fruitful discussions, and the department of computer science for financial support.

References

[Anantharaman et al., 1990] T.Anantharaman, M. Campbell, F. Hsu. Singular Extensions: Adding Selectivity to Brute Force Searching. *Artificial Intelligence*, 43(1), 1990.

[Dean and Boddy, 1988] T. Dean and M. Boddy. An Analysis of Time-dependent Planning. AAAI-88 Proceedings, Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[Fox and Smith, 1984] M.S. Fox and S.F. Smith. ISIS: A Knowledge-based System for Factory Scheduling. *Int. J. Expert Systems*, 1(1), 1984.

[Fox, 1987] M.S. Fox. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling.* Morgan Kaufmann, Los Altos, CA, 1987.

[Kempf et al., 1991] K. Kempf, C.L. Pape, S.F. Smith, and B.R. Fox. Issues in the Design of AI-Based Schedulers: A Workshop Report. *AI Magazine*, 11(5), 1991.

[Kim, 1990] S. Kim. *Schedule-Based Material Requirements Planning: An Artificial Intelligence Approach* M.S. Thesis, Department of Industrial and Manufacturing Engineering, Oregon State University, Corvallis, OR, 1990.

[Korf, 1985] R. Korf. Depth-first Iterative-deepening: An Optimal Admissible Tree Search. Artificial Intelligence, 27, 1985.

[Korf, 1990] R. Korf. Real Time Heuristic Search. Artificial Intelligence, March, 1990.

[Lee and Mahajan, 1988] K.F. Lee and S. Mahajan. A Pattern Classification Approach to Evaluation Function Learning. Artificial Intelligence, 36, 1988.

[Sadeh, 1991] N. Sadeh. Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling. Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, available as CMU-CS-91-102, 1991.

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

[Smith et al., 1986] S. Smith, M.S. Fox and P.S. Ow. Constructing and Maintaining Detailed Production Plans: Investigations into the Development of Knowledge-based Factory Scheduling Systems. *AI Magazine*, Vol. 7, No. 4, 1986.

[Vollmann et al., 1988] T.E. Vollmann, W.L. Berry, and D.C. Whybark. Manufacturing Planning and Control Systems. Irwin, Homewood, IL, 1988.

[Zweben and Eskey, 1989] M. Zweben and M. Eskey. Constraint Satisfaction with Delayed Evaluation. IJCAI-89 Proceedings, AAAI Press, Menlo Park, 1989.