Performance Analysis of Beam Search with Look Ahead

Suranjan De[1] and Anita Lee–Post[2]

[1] A. Gary Anderson Graduate School of Management, University of California, Riverside, USA
[2] Decision Science & Information Systems Area, School of Management, Gatton College of Business & Economics, University of Kentucky, Lexington, USA

This paper presents a heuristic search strategy called beam search and investigates its effectiveness by applying it to classical job shop scheduling problems. A look-ahead feature is incorporated in the strategy to detect potential resource conflicts. Benchmark problems and existing test cases are used in a performance analysis of the search strategy. The search strategy is implemented in a SUN SPARC station using Common-LISP. Our computational experience using this enhanced version of beam search on job shop scheduling problems are reported.

Keywords: Scheduling, Heuristic Search, Beam Search, Look-ahead

Introduction

This paper presents a heuristic solution strategy called beam search and the application of this strategy to selected job shop scheduling problems reported in the literature. Although we have reported on the use of beam search in our earlier research (De and Lee 1990, 1993), this paper has some unique features. First, we present the basic method, that is beam search. Second, we augment the performance of the knowledge-based beam search method by using look-ahead features. Finally, we report the effectiveness of our method by applying it to job shop scheduling, a well-researched problem, and comparing our results with the reported results in job shop scheduling.

Beam Search Strategy

The complexity of job shop scheduling problems seems well suited for the application of artificial intelligence (AI) methodology. Steffen surveyed some 50 AI-based scheduling systems and indicated that among the 15 systems used for job shop application, half of them relied on certain types of searching strategy to generate the schedule (Steffen 1986). For example, Intelligent Scheduling and Information System (ISIS) used a search method called constraint-directed search for job shop scheduling (Fox and Smith 1984). In fact, the most widely used problem solving technique in AI is searching, which involves choosing a path through a problem space from an initial configuration to a goal state. The objective of searching is to make use of a search technique to locate a "good" solution path efficiently and effectively by limiting the number of alternatives examined.

In general, there are two ways of limiting the search space to manageable size: (1) search effort concentration — search effort is directed toward the promising paths and away from the implausible ones, (2) search effort reduction — no search effort is wasted in exploring the unpromising paths by pruning these paths away from the search. Search techniques such as best-first, A*, and various extensions of A* such as B (Martelli 1977), C (Bagchi and Mahanti 1983), PropA, PropC, and MarkA (Bagchi and Mahanti 1985) belong to the first category. A heuristic estimate called evaluation function, f*, is used to guide the search along solution paths in the order f*. Since none of the paths are

pruned to allow for backtracking, the complexity of these search techniques can be quite large. For example, A* can require $O(2^N)$ node expansions when the search is performed on a graph with N nodes (Martelli 1977, Mero 1984). A search technique such as branch and bound belongs to the second category. It relies on a bounding algorithm to calculate a lower bound which is used to decide whether a solution path can be fathomed or not. However, the bounding algorithm can be computational exhaustive in order to arrive at a good lower bound (Nau et al. 1984).

In this paper, we will focus on a special kind of heuristic search method called beam search. Beam search is sought to improve the search efficiency by incorporating both the guiding and pruning features. It represents the search process as a tree with each node being a partial solution to the problem. As the tree expands in a process called branching or sprouting, successors are generated from each node until a goal node (a complete solution to the problem) is reached. Out of all these successors, only w (w=beam width) will be chosen to expand into the next level, the rest will be pruned. Whether a node is chosen to be expanded or pruned is determined by an evaluation function. Ties are broken arbitrarily, for instance, nodes that are generated first are preferred. There is no backtracking and search efforts are directed to a number of search paths that are conducted parallely. Effectiveness of the search method depends on its ability to locate a goal node that is close to the optimum solution of the problem with minimal computational overhead. The essence of the algorithm is given below:

The Beam Search Algorithm

1. Put the start node, NODE0, on a list, called OPEN, of unexpanded nodes. Calculate f*(NODE0) and associate its value with NODE0. f*(n) is the evaluation function of node n.
2. If OPEN is empty, exit with failure; no solution exists.
3. Select from OPEN the w best nodes for expansion (i.e., the nodes with the w least values of f*; resolve ties arbitrarily); w is the beam width. Move these nodes from OPEN to the list, CLOSED, of expanded nodes. The remaining nodes in OPEN are pruned.
4. If any of the w nodes is a goal node, exit with success; a solution has been found.
5. Expand the w nodes, creating all successors for each node. For every successor node n_s of n, calculate $f^*(n_s)$, and add it to OPEN.
6. Go to Step 2.

End of algorithm.

The success of beam search method depends on its ability to avoid pre-mature pruning, i.e., promising nodes are not omitted in exploration. There are two ways to enhance the search performance in locating the goal node: (1) use a larger beam width, and (2) increase the accuracy of f*. The fact that a larger beam width increases the likelihood of finding the goal node using beam search can be demonstrated using a probabilistic analysis, as shown in the appendix. Intuitively, it can be argued that, since w different paths are selected in the search tree using a beam width w, the probability of finding a goal node is w times as large as for a unit beam width.

The second way to enhance the performance of beam search is to increase the accuracy of f*. f* estimates the likelihood of reaching the goal from a given state, which, when used in conjunction with the beam width, determines which node should be pruned from the search tree. Clearly, if f* is not accurate, a "good" node can be pruned away pre-maturely, hence, hampering the quality of the search result. The focus of this paper is on exploring ways to improve the accuracy of f*. A discussion on this will be provided following a description of the job shop scheduling problem in the next section.

The Job Shop Scheduling Problem

A classical job shop consists of φ jobs $\{J_1, J_2, \ldots, J_\varphi\}$ and μ machines $\{M_1, M_2, \ldots, M_\mu\}$. Each job has μ operations to be performed on one of the μ machines in a unique sequence according to its technological constraints. The scheduling problem is to determine a time table for each O_{jm} (the operation of jth job on mth machine) to achieve certain objective such as minimum makespan, minimum sum of completion times, minimum lateness, minimum tardiness

while satisfying the technological and resource constraints.

A number of assumptions are considered. With regard to the machines, we assume that (i) the number of machines and their capabilities are known and fixed, (ii) all machines are available at the same time and remain available throughout the scheduling period, (iii) there is only one machine of each type, (iv) no machine may process more than one operation at a time, (v) all processing times are sequence-independent, (vi) no pre-emption is allowed, i.e., each operation, once started, must be completed without interruption, (vii) machines may be idle. With regard to the jobs, we assume that (i) the number of jobs to be scheduled is known and fixed, (ii) each job is an entity, i.e., no two operations of the same job may be processed at the same time, (iii) each job has a known and fixed job routing and no alternative routings are allowed, (iv) the jobs are assumed to be independent and available at the same time, (v) no cancellation of job is allowed, i.e., each job must be processed to completion, (vi) in-process inventory is allowed, i.e, jobs may wait for their next machine to be free. Based on these assumptions, we address a job shop scheduling problem that is deterministic (i.e., the parameters that define the problem are known and fixed) and static (i.e., the number of jobs and their ready times are known and fixed).

The interest in scheduling has been stimulated by a number of pragmatic and theoretical considerations. From a practical point of view, a good production schedule provides a competitive edge to the manufacturer through low work-in-process, high machine utilization, increase in productivity, efficient operations management, to mention just a few of them. From a theoretical point of view, traditional approaches have been preoccupied with the design of optimization algorithms for small-sized problems. Recent theoretical advances in complexity theory and AI techniques have redirected the body of scheduling research to examine intelligent heuristic search methods which can be applied to more realistic scheduling problems (Van Laarhoven et al. 1992, Aarts et al. 1994, Taillard 1994). The purpose of this research is to examine an intelligent search strategy called beam search, to solve job shop scheduling problems of reasonable size. Various sources of knowledge are employed to reduce the search time and space in locating a "good" job shop schedule.

Enhancing the Performance of Beam Search

The Look Ahead Feature

The effort to improve the accuracy of f^* on the performance of beam search is demonstrated in this section with an example. Generally, $f^*(n)$ is an estimate of the minimum cost of a solution path passing through node n. It has two components:

$g(n)$, which is the cost of reaching node n from the starting node

$h(n)$, which is the cost of reaching the goal node from node n.

When beam search is applied to job shop scheduling with the objective to locate a minimum makespan schedule, $f^*(n)$ can be defined as $\max_{j\in\{job\}}\{g_j(n) + h_j(n)\}$, where {job} is the set of jobs to be considered, $g_j(n)$ is the completion time of the current operation scheduled for job j, and $h_j(n)$ is the completion time for the remaining operations of job j. $g_j(n)$ is a perfect estimate once an operation is scheduled, hence, it has no effect on the accuracy of $f^*(n)$. We propose the use of a look-ahead feature to detect any resource conflicts that may exist in performing the remaining operations of job j in estimating $h_j(n)$. In other words, $h_j(n)$ is the sum of the total processing time for the remaining operations of job j and the total time of waiting for resources in conflict to complete the remaining operations in job j. The total waiting time is determined by projecting the future resource allocation by completing the partial schedule at node n with the use of some heuristic dispatching rules such as the shortest processing time (SPT), the longest processing time (LPT), and most work remaining (MWRK). These dispatching rules resolve resource conflicts among jobs by prioritizing jobs competing for the same resource in the following manner:

SPT: job with the shortest processing time goes first

LPT: job with the longest processing time goes first

		Machine		
		1	2	3
Job	1	129	78	9
	2	43	28	90
	3	71	81	85

Table 1. Processing times

		Machine		
		1	2	3
Job	1	1	2	3
	2	1	3	2
	3	3	1	2

Table 2. Routings

MWRK: job with the most work remaining goes first.

An Example

The following 3–job, 3–machine case, modified from Fisher and Thompson (1963) illustrates how the use of a look-ahead feature in f* improves search quality. The operation time and machine assignment for each job are given in tables 1 and 2 respectively. The problem is to construct a schedule to minimize the makespan.

Figure 1 shows a search tree generated by using a beam search of beam width 5 (equivalent to expanding at most 5 successors at each level). Note that a beam width exceeding 5 has no effect on the search, since all the possible active schedules are enumerated in this case. Each node of the tree represents a partial schedule with a collection of triplets (i, j, k), indicating that job i finishes using machine j at time k. The numbers next to each node n are the values of two evaluation functions f*(n) and f*(n)'. f*(n) is the evaluation function without the look-ahead feature, i.e., $h_j(n)$ is simply the sum of remaining operation times of job j. f*(n)' is the enhanced evaluation function with look-ahead feature and is given in parentheses.

In the above example, the partial schedule at any node n is projected to completion by using the SPT rule. With this look-ahead information, f*(n)' gives a more accurate portrayal of the behavior of a solution path. It indicates which

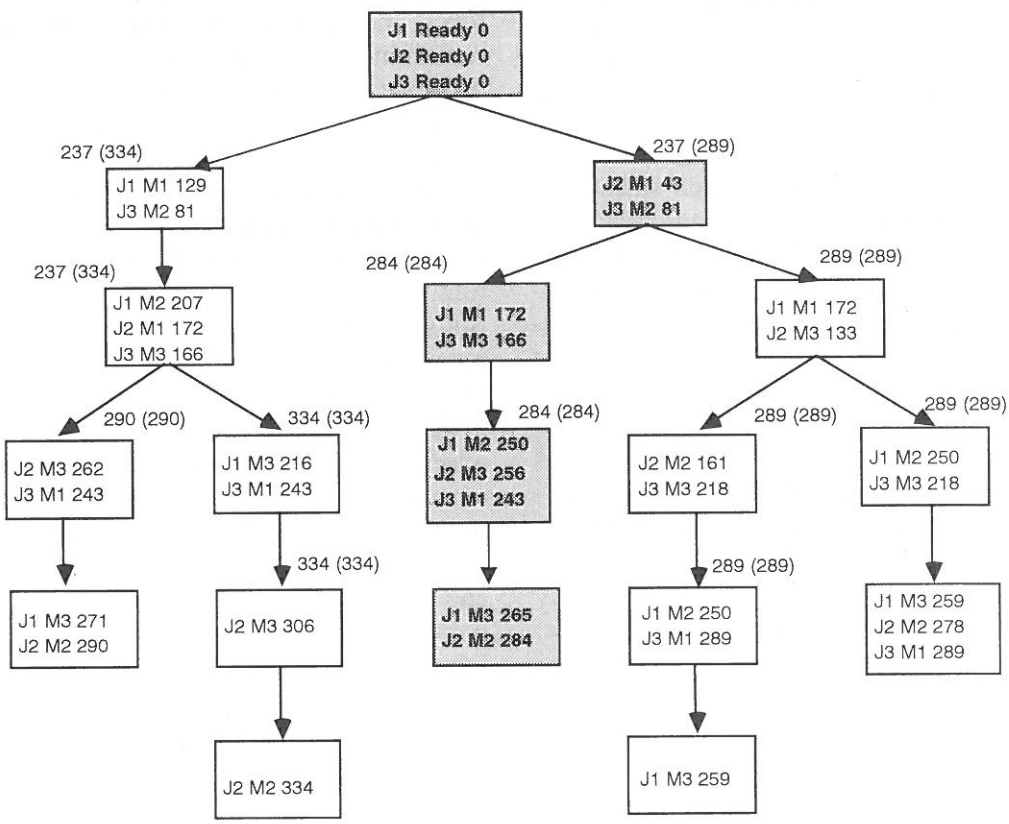

Fig. 1. A beam search tree (w=5)

branch is more promising much earlier than by using f*(n). With the additional knowledge to guide the search, a smaller search tree can be used without jeopardizing the quality of the solution. In the above example, a beam search tree with the beam width as low as 1 is sufficient to generate the optimal solution (the path with shaded nodes, giving a makespan of 284) if f*(n)' is used. The total number of nodes generated is 5. Whereas, if f*(n) is used, a beam width of 2 is required in order to prevent premature pruning of the left side branch. As a result, the total number of nodes generated in the latter case is 10, doubling the number of nodes needed in the former case.

Computational Experience

A set of 20 job shop problems is selected from Adams et al. (1988) to investigate the effects of (1) beam width and (2) different look-ahead heuristics on the performance of beam search. Two benchmark problems from Fisher and Thompson (1963), namely, the 10–jobs 10–machines and 20–jobs 5–machines problems are also tested. Solution quality is measured by the value of makespan of schedule generated using these 22 different test cases. Computational complexity is indicated by the total number of nodes generated and by the amount of running times spent in producing the schedule. The running times include the input, output and CPU times in our SUN SPARC station 1+ using

Problem	m × j	SB	w=1	w=3	w=5	others
1	5 × 10	**666**	712	681	681	**666** (w=20)
2	5 × 10	720	766	728	718*	694* (w=20)
3	5 × 10	623	662	662	654	619* (w=50)
4	5 × 10	597	632	632	632	618 (w=20)
5	5 × 10	**593**	**593**			
6	5 × 15	**926**	**926**			
7	5 × 15	**890**	910	910	910	892 (w=10)
8	5 × 15	868	898	892	892	863* (w=20)
9	5 × 15	951	966	951		
10	5 × 15	959	961	958*		
11	5 × 20	**1222**	**1222**			
12	5 × 20	**1039**	1114	1113	1107	**1039** (w=10)
13	5 × 20	**1150**	1203	1170	1173	**1150** (w=15)
14	5 × 20	**1292**	**1292**			
15	5 × 20	**1207**	1268	1251	1246	1232 (w=10)
16	10 × 10	1021	1081	1078	1076	1012*(w=15)
17	10 × 10	796	812	812	812	794* (w=15)
18	10 × 10	891	932	932	932	891 (w=15)
19	10 × 10	875	947	898	896	891 (w=10)
20	10 × 10	924	1052	1028	971	941 (w=15)
F&T 1	10 × 10	1015	1001*	1001*	1001*	
F&T 2	5 × 20	1290	1287*	1270*	1234*	

Notation: SB: Adams et al.'s straight version shifting bottleneck
m: no. of machines
j: no. of jobs
w: beam width
F&T: Fisher & Thompson

Note: (1) SPT is the look-ahead heuristic unless stated otherwise
(2) number in bold face is the optimal makespan
(3) number with * is the improved makespan

Table 3. Makespan using different beam widths

Problem	m × j	No Look Ahead	MWKR	SPT	LPT
1	5 × 10	772	771	681	756
2	5 × 10	793	758	728	811
3	5 × 10	699	753	662	690
4	5 × 10	734	702	632	746
5	5 × 10	593	619	593	666
6	5 × 15	1110	1054	926	964
7	5 × 15	975	1099	910	954
8	5 × 15	974	914	892	957
9	5 × 15	1065	1004	951	980
10	5 × 15	1044	977	958	964
11	5 × 20	1330	1353	1222	1324
12	5 × 20	1128	1146	1113	1115
13	5 × 20	1343	1312	1170	1237
14	5 × 20	1489	1292	1292	1292
15	5 × 20	1330	1324	1232	1328
16	10 × 10	1054	1145	1076	1092
17	10 × 10	909	1010	812	836
18	10 × 10	1036	1003	932	960
19	10 × 10	939	1026	898	1064
20	10 × 10	1081	1128	1028	1072

Notations:
m: no. of machines
j: no. of jobs
MWKR: Most WorK Remaining
SPT: Shortest Processing Time
LPT: Longest Processing Time

Table 4. Makespan using different look-ahead heuristic rules in beam search with a beam width of 3

Common-LISP.

The effects of beam width and look-ahead heuristics on the solution quality are reported in Tables 3 and 4. Table 3 compares the makespan of schedules obtained by using beam search augmented with SPT look-ahead heuristic under various beam widths. Table 4 compares the makespan of schedules obtained by using three different look-ahead heuristics, namely, SPT, LPT, and MWK. Makespans of schedules generated without using the look-ahead heuristics (No Look-Ahead) are also included.

The effect of beam width on the solution quality and search complexity are illustrated in Figures 2 to 4. In general, the solution quality improves as the beam width increases, but at the expense

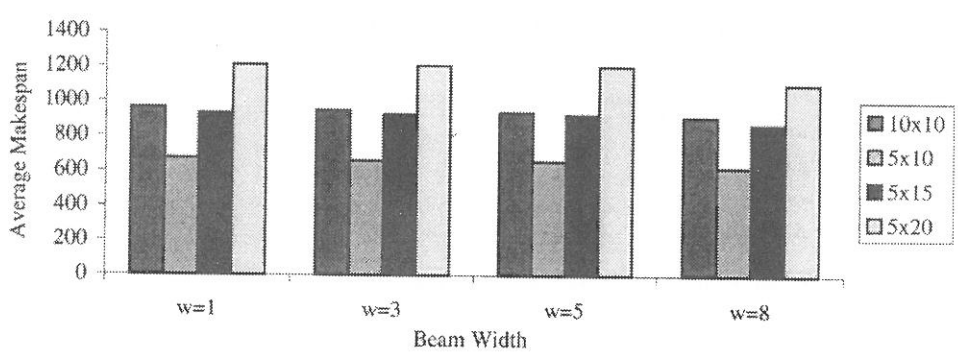

Fig. 2. Effect of beam width on average makespan

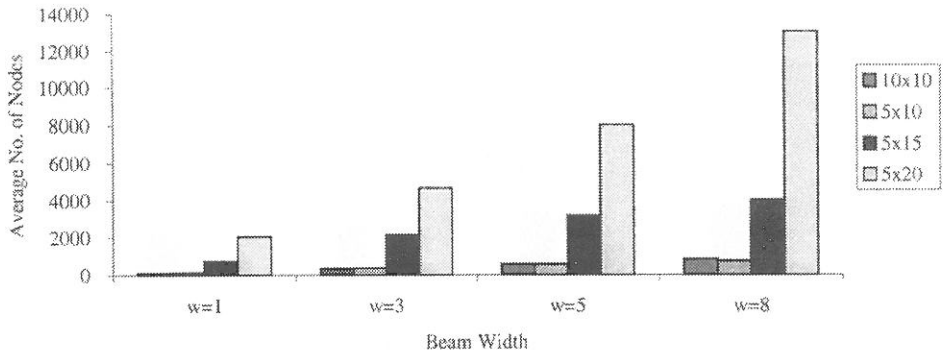

Fig. 3. Effect of beam width on average no. of nodes generated

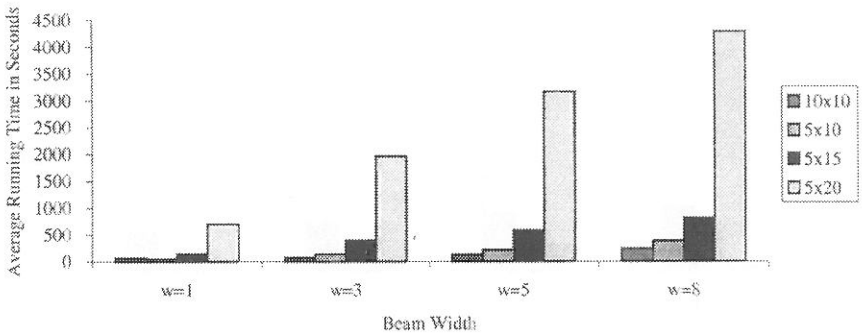

Fig. 4. Effect of beam width on average running time

of a larger search space and longer running time.

The effect of look-ahead heuristics on the solution quality and search complexity are shown in Figures 5 to 7. It is interesting to note that solution quality improves with the use of look-ahead heuristics but the computational overhead involved is not significant. Among the three look-ahead heuristics rules, SPT performs the best in terms of its ability to generate schedules with the shortest makespan. It is also worth mentioning that complexity of the problem here depends on the machines to jobs ratio. The 5–machine and 20–job cases (a machine to job ratio of 1:4) are computationally more complex than the 10–machine and 10–job cases (a machine to job ratio of 1:1).

The results of our experiment support the use of look-ahead heuristics in improving the quality of the schedule. Among different look-ahead heuristics, the SPT performs best in obtaining schedules with the shortest makespan. The additional computational overhead involved in using the look-ahead is insignificant. The performance of our beam search augmented with

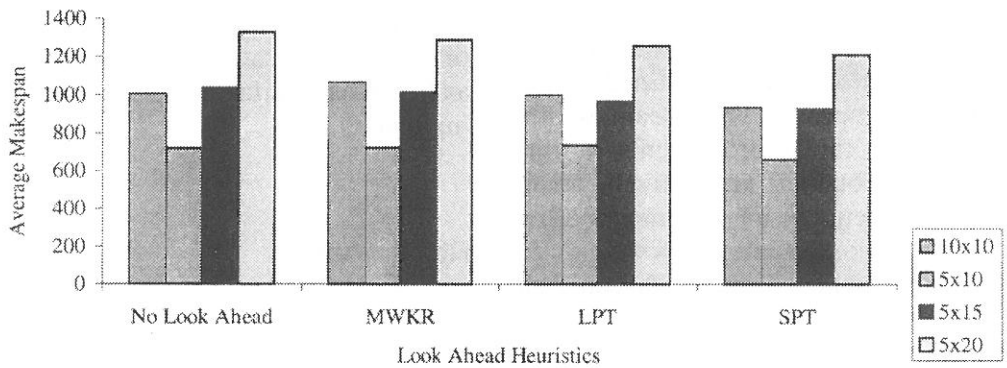

Fig. 5. Effect of look ahead on average makespan

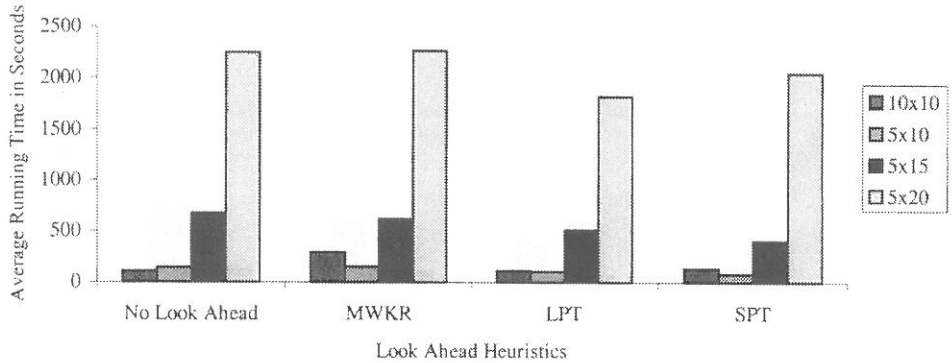

Fig. 6. Effect of look ahead on average no. of nodes generated

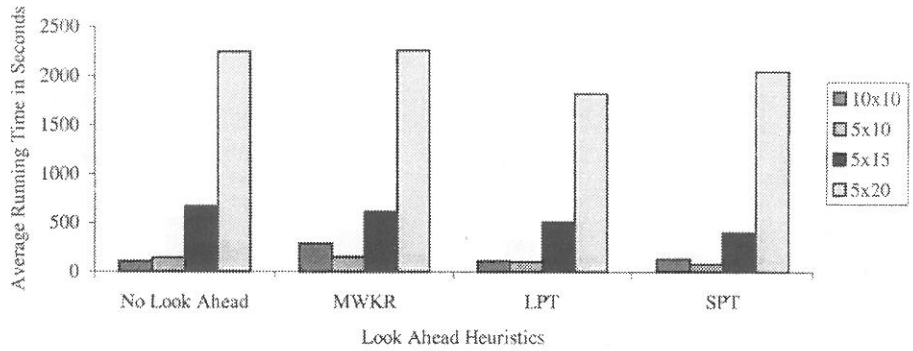

Fig. 7. Effect of look ahead on average running times

SPT look-ahead is comparable to Adams et al.'s straight version of shifting bottleneck algorithm. In 8 out of the 22 test cases, we are able to produce a shorter makespan. Also, we are able to obtain 7 optimal makespans (out of 9), 4 of which by using a beam width as low as 1.

Conclusion

This paper reports the performance of a knowledge-based heuristic search strategy called beam search on well-known job shop scheduling problems. Some of the key features of our computational experience are summarized below. First, the solution quality obtained by using beam search improves with beam width. Moreover, most of the improvements in performance can be obtained at relatively small beam widths. Subsequent improvements often require substantial computational overhead. In other words, our proposed approach can find "good" solutions with very modest computational overhead. Second, improvements in the solution quality are obtained by using domain knowledge stored in the knowledge base; this domain knowledge is used to identify the most promising branches. Third, improvements in the solution quality are obtained by using a search mechanism whose tree size can be controlled by the user. Finally, we have incorporated a look-ahead feature to detect potential resource conflicts; the use of the look-ahead feature also improves the quality of the solution obtained.

The beam search strategy reported here in this paper is by no means limited to generating makespan-based job shop schedules. In fact, our proposed approach can be easily applied to due-date oriented job shop scheduling problems or more complicated scheduling problems, such as flexible manufacturing, with equally promising results.

References

AARTS, E.H.L., VAN LAARHOVEN, P.J.M. AND ULDER, N.L.J., (1994) A Computational Study of Local Search Algorithms for Job Shop Scheduling, *ORSA Journal on Computing*, 6, 2, 118–125.

ADAMS, J., BALAS, E. AND ZAWACK, D., (1988) The Shifting Bottleneck Procedure for Job Shop Scheduling, *Management Science*, 34, 3, 391–401.

AMAR, A.D. AND GUPTA, J.N.D., (1986) Simulated Versus Real Life Data in Testing the Efficiency of Scheduling Algorithms, *IIE Transactions*, March, 1986, 16–25.

BAGCHI, A. AND MAHANTI, A., (1983) Search Algorithms Under Different Kinds of Heuristics — A Comparative Study, *Journal of the Association for Computing Machinery*, 30, 1–21.

BAGCHI, A. AND MAHANTI A., (1985) Three Approaches to Heuristic Search in Networks, *Journal of the Association for Computing Machinery*, 32, 1–27.

DE, S. AND LEE, A., (1990) FMS Scheduling Using Filtered Beam Search, *Journal of Intelligent Manufacturing*, 1, 3, 165–183.

DE, S. AND LEE, A., (1993) Flexible Assembly Scheduling Using a Knowledge-based Approach, *Expert Systems with Applications : An International Journal*, 6, 3, 309–326.

FISHER, H. AND THOMPSON, G.L., (1963) *Industrial Scheduling*, Englewood Cliffs, NJ: Prentice–Hall.

FOX, B.R. AND SMITH, S.F., (1984) ISIS: A Knowledge Based System for Factory Scheduling, *Expert Systems*, 1, 1, 25–49.

MARTELLI, A., (1977) On the Complexity of Admissible Search Algorithms, *Artificial Intelligence*, 8, 1–13.

MERO, L., (1984) A Heuristic Search With Modifiable Estimate, *Artificial Intelligence*, 23, 13–27.

NAU, D.S., KUMAR, V. AND KANAL, L., (1984) General Branch and Bound, and Its Relation to A* and AO*, *Artificial Intelligence*, 23, 29–58.

STEFFEN, M.S., (1986) A Survey of Artificial Intelligence-Based Scheduling Systems, *IIE Fall Industrial Engineering Conference Proceedings*, 395–405.

TAILLARD, E.D., (1994) Parallel Taboo Search Techniques for the Job Shop Scheduling Problem, *ORSA Journal on Computing*, 6, 2, 108–117.

VAN LAARHOVEN, P.J.M., AARTS, E.H.L. AND LENSTRA, J.K., (1992) Job Shop Scheduling By Simulated Annealing, *Operations Research*, 40, 113–125.

Appendix

Given a search tree with a constant branching factor b, b successors will be generated from any node of the tree. Assume there is only one goal node at the bottom of the tree at level D, where the goal node represents a schedule with a minimum makespan. We would like to show that the probability of success (finding the goal node) by using beam search, P_s, is dependent on the beam width, w. To isolate the effect of w on P_s, f^* will not be used to guide the search. In other words, at each level of the search tree, w nodes will be selected randomly to expand into the next level.

Since there is only one goal node, the path from the root to the goal node (the optimal solution path) will be unique. Hence, P_s can be viewed as the probability of selecting a successful node (a node on the optimal solution path) repeatedly at various depths of the tree. If $p(n_d)$ denotes the probability that node n at level d is successful, P_s can be computed as:

$$(1) \qquad P_s = \prod_{d=1}^{D} p(n_d)$$

For d=0 (the root node), $p(n_0) = 1$, since the root node is always successful. For various d, $p(n_d)$ can be found by noting if a successful node is reached at level d, then one of the nodes on level d-1 is also a successful node. This yields (if $b > w$):

$$(2) \quad p(n_d) = \begin{cases} \dfrac{\binom{wb-1}{w-1}}{\binom{wb}{w}} = \dfrac{1}{b} & \text{if } d > 1 \\ \dfrac{w}{b} & \text{if } d = 1 \end{cases}$$

By putting expressions (1) and (2) together, we have $P_s = \dfrac{w}{b^D}$.

Hence, it can be concluded that $P_s \alpha w$ for a given search tree.

Received: April, 1997
Accepted: December, 1997

Contact address:
Suranjan De
A. Gary Anderson Graduate School of Management
University of California
Riverside, CA 9251–0203
USA
E-mail: suranjan.de@ucr.edu
Phone: (818) 986-4652

Anita Lee–Post
Decision Science & Information Systems Area
School of Management
Gatton College of Business & Economics
University of Kentucky
Lexington, KY 40506
E-mail: dsianita@ukcc.uky.edu
Phone: (606)257-1948
Fax: (606) 257-8031

SURANJAN DE received his Ph.D. degree in Management from Purdue University. He is now a Visiting Associate Professor in the A. Gary Anderson Graduate School of Management at the University of California, Riverside. In the past, Professor De has served on the faculty at Santa Clara University, the University of Iowa and Purdue University. His research interests include decision support systems and knowledge-based systems.

ANITA LEE–POST is an Associate Professor of the Decision Science and Information Systems area at the University of Kentucky. She received her Ph.D. in Business Administration from the University of Iowa in 1990. Her research interests include artificial intelligence, machine learning, knowledge-based systems, computer integrated manufacturing, and group technology. She has published extensively in journals such as International Journal of Production Research, AI Magazine, Expert Systems, Expert Systems with Applications, IEEE Expert, Journal of the Operational Research Society, and OM Review. She is the author of Knowledge-based FMS Scheduling: An Artificial Intelligence Perspective. She serves on the editorial review boards of International Journal of Computational Intelligence and Organization, Journal of Managerial Issues, and Journal of Database Management.