From: AAAI Technical Report SS-93-03. Compilation copyright © 1993, AAAI (www.aaai.org). All rights reserved.

An Incremental Search Approach to Real-Time Planning: Preliminary Results for a Scheduling Problem

Joseph C. Pemberton and Richard E. Korf*
pemberto@cs.ucla.edu *korf@cs.ucla.edu*
Computer Science Department
University of California, Los Angeles
Los Angeles, CA. 90024

Abstract

We propose an incremental search method for making planning decisions in real time. As an example, we present the problem of scheduling jobs in a factory as a real-time decision problem, and model the real-time constraints as strict limits on the amount of computation that can be performed before a scheduling decision must be made. Our approach is to use incremental heuristic search techniques to simulate and evaluate the effects of future decisions, and thereby generate each individual scheduling decision as needed. We present four incremental search algorithms that can be used to make real-time planning decisions. We also present preliminary results for the two-machine flowshop scheduling problem which indicate that this is a promising method for real-time planning.

1 Introduction

Real-time decision making is an important part of many real-life planning and scheduling problems. One example is air traffic control where the time taken to schedule the arrival of airplanes cannot exceed the maximum amount of time an airplane can fly. Another example is scheduling the flow of trains on a network of tracks. A third example is planning the flow of jobs in a factory, where time spent optimizing a job schedule must be traded-off against the cost of delaying production. In general, costs are associated with any delay caused by the scheduler. For this work, we impose strict decision deadlines, and thus do not allow the planner to delay the schedule. This corresponds to a fixed amount of computation available before each decision must be made. In addition, real-time planning problems typically consist of a sequence of decisions, thus computation spent making one decision will impact the amount of computation available to make subsequent decisions.

*This research was supported by NSF Grant #IRI-9119825, and a grant from Rockwell International. An earlier version of this paper appeared in *Proc. NSF Workshop on Intelligent, Dynamic Scheduling for Manufacturing Systems*, January, 1993.

For example, consider the problem of scheduling the processing of a set of jobs on two different machines, $M1$ and $M2$. Each job consists of one task for each machine, and it must be processed on the first machine, $M1$, before it can be processed on the second machine, $M2$. Associated with each task is the amount of time it takes to process the task on its respective machine. An additional constraint is that each machine can only process one job at a time. One example of this problem is a set of typesetting jobs which must be preprocessed on a computer before being sent to a printer. We assume that the task processing times are randomly distributed and uncorrelated. In the static version of this problem, all jobs are available at the beginning, whereas in the dynamic flowshop scheduling problem, new jobs may continually enter the system.

The task of the scheduler is to determine the order in which jobs should be processed on the two machines subject to the strict decision deadlines that we have imposed. Each scheduling decision is made before the current job finishes processing, thus there is no additional processing cost associated with the scheduler. The cost of a schedule is measured as the mean-time to completion of the jobs scheduled, which is equal to the sum of the completion times for each job, divided by the number of jobs scheduled. The problem of determining which schedule minimizes the mean-time to completion of a set of jobs is NP-complete [10]. Without loss of generality, we assume that the order of processing jobs is the same on both machines [5]. An example two-machine scheduling problem is shown in figure 1, with the optimal schedule for this problem shown in figure 2.

2 What is Incremental Search?

The main idea behind incremental search is to perform a lookahead search on a problem space representation of the planning task, and interrupt the lookahead search when time runs out. At that point the current best decision is executed, and the problem space is updated to reflect the decision. The search process is then repeated for the next decision. We call this approach incremental search because the sequence of decisions is generated incrementally over time as needed.

One way to formulate real-time flowshop scheduling as an incremental heuristic search problem is to associate

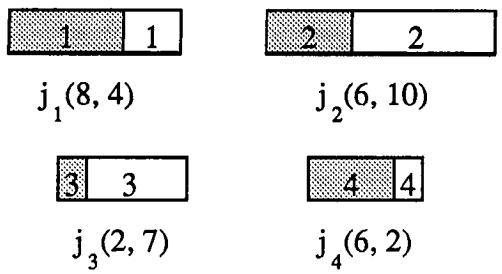

Figure 1: Sample flowshop scheduling problem.

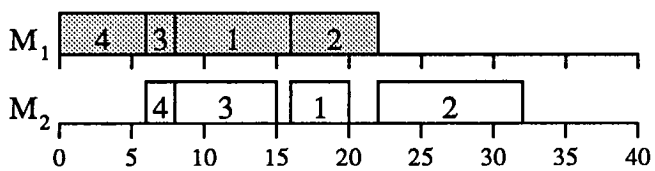

Figure 2: Sample flowshop schedule.

nodes with partial schedules. In this approach, search involves generating new nodes by adding one of the remaining jobs to the current schedule. An example of this approach is shown in figure 3 for the four-job scheduling problem in figure 1. Here the nodes are labeled with the number of the job which is added to the partial schedule at that node. The partial schedule associated with a node can be found by tracing the path from the root to that node. In this figure, the vertical position of the nodes reflects the decision deadline for scheduling the next job, which is when the previously scheduled job will finish on $M1$. The deadline for the first decision depends on the starting conditions, and was chosen arbitrarily for this example.

The cost of a search space node in this representation is the sum of the cost of the partial schedule, and an estimate of the cost of the remaining schedule. Our search procedures use an admissible heuristic function that was developed by Ignall and Schrage [5] to estimate the cost of the remaining schedule.

All incremental search algorithms must address the following two questions: which nodes should be explored, and which decision should be made based on the available information. For traditional search algorithms, it is generally assumed that there is sufficient time to explore the whole problem space in the worst case before any decisions need to be made. This makes it possible for traditional search algorithms to produce optimal solutions. In the real-time setting, there is typically not sufficient time to explore the whole problem space, thus the quality of the solution produced must be traded off against the timeliness of the decisions made.

3 Incremental Search Algorithms

We have considered two general search strategies: depth-first branch-and-bound and best-first search. In this section, we present two incremental search variations for each general search strategy. In all four algorithms, the default is a greedy decision based on a search of depth one from the current root node.

Depth-First Branch-and-Bound

Traditional depth-first branch-and-bound (DFBnB) operates as follows. An initial path is generated in a depth-first manner until a goal node is discovered. Once an initial solution is found, a cost bound is set to the cost of this solution, and the remaining solutions are explored depth-first. If the cost of a partial solution exceeds the current bound, then that partial solution is pruned since the complete solution cost cannot be less. This assumes that the heuristic cost of a child is at least as great as the cost of the parent (*i.e.*, the node costs are monotonic non-decreasing). This assumption is necessary to allow pruning and still guarantee that the solution found is optimal, provided there is sufficient time to complete the search. If a new goal node is found with a lower cost than the current bound, then the bound is updated to the cost of the new goal. The search continues until all paths are either explored to a goal or pruned. At this point, the path associated with the current bound is an optimal solution path. One useful method for ordering the search is to explore the children of a node in increasing order of their heuristic values. This method is called static node ordering, and is employed by both DFBnB algorithms described below.

The problem with DFBnB is that in the worst case it may explore all the nodes in the problem space. If the problem space is a tree, then this corresponds to a worst-case time complexity that is exponential in the depth of the search space, although the space complexity is linear in the depth of the search. In order to adapt this general algorithm to real-time decision making, we have considered two options. Our first is called iterative-deepening branch-and-bound (IDBnB). IDBnB operates by performing a DFBnB search to successively greater search depths while there is time available. The scheduler then commits to the first job along the best partial schedule found by the last completed iteration. This process is repeated for each scheduling decision.

Our second approach is called incremental branch-and-bound (IncBnB). IncBnB first calculates the maximum search depth for which it can guarantee completion in the available time under worst-case conditions. This calculation is based on the branching factor of the problem space. It then performs a depth-first branch-and-bound search to this depth, treating all nodes at the maximum search depth as if they were goal nodes. The scheduler then commits to the first job along the best partial schedule found, and the search is repeated for each scheduling decision. The conservative estimate is used to guarantee completion so that there is search information available for each decision option under the current decision node. Note that IncBnB is a conservative algorithm since due to pruning it is unlikely that the DFBnB search will require the worst-case number of computations. Any time remaining after the DFBnB search is completed is made available for the next decision step.

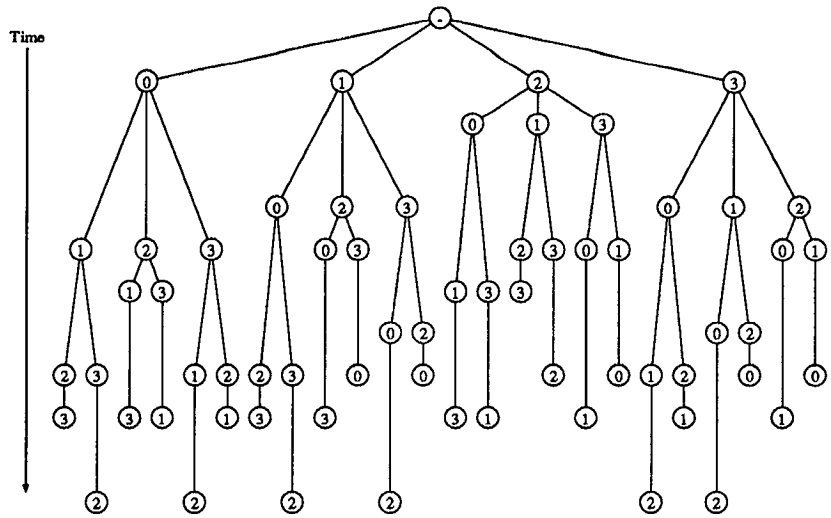

Figure 3: Problem space representation for a four job scheduling problem.

IDBnB does not need to know the branching factor of the search space, and therefore is more versatile than IncBnB. If the branching factor is known (or in some other way it is possible to determine the number of nodes at a given depth from the root), then the initial search depth used by IDBnB can be set to the maximum conservative search depth used by IncBnB. As with IncBnB, any additional time not used by IDBnB is carried over to the next decision step. This will occur when the remaining time available is less than the time taken by the last iteration of the depth-first branch-and-bound. We are able to make this optimization because we have assumed that the heuristic costs of nodes are monotonic non-decreasing along a path towards a complete schedule. This means that each subsequent branch-and-bound search to a greater depth which uses the same node ordering method will take at least as long as the previous iteration, because any nodes in the previous iteration must also be explored in the current iteration.

Best-First Search

We have also considered two incremental variations of best-first search. Traditional best-first search (BFS) operates by maintaining an open list of nodes that have been generated but not yet expanded. Initially the open list contains only the root node. Each step consists of removing a minimum-cost node from the open list, expanding it, and adding its children to the open list. This process is continued until a goal node is removed from the open list, at which point the path to that goal node is an optimal solution path. This assumes that the heuristic cost function is non-overestimating. As with DFBnB, it may be necessary to explore every problem space node before finding an optimal goal node. In this case the worst-case time complexity is exponential in the depth of the search tree. The main problem with best-first search is that the space complexity is also exponential. A related problem is that the time required to process the open list grows with the size of the list (*e.g.*, logarithmically if a heap is used to store the list). Thus even if there is enough memory to store the complete problem space in the open list, time constraints may still make this impractical.

Best-first search algorithms suffer from an additional problem because they explore the best path until the cost of that path exceeds the cost of some other path. When the cost function is monotonic non-decreasing, eventually all paths will look bad if explored deep enough. Thus, eventually the leaf nodes attached to shallow paths will dominate the leaf-nodes associated with deeper paths on the open list, even though these shallow paths do not necessarily correspond to better decisions. DFBnB algorithms avoid this problem by comparing the costs of equal-depth paths.

One solution to this problem is to use two different heuristic functions: one to drive the node exploration decisions, and the other to support the move decisions. Both algorithms described below perform a best-first exploration of the problem space using the heuristic values of the nodes to order the search. However, both algorithms base their move decisions on the average cost of the partial schedule, by dividing the cost of a leaf node by its depth in the problem space. If the exploration and move decisions are based on the actual estimated costs, then shallow nodes are favored. If instead the exploration and move decisions are both based on the average cost of a partial schedule, then an optimal schedule that has a relatively large heuristic cost for the first job scheduled may never be considered. The experimental results presented in the next section confirm that this method of combining best-first exploration and best-average decisions is a reasonable solution to this problem.

We have considered two incremental variations on best-first search that use less than exponential memory, and use the best-first exploration, best-average decision heuristic method. The first is called bounded memory best-first search (BMBFS). BMBFS operates like a traditional best-first search with the additional constraint that the open list size cannot exceed a preset bound.

This guarantees that the amount of time needed to process a node is bounded by a constant. Decisions are made either when the time runs out or the open list is full. In either case the decision is made to move toward the best-average-cost node on the open list. After the decision is made, the problem space and open list are pruned to remove nodes whose partial schedules are no longer possible. The process is repeated until all jobs are scheduled.

The second incremental best-first search algorithm that we have considered is called incremental recursive best-first search (IRBFS), which is an incremental version of recursive best-first search (RBFS) [8]. RBFS is a linear-space best-first search algorithm that operates by maintaining the current search path, and backed-up heuristic values for all siblings of nodes on the path. The heuristic cost of each sibling is the best estimate of the completion cost of the partial schedule associated with that node, based on the entire subtree previously explored under that node. See [8] for a complete description of RBFS. IRBFS performs the recursive best-first search algorithm, one node expansion or backtracking step at a time, while there is time available. When time runs out, a decision is made to move toward the best-average leaf node seen so far, after which the problem space is pruned to eliminate nodes associated with partial solutions that are no longer possible. If the current search node was pruned, then the search process starts over. Otherwise, the search is continued after the cost bounds are updated to reflect the changes to the backtracking options that resulted from the move decision.

4 Experimental Results

In order to compare these four incremental search algorithms (IncBnB, IDBnB, IRBFS, and BMBFS), we performed some experiments on random instances of the real-time flowshop scheduling problem with the mean-completion-time cost function. For each job, processing times for each of two machines were generated at random over the range $[0, 1]$. The actual amount of computation available was varied by associating a different number of node generations with one unit of processing time (*i.e.*, generations per time unit). In addition, the algorithms were given 0.5 time units to schedule the first job.

The algorithms were used to schedule a set of 20 jobs and the results are averaged over 100 random trials. The results in figure 4 show the mean-completion-time versus the number of generations per time unit. The open list size for BMBFS was chosen to be 100 nodes since this produced reasonable results for this scheduling problem.

It is interesting to observe that for smaller amounts of available computation, BMBFS is the best algorithm followed closely by IncBnB. As the maximum number of available generations becomes relatively large, both IRBFS and IDBnB outperform IncBnB. This is due to the fact that IncBnB does not make full use of the computation available. As the maximum number of available generations is increased, BMBFS makes a larger fraction of its scheduling decisions because the open list is full,

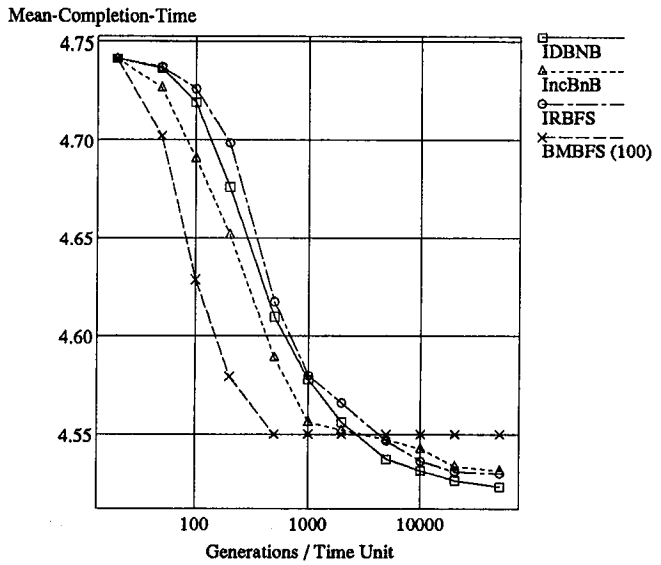

Figure 4: Mean-completion-time vs. maximum node generations available per scheduling decision for 20-job scheduling problems.

as opposed to running out of time. Eventually all of the decisions are made because the open list is full, resulting in a constant level of performance for BMBFS (*e.g.*, when the maximum generations available equals or exceeds 500). If the open list bound is increased, then the change to a constant level of performance for BMBFS will occur later, and the final mean-completion-time will be less. Note that a simple greedy algorithm has a mean-completion-time of 4.74 time units, which corresponds to the first data point for all four algorithms.

Another interesting observation is that the two algorithms that initially perform well (BMBFS and IncBnB), do not make full use of the available computation time because they are overly conservative. This means that there is a potential to improve these two algorithms, if there exists some way to make use of the additional time.

5 Related Work

Dean and Boddy [1, 2] have proposed an area of research called *anytime algorithms*. Anytime algorithms have two basic properties: they always have a decision ready, and in general they produce better quality decisions as the time available increases. Incremental search is one method of generating an anytime algorithm, and thus falls into this area. The main distinction we have made is that the decision deadline is available to the search process, so our incremental search algorithms can use this information to their advantage (*e.g.*, to calculate the maximum search depth (IncBnB)).

Many other researchers are interested in the problem of making planning decisions under time constraints (*i.e.*, anytime planning). For example, Zweben *et al.* [15] have proposed a simulated annealing approach to the rescheduling problem, while Minton and Phillips [9] have

proposed a local-search method for generating a schedule for the Hubble Space Telescope. Still others (*e.g.*, [4]) have proposed expert system approaches for making scheduling decisions. This work differs from our work in that it has focused on generating a complete solution under time constraints, rather than incrementally generating the solution as needed.

Drummond and Bresina [3] have proposed a probablistic technique for incremental plan synthesis that is called ERE (for estimated remaining entropy). This work is closely related to ours, although it is more directed to classical planning domains and does not explicitly handle deadline information. Sutton [13] has proposed a dynamic-programming based approach to incremental planning which is an extension of his original unsupervised learning work. Future work will involve incorporating some of these methods into our incremental search techniques so that we can better address more standard planning problems.

The work most closely related to our incremental search approach is on real-time A* (RTA*) [7]. RTA* is a single-agent incremental search algorithm that used an arbitrary lookahead-depth bound, in order to generate solutions to large search problems. Our work has built on this initial result, but instead of arbitrary lookahead bounds, we have investigated the situation where the problem itself constrains the amount of computation available between subsequent decisions. In addition, we have looked at other decision-making algorithms beyond the simple RTA* algorithm.

Finally, Russell and Wefald[12] have presented a very general decision-theoretic framework for making rational decisions with bounded resources. For this paper, we have investigated simple heuristic search methods for this problem, although we intend to explore the feasibility of making rational meta-level search decisions in real-time as an extension to incremental search. This approach will have to be re-evaluated with respect to the real-time constraints we have described.

There is also a vast literature on scheduling problems (*e.g.*, [4, 6]) that we do not have space to describe in any detail. This area of research is a rich source of heuristic techniques and alternative problems.

6 Conclusions

We have proposed an incremental heuristic search approach to solving real-time planning and scheduling problems. Specifically, we have presented four incremental search adaptations to traditional algorithms for exploring a planning or scheduling problem space in real time. Our preliminary results indicate that these algorithms are promising, since their average performance is a considerable improvement over greedy decisions. Future work will include relaxing the assumption of strict decision deadlines. This will allow the scheduler to trade off the cost of delaying a decision, against the potential benefit from the additional computation.

References

[1] Boddy, M., Problem Solving Using Dynamic Programming, *Proc. AAAI-91*, Anaheim (1991) 738-743.

[2] Dean, T. and M. Boddy, An Analysis of Time-Dependent Planning, in: *Proc. AAAI-88*, St. Paul, Minnesota (1988), 49-54.

[3] Drummond, M. and J. Bresina, Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction, *Proc. AAAI-90*, MIT Press, 1990, pp. 138-144.

[4] Hadavi, K, W. L. Hsu, T. Chen, and C. N. Lee, An Architecture for Real-Time Distributed Scheduling, *AI Magazine*, Fall 1992, pp. 46-56.

[5] Ignall, E. and L. Schrage, Application of the Branch and Bound Technique to some Flow-Shop Scheduling Problems, *Oper. Res.*, Vol. 13, 1965, pp. 400-412.

[6] Kohler, W. H. and K. Steiglitz, Enumerative and Iterative Computational Approaches, in Computer and Job-Shop Scheduling Theory, Ed. E. G. Coffman, Jr., John Wiley and sons, 1976, pp. 229-296.

[7] Korf, R. E., Real-time heuristic search, *Artificial Intelligence*, Vol. 42, No. 2-3, March 1990, pp. 189-211.

[8] Korf, R. E., Linear-Space Best-First Search: Summary of Results, *Proc. AAAI-92*, San Jose, CA, July, 1992.

[9] Minton, S. and A. B. Phillips, Applying a Heuristic Repair Method to the HST Scheduling Problem, *Proc. Workshop on Innovative Approaches to Planning, Scheduling and Control*, Ed. K. Sycara, Morgan Kaufman, November, 1990, pp. 215-219.

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

[11] Pemberton, J. C. and R. E. Korf, Incremental Planning on Graphs with Cycles, *Proc. of the First Annual Conf. on AI Planning Systems (AIPS-92)*, (Editor: J. Hendler, Morgan Kaufmann), College Park, Maryland, June, 1992, pp. 179-188.

[12] Russell, S., and E. Wefald, Do the Right Thing; Studies in Limited Rationality, (M.I.T. Press, Cambridge, Massachusetts, 1991).

[13] Sutton, R. S., Planning by Incremental Dynamic Programming, *Proc. of the Eighth International Workshop on Machine Learning (ML91)*, Ed. L. A. Birnbaum and G. C. Collins, Morgan Kaufmann, 1991, pp. 353-357.

[14] Wefald, E., and S. Russell, Estimating the value of computation: The case of real-time search, *Proc. of the AAAI Spring Symp. on AI and Limited Rationality*, Stanford, CA., March, 1989, pp. 106-110.

[15] Zweben, M, M. Deale, and R. Gargan, Anytime Rescheduling, *Proc. Workshop on Innovative Approaches to Planning, Scheduling and Control*, Ed. K. Sycara, Morgan Kaufman, November, 1990, pp. 251-259.