IEMS Vol. 8, No. 4, pp. 239-246, December 2009.

A Flexible Branch and Bound Method for the Job Shop Scheduling Problem

Katsumi Morikawa †
Department of Artificial Complex Systems Engineering, Graduate School of Engineering,
Hiroshima University, 1-4-1, Kagamiyama, Higashi-Hiroshima 739-8527, JAPAN
E-mail: mkatsumi@hiroshima-u.ac.jp

Katsuhiko Takahashi
Department of Artificial Complex Systems Engineering, Graduate School of Engineering,
Hiroshima University, 1-4-1, Kagamiyama, Higashi-Hiroshima 739-8527, JAPAN
E-mail: takahasi@hiroshima-u.ac.jp

Received, March 31, 2009; Revised, September 25, 2009; Accepted, October 12, 2009

Abstract. This paper deals with the makespan minimization problem of job shops. The problem is known as one of hard problems to optimize, and therefore, many heuristic methods have been proposed by many researchers. The aim of this study is also to propose a heuristic scheduling method for the problem. However, the difference between the proposed method and many other heuristics is that the proposed method is based on depth-first branch and bound, and thus it is possible to find an optimal solution at least in principle. To accelerate the search, when a node is judged hopeless in the search tree, the proposed flexible branch and bound method can indicate a higher backtracking node. The unexplored nodes are stored and may be explored later to realize the strict optimization. Two methods are proposed to generate the backtracking point based on the critical path of the current best feasible schedule, and the minimum lower bound for the makespan in the unexplored sub-problems. Schedules are generated based on Giffler and Thompson's active schedule generation algorithm. Acceleration of the search by the flexible branch and bound is confirmed by numerical experiment.

Keywords: Scheduling, Job Shop, Makespan, Depth-first Branch and Bound, Heuristic

1. INTRODUCTION

This paper deals with makespan minimization in job shops. The makespan is the length of the schedule. It is well known that the makespan minimization in job shops is one of the most difficult problems to optimize. Although several papers, e.g., Carlier and Pinson (1989), Brucker *et al*. (1994), Perregaard and Clausen (1998), Brinkkötter and Brucker (2001), have proposed strict optimization procedures, one of famous 20-job, 10-machine problems still remains unsolved (Pezzella and Merelli, 2000).

Many of these papers have been based on the depth-first branch and bound method, and their primary focus has been on eliminating meaningless nodes or on finding nodes that would lead to better schedules. The branch and bound approach controls the search flow as follows: if a node in the search tree is unsuccessful, i.e., it cannot lead to a better solution, the search engine will go to the immediate parent node and examine different child nodes generated from this parent. Once all the child nodes have been examined, it will go back to the immediate parent of the current node, and continue the search until all remaining nodes have been examined. The superiority of this flow control is simplicity in programming and less consumption of computer memory.

There is a different approach to accelerating the search of the depth-first branch and bound by utilizing search history. The main idea is to collect the reasons of unsuccessful conditions if a node is judged unsuccessful, and then infer that a new candidate node can be considered as a promising node or not. Guéret *et al*. (2000) applied the search history to the makespan minimization of open shops, and proved its effectiveness by solving benchmark problems. Although the acceleration of the search by utilizing search history was also confirmed by Morikawa *et al*. (2005) for job shops, its mechanism is rather complex.

This paper proposes another approach to the acceleration of the depth-first branch and bound method, es-

† : Corresponding Author

pecially in finding good solutions in an earlier stage of the search. To realize this goal we introduce a flexible control method within the search engine. In general, if a node is judged as unsuccessful, the search engine will go back to the immediate parent node as described before. However the search engine can also indicate a higher backtracking node without examining whole unexplored child nodes in our proposed method. The unexplored nodes are stored and will be examined later to realize the strict optimization of the search. Apparently this method can also be considered as a heuristic method by discarding unexplored nodes, interrupting the search after a certain amount of time, or terminating the search when a near-optimal solution is obtained.

Giffler and Thompson's active schedule generation algorithm (Giffler and Thompson, 1960) is adopted as a basic algorithm of makespan minimization in this study. Although their algorithm may not be the best to use for makespan minimization, it is possible to enumerate all minimum makespan schedules. Therefore, if a schedule is required that optimizes a secondary criterion from the set of minimum makespan schedules, Giffler and Thompson's algorithm may be necessary. Finding schedules that are less sensitive to the uncertainty of processing times from the set of minimum makespan schedules (Kawata, *et al*., 2003) is an example of such cases.

This paper first describes the makespan minimization procedure. An example problem is then introduced to explain the motivation of developing a flexible branch and bound method, and to illustrate key ideas when considering candidate backtracking nodes. The control mechanism is then explained in detail and the effectiveness of the proposed approach is discussed by solving benchmark problems.

2. MAKESPAN MINIMIZATION BY BRANCH AND BOUND

2.1 Assumptions

1) Let n be the number of jobs, and m the number of machines. Each job must visit each machine exactly once. The process route for jobs is given in advance.
2) The operation time is deterministic and sequence-independent. All jobs arrive at the shop at time zero and their moving times between machines are negligible.

2.2 Disjunctive Graph

Figure 1 is an initial disjunctive graph for the job shop scheduling problem. Nodes correspond to operations and directed arcs correspond to precedence constraints within each job. Operations are numbered sequentially from 0 to $nm + 1$, where 0 and $nm + 1$ are dummy operations, and the jth ($j = 1, \cdots, m$) operation of job i($i = 1, \cdots, n$) is numbered $(i-1)m+j$. The length of the directed arc(s) starting from node k is the processing time for operation k, denoted by p_k, where $p_0 = 0$ has been assumed. If there is a directed arc, $c \rightarrow d$, this means that the process for operation d can only be started after operation c has been completed. Although there are undirected disjunctive arcs between operations that must be processed on the same machine, these arcs have not been included in the figure.

As the initial graph has only taken the precedence constraints within each job into consideration, the corresponding schedule is generally infeasible because two or more operations require the same machine simultaneously. It is necessary to determine the process order for these operations to resolve this operation conflict. This decision corresponds to determining the direction of disjunctive arc(s) and directed arcs between conflicting operations must be added to the graph. The length of the added arcs is also defined by the processing time for the operation corresponding to the starting node. If there are no operation conflicts then the corresponding schedule is feasible, and its makespan equals the length of the longest path from dummy operation 0 to $nm + 1$. Therefore, solving the scheduling problem involves making the length of the longest path as short as possible when resolving operation conflicts.

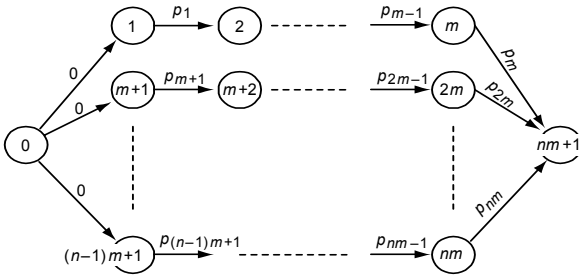


Figure 1. Initial disjunctive graph.

2.3 Head and Tail Times

For any operation c($c = 0, 1, \cdots, nm + 1$) in Figure 1, the length of the longest path from operation 0 to c is the head time for c, denoted by h_c. By subtracting p_c from the length of the longest path from c to $nm + 1$, the value obtained is the tail time for c, denoted by t_c. Undirected disjunctive arcs are ignored when calculating these values. The values of head and tail times are non-decreasing with the addition of directed disjunctive arcs, and are utilized when calculating lower bound for the makespan of the current partial schedule.

2.4 One-machine Preemptive Scheduling

Suppose that a disjunctive graph is given containing some directed disjunctive arcs. Consider the one-machine scheduling problem based on this graph. First, select a machine, calculate the head and tail times for operations to be processed on the machine, and then preemptively schedule the operations on the machine.

The scheduling method involves allocating an operation on the machine when it arrives (at its head time), or when the machine has completed the process for some other operation. If two or more operations are ready to process, select the one with the longest tail time. The schedule obtained is called JPS (Jackson's Preemptive Schedule) and its makespan is a lower bound for the makespan of the non-preemptive, one-machine sequencing problem. The JPS makespan can be obtained for each machine separately, and the maximum makespan is a lower bound for the makespan of the job shop to be solved (Carlier, 1982).

2.5 Standard Makespan Minimization Procedure

It is well known that a minimum makespan schedule can be obtained from the set of active schedules. This study therefore attempts to find a minimum makespan schedule based on Giffler and Thompson's active schedule generation algorithm. Hopeless partial schedules are pruned based on the lower bound. The optimal schedule is the last schedule produced by the procedure. The initial value of upper bound *ub* for the makespan is the makespan of the SPT (Shortest Processing Time) schedule applied to the original problem. The proposed procedure is as follows:

0°: Let S be the set of operations that can be scheduled. Initially the elements in S represent the first operation of all jobs.

1°: Calculate the head and tail times for all operations. If at least one of the following two unsuccessful conditions is satisfied, stop the search under the current node, go back to the immediate parent problem, and then go to 5° after selecting an unsolved sub-problem based on the priority described in 4.2°.

- For any operation c, the following condition is satisfied.

$$h_c + p_c + t_c \geq ub \tag{1}$$

- The lower bound given by JPS is equal to, or greater than *ub*.

2°: Let g be an operation that yields $\min_{j \in S}(h_j + p_j)$, and $\hat{m}$ be the machine on which g must be processed.

3°: Let S' be a subset of operations in S where the head time is less than $(h_g + p_g)$ and must be processed on machine $\hat{m}$.

4°: If $|S'| = 1$ then go to 4.1°, otherwise go to 4.2°.

4.1°: Remove the operation in S' from S, and add its job-successor operation to S, if it exists. Go to 5°.

4.2°: Generate $|S'|$ sub-problems by assigning each operation in S' as the first operation for processing. More precisely, if operation $g' \in S'$ is selected as the first operation, then the unscheduled operations that must be processed on $\hat{m}$ are all scheduled after g'. Thus, the direction of disjunctive arcs between them is determined. Operation g' is removed from S, and the job-successor operation of g' is added to S, if it exists. After generating $|S'|$ sub-problems, assign priorities to sub-problems based on the following criteria, select the one with the highest priority, and store other sub-problems. Go to 5°.

[Priority criteria]

First: The lower bound for the makespan calculated by considering all the machines in each sub-problem.

Second: The lower bound for the makespan on machine $\hat{m}$.

Third: The head time for the selected operation.

Fourth: The tail time for the selected operation.

A smaller value in the above criteria means a higher priority except for the fourth criterion. If two or more sub-problems are given the same priority based on the first criterion, then use the second criterion, and then the third, and lastly the fourth.

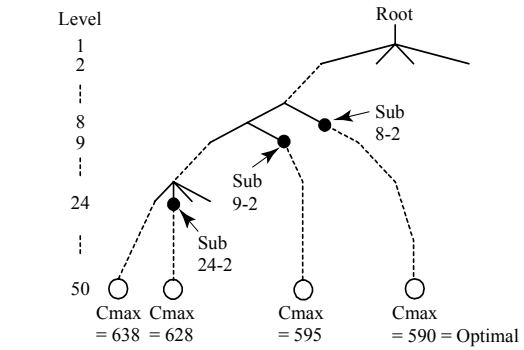
5°: If S is not empty then go back to 1°. Otherwise, the procedure finds a feasible schedule. If the makespan of the schedule, T', is less than *ub*, then output the current schedule, and set $ub = T'$. Go back to the immediate parent problem, and continue the search if there is an unsolved sub-problem.

3. A FLEXIBLE BRANCH AND BOUND METHOD

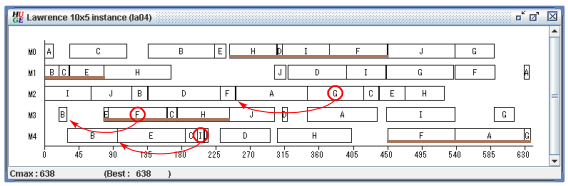
3.1 Explanatory Example

Let us minimize the makespan, *C*max, of problem la04, a famous benchmark problem with 10 jobs (A to J) and 5 machines (M0 to M4) using a depth-first branch and bound method. The search tree of this problem is shown in Figure 2 (a). As this problem has 50 operations, the number of levels becomes 50, and a first feasible solution obtained is shown in Figure 2 (b) of which makespan is 638. By analyzing the search thoroughly, we can find a better feasible solution (*C*max = 628) if we select the second sub-problem generated on level 24 as shown in Figure 2 (c). A further improvement on the makespan is realized by selecting the second sub-problem generated on level 9 shown in Figure 2 (d). An optimal schedule that realizes *C*max = 590 shown in Figure 2 (e) is obtained by selecting the second sub-problem generated on level 8.

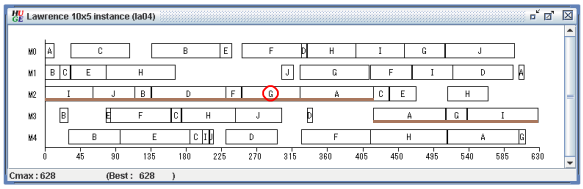
If we can go back to the node 'sub 8-2', the second sub-problem on level 8, immediately after obtaining the first feasible schedule, a certain amount of useless search could be eliminated. However, it is difficult to find nodes that can lead to better schedules from a feasible schedule on hand. It is well-known that in order to reduce the makespan of a schedule, we need to change the



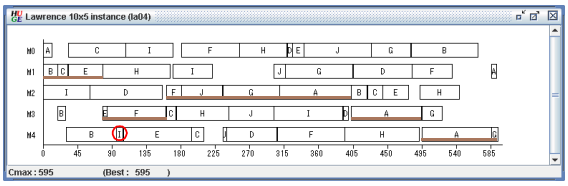
(a) Search tree of problem 1a04



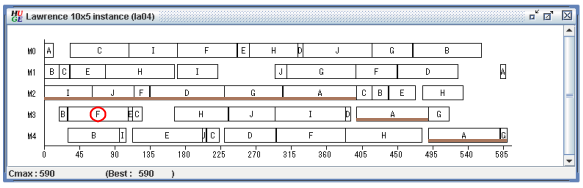
(b) Initial feasible schedule (Cmax = 638)



(c) An improved schedule (Cmax = 628) generated from sub-problem (24-2)



(d) An improved schedule (Cmax = 595) generated from sub-problem (9-2)



(e) Optimal schedule (Cmax = 590) generated from sub-problem 8-2

Figure 2. Search tree and four schedules found along with the search.

processing order of jobs on the critical path (Brucker *et al.*, 1994). The critical path in the initial schedule is shown in Figure 2 (b) with gray bold lines, i.e., the staring job is B on M1, and after visiting M3, and M0, the last job is G on M4. The sub-problem 24-2 corresponds to the assignment of job G on M2 after finishing the processing of F. However, without complicated examinations, it is difficult to find why starting job G earlier could produce a better schedule. Similarly, the effect of starting job I earlier on M4 (sub-problem 9-2) is not so clear. Probably a skilled scheduler can find that by starting job I on M4 at time 96, the remaining operations of job I on M0, M1, and M3 can be started earlier and these movements can utilize unused time intervals involved in the current schedule. The sub-problem 8-2 fixes the start of job F on M3 immediately after B. As job F is involved in the critical path, this sub-problem can be considered as a hopeful node.

3.2 Algorithm of the Flexible Branch and Bound

3.2.1 Sub-problem handling mechanism

As sub-problems generated through a depth-first search are examined in LIFO (last-in, first-out) manner, it is a good way to use a *stack* to store sub-problems. A sub-problem has (i) the set of operations, *S*, to be considered on this level, (ii) the selected operation to be scheduled on this node, and (iii) the current condition of the graph. Under the standard depth-first branch and bound method, the graph information can be maintained by using arrays for storing directed disjunctive arcs added from the root node to the current node. Each sub-problem can retrieve the graph information by storing the indices for these arrays. Storing head and tail times, and lower bound is also desirable to reduce the computational burden.

If we need to divide the stack of sub-problems into two parts, however, a dual-linked list structure instead of a general array-based stack is useful. In addition we need an array for storing lists of unsolved sub-problems. Figure 3 (a) shows the storing structure of sub-problems ready for sub-problem separations. Figure 3 (b) indicates a case that after finding a feasible schedule, the node 'sub 24-2' is selected as a backtracking node. The list of sub-problems stored in the main is shortened, and the unexamined sub-problems are stored in the 1st element of the array with the value of lower bound for later explorations. The last sub-problem element must hold the current graph information in full form as shown in Figure 3 (b). The word full form means that all directed disjunctive arcs added from the root node to the current node are stored. When starting the search from a stored sub-problem, the graph information with it is first overwritten to the current graph information and the search is then activated.

3.2.2 Backtracking node decision

The flexible branch and bound method needs solutions to the following questions: (i) when the search engine calls a halt to the current search, and (ii) which node to backtrack. As these questions are difficult to solve perfectly, we introduce simple methods as answers to these questions.

The stopping condition for the current ordinal search is defined as follows: if the number of sub-problems taken from the main list without improving the current feasible solution reaches a pre-specified threshold value,

stop the current search. On the other hand, the backtracking point is decided by the following two methods:

Method CP: Based on the critical path of the current best feasible schedule, start the search from the bottom of the list, and find a sub-problem of which the selected operation belongs to a block of the critical path, and it can be started by the start time of the first operation of the block. A block is a set of two or more operations on the critical path to be processed on a machine successively. If there is no such operation, then repeat the search to find a sub-problem of which the selected operation belongs to the critical path.

Method LB: First find the minimum lower bound from sub-problems in the main list, and then find the sub-problem with the minimum lower bound and the maximum level number.

Method CP considers the critical path of the current best feasible schedule, while Method LB focuses on the lower bound of the sub-problems. As described in section 3.1, focusing on the critical path is reasonable but it is not enough in general. In addition, there is no guarantee that at least one sub-problem can be found by Method CP. Therefore, Method LB is also introduced that considers the lower bound, an important value when realizing an efficient search in branch and bound. When applying Method CP, if Method CP cannot find a sub-problem, then Method LB is invoked after that. Therefore, this combined method is termed Method CP & LB hereafter.

3.2.3 Control mechanism of flexible branch and bound

Our proposed method differs from the standard branch and bound method in several points. Basically sub-problems in the main list are solved by the standard depth-first branch and bound manner. When the number of sub-problems in the main list reaches zero, however, one sub-problem list from the array with minimum lower bound is moved to the main list and then the search en-

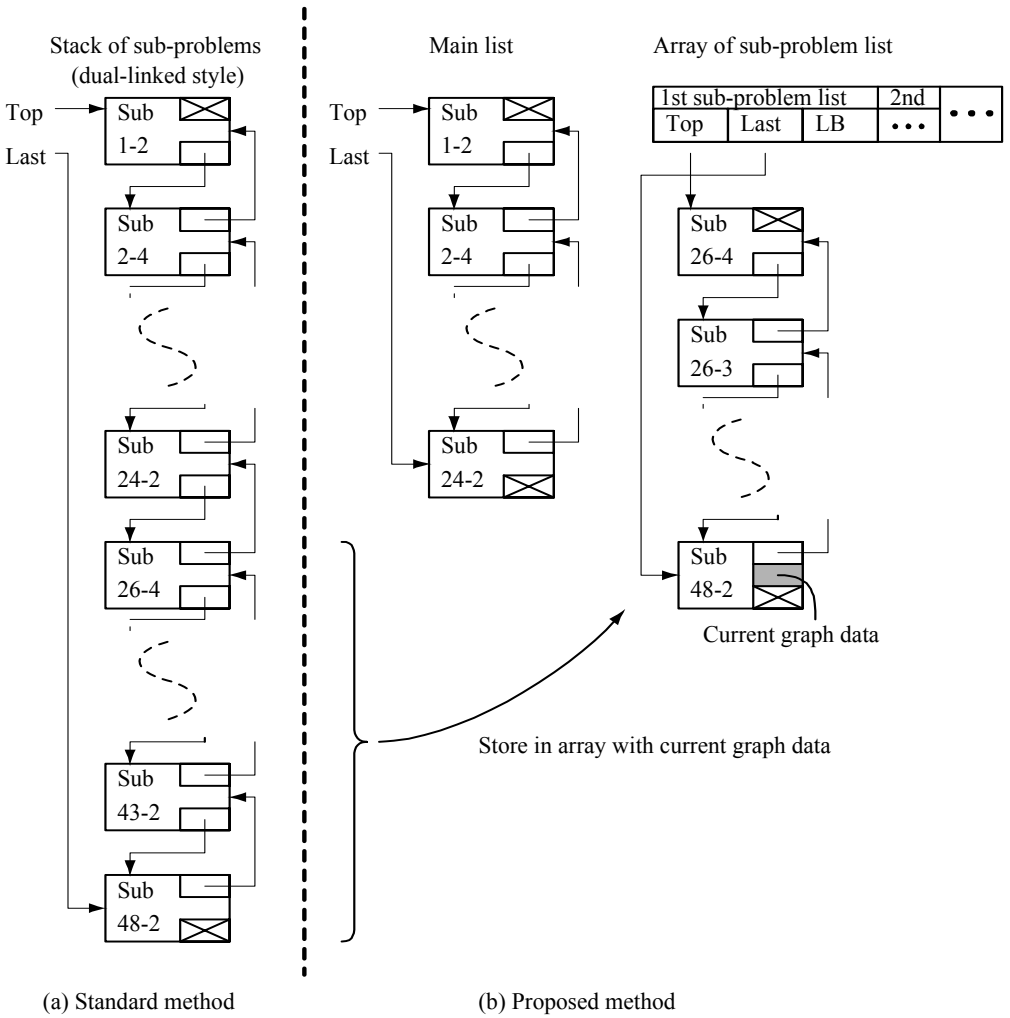


Figure 3. Example to illustrate the structure for storing sub-problems.

gine examines this moved list, if at least one sub-problem list exists in the array.

The normal LIFO operation to the sub-problems in the main list is interrupted when the number of sub-problems solved without improving the current best solution reaches a pre-specified threshold value. A sub-problem is then selected from the main list as a backtracking point by Method CP or Method LB, and a list of unsolved sub-problems under the selected sub-problem is moved to the array, and the search is started from the backtracking point. When moving the list of unsolved sub-problems, the graph information is attached to the bottom of the list, and the value of lower bound of this list is also stored. If there is no space in the array, then continue the search of the current list in the main list, or exchange the list of sub-problems between the main list and one in the array based on the lower bound of them. The search is applied to the list of sub-problems with a smaller lower bound value. These modifications to the standard branch and bound method are summarized in Figure 4.

[Handling of unsolved sub-problems]
If there is no sub-problem in the main list,
→ If there is no sub-problem list in the array then STOP, else move one sub-problem list with minimum lower bound to the main list.

[Termination of the current search]
Whenever a sub-problem is taken from the main list, set counter = counter + 1.

If a better schedule is found, set counter = 0.

If counter is greater than the threshold value,
→ If there is one free space in the array, then select a backtracking node, and store the remaining unsolved sub-problems in the array together with the current graph data.
Update the main list, and set counter = 0.
→ If no space, then
→ If the lower bound of the current sub-problem in main list is not greater than the minimum lower bound stored in the array, then continue the search.
Otherwise, select one sub-problem list with minimum lower bound from the array, and exchange it with the main list.
Set counter = 0.

Figure 4. Steps modified from the standard depth-first branch and bound method.

4. NUMERICAL EXPERIMENT

4.1 Purpose of Experiment and Problems Solved

Although the flexible branch and bound method is expected to find better schedules faster than the standard branch and bound method, its actual performance cannot be discussed without solving several benchmark problems. In addition, the performance of the proposed method may depend on the value of parameters, and methods of selecting a backtracking point. In this experiment, we focus on Method CP& LB and Method LB, both decide the backtracking node, and parameter values were fixed based on preliminary experiments.

Forty-four job-shop scheduling problems presented by Lawrence (la01-la40), Adams *et al*. (abz5, abz6), and Fisher and Thompson (ft10, ft20) were firstly selected. A preliminary experiment revealed that 21 problems (la01-15, la31-35, and ft20) could be solved optimally by the standard branch and bound method described in section 2.5 in a short time. Therefore we applied two backtracking methods to the remaining 23 problems. The program was written in Java language, and the experiment was conducted using Java 2 SDK 1.6.0 by Sun Microsystems, Inc. on a notebook computer with a Pentium M, 2.0-GHz CPU. The length of the array for storing sub-problem lists was 10, and the threshold value to interrupt the current search was 100,000. All computations were terminated after five minutes of searches if the flexible branch and bound method could not find an optimal solution with the confirmation of its optimality.

4.2 Results and Discussion

The values of minimum makespan found or consumed computation times using Method CP&LB and Method LB are summarized in Table 1. This table also includes the best makespan found within five minutes by the standard branch and bound method, and by applying the history-guided search method both reported in Morikawa *et al*. (2005). Please note that these results were obtained on a personal computer with a Pentium 4, 2.6-GHz CPU. The history-guided search accelerated searches in 13 out of 23 problems, but the optimal solution was found only in one problem.

A clear difference in best solutions between Method CP & LB and Method LB could not be found at a first glance. Method CP & LB found better solutions in 6 problems, while Method LB in 9 problems. Both methods found the same solutions in terms of the makespan in 8 problems. However, if the problems are divided into two cases; $n = m$, and $n > m$, then Method CP&LB produced better results in 5 problems, the same results in 7 problems, and a worse result in 1 problem under $n = m$. On the other hand, Method LB performed better in 8 problems, found the same solution in 1 problem, and could not find a better solution in 1 problem under the condition of $n > m$. From the numbers of backtracking points decided by Method CP and Method LB shown in the column of Method CP&LB, we can find that Method CP almost determined the backtracking points under $n >$

m. On the contrary, Method LB was often activated in Method CP&LB under $n = m$. From these results we can expect that blending these two methods properly will realize a more efficient search.

Table 1 indicates that as an exact optimization procedure, the performance of the proposed flexible branch and bound method was not so stable but it produced the same or better solutions than the solutions found by the history-guided branch and bound method. The history-guided branch and bound found a better schedule for problems la19 and ft10, solved la23 quickly, and performed better than Method CP&LB or Method LB in 3 problems. However, the proposed flexible branch and bound often found better schedules in earlier stages of the search, particularly in 10-machine problems with 15 or 20 jobs. Method LB found optimal schedules of problems la23 and la30 within about two minutes including the guarantee of the optimality. Method CP&LB also solved la23 optimally. Under these scale problems, the backtracking to upper nodes accelerated the search to some extent. For 15-job, 15-machine problems, nevertheless, the effect of proposed backtracking was limited. More elaborated methods that find a good backtracking point and guide the search from that point will be required to solve this scale problems.

Table 2. Computation time (in seconds) required to obtain an optimal solution and confirm its optimality.

la18	la19	la20	abz5	abz6
[2425s]	[2712s]	[6452s]	[21434s]	[342s]

The smallest scale of the problems shown in Table 1 is 10 jobs and 10 machines, and the proposed method found optimal solutions in two problems; la18 and abz6. However, the optimality of these solutions could not be proved in five minutes. Therefore an additional lengthy experiment was implemented to further investigate the proposed flexible branch and bound method as a strict optimization method by solving eight problems with 10 jobs and 10 machines. Based on the discussion described above, the backtracking node is firstly decided ba-

Table 1. Best makespan or computation time (in seconds) within five minutes of computation.[†1]

n×*m*	Name	Optimal[†3]	Method CP & LB[†4]	Method LB	Standard B & B[†2]	History-guided B & B[†2]
10×10	la16	945	975 (3, 7)	968	988	979
10×10	la17	784	792 (7, 4)	792	792	792
10×10	la18	848	848 (4, 7)	854	859	854
10×10	la19	842	854 (2, 8)	854	846	846
10×10	la20	902	907 (5, 5)	908	915	915
15×10	la21	1046	1101 (9, 0)	1092	1193	1128
15×10	la22	927	974 (3, 1)	969	1029	994
15×10	la23	1032	[51s] (0, 0)	[51s]	[65s]	[19s]
15×10	la24	935	970 (4, 1)	967	985	985
15×10	la25	977	1023 (6, 0)	1036	1054	1048
20×10	la26	1218	1290 (6, 0)	1274	1290	1290
20×10	la27	1235	1356 (4, 0)	1321	1371	1353
20×10	la28	1216	1234 (6, 0)	1229	1251	1245
20×10	la29	(1153)	1278 (4, 0)	1244	1291	1278
20×10	la30	1355	1394 (6, 0)	[122s]	1394	1390
15×15	la36	1268	1331 (1, 3)	1332	1350	1331
15×15	la37	1397	1449 (0, 3)	1449	1449	1449
15×15	la38	1196	1295 (4, 0)	1295	1295	1295
15×15	la39	1233	1289 (4, 0)	1289	1289	1289
15×15	la40	1222	1301 (2, 1)	1301	1386	1345
10×10	abz5	1234	1238 (8, 2)	1249	1238	1238
10×10	abz6	943	943 (1, 8)	943	966	966
10×10	ft10	930	952 (3, 7)	958	956	947

Note) [†1] If an optimal schedule was found and its optimality was confirmed within five minutes, then the computation time in seconds is shown within square brackets. (Problems la23 and la30).
[†2] These results are adopted from Morikawa *et al*. (2005).
[†3] The value 1153 in parenthesis in problem la29 means that this is the best known value.
[†4] The numbers in parentheses are the number of backtracking nodes decided by Method CP, and Method LB, respectively.

sed on the critical path of the current best feasible schedule, and the size of the space for storing sub-problem lists was set to 10,000 in order to avoid the additional computational burden caused by list exchange operations. The computation time was limited to six hours, and the required time to find an optimal solution and then confirm its optimality was obtained. In this experiment, the proposed method found optimal solutions of all these problems. This fact reinforces the validity of the proposed flexible branch and bound method as an optimization approach. In terms of the computation time, on the other hand, the method did not terminate in six hours in problems la16, la17, and ft10. Table 2 indicates the computation time of other five problems. The minimum time was less than six minutes, while the maximum time was nearly six hours. This result indicates the difficulty of confirming the solution optimality within the active schedule generation mechanism alone described in section 2.5.

5. CONCLUSION

A flexible depth-first branch and bound method that can indicate a higher backtracking node, and store the remaining unsolved sub-problems to later explorations is proposed in this study. The focused problem is the makespan minimization of job shops based on Giffler and Thompson's active schedule generation algorithm. Two methods are proposed for deciding backtracking points. One is based on the critical path of the current best feasible solution, and the other is on the lower bound for the makespan of unsolved sub-problems. A dual-linked list structure is introduced for dividing and retrieving unsolved sub-problems easily. Acceleration of the search by the flexible branch and bound is confirmed by numerical experiment.

Further acceleration will be achievable by utilizing other beneficial information involved in the feasible schedules. In addition the priority assignment rule among sub-problems has also impact on the search speed. Exhaustive examinations of experimental results will give some suggestions for resolving these issues. Applying the flexible branch and bound under other branching strategies may be also an interesting research theme.

ACKNOWLEDGMENT

This research was partially supported by the Japan Society for the Promotion of Science with a Grant-in-Aid for Scientific Research, 19510149. The authors are grateful to Mr. Kazuya Hasegawa for preparing an explanatory example.

REFERENCES

Brinkkötter, W. and Brucker, P. (2001), Solving open benchmark instances for the job-shop problem by parallel head-tail adjustments, *Journal of Scheduling*, **4**, 53-64.

Brucker, P., Jurisch, B., and Sievers, B. (1994), A branch and bound algorithm for the job-shop scheduling problem, *Discrete Applied Mathematics*, **49**, 107-127.

Carlier, J. (1982), The one-machine sequencing problem, *European Journal of Operational Research*, **11**, 42-47.

Carlier, J. and Pinson, E. (1989), An algorithm for solving the job-shop problem, *Management Science*, **35**, 164-176.

Giffler, B. and Thompson, G. L. (1960), Algorithms for solving production-scheduling problems, *Operations Research*, **8**, 487-503.

Guéret, C., Jussien, N., and Prins, C. (2000), Using intelligent backtracking to improve branch-and-bound methods: An application to open-shop problems, *European Journal of Operational Research*, **127**, 344-354.

Kawata, Y., Morikawa, K., Takahashi, K., and Nakamura, N. (2003), Robustness optimization of the minimum makespan schedules in a job shop, *International Journal of Manufacturing Technology and Management*, **5**, 1-9.

Morikawa, K., Takahashi, K., and Tabata, K. (2005), Branch and bound based makespan minimization in job shops guided by search history, *Proceedings of the 18th International Conference on Production Research*, *Salerno, Italy*, (in CD-ROM).

Perregaard, M. and Clausen, J. (1998), Parallel branch-and-bound methods for the job-shop scheduling problem, *Annals of Operations Research*, **83**, 137-160.

Pezzella, F. and Merelli, E. (2000), A tabu search method guided by shifting bottleneck for the job shop scheduling problem, *European Journal of Operational Research*, **120**, 297-310.