ELSEVIER

European Journal of Operational Research 118 (1999) 390–412

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/orms

Theory and Methodology

Job shop scheduling with beam search

I. Sabuncuoglu *, M. Bayiz

Department of Industrial Engineering, Bilkent University, 06533 Ankara, Turkey

Received 1 July 1997; accepted 1 August 1998

Abstract

Beam Search is a heuristic method for solving optimization problems. It is an adaptation of the branch and bound method in which only some nodes are evaluated in the search tree. At any level, only the promising nodes are kept for further branching and remaining nodes are pruned off permanently. In this paper, we develop a beam search based scheduling algorithm for the job shop problem. Both the makespan and mean tardiness are used as the performance measures. The proposed algorithm is also compared with other well known search methods and dispatching rules for a wide variety of problems. The results indicate that the beam search technique is a very competitive and promising tool which deserves further research in the scheduling literature.

Keywords: Scheduling; Beam search; Job shop

1. Introduction

Beam search is a heuristic method for solving optimization problems. It is an adaptation of the branch and bound method in which only some nodes are evaluated. In this search method, at any level only the promising nodes are kept for further branching and the remaining nodes are pruned off permanently. Since a large part of the search tree is pruned off aggressively to obtain a solution, its running time is polynomial in the size of the problems.

This search technique was first used in artificial intelligence for the speech recognition problem (Lowerre, 1976). There have been a number of applications reported in the literature since then. Fox (1983) used beam search for solving complex scheduling problems by a system called ISIS. Later, Ow and Morton (1988) studied the effects of using different evaluation functions to guide the search and compare the performance of beam search with other heuristics for the single machine early/tardy problem and the flow shop problem. They also proposed a variation of this technique called the filtered beam search and found optimal settings of the search parameters.

In another study, Chang et al. (1989) used beam search as a part of their FMS scheduling algorithm called bottleneck-based beam search (BBBS). Results indicate that BBBS outperforms widely used dispatching rules for the makespan criterion.

* Corresponding author. Tel.: 90 312 266 4126; fax: 90 312 266 4126; e-mail: sabun@bilkent.edu.tr

0377-2217/99/$ – see front matter
PII: S0377-2217(98)00319-1

Another beam search application to FMSs is reported by De and Lee (1990) who showed that the solution quality of filtered beam search algorithm is better than depth-first type heuristics in terms of the average maximum lateness and average flowtime measures. The authors also showed that beam search was better than breadth first type heuristics in terms of number of nodes created during the search. In another study, Hatzikonstantis and Besant (1992) proposed a heuristic called A^* for the job shop problem with the makespan criterion. A^* algorithm is very similar to the beam search method. The only difference is that A^* algorithm is a best-first search based heuristic and aims to find minimum-cost paths in search trees. Computational tests indicate that this heuristic search algorithm performs better than dispatching rules. Finally, Sabuncuoglu and Karabuk (1998) proposed a filtered beam search algorithm for more complex FMS environment in which AGVs are explicitly modeled in addition to the routing and sequence flexibilities. Their computational experiments show that the beam search performs better than the machine and AGV scheduling rules under all experimental conditions for the makespan, mean flow time and mean tardiness criteria. Their results also indicate that the beam search based scheduling algorithm exploits flexibilities inherent in FMS more effectively than other methods. An overview of the beam search and its applications to optimization problems can be found in Morton and Pentico (1993).

Even though beam search has been used to solve a wide variety of optimization problems, its performance is not generally known for scheduling problems. Because, in the existing research work, beam search is primarily applied to the FMS scheduling problem with additional considerations on MHS finite buffer capacities and flexibilities and compared with only some dispatching rules. Hence, its relative performances with respect to the known optimum solution and other recently developed heuristics are not known. Besides, it has not been thoroughly studied as a problem solving strategy with certain evaluation functions and search parameters.

This paper attempts to achieve some of these objectives. First of all, we measure the performance of beam search (with respect to optimum solutions) and compare it with other well-known algorithms. In addition, we investigate the effectiveness of various rules as local and global functions of the beam search applications. The previous research indicates that the values of filter and beam width affect the performance of the beam search. Hence, we also examine the performance of beam search for various values of filter and beam width and find their proper values. Furthermore, we test two well-known schedule generation schemes (active and nondelay schedule generation schemes) in the context of the beam search applications to the job shop problems.

The rest of the paper is organized as follows. Section 2 gives definitions of the job shop problem. Then a beam search based algorithm is developed for the problem. This is followed by a discussion on test problems and computational experience with the proposed algorithm. The paper ends with concluding remarks and suggestions for further research.

2. Problem definition

The job shop problem is to determine the start and completion time of operations of a set of jobs on a set of machines, subject to the constraint that each machine can handle at most one job at a time (capacity constraints) and each job has a specified processing order through the machines (precedence constraints). Explaining the problem more specifically, there are a finite set J of jobs and a finite set M of machines. For each job $j \in J$, a permutation $(\sigma_1^j, \ldots, \sigma_m^j)$ of the machines (where $m = |M|$) represents the processing order of job j through the machines. Thus, j must be processed first on σ_1^j, then on σ_2^j, etc. Also, for each job j and the machine i, there is a nonnegative integer p_{ji}, the processing time of job j on machine i.

Since, this problem is NP-Hard (Garey and Johnson, 1979) and very difficult to solve, early studies on this problem were directed at development of effective priority dispatching rules. But later, due to the general deficiencies exhibited by priority dispatching rules, researchers concentrated on more complex techniques. Tabu search

(Glover, 1989, 1990), large step optimization (Martin et al., 1989), simulated annealing (Matsua et al., 1988; Aarts et al., 1991), neural networks (Sabuncuoglu and Gurgun, 1996) and genetic algorithms (Nakano and Yamada, 1991) are examples of the formalized applications of such scheduling techniques to the job shop problem. A comprehensive bibliography of these studies for the job shop problem is given by Jain and Meeran (1996). In this paper, we measure the performance of beam search for the makespan and mean tardiness criteria. Makespan, $C_{\max}$ is the duration in which all operations for all jobs are completed. Tardiness is the positive difference between the completion time and due date of a job. The objective is to determine starting times for each operation in order to minimize the makespan or mean tardiness while satisfying all the capacity and precedence constraints:

$$C^*_{\max} = \min(C_{\max}) = \min_{\text{feasible schedules}} (\max(C_i) : \forall_i \in J),$$

$$T^* = (1/|J|) \min_{\text{feasible schedules}} \left(\sum_{i \in J} \max(0, C_i - d_i) \right),$$

where C_i and d_i are the completion time and due date of job i, respectively.

3. Beam search

Beam search is like breadth-first search since it progresses level by level without backtracking. But unlike breadth-first search, beam search only moves downward from the best β promising nodes (instead of all nodes) at each level and β is called the *beam width*. The other nodes are simply ignored. In order to select the best β nodes, promise of each node is determined. This value can be determined in various ways. One way is to employ an evaluation function which estimates the minimum total costs of the best solution that can be obtained from the partial schedule represented by the node. Such an evaluation function may require as little effort as computing some priority rating or as much as completing the partial schedule by some method. The former method is called *one-step priority evaluation function*, and the latter case is called *total cost evaluation function*. The one-step priority evaluation function has a *local view*, whereas, the total cost evaluation employs a projecting mechanism to estimate costs from the current partial solution. Therefore, evaluation is based on a *global* view of the solution. Unfortunately, there is a trade-off between these two approaches: one-step (local) evaluation is quick but may discard good solutions. On the other hand, more thorough evaluation by the global function is more accurate but computationally more expensive.

A filtering mechanism is also proposed in the literature to reduce the computational burden of beam search. During filtering some nodes are discarded permanently based on their local evaluation function values. Only the remaining nodes are subject to global evaluation. The number of nodes retained for further evaluation is called the *filter width* (α).

As shown in Fig. 1, we determine the promising nodes (beam nodes) by performing local and global evaluations and proceed with the search through these selected nodes. After determining the first beam nodes at level 1, we apply the algorithm to these nodes independently and generate one partial tree from each of them. We refer to these partial trees as beams. For each beam after filtering based on the outcome of the global evaluation, one node (beam node for the next level) is selected among the descendants of each node. Since we have beam width number of nodes in the former level while keeping one descendant, we again have beam width number of nodes in the next level and therefore the search progresses through β parallel beams. In the application of beam search, one can have as many beams as possible. On the other hand, filter width is defined for each beam independently. Thus, the number of beams (which is defined by the beam width parameter) can be feasibly greater than the filter width value.

Instead of performing the beam search independently through the different beams, we could pool at one level all the descendants nodes generated from all the beam nodes and perform local

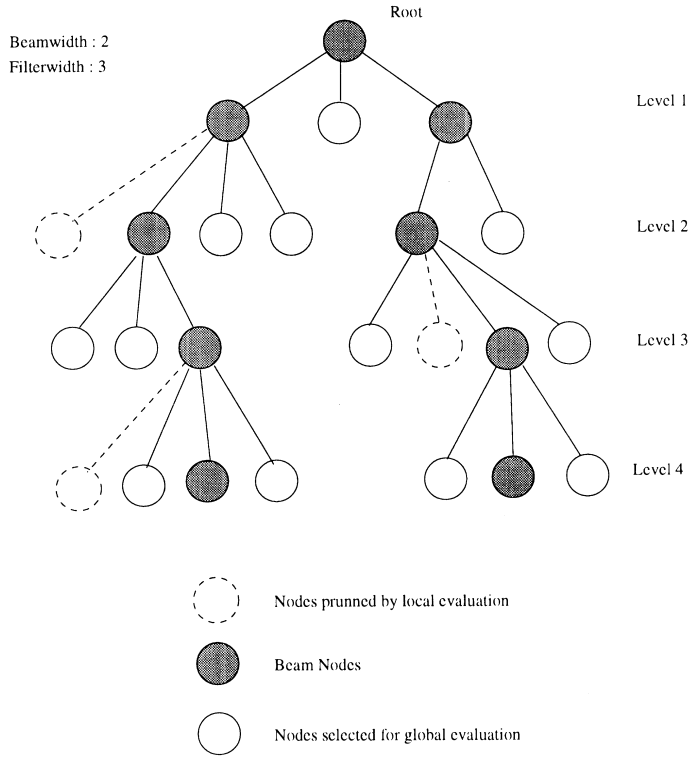


Fig. 1. Representation of beam search tree.

evaluation for all of them. Then we could apply global evaluation for the filtered nodes and select beam nodes for the next level. However we would not rather use this approach because different nodes at the same level represent different partial schedule. If the local evaluation is a function of the partial schedule (as in the case of the lower bound based local evaluation function to minimize makespan), values of the local evaluation function obtained for expanding one beam node cannot be compared legitimately with the values of the local evaluation functions obtained for expanding another beam node at the same level. Therefore, nodes in each of the parallel beams are evaluated separately and only one node is selected for each.

3.1. The proposed beam search based algorithm

In an algorithm like beam search, there are two important issues: (1) search tree representation and (2) application of a search methodology. As mentioned earlier, each node in the search tree, corresponds to a partial schedule. A line between two nodes represents the decision to add a job to the existing partial schedule. Consequently, leaf nodes at the end of the tree correspond to complete schedules. Baker (1974) describes two search tree generation procedures (active and nondelay) schedules for the job shop systems. In the proposed algorithm, these procedures are used to generate branches from a given node.

The second issue in beam search is the determination of search methodology. In the proposed algorithm, the filtered beam search method is used to perform a search in the tree. All the nodes at level 1 are globally evaluated to determine the best β number of promising nodes. The selected nodes become the first nodes of the β number of parallel beams. In the subsequent levels, descendants of the beam nodes are first locally evaluated to find number of promising nodes and then these nodes

are further globally evaluated to select the next beam node. If the number of nodes expanded in the first level are less than the specified beam width, then all the nodes are expanded until the number of nodes are greater than the beam width in the next level.

Since the quality of the filtered beam search depends on the quality of local and global evaluation functions as well as the beam and filter width parameters, a thorough analysis must be carried out to determine the nature of these functions and parameters. In this study, local evaluation is performed by using simple dispatching (or priority) rules. Global evaluation of a node is determined as the estimation of the upper bound value for the solutions that can be generated if that node is added to the partial schedule. This is performed by generating a complete schedule from a given partial schedule by applying dispatching rules and reading the value of the objective function. Priority rules in local and global evaluation functions are not necessarily the same. In this study, we test several rules for this purpose.

In the proposed algorithm, all the nodes at level 1 are globally evaluated to determine the best β number of promising nodes. The selected nodes become the first nodes of the β number of parallel beams. In the subsequent levels, descendants of the beam nodes are first locally evaluated to find α number of promising nodes and then these nodes are further globally evaluated to select the next beam node. If the number of nodes expanded in the first level are less than specified beam width, then all the nodes are expanded until the number of nodes are greater than beam width in the next level.

To be more specific, procedural form of the beam search based algorithm is given as follows.

Procedure (*BEAM SEARCH*): In the filtered beam search algorithm, we use active and nondelay schedule generation methods discussed in Baker (1974, pp. 189), in order to generate a search tree. At each level of the tree, operations with assigned starting times form a partial schedule. Hence, given a partial schedule for any job shop problem, a set of schedulable operations is first constructed.

Let PS_t be a partial schedule containing t scheduled operations, S_t be the set of schedulable operations at stage t, corresponding to a given PS_t, σ_t the earliest time at which operation $j \in S_t$ could be started, and ϕ_t the earliest time at which operation $j \in S_t$ could be completed.

Active schedule generation subroutine (*ACTIVE*):

Step 1: Determine $\phi^* = \min_{j \in S_t}\{\phi_j\}$ and the machine m^* on which ϕ^* could be realized

Step 2: For each operation $j \in S_t$ that requires machine m^* and for which $\sigma_j < \phi^*$, generate a new node which corresponds to the partial schedule in which operation j is added to PS_t and started at time σ_j.

Nondelay schedule generation subroutine (*NONDELAY*):

Step 1: Determine $\sigma^* = \min_{j \in S_t}\{\sigma_j\}$ and the machine m^* on which σ^* could be realized

Step 2: For each operation $j \in S_t$ that requires machine m^* and for which $\sigma_j = \sigma^*$, generate a new node which corresponds to the partial schedule in which operation j is added to PS_t and started at time σ_j.

We now give the steps of our beam search based algorithm.

Beam Search:

Step 0 (*Node generation*). Generate nodes from the parent node by using the procedure ACTIVE (or NONDELAY) with PS_t as the null partial schedule.

Step 1 (*Checking the number of nodes*). If the total number of nodes generated is less than beamwidth, then move down to one more level, generate new nodes by using the procedure ACTIVE (or NONDELAY) with PS_t as the partial schedule represented by the node, and go to Step 1. Else, go to Step 2.

Step 2 (*Computing global evaluation functions*). Compute the global evaluation function values for all the nodes and select the best beamwidth (b) number of nodes (initial beam nodes).

For each initial beam node:

Step 3 (*Determining beam nodes*). While the number of levels is less than the number of operations (denoted as n).

Step 3.1 (*Node generation*). In the next level, generate new nodes from the beam node according

to the procedure ACTIVE (or NONDELAY) with PS_t as the partial schedule represented by the beam node. Let k is the number of nodes generated.

Step 3.2 (*Computing local evaluation functions*). Compute local evaluation function values for each node.

Step 3.3 (*Filtering*). Choose the best f' number of nodes according to local evaluation function values (f is the filterwidth and $f' = \min(k, f)$).

Step 3.4 (*Computing global evaluation functions*). Compute global evaluation values of each f' number of selected nodes.

Step 3.5 (*Selecting the beam nodes*). Select the node with the lowest global evaluation value (i.e., beam node). For the partial schedule represented by the beam node update the data set as follows:

(a) Remove operation j from S_t

(b) Form S_{t+1} by adding the direct successor of operation j to S_t

(c) Increment t by one

Step 4 (*Selecting the solution schedule*). Among the beamwidth number of schedules, select the one with the best objective function value.

As given Sabuncuoglu and Karabuk (1998), the complexity of the beam search is $O(n^3)$, where n is the number of operations to be scheduled in the job shop problem. Next we give an illustrative example that shows the basic steps of the algorithm.

Numeric example: Consider the following job shop system with two machines and four jobs and each job has two operations. The processing time and routing information of the jobs is given in Table 1 below. The performance measure is makespan. Suppose that both the beamwidth and the filterwidth are equal to 2. Nondelay branching scheme is used to generate a search tree. The "most work remaining" (MWR) rule is used as the local evaluation function and the global evaluation function is represented by the MWR dispatching rule.

The beam search tree of the problem is shown in Fig. 2 in which GF and LF refer to global and local evaluation function values, respectively. The shaded nodes are the beam nodes and the nodes with crosses are the ones that are pruned off as a result of the global or local evaluation.

In the beam search algorithm the first stage is to determine the initial beam nodes. We determine three nodes (i.e., J1-01, J2-02 and J4-01) by using the nondelay branching scheme. The node J1-01 corresponds to the first operation of the first job. Since the number of nodes generated (3 in this case) are greater than the beam size of 2, we perform global evaluation and select 2 of them. For this step of the beam search algorithm we calculated global evaluation function values (makespan measure of the complete schedule generated by the MWR rule from the partial schedule represented by the node) and selected the best two nodes on the basis of their global evaluation values. These nodes are J1-01 and J4-01. If the number of nodes were smaller than the beam size in this first level, we would continue to expand until the number of nodes are greater than the beam size at the next level.

After determining the initial beam nodes, we apply the beam search algorithm for each of the beam nodes independently. At each level, we generate the new nodes, perform filtering, compute the global evaluation function values for the filtered nodes and finally select the node which corresponds to the operation added to the current partial schedule. At the end of the search tree we obtain the Gantt chart of the resulting schedule displayed in Fig. 3(b). Note that at the end of the algorithm, we have two schedules. In the last step of the beam search algorithm we compute the makespan of the schedules and select the one with the minimum value. It turns out that, in Fig. 2, the beam on the left-hand side gives the better schedule.

Table 1
Job information

	Operation 1		Operation 2	
Job	Processing time	Machine	Processing time	Machine
1	13	1	53	2
2	54	1	42	2
3	1	2	9	1
4	78	1	50	2

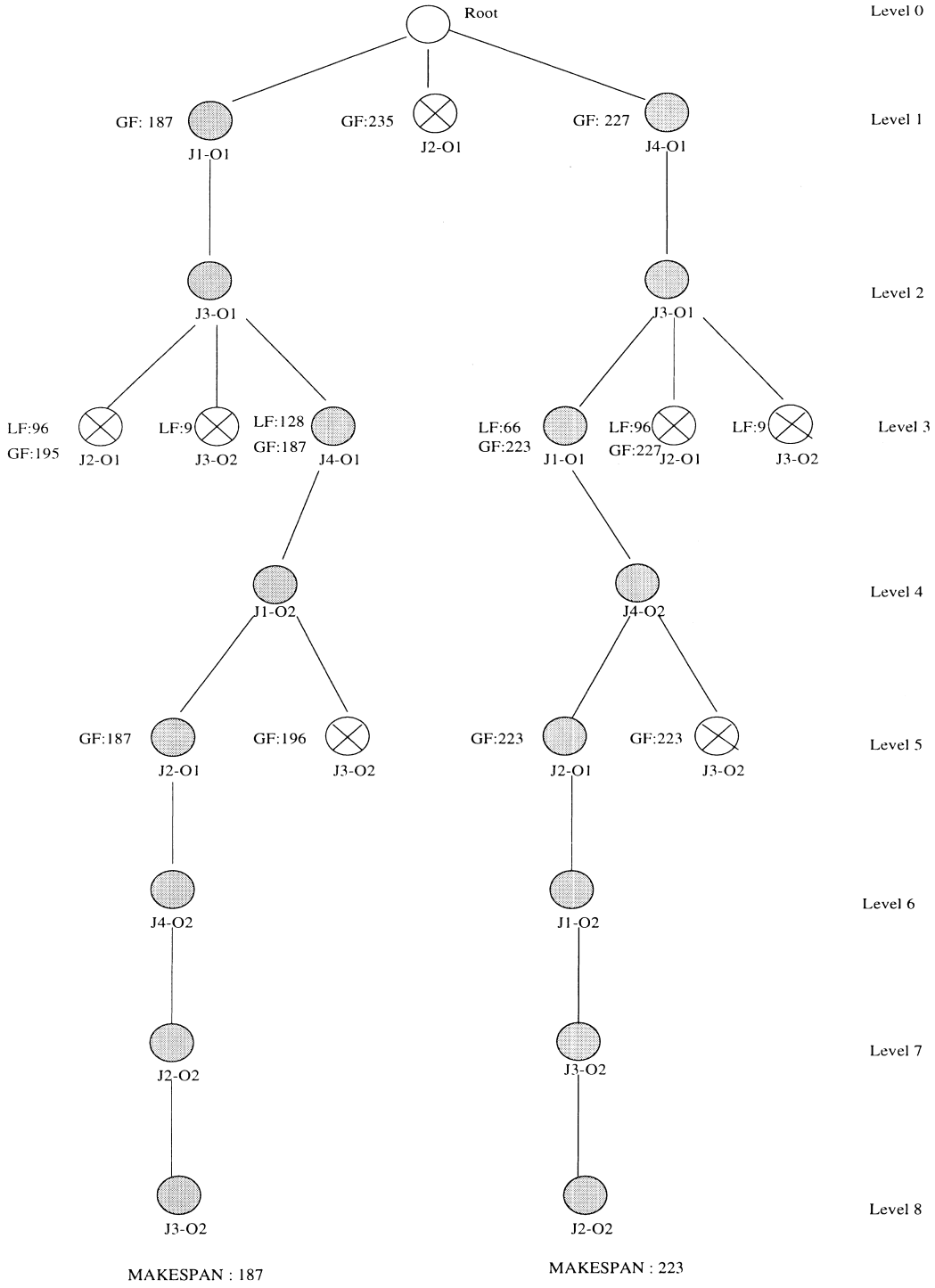
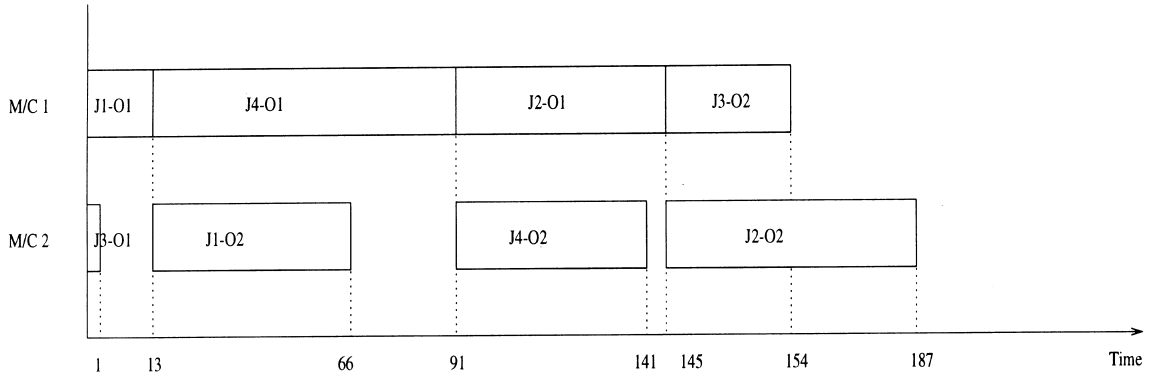
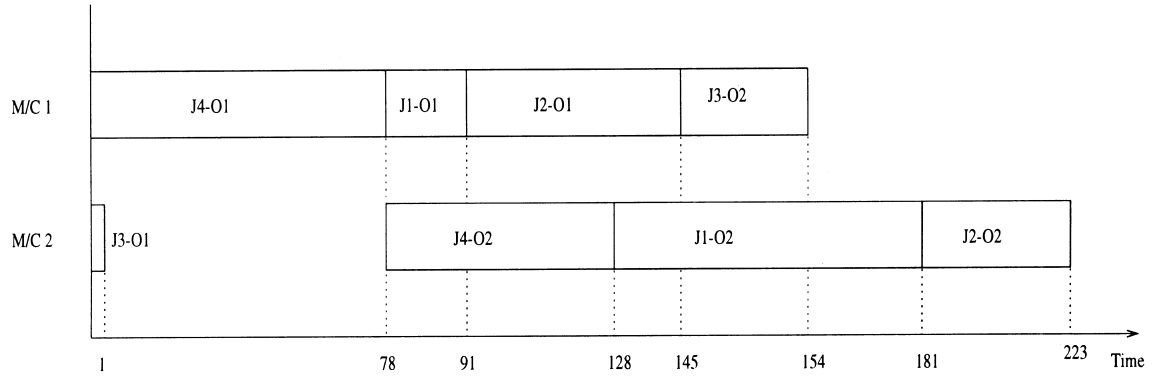


Fig. 2. Beam search tree for the numerical example.



(a) Schedule generated on the left beam



(b) Schedule generated on the right beam

Fig. 3. Schedules generated by the beam search algorithm.

4. Makespan case

The performance of the scheduling algorithm is first measured for the makespan criterion. Effects of different local and global functions and various beam and filter width levels are also evaluated in the experiments. Since the optimal results of some the test problems are known in the literature for the makespan criterion, the performance of the algorithm is tested in terms of the percent deviation from optimality. Some of these problems are given in Applegate and Cook (1990). These problems are generated according to format described in the previous section. Each of the test problems used in this study have 10 machines and 10 jobs. The problems ABZ5 and ABZ6 are from Adams et al. (1988); the problems LA16 through LA20 are from Lawrence (1984); the problems ORB3, ORB4 and ORB5 are given in Applegate and Cook (1990).

Since the proposed algorithm is a heuristic, it is also compared with well-known dispatching rules, such as MWR, MTWR and LPT. These rules are implemented via the nondelay scheduling scheme proposed by Baker (1984).

Based on pilot runs, MWR is also used as the nondelay dispatching rule in the global evaluation function to complete the partial schedule from the nodes. For the local evaluation function, however, both MWR and a lower bound proposed by Baker (1974) are considered in the algorithm. The complete definition of these rules and the lower bound are given in Table 2 where S_t is the set of unscheduled operations at level t; σ_j is the earliest

Table 2
Descriptions of priority rules used in makespan analysis

Rule	Description
MWR (Most Work Remaining)	$\mathrm{MWR}_{ij} = \sum_{q=j}^{m_i} p_{iq}$
MTWR (Most Total Work)	$\mathrm{MWR}_{ij} = \sum_{q=1}^{m_i} p_{iq}$
LPT (Longest Processing Time)	$\mathrm{LPT}_{ij} = p_{ij}$
LB (Lower Bound)	

$\mathrm{LB}_{ij} = \max(\max_{j \in S_i}(\sigma_j + R_j), \max_{1 \leqslant k \leqslant m}(f_k + M_k))$.

time at which operation j could be started; R_j is the unscheduled processing for the job corresponding to operation j; f_k is the latest completion of an operation on machine k; M_k is the unscheduled processing that will require machine k.

In addition, two different schedule generation (or search tree representation) schemes are tested in this study. As discussed in Baker (1974), the search trees are expanded by either the active or nondelay schedule generation procedures. In an active schedule, no operation is started earlier without delaying some other operation whereas in a nondelay schedule, no machine remains idle when there exists a schedulable operation. A combination of two evaluation functions and two schedule generation schemes results in four versions of the proposed beam search algorithm (see Table 3).

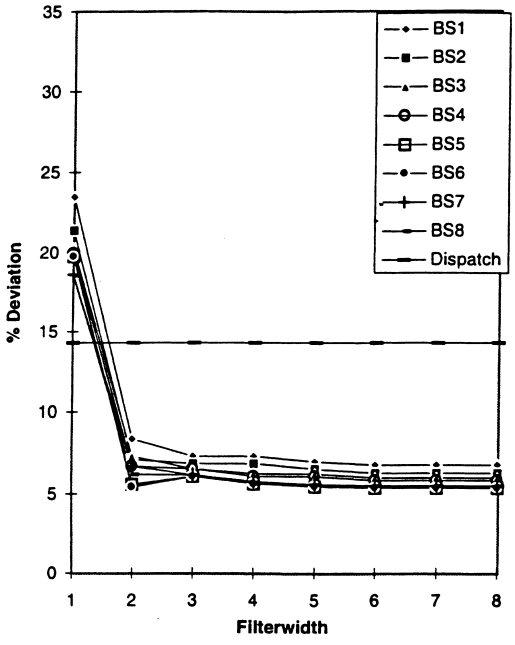
The results of the experiments are depicted in Fig. 4 for different beam and filter width values. The vertical axis shows the average (over 10 problems) deviation from optimality. Each curve represents the results of a particular beam width (i.e., BS1, BS2, etc). The horizontal axis measures the filter width. The horizontal lines in the graphs are the performances of the nondelay MWR dispatching heuristic.

In general, the performance of the algorithm changes for different search tree representation schemes, local evaluation functions and beam and filter width values (Fig. 4). Therefore, each of these factors is tested separately in order to use the best version of the filtered beam search method in the later stages of the research. In the L-shaped graphs, one can observe some erratic behaviors. These behaviors are mainly due to the imperfectness of the local and global evaluation functions. In general, local evaluations are cheaper but also less accurate. Thus, too small a filter width makes it more likely that errors in the local estimate prevents good nodes from being passed to global evaluation. Therefore, if we increase the filter width, more nodes enter the second stage, and nodes erroneously valued by the local estimate will have a second chance. If the global estimate happens to be more accurate, then the node will be saved as appropriate in the second stage. However, both estimates could be imperfect. Hence, the larger filter width in this case forces a poor node, according to the local evaluation function, to pass to the second stage of the evaluation. The global method may then erroneously save these bad nodes. When this happens, the performance of the search may deteriorate as the filter width increases while keeping the beam width constant. Since either estimate is not very accurate the above behavior is observed.

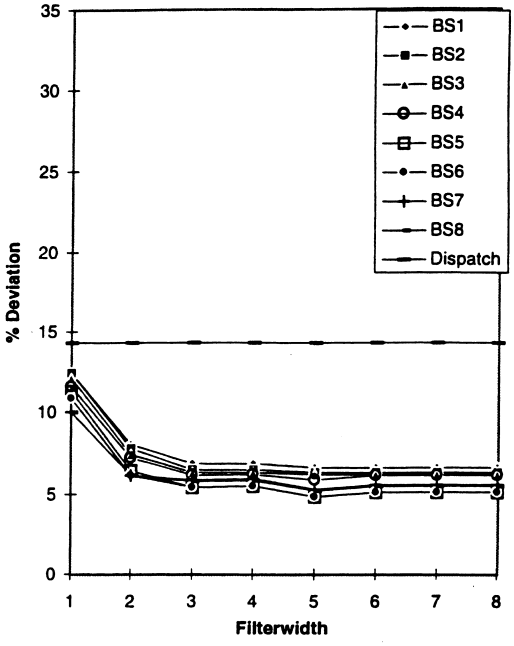
We also note that, if there are fewer nodes expanded than the size of the beam width at the level 1, all the nodes are further expanded in the next level until the number of nodes are greater than the beam width. Then all the nodes are globally evaluated to select beam nodes. If we increase the beam width, we may have more such nodes to evaluate by the global function. Similarly, if the global estimate is imperfect, global function may then mistakenly select inferior nodes as the beam nodes. In this case, the performance of the search

Table 3
Searching methods used in the makespan Case

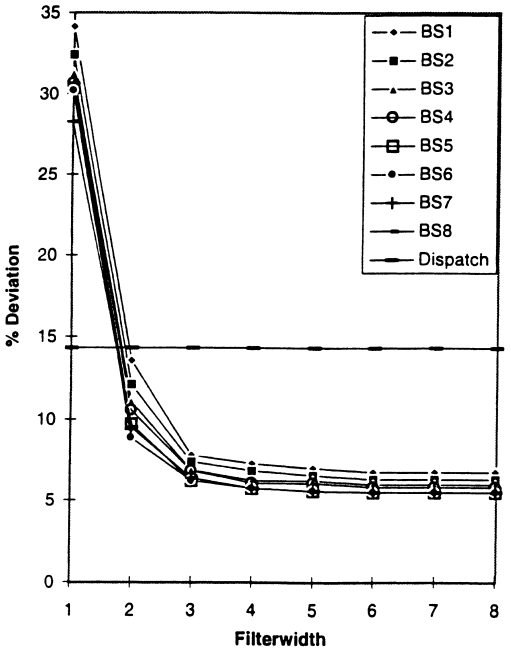
Search tree representation	Local evaluation rule	Global evaluation function
Active	MWR	Nondelay schedule generation using MWR
Nondelay	MWR	Nondelay schedule generation using MWR
Active	LB	Nondelay schedule generation using MWR
Nondelay	LB	Nondelay schedule generation using MWR



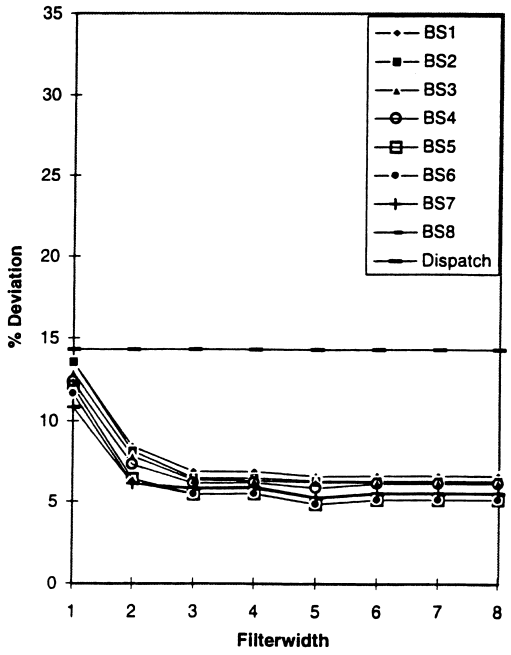
(a) Active branching, Local evaluation:MWR, Global evaluation: Nondelay dispatch (MWR)



(b) Nondelay branching, Local evaluation:MWR Global evaluation: Nondelay dispatch (MWR)



(c) Active branching, Local evaluation:LB, Global evaluation: Nondelay dispatch (MWR)



(d) Nondelay branching, Local evaluation:LB, Global evaluation: Nondelay dispatch (MWR)

Fig. 4. Percent deviation from optimal solution vs filterwidth.

heuristics may deteriorate as the beam width increases if the filter width is kept constant. We observe such a behavior in the nondelay schedule generation scheme. But the amount of deterioration on the schedule is negligible.

Active vs. nondelay schedules: In the analysis, both active and nondelay branching methods are used to generate search trees. Our computational experiments indicate that overall performance of the nondelay branching scheme is better than the active branching approach for small filter widths. However, as the filter width increases beyond a certain limit, the active scheme starts performing slightly better than the nondelay scheme. We also observe that the best result found by the active scheme is worse than the one found by the nondelay scheme. Even though this finding may seem to be counter intuitive, it is consistent with the generally held view that the nondelay scheduling scheme produces better results than the active scheduling scheme when used with dispatching rules for the makespan criterion (Baker, 1974).

Local evaluation functions: We measure the performances of two local evaluation functions: the lower bound discussed in Baker (1974) and the MWR rule. A lower bound value of a partial schedule is computed as the maximum of a job based bound value and machine based bound value. The job based bound value is equal to MWR plus earliest possible start time (Table 2). This is generally greater than a machine based value. Since evaluation function based on the lower bound resembles the MWR rule, and the results of MWR and lower bound are not expected to differ too much. However, as depicted in Fig. 4, the MWR local function yields better results than the lower bound estimate. The details of these results are given in Appendix A, Table 10.

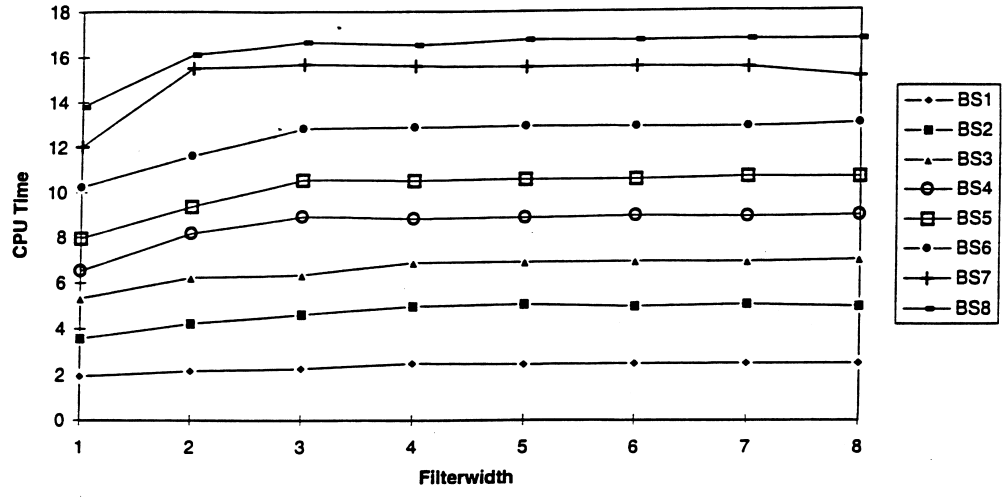
Filter and beam width: In the classical job shop problem, there is not as much flexibility as in the case of FMS. Hence, relatively fewer number of nodes are expanded in the search tree. If the filter width is set very high, the algorithm locally evaluates all the nodes expanded from a beam node. For instance if there are three nodes expanded from a beam node, it does not matter whether filter width is greater than three. Consequently, the performance of the algorithm does not change significantly as the filter width increases beyond a certain limit. From our experiments, it appears that after filter width of five, performance of the algorithm almost remains the same. Thus, we decide to set the filter width value to five for the later stages of the algorithm.

According to experiments, increasing the beam width improves the solution quality. This result is consistent with our expectations. Recall that there are beam width number of parallel beams in the search tree. Thus, if we perform an algorithm through the higher number of beams, we have the larger pool of nodes to evaluate and consequently have better solution possibilities. However in order to have a real-time scheduling capability in the algorithm, the computational aspect should also be taken into account. As seen from Fig. 5, the higher the beam width, the higher the CPU time needed to execute the algorithm. In the experiments, it is observed that the relative improvement on solution quality gets smaller when the values of the beam width parameter increases. It appears that the best beam width is five when considering both the CPU times and the solution quality. Thus, this setting is used in the later applications of the algorithm.

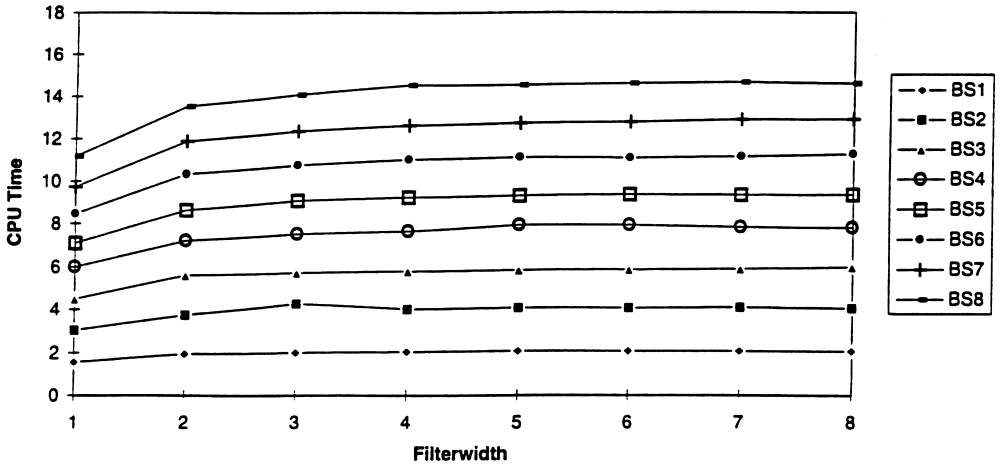
4.1. Computational results

In order to measure the makespan performance of beam search algorithm in the job shop environment, we used well-known benchmark problems reported in the literature. These are: 40 problems generated by Lawrence (1984), 2 problems used by Adams et al. (1988), and 5 problems mentioned in Applegate and Cook (1990). The famous FT10 problem is also included in the experiments. Both the proposed algorithm and the rules are run on Sparc Station with one 60 MHz micro SPARC 8-CPU and 1 GB memory. The codes are written in the C language.

The computational results are given in Table 4. Each cell in that table represents the average deviation from optimality and the computation times in CPU seconds for the corresponding problem instance. Since solution times of the dispatching



(a) CPU Time vs filterwidth for the makespan objective



(b) CPU Time vs filterwidth for the tardiness objective

Fig. 5. CPU time vs. filterwidth.

rules are very small (e.g. less than 10^{-2} s), the CPU times of the rules are no included in this table.

As expected, the performance of the algorithm in terms of the average percent deviation from optimality is much better than the rules. The average deviation of the algorithm is 4.26% for all the test problems, while it is 16%, 29%, and 13% for SPT, LPT, MWR, respectively. These rules solve only three problem instances to optimality, but the proposed algorithm solves 16 out of 48 instances, including the most difficult problems FT10, LA36,...,LA40. For the instances which we do not know optimal solutions, the percent deviations from optimality cannot be calculated and thus these cell are filled by '*'.

Even though beam search is a branch and bound based algorithm, the CPU times is not very high. It appears that the number of jobs (rather

Table 4
Results of test problems for the makespan analysis

Problem	Optimum	Algorithm $b=5, f=5$			SPT		LPT		MWR	
		Solution	% Dev.	CPU	Solution	% Dev.	Solution	% Dev.	Solution	% Dev.
10*5										
LA01	666	666	0	2.5	751	12.8	933	40.01	735	10.4
LA02	655	704	7.84	2.9	821	25.3	830	26.7	817	24.7
LA03	597	650	8.88	3	672	12.6	822	37.7	696	16.6
LA04	590	620	5.09	2.8	711	20.5	833	41.2	758	28.5
LA05	593	593	0	3.2	610	2.9	766	29.2	593	0
Averages			4.29			14.82		34.98		16.04
15*5										
LA06	926	926	0	13.6	1200	29.6	1067	15.2	926	0
LA07	890	890	0	12.2	1034	16.2	1136	27.6	970	9
LA08	836	863	0	14.7	942	9.2	1176	36.3	957	10.9
LA09	951	951	0	12.1	1045	9.9	1334	40.3	1015	6.7
LA10	958	958	0	14.2	1049	9.5	1312	37	966	0.8
Averages			0			14.88		31.28		5.48
20*5										
LA11	1222	1222	0	45.4	1473	20.5	1525	24.8	1268	3.8
LA12	1039	1039	0	39.8	1203	15.8	1305	25.6	1137	9.4
LA13	1150	1150	0	44.9	1275	10.9	1354	17.7	1166	1.4
LA14	1292	1292	0	43.3	1427	10.4	1725	33.5	1292	0
LA15	1207	1207	0	41.6	1339	10.9	1648	36.5	1343	11.3
Averages			0			13.7		27.62		5.18
10*10										
LA16	945	988	4.55	10.7	1156	22.3	1347	42.5	1054	11.5
LA17	784	827	5.49	9.6	924	17.9	1203	53.4	846	7.9
LA18	848	881	3.89	10.2	981	15.7	1154	36.1	970	14.4
LA19	842	882	4.75	8	940	11.6	986	17.1	1013	20.3
LA20	902	948	5.1	8.8	1000	10.9	1232	36.6	964	6.9
FT10	930	1016	9.2	74	1074	15.5	1324	42.4	1108	19.1
ABZ5	1234	1288	4.4	10.8	1352	9.6	1735	40.6	1369	10.9
ABZ6	943	980	3.9	14.6	1097	16.3	1110	17	987	4.7
ORB1	1059	1174	10.85	14.5	1478	39.6	1398	32	1359	28.3
ORB2	888	926	4.27	10.9	1175	32.3	1170	31.8	1047	17.9
ORB3	1005	1087	7.54	12.9	1179	17.3	1389	38.2	1247	24
ORB4	1005	1036	3.09	11.9	1236	23	1432	42.5	1172	16.6
ORB5	887	968	9.13	11.2	1152	29.9	1175	32.5	1173	32.2
Averages			5.86			20.14		35.58		16.53
15*10										
LA21	1040–1053	1154	*	44	1324	*	1518	*	1264	*
LA22	927	985	6.26	44.3	1180	27.3	1589	71.4	1079	16.4
LA23	1032	1051	1.84	39.8	1162	12.6	1347	33.1	1185	14.8
LA24	935	992	6.1	39.3	1203	28.7	1214	29.8	1101	17.8
LA25	977	1073	9.83	43.1	1449	48.3	1487	52.2	1166	19.3
Averages			6.01			29.23		46.63		17.08
20*10										
LA26	1218	1269	4.19	136.1	1498	23	1606	31.9	1435	17.8
LA27	1235–1269	1316	*	129.8	1784	*	1728	*	1442	*
LA28	1216	1373	12.91	137.2	1610	32.4	1750	43.9	1487	22.3

Table 4 (Continued)

Problem	Optimum	Algorithm $b=5, f=5$			SPT		LPT		MWR	
		Solution	% Dev.	CPU	Solution	% Dev.	Solution	% Dev.	Solution	% Dev.
LA29	1120–1195	1252	*	140.7	1556	*	1665	*	1337	*
LA30	1355	1435	5.9	144.6	1792	32.3	2067	52.5	1534	13.2
Averages			7.67			29.23		42.77		17.77
30∗10										
LA31	1784	1784	0	810.3	1951	9.4	2322	30.2	1931	8.2
LA32	1850	1850	0	806	2165	17	2341	26.5	1875	1.4
LA33	1719	1719	0	818.9	1901	10.6	2125	23.6	1875	9.1
LA34	1721	1780	3.42	823.5	2070	20.3	2223	29.2	1935	12.4
LA35	1888	1888	0	684.2	2118	12.2	2316	22.7	2118	12.2
Averages			0.68			13.89		26.43		8.66
15∗15										
LA36	1268	1401	10.47	98.7	1681	32.6	1908	50.5	1521	20
LA37	1397	1503	7.59	99.2	1693	21.2	1884	34.9	1643	17.6
LA38	1171–1184	1297	*	93.7	1509	*	1686	*	1477	*
LA39	1233	1369	11.03	95.8	1447	17.4	1894	53.6	1443	17
LA40	1222	1347	10.23	100	1495	22.3	1661	35.9	1475	20.7
Averages			9.83			23.36		43.72		18.82
Average of means			4.26		15.99		28.62		12.16	

Table 5
Comparison of scheduling algorithms

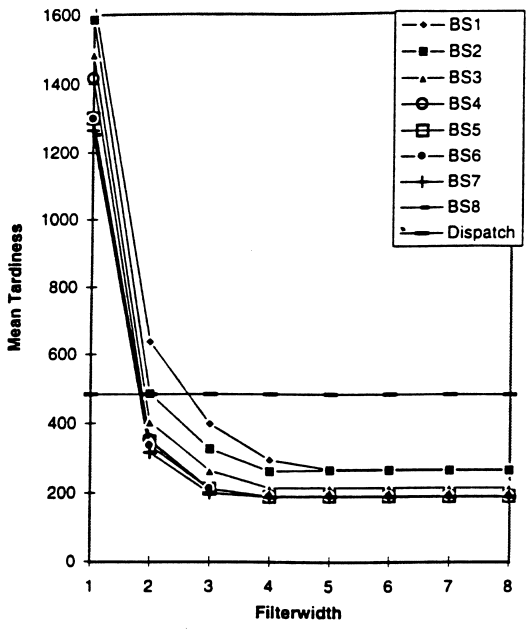
	MSII1	MSII2	TA1	SA1	SA2	GLS1	GLS2	BS
Avr. % dev.	11.73	8.96	3.73	1.52	1.51	2.22	1.92	4.47
Max. % dev.	32.61	27.36	12.98	10.13	9.45	15.20	12.50	11.79
# of optimum	4	7	15	19	18	16	16	16

Table 6
Description of priority used in tardiness analysis

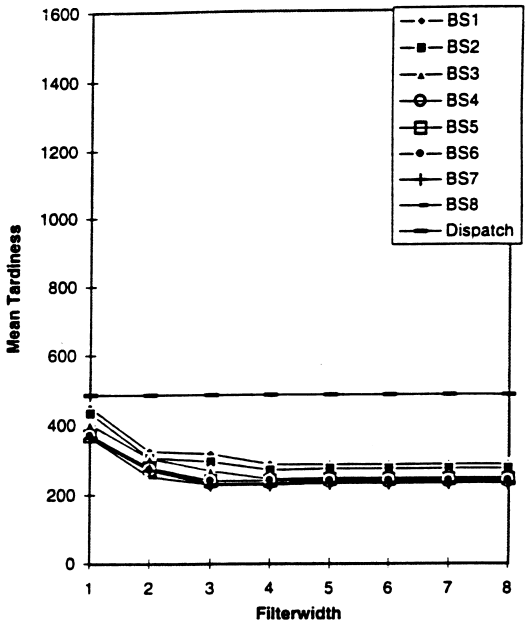
Rule	Description
SPT (Shortest Processing Time)	$\mathrm{SPT}_{ij} = p_{ij}$
LWR (Least Work Remaining)	$\mathrm{LWR}_{ij} = \sum_{q=j}^{m_i} p_{iq}$
EDD (Earliest Due Date)	$\mathrm{EDD}_{ij} = \mathrm{duedate}_i$
MOD (Modified Operational Due Date)	$\mathrm{MODD}_{ij} = (\mathrm{duedate}_i / \sum_{q=1}^{m_i} p_{iq}) \sum_{q=1}^{j} p_{iq}$
MD (Modified Due Date)	$\mathrm{MDD}_{ij} = \max(\mathrm{duedate}_i, t + \sum_{q=j}^{m_i} p_{ij})$

Table 7
Search methods used in the first part of the experiments

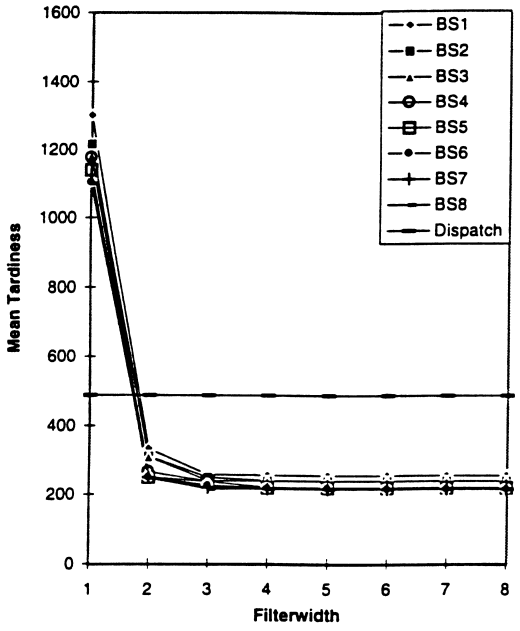
Search tree representation	Local evaluation rule	Global evaluation function
Active	SPT	Nondelay SPT dispatch heuristic
Nondelay	SPT	Nondelay SPT dispatch heuristic
Active	EDD	Nondelay EDD dispatch heuristic
Nondelay	EDD	Nondelay EDD dispatch heuristic
Active	MDD	Nondelay MDD dispatch heuristic



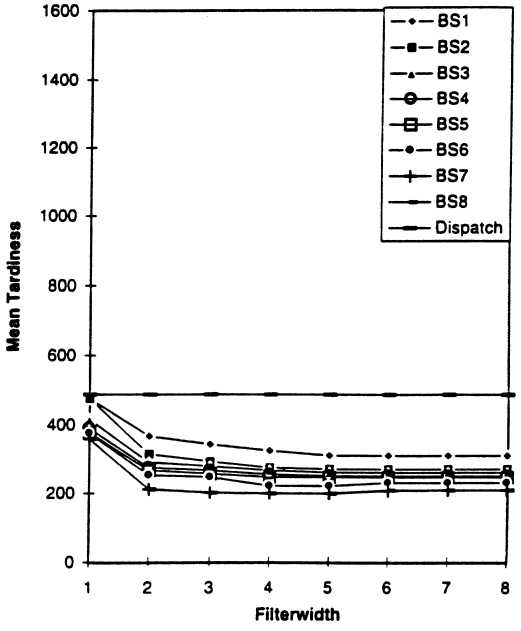
(a) Active branching, Local evaluation:SPT, Global evaluation: Nondelay dispatch (SPT)



(b) Nondelay branching, Local evaluation:SPT, Global evaluation: Nondelay dispatch (SPT)



(c) Active branching, Local evaluation:EDD, Global evaluation: Nondelay dispatch (EDD)



(d) Nondelay branching, Local evaluation:EDD, Global evaluation: Nondelay dispatch (EDD)

Fig. 6. Mean tardiness vs. filterwidth analysis.

Table 8
Search methods used in the second part of the experiments

Search tree representation	Local evaluation rule	Global evaluation function
Active	MDD	Nondelay SPT dispatch heuristic
Active	EDD	Nondelay SPT dispatch heuristic
Active	MODD	Nondelay SPT dispatch heuristic

than machines) is the most determining factor for its computational time requirement.

We also note that the performance of the algorithm changes for different problem types. It seems that the algorithm performs very well if the number of jobs is greater than the number of machines which we call *rectangular instances*. It also appears that the *square type instances* (number of machines equals to number of jobs) are hard instances for the beam search algorithm.

We also compare the beam search algorithm with other heuristic methods whose performances are reported for some selected problems. In a recent study, Aarts et al. (1994) propose multi-start iterative improvement (MSII), threshold accepting (TA), simulated annealing (SA) and genetic local search (GLS) algorithms for the job shop problems. The authors use two neighborhood structures in their algorithms and apply them to 43 problem instances, among which 40 of them are also common in our test set. The performances of their algorithms and our beam search based algorithm (referred to as BS) are given in Table 5.

In terms of average % deviation from optimality, the solution quality of BS is better than the multi-start iterative improvement methods (MSII1, MSII2) and close to threshold accepting method (TA1). In terms of the maximum % deviation and number of optimally solved instances, the performance of the beam search based algorithm is still better than MSII1, MSII2 and TA1 and competes with the well-known searching schemes (simulating annealing and genetic local search methods).

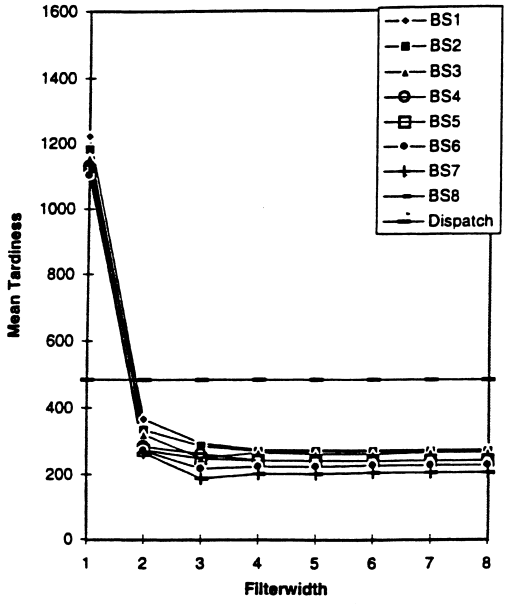
Bear in mind that BS is a constructive algorithm whereas others are iterative procedures whose computation times can be indeed very long. In Aarts et al. (1994) the average of running times reported for each problem are much larger than the CPU time requirements of BS. Even for small-sized problems, the differences between CPU times are significant, (i.e., the beam search method is approximately 10 and 6 times faster than the other methods for 10 jobs 10 machines and 15 jobs 15 machines problems, respectively).

5. Mean tardiness case

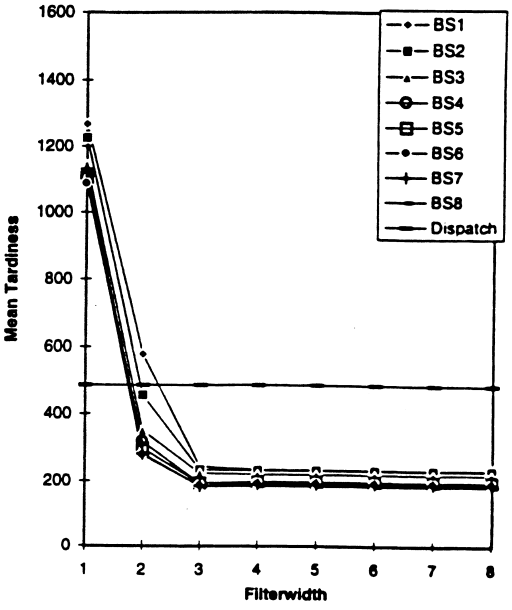
The performance of the beam search based algorithm is also measured in terms of the mean tardiness. Again, we first examine evaluation functions and find proper settings of the beam search parameters. Since optimum solutions of the job shop problems are not generally known in the tardiness case, we only compare the performance of the algorithm with dispatching rules.

During computational experiments, we use the modified version of the problem data used in the makespan case. Specifically, we append the due-date information to the data sets using the TWK due date assignment method (Baker, 1984). Based on pilot runs, the proportionality constant (tardiness factor) of TWK is set to 1.5 for the tight due date case (corresponds to 50% tardy jobs) and 2 for the loose due date case (corresponds to 6% tardy jobs). These two tardiness levels are very close to each other because the due dates are assigned in proportion to total processing time, instead of average processing time. We also know that in low utilization rates (such as the case in our model with 62% utilization), the closeness of tardiness factors is expected as indicated by Baker (1984, pp. 1099). In this analysis we use the tardiness factor of 1.5 to determine the due dates of the jobs.

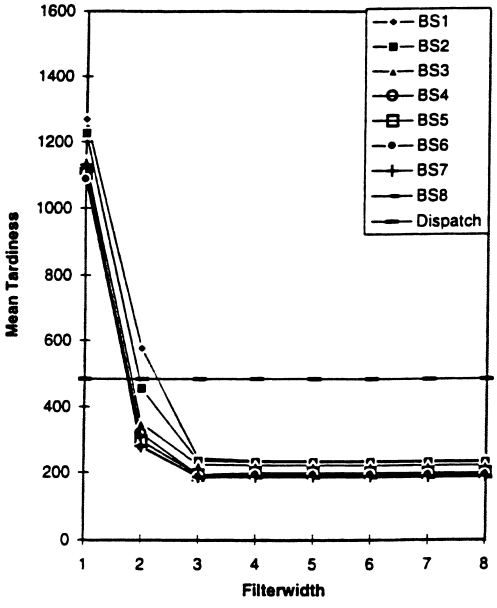
After determining the due date settings for the jobs, the performances of non-delay dispatching



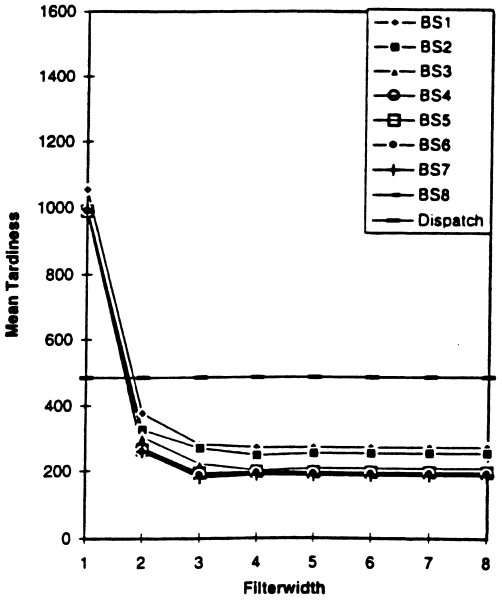
(a) Active branching, Local evaluation:MDD, Global evaluation: Nondelay dispatch (MDD)



(b) Nondelay branching, Local evaluation:MDD, Global evaluation: Nondelay dispatch (SPT)



(c) Active branching, Local evaluation:EDD, Global evaluation: Nondelay dispatch (SPT)



(d) Nondelay branching, Local evaluation:MOD, Global evaluation: Nondelay dispatch (SPT)

Fig. 7. Mean tardiness vs. filterwidth analysis.

Table 9
Results of test problems for the mean tardiness analysis

	Algor. $b=5, f=5$		EDD	LWR	MDD	MODD	SPT	EDD	LWR	MDD	MODD	SPT
Prob.	Sol.	CPU	Sol.	Sol.	Sol.	Sol.	Sol.	% Dev.	% Dev.	% Dev.	% Dev.	% Dev.
10∗5												
LA01	97.0	3.4	118.2	118.2	118.2	147.9	135.6	21.8	21.8	21.8	52.4	39.7
LA02	62.5	3.9	103.9	105.8	107.5	132.4	89.0	66.2	69.2	72.0	111.8	42.4
LA03	70.0	3.4	98.7	97.0	97.0	132.4	85.9	41.0	38.5	38.5	89.1	22.7
LA04	86.8	8.7	133.3	133.8	135.8	151.4	130.8	53.5	54.1	56.4	74.4	50.7
LA05	95.8	3.6	106.9	112.2	106.1	144.5	118.0	11.60	17.1	10.7	50.8	23.1
Averages								38.9	40.2	39.9	75.7	35.7
15∗5												
LA06	207.8	12.9	233.7	227.2	217.9	350.5	255.6	12.4	9.3	4.8	68.6	23.0
LA07	195.6	15.7	238.9	241.0	238.5	297.6	214.5	22.1	23.2	21.9	52.1	9.6
LA08	197.3	13.8	220.3	240.3	240.3	310.8	242.9	11.6	21.7	21.7	57.5	23.1
LA09	227.0	14.5	284.0	265.0	232.2	341.3	284.6	25.1	16.7	2.1	50.3	25.4
LA10	218.2	13.1	237.9	228.8	225.2	374.2	251.1	8.9	4.8	3.1	71.4	15.0
Averages								16.0	15.2	10.7	60.0	19.2
20∗5												
LA11	343.7	36.2	365.5	353.6	341.0	564.4	389.8	6.3	2.8	−0.7	64.2	13.4
LA12	302.3	38.5	305.7	286.7	295.7	466.3	334.3	1.1	5.1	−2.1	54.2	10.6
LA13	329.8	39.3	352.9	335.7	340.5	515.1	371.0	7.0	1.8	3.2	56.1	12.5
LA14	398.6	38.5	403.3	370.4	377.5	566.0	411.0	1.1	−7.1	−5.3	41.9	3.1
LA15	386.1	37.2	396.6	419.7	412.9	566.6	429.1	2.7	8.7	6.9	46.7	11.1
Averages								3.6	0.2	0.4	52.7	10.2
10∗10												
LA16	25.5	13.3	64.0	73.0	64.0	30.5	48.4	150.9	186.2	150.9	19.6	89.8
LA17	20.5	13.2	39.3	80.4	39.0	76.1	54.0	91.7	292.2	91.7	271.2	163.4
LA18	6.6	13.0	50.3	55.0	44.2	47.0	18.2	662.1	733.3	569.7	612.1	175.7
LA19	11.3	12.4	27.2	39.8	34.7	50.9	51.7	140.7	252.2	207.1	350.4	357.5
LA20	9.5	12.6	40.0	74.0	40.0	28.1	38.2	321.0	678.9	321.0	195.7	302.1
FT10	56.7	14.2	106.0	117.0	106.0	148.0	95.5	86.9	106.3	86.9	161.0	68.4
ABZ5	18.4	13.5	57.2	171.6	57.2	31.9	41.5	210.9	832.6	210.8	73.3	125.5
ABZ6	0.0	12.3	11.0	17.9	11.0	10.2	4.7	*	*	*	*	*
ORB1	113.2	15.6	194.7	157.1	205.4	199.8	241.0	72.0	38.8	81.4	76.5	112.9
ORB2	23.2	13.7	76.0	61.1	80.5	77.8	50.1	227.6	163.3	246.9	235.3	115.9
ORB3	105.8	15.5	135.1	156.3	136.2	298.9	184.6	27.7	47.7	28.7	182.5	74.5
ORB4	39.6	14.9	74.2	175.9	91.7	155.6	91.3	87.4	344.2	131.6	292.9	130.5
ORB5	40.3	13.6	78.7	97.4	78.7	96.5	87.5	95.3	141.7	95.3	139.4	117.1
Averages								181.2	318.1	185.2	217.5	152.8
15∗10												
LA21	90.8	55.4	150.0	175.1	156.7	207.5	175.7	65.0	92.6	72.4	128.3	93.3
LA22	114.5	54.7	241.5	235.2	210.7	225.6	172.3	111.0	105.5	84.1	97.1	50.5
LA23	96.2	52.7	130.3	159.9	147.9	177.4	143.1	35.4	66.0	53.6	84.3	48.7
LA24	95.6	53.9	129.8	131.1	121.7	173.0	131.0	35.8	37.2	27.3	80.9	37.0
LA25	106.7	53.6	176.9	151.7	163.2	187.5	162.9	65.8	42.1	52.9	75.6	52.6
Averages								62.6	68.7	58.0	93.3	56.4
20∗10												
LA26	225.6	156.0	345.0	241.1	288.3	344.4	250.1	52.9	6.9	27.8	52.6	10.8
LA27	226.3	148.4	303.8	292.4	286.1	358.0	271.3	34.3	29.2	26.4	58.2	18.9
LA28	212.4	155.9	301.4	330.8	227.3	410.2	342.3	41.9	55.7	30.5	93.1	61.1
LA29	216.1	150.7	270.2	276.2	266.9	443.4	269.9	25.0	27.8	23.5	105.1	24.9
LA30	233.8	155.2	396.2	372.9	382.9	425.4	337.6	69.4	59.5	63.7	81.9	44.4
Averages								44.7	35.8	34.4	78.2	32.2

Table 9 (Continued)

	Algor. $b=5, f=5$		EDD	LWR	MDD	MODD	SPT	EDD	LWR	MDD	MODD	SPT
Prob.	Sol.	CPU	Sol.	Sol.	Sol.	Sol.	Sol.	% Dev.	% Dev.	% Dev.	% Dev.	% Dev.
30*10												
LA31	348.3	915.4	566.6	566.4	564.2	783.3	625.9	62.7	62.6	61.9	124.8	79.7
LA32	364.8	942.3	565.8	563.8	571.4	813.5	652.9	55.1	54.5	56.6	123.0	78.9
LA33	378.1	940.1	565.4	501.3	584.9	755.2	540.0	49.5	32.6	54.7	99.7	42.8
LA34	333.3	900.7	582.4	549.4	537.2	776.4	591.9	74.7	64.8	61.1	132.9	77.5
LA35	335.3	866.5	605.6	567.1	587.2	815.1	603.4	80.6	69.1	75.1	143.0	79.9
Averages								64.5	56.7	61.9	124.7	71.8
15*15												
LA36	33.53	123.03	113.93	115.20	113.9	142.2	140.7	239.7	243.5	239.7	324.1	319.7
LA37	26.8	125.4	48.0	109.4	54.1	172.1	118.6	79.3	308.2	101.9	542.2	342.8
LA38	27.1	126.3	128.6	136.6	130.0	87.4	53.1	374.2	403.5	379.1	222.1	95.8
LA39	24.5	121.9	87.3	114.3	87.3	93.3	76.1	256.0	366.0	256.0	280.4	210.3
LA40	43.6	125.5	92.1	123.2	106.4	126.2	88.9	110.9	182.1	143.8	188.9	103.6
Averages								212.0	300.7	224.1	311.6	214.7

Sol.: Solution; % Dev: Percent deviation from algorithm's solution.

rules are measured in 10 test problems to find the appropriate local and global functions for the beam search algorithm. Five rules (SPT, EDD, LWR, MDD and MODD) are used in the experiments (see Table 6 for the complete description). Results indicate that the solution quality of EDD, MDD and SPT are comparable with each other and they are all better than the other rules with the average tardiness values of 517.9, 540.3 and 485.6, respectively. Similar to the makespan case, we determine most suitable branching scheme, evaluation functions and search parameters. Computational experiments are carried out in two stages: we first investigate the branching scheme and then local evaluation function and search parameters. Descriptions of these search methods are given in Table 7.

Active vs. nondelay schedules: The results indicate that the performance of nondelay schedules is better than the active schedules for only small filter widths (Fig. 6). However, after the filter width value of 3, the performance of the active generation method becomes better than the nondelay method (see Appendix A, Table 11 for the details of the experiments). This behavior is observed due to the fact that the number of active schedules generated by the beam search algorithm is greater than the number of nondelay schedules. Hence, there is more chance to obtain better results by searching active schedules. For that reason, in contrast to the makespan measure, the active scheduling scheme is used in this case.

Local and global evaluation functions: In the experiments, as a part of the active schedule generation scheme, MODD, SPT and EDD rules are used in both local and global evaluation functions. Results shows that SPT is better than the other rules. This result is consistent with results of the earlier studies reported by Kiran and Smith (1984) that SPT is one of the best priority rules in terms of all due-date related measures.

However, due-date based rules were expected to perform well for the mean tardiness criterion. Hence, the second set of experiments is carried out with EDD, MDD and MODD as the local evaluation functions. Due to better performance of the beam search with evaluation functions of SPT rule in the previous analysis, global estimation is again performed by this rule (see Table 8). The experimental results indicate that the due-date based rules perform better than SPT (Fig. 7). As given in Table 12 of Appendix A, due-date based rules find promising nodes easily for small filter widths. For that reason, their performances are considerably better than SPT. However, for the large beam and filter widths, they only perform slightly better than SPT. Overall, MODD displays the best perfor-

mance, and hence is selected as the local evaluation function.

Filter and beam widths: As previously discussed in the makespan case, only a few nodes are expanded from a beam node in the job shop problem. Hence, increasing the filter width does not improve the solution quality. It seems that the appropriate value is 5 (see also Figs. 6 and 7). For the beam width parameter, the test results show that the beam width of 5 is also a proper value for the tardiness case when considering the CPU times and the quality of solutions (Fig. 5). As a result, we decide to use active schedule generation method with local evaluation function of the MODD rule and global evaluation function of the SPT rule.

5.1. Computational results

After determining the proper evaluation functions and parameter settings, the resulting beam

Table 10
Percent deviation from optimal solution vs. filterwidth

f	*b*:1	*b*:2	*b*:3	*b*:4	*b*:5	*b*:6	*b*:7	*b*:8	Dispatch
Active branching, Local: MWR, Global: MWR dispatching rule									
1	23.45	21.38	20.28	19.95	19.73	19.73	18.59	18.56	14.34
2	8.36	7.09	7.29	6.68	5.58	5.41	6.67	6.20	14.34
3	7.29	6.84	6.52	6.51	6.05	6.05	6.16	6.16	14.34
4	7.28	6.83	6.21	6.07	5.61	5.61	5.72	5.62	14.34
5	6.97	6.53	6.21	6.07	5.47	5.47	5.57	5.57	14.34
6	6.74	6.29	5.97	5.83	5.37	5.37	5.48	5.48	14.34
7	6.74	6.29	5.97	5.83	5.37	5.37	5.48	5.48	14.34
8	6.74	6.29	5.97	5.83	5.37	5.37	5.48	5.48	14.34
Nondelay branching, Local: MWR, Global: MWR dispatching rule									
1	12.47	12.47	12.12	11.59	11.32	10.85	9.99	9.99	14.34
2	8.02	7.78	7.41	7.20	6.42	6.15	6.16	6.13	14.34
3	6.86	6.46	6.32	6.15	5.43	5.43	5.88	5.78	14.34
4	6.85	6.45	6.31	6.19	5.48	5.48	5.93	5.82	14.34
5	6.61	6.30	6.21	5.87	4.86	4.86	5.31	5.22	14.34
6	6.61	6.31	6.21	6.13	5.13	5.13	5.58	5.29	14.34
7	6.61	6.31	6.21	6.13	5.13	5.13	5.58	5.29	14.34
8	6.61	6.31	6.21	6.13	5.13	5.13	5.58	5.29	14.34
Active branching, Local: LB, Global: MWR dispatching rule									
1	34.16	32.43	31.25	30.68	30.33	30.17	28.28	28.28	14.34
2	13.56	12.12	11.06	10.57	9.69	8.87	9.49	9.49	14.34
3	7.77	7.38	6.85	6.84	6.24	6.24	6.39	6.39	14.34
4	7.28	6.83	6.21	6.07	5.72	5.72	5.72	5.72	14.34
5	6.98	6.53	6.21	6.07	5.57	5.72	5.72	5.72	14.34
6	6.74	6.28	5.97	5.83	5.48	5.48	5.48	5.48	14.34
7	6.74	6.28	5.97	5.83	5.48	5.48	5.48	5.48	14.34
8	6.74	6.28	5.97	5.83	5.48	5.48	5.48	5.48	14.34
Nondelay branching, Local: LB, Global: MWR dispatching rule									
1	13.55	13.55	12.91	12.38	12.09	11.63	10.84	10.84	14.34
2	8.41	8.16	7.82	7.29	6.43	6.16	6.16	6.16	14.34
3	6.86	6.46	6.33	6.15	5.44	5.44	5.88	5.77	14.34
4	6.85	6.45	6.31	6.19	5.48	5.48	5.93	5.82	14.34
5	6.60	6.29	6.21	5.87	4.86	4.86	5.31	5.23	14.34
6	6.61	6.31	6.21	6.14	5.13	5.13	5.58	5.49	14.34
7	6.61	6.31	6.21	6.14	5.13	5.13	5.58	5.49	14.34
8	6.61	6.31	6.21	6.14	5.13	5.13	5.58	5.49	14.34

Table 11
Mean tardiness vs filterwidth analysis

f	*b*:1	*b*:2	*b*:3	*b*:4	*b*:5	*b*:6	*b*:7	*b*:8	Dispatch
Active branching, Local: SPT, Global: SPT dispatching rule									
1	1641.7	1585.7	1485.6	1415.2	1300.9	1300.6	1266.4	1254.2	485.6
2	636.1	486.7	403.7	352.2	349.4	335.7	317.1	317.1	485.6
3	398.3	326.0	264.0	212.1	212.1	212.1	201.0	201.0	485.6
4	295.1	263.2	213.9	200.2	200.2	200.2	198.5	198.5	485.6
5	268.5	265.5	216.2	202.2	202.2	202.2	200.5	200.5	485.6
6	268.5	265.5	216.2	202.2	202.2	202.2	200.5	200.5	485.6
7	268.5	265.5	216.2	202.2	202.2	202.2	200.5	200.5	485.6
8	268.5	265.5	216.2	202.2	202.2	202.2	200.5	200.5	485.6
Active branching, Local: SPT, Global: SPT dispatching rule									
1	450.7	434.3	402.9	371.8	369.4	369.4	364.4	364.4	485.6
2	323.6	307.0	303.7	274.6	277.4	271.1	270.9	252.5	485.6
3	316.9	293.5	267.1	232.9	241.0	241.0	228.3	228.3	485.6
4	287.0	270.4	244.0	232.6	240.7	240.7	228.0	228.0	485.6
5	287.0	274.7	248.3	236.9	245.0	240.7	232.3	232.3	485.6
6	287.0	274.7	248.3	236.9	245.0	240.7	232.3	232.3	485.6
7	287.0	274.7	248.3	236.9	245.0	240.7	232.3	232.3	485.6
8	287.0	274.7	248.3	236.9	245.0	240.7	232.3	232.3	485.6
Active branching, Local: EDD, Global: EDD dispatching rule									
1	1300.9	1214.5	1181.9	1175.1	1139.4	1104.4	1124.2	1100.0	485.6
2	333.5	311.2	309.9	266.3	249.3	249.3	249.3	248.9	485.6
3	258.4	250.5	240.0	237.5	237.5	226.9	220.2	216.2	485.6
4	256.8	240.3	239.6	220.6	220.6	218.3	217.0	216.2	485.6
5	256.8	240.3	239.6	220.6	220.6	218.3	217.0	216.2	485.6
6	256.8	240.3	239.6	220.6	220.6	216.6	216.6	216.6	485.6
7	256.8	240.3	239.6	220.6	220.6	216.6	216.6	216.6	485.6
8	256.8	240.3	239.6	220.6	220.6	216.6	216.6	216.6	485.6
Nondelay branching, Local: EDD, Global: EDD dispatching rule									
1	475.3	475.3	414.8	388.4	376.7	374.9	360.0	360.0	485.6
2	366.4	316.4	291.1	276.1	268.0	254.4	211.6	211.6	485.6
3	342.7	293.9	280.3	268.2	258.7	247.9	202.6	202.6	485.6
4	324.7	275.7	267.1	256.9	247.4	223.0	200.3	200.3	485.6
5	312.3	271.2	262.6	252.4	247.4	223.0	200.3	200.3	485.6
6	312.3	271.2	262.6	252.4	247.4	232.3	209.3	209.6	485.6
7	312.3	271.2	262.6	252.4	247.4	232.3	209.3	209.6	485.6
8	312.3	271.2	262.6	252.4	247.4	232.3	209.3	209.6	485.6

search algorithm is applied to the 48 test problems described earlier. The detailed results of the algorithm and five dispatching rules are given in Table 9. Since we do not know optimal solutions in the tardiness case, we only compare the algorithm with the rules. Note that the percent deviations of dispatching rules from the solution of the proposed algorithm are also reported in that table.

The results indicate that, the proposed algorithm outperforms the rules for all problem instances. Among the dispatching rules, there is not a clear winner in the experiments. Their relative performances usually vary for different problem types (i.e., square or rectangular). We also note that the mean tardiness values are quite low for the square type instances (i.e., number of jobs equals to number of machines). For that reason, these small numbers in Table 9 results in high percent deviation of the rules for these problem instances.

Table 12
Mean tardiness vs. filterwidth analysis

f	*b*:1	*b*:2	*b*:3	*b*:4	*b*:5	*b*:6	*b*:7	*b*:8	Dispatch
Active branching, Local: MDD, Global: MDD dispatching rule									
1	1221.2	1182.0	1155.0	1129.6	1117.6	1101.9	1133.8	1116.2	485.6
2	365.8	333.9	319.4	285.3	274.4	273.2	266.9	265.7	485.6
3	294.1	286.0	250.4	262.6	248.3	217.5	189.1	189.1	485.6
4	274.0	270.4	265.6	243.2	241.5	224.1	202.7	202.7	485.6
5	274.0	270.4	262.0	243.2	241.5	224.1	202.7	202.7	485.6
6	274.0	270.4	262.0	243.2	241.5	227.8	206.4	206.4	485.6
7	274.0	270.4	266.1	243.2	241.5	227.8	206.4	206.4	485.6
8	274.0	270.4	266.1	243.2	241.5	227.8	206.4	206.4	485.6
Active branching, Local: MDD, Global: SPT dispatching rule									
1	1267.3	1224.9	1141.9	1103.0	1103.0	1085.4	1129.6	1110.2	485.6
2	577.4	455.8	351.3	317.6	296.5	276.0	281.2	281.2	485.6
3	242.7	233.4	222.5	192.6	192.6	186.1	185.9	183.8	485.6
4	233.7	230.7	219.8	198.2	198.2	191.7	188.3	188.3	485.6
5	233.7	230.7	219.8	198.2	198.2	191.7	188.3	188.3	485.6
6	233.7	230.7	219.8	198.2	198.2	191.7	188.3	188.3	485.6
7	233.7	230.7	219.8	198.2	198.2	191.7	188.3	188.3	485.6
8	233.7	230.7	219.8	198.2	198.2	191.7	188.3	188.3	485.6
Active branching, Local: EDD, Global: SPT dispatching rule									
1	1282.9	1256.6	1166.6	1134.4	1131.7	1095.3	1086.7	1075.7	485.6
2	569.4	450.1	340.9	313.5	291.8	276.0	276.5	276.5	485.6
3	242.7	232.7	221.8	192.6	192.6	186.1	185.9	183.8	485.6
4	232.9	229.9	219.0	198.2	198.2	191.7	188.3	188.3	485.6
5	232.9	229.9	219.0	198.2	198.2	191.7	188.3	188.3	485.6
6	232.9	229.9	219.0	198.2	198.2	191.7	188.3	188.3	485.6
7	232.9	229.9	219.0	198.2	198.2	191.7	188.3	188.3	485.6
8	232.9	229.9	219.0	198.2	198.2	191.7	188.3	188.3	485.6
Active branching, Local: MOD, Global: SPT dispatching rule									
1	1053.5	995.6	989.7	989.7	989.7	989.7	994.8	986.7	485.6
2	376.0	326.0	304.1	269.6	264.4	259.2	259.2	259.2	485.6
3	281.1	267.4	220.7	192.0	192.0	186.8	179.0	179.0	485.6
4	269.4	246.6	199.9	196.9	196.9	191.7	186.9	186.9	485.6
5	269.4	252.7	206.0	194.9	194.9	189.7	184.9	184.9	485.6
6	269.4	252.7	206.0	194.9	194.9	189.7	184.9	184.9	485.6
7	269.4	252.7	206.0	194.9	194.9	189.7	184.9	184.9	485.6
8	269.4	252.7	206.0	194.9	194.9	189.7	184.9	184.9	485.6

6. Conclusion

In this paper, we used the beam search to solve the classic job shop scheduling problem for the makespan and mean tardiness criteria. We also examined active and nondelay schedule generation schemes, different priority rules for both local and global evaluation functions and various values of beam and filter width parameters. According to our computational experience, we identified the proper values of these parameters which can also be used in future applications of this method. The results also indicate that the beam search method is a very good heuristic for the job shop problems.

As compared to other algorithms, the speed and the performance of a beam search based algorithm are manipulated by changing search parameters and evaluation functions. In addition, it is also quite possible to generate partial schedules with the beam search method, since as the schedules are built progressively from the first operation to the last one in a forward direction. This is an

important property of this search technique because in a stochastic and dynamic environment where unexpected events can easily upset schedules, this feature of the beam search can be well utilized to generate partial schedules in a rolling horizon scheme. Finally, we should also point out that, coding of the algorithm is very simple and hence it can easily be implemented by practitioners.

One drawback of the beam search algorithm is imperfect assessing of the promise of nodes. As a result of this, the nodes that can lead to good solutions, are sometimes erroneously discarded. For the makespan problems, however, strong lower bounds are available in the literature. Hence, these lower bounds can be used as a part of evaluation functions to improve the performance of the proposed beam search algorithm in future studies.

Appendix A. Numerical results

See Tables 10–12.

References

Aarts, E.H.L., Van Laarhoven, P.J.M., Ulder, N.L.J., 1991. Local search based algorithms for job shop scheduling. Working paper, Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands.

Aarts, E.H.L., Van Laarhoven, P.J.M., Lenstra, J.K., Ulder, N.L.J., 1994. A computational study of local search algorithms for job shop scheduling. ORSA Journal on Computing 6 (2).

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

Applegate, D., Cook, W., 1990. A Computational Study of Job-Shop Scheduling. Technical report CMU-CS-90-145, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA, 1990.

Baker, K.R., 1974. Introduction to Sequencing and Scheduling. Wiley, New York.

Baker, K.R., 1984. Sequencing rules and due-date assignments in job shop. Management Science 30 (9), 1093–1104.

Chang, Y., Matsuo, H., Sullivan, R.S., 1989. A bottleneck-based beam search for job scheduling in a flexible manufacturing system. International Journal of Production Research 27 (11), 1949–1961.

De, S., Lee, A., 1990. Flexible manufacturing system (FMS) scheduling using filtered beam search. Journal of Intelligent Manufacturing 1, 165–183.

Fox, M.S., 1983. Constraint directed search: A case study of job shop scheduling. Ph.D Thesis, Carnegie Mellon University, Pittsburgh, PA.

Garey, M.R., Johnson, D.S., 1979. Computers abd Intertracability: A Guide to the Theory of NP-Completeness, W.H. Freeman, San Francisco, California.

Glover, F., 1989. Tabu Search, Part-I. ORSA Journal on Computing 1 (3), 190–206.

Glover, F., 1990. Tabu Search, Part II. ORSA Journal on Computing 2 (1), 4–32.

Hatzikonstantis, L., Besant, C.B., 1992. Job-shop scheduling using certain heuristic search algorithm. International Journal of Advance Manufacturing Technology 7, 251–261.

Jain, A.S., Meeran, S., 1996. The Job Shop Problem: Past, Present and Future. Department of Applied Physics and Electronic and Mechanical Engineering, University of Dundee, Dundee, UK.

Kiran, A.S., Smith, M., 1984. Simulation studies in job shop scheduling – I. A survey. Computer and Industrial Engineering 8 (2), 87–93.

Lawrence, S., 1984. Resource Constrained Project Scheduling: An experimental Investigation of Heuristic Scheduling Techniques. GSIA, Carnegie Mellon University, 1984.

Lowerre, B.T., 1976. The HARPY speech recognition system. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA.

Martin, O., Otto, S.W., Felten, E.W., 1989. Large-step Markov chains for traveling salesman problem. Complex Systems 5, 299–326.

Matsua, H., Suh, C.J., Sullivan, R.S., 1988. A controlled search simulated annealing method for the general job shop scheduling problem. Working paper 03-04-88, Graduate School of Business, The University of Texas at Austin, Texas, USA.

Morton, T.E., Pentico, D.W., 1993. Heuristic Scheduling Systems. Wiley, New York, 1993.

Nakano, R., Yamada, T., 1991. Conventional genetic algorithm for job shop problems. Proceedings of the Fourth International Conference on the Genetic Algorithms and Their Applications, San Diego, California, USA, pp. 474–479.

Ow, P.S., Morton, T.E., 1988. Filtered beam search in scheduling. International Journal of Production Research 26 (1), 35–62.

Sabuncuoglu, I., Gurgun, B., 1996. A neural network model for scheduling problems. European Journal of Operations Research 2 (93), 288–299.

Sabuncuoglu, I., Karabuk, S., 1998. A beam search algorithm and evaluation of scheduling approaches for FMSs. IIE Transactions 30 (2), 179–191.