Trans. of the Society of Instrument and Control Engineers
Vol. E-3, No.1, 1 / 7 (2004)

Proposition of New Neighborhood Structures in a Local Search Method for Job Shop Scheduling Problems

Hitoshi IIMA* and Nobuo SANNOMIYA**

Local search methods are effective for solving scheduling problems in the case where the computation time is limited. However, the performance of the local search methods depends on the neighborhood structure, that is, the procedure to generate a new solution In this paper, a local search method with new neighborhood structures is proposed for the job shop scheduling problem of minimizing the makespan. The solution in the proposed method is represented by an operation sequence. In this representation a same schedule is generated generally from different operation sequences, and such redundancy reduces the search efficiency. The proposed neighborhood structures generate necessarily a schedule which is different from the incumbent one. Moreover, two neighborhood structures based on this idea are used alternately for escaping from local optimal solutions. The proposed method is applied to some benchmark instances. It is shown from the computational result that the proposed method outperforms other local search methods.

Key Words: scheduling, local search method, job shop, neighborhood structure

1. Introduction

Scheduling problems [1),2)] in manufacturing systems are one of the most important problems in the manufacturing industry. Typical scheduling problems are classified into the one-machine, the parallel-machine, the flow shop and the job shop problems. Of these the job shop problem is hard to solve. Local search based algorithms, in which a procedure is iteratively executed to update a solution, are promising for solving the job shop problems with the advance of computer technology. Typical local search based algorithms are the local search method, the tabu search method, the simulated annealing method and the genetic algorithm. The performance of these methods depends on the solution representation and the neighborhood structure. Gen et al. [3)], Ono et al. [4)] and Shi et al. [5)] propose, respectively, similar solution representations in which a solution is represented by sequencing jobs or operations.

In the real job site, a solution, that is, a schedule must be determined in a limited time. Hence, a heuristic rule is often used for obtaining a solution in short time. The tabu search method, the simulated annealing method and the genetic algorithm can be regarded as a method enhancing the local search method, and consume much time to obtain a slightly better solution than the local search method. Therefore, they are not desirable in real time applications. To obtain a solution in short time, Nakano et al. [6)] propose a local search method in which a solution is represented by an operation sequence. In this method, the local search and the global search are used effectively. However, the operation sequence is redundant, because different sequences are decoded to the same schedule. Consequently, a schedule is often unchanged, even if the operation sequence is changed. Such redundancy reduces the search efficiency.

In order to overcome this disadvantage, new neighborhood structures are proposed for the local search method to solve the job shop scheduling problems in this paper. In these neighborhood structures, a new operation sequence is generated by shifting an operation to a forward or backward position. In this shift, an idea not to generate useless solutions is introduced in consideration to the redundancy of operation sequence. Moreover, these neighborhood structures are used alternately for escaping from local optimal solutions. The proposed method is applied to several benchmarks, and the computational result is shown.

2. Job Shop Scheduling Problem

A set of I kinds of jobs J_i $(i = 1, 2, \cdots, I)$ is processed by using K kinds of machines M_k $(k = 1, 2, \cdots, K)$. A machine can process only one job at a time. A job J_i consists of K operations O_{ij} $(j = 1, 2, \cdots, K)$ which are executed on the respective machines given in advance. No preemption of operation is allowed. There exists a precedence constraint between operations belonging to a job, and the operations must be executed in the order of j.

* Kyoto Institute of Technology, Faculty of Engineering and Design, Sakyo-ku, Kyoto
** Okayama Prefectural University, Soja, Okayama
(Received April 25, 2002)
(Revised November 5, 2002)

For this problem, the following data set is given:

R_{ij} : Machine number on which operation O_{ij} is executed ($R_{ij} \in \{1, 2, \cdots, K\}$).

P_{ij} : Processing time for operation O_{ij}.

Q : Total number of operations. $Q = IK$.

This problem is to determine the completion time of operations in such a way that the makespan should be minimized. The makespan Z is formulated as

$$Z = \max_i c_{iK} \tag{1}$$

where c_{ij} is the completion time of operation O_{ij}.

3. Design of Local Search Method

The local search method is a method to find a local optimal solution by the following algorithm.

Step 1 Generate an initial solution as the incumbent solution.

Step 2 Generate a new solution existing in the neighborhood of the incumbent solution.

Step 3 If the new solution is better than the incumbent solution, change the incumbent solution to the new one.

Step 4 If the termination condition is satisfied, terminate this algorithm. The incumbent solution is adopted as the suboptimal solution. Otherwise, return to Step 2.

The performance of the local search method depends on the solution representation and the neighborhood structure. Hence, these procedures should be defined in a suitable form. In this section, a local search method is designed for solving the job shop scheduling problems.

3. 1 Solution Representation

Direct solution representation for job shop problems is to use the processing order for each machine. However, a set of the processing orders is often infeasible. To give such an example, consider the set of the processing order {O_{22}, O_{11}} for a certain machine and {O_{12}, O_{21}} for another. According to the precedence constraint, O_{11} and O_{21} must be executed before O_{12} and O_{22}, respectively. Therefore, this set of processing orders is infeasible. In this direct representation, there does not exist a simple rule to judge whether a set of processing orders is infeasible or not. Hence, it is hard to generate a feasible neighboring solution.

In the proposed method, the indirect solution representation proposed by Nakano et al. is used to generate a feasible neighboring solution easily. In this representation, all the operations are sequenced. The length of this operation sequence is equal to the number Q of operations. The operation sequence represents basically a processing order, and the schedule of operations is determined in turn according to the schedule decision procedure. Let ℓ_{ij} be the position of operation O_{ij} in the operation sequence. If $\ell_{ij+1} < \ell_{ij}$ ($\exists(i,j)$; $i = 1, 2, \cdots, I$, $j = 1, 2, \cdots, K-1$), the operation sequence is infeasible because of the violation of the precedence constraint. (A feasible schedule can not be generated from the operation sequence by using the schedule decision procedure.) Since there exists the simple rule to judge whether an operation sequence is infeasible or not, the proposed method can be designed in such a way that only feasible solutions are searched.

The procedure to determine a schedule from an operation sequence is explained as follows:

[Schedule decision procedure]

Step 1 Set $m(k) \leftarrow 0$ ($\forall k = 1, 2, \cdots, K$). The variable $m(k)$ represents the completion time of operation to be executed last on machine M_k.

Step 2 Set $\ell \leftarrow 1$. The variable ℓ represents the position in the operation sequence.

Step 3 The completion time of operation $O_{i_\ell j_\ell}$ ($i_\ell \in \{1, 2, \cdots, I\}$, $j_\ell \in \{1, 2, \cdots, K\}$) at the ℓ-th position is given by

$$c_{i_\ell j_\ell} = \max(m(R_{i_\ell j_\ell}),\ c_{i_\ell\ j_\ell - 1}) + P_{i_\ell j_\ell} \tag{2}$$

It is noted that c_{i0}=0 ($\forall i$).

Step 4 Set $m(R_{i_\ell j_\ell}) \leftarrow c_{i_\ell j_\ell}$.

Step 5 If $\ell = Q$, terminate the procedure. If not, set $\ell \leftarrow \ell + 1$ and return to Step 3.

Operation $O_{i_\ell j_\ell}$ at the ℓ-th position in the operation sequence is denoted as $O(\ell)$ simply.

Example 1 . In instance EX shown in **Table 1**, operation sequence {O_{21}, O_{11}, O_{22}, O_{31}, O_{32}, O_{23}, O_{12}, O_{33}, O_{13}} is decoded to the schedule shown in **Fig. 1**. In this figure, the symbol O for an operation is omitted.

As for the operation sequence, the following theorem holds.

Theorem 1 . For operation O_{ij} executed on the machine $M_{R_{ij}}$ in the operation sequence, the following variables are defined:

ℓ^a : Position of O_{ij-1}.

ℓ^b : Position of operation executed just before O_{ij} on the same machine $M_{R_{ij}}$.

ℓ^c : Position of O_{ij+1}.

ℓ^d : Position of operation executed just after O_{ij} on the same machine $M_{R_{ij}}$.(1)

(1) We assume a pseudo operation $O(0)$ before $O(1)$ if O_{ij-1} or $O(\ell^b)$ does not exist. Similarly, we assume a pseudo operation $O(Q+1)$ after $O(Q)$ if O_{ij+1} or $O(\ell^d)$ does not exist.

Table 1 Illustrative instance EX ($I = K = 3$)

Operation	R_{ij}	P_{ij}
O_{11}	1	2
O_{12}	2	3
O_{13}	3	4
O_{21}	2	5
O_{22}	1	3
O_{23}	3	2
O_{31}	3	4
O_{32}	1	5
O_{33}	2	3

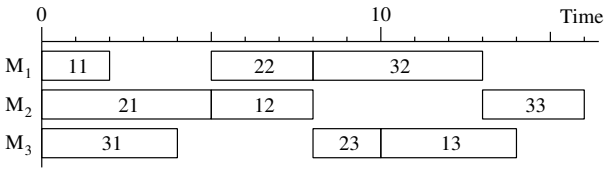

Fig. 1 Schedule for operation sequence {O_{21}, O_{11}, O_{22}, O_{31}, O_{32}, O_{23}, O_{12}, O_{33}, O_{13}}

The new operation sequence generated by shifting O_{ij} after $O(\ell)$ ($\ell = \max(\ell^a, \ell^b), \cdots, \min(\ell^c, \ell^d) - 1$) is decoded to the same schedule as the original operation sequence.

Proof. The schedule of $O(\ell)$ ($\ell = 1, 2, \cdots, \max(\ell^a, \ell^b)$) in the new operation sequence is unchanged, because the order of the sequence is unchanged. As for O_{ij}, there does not exist an operation executed on the same machine $M_{R_{ij}}$ in the positions between $\max(\ell^a, \ell^b)$ and $\min(\ell^c, \ell^d)$. Therefore, the value of $m(R_{ij})$ is unchanged in equation (2) of the schedule decision procedure. The value of c_{ij-1} is also unchanged, because there is no operation belonging to the same job as O_{ij} in these positions. Therefore, the completion time of O_{ij} is unchanged. Similarly, the completion times of the other operations in the positions are unchanged. Finally, the completion time of $O(\ell)$ ($\ell = \min(\ell^c, \ell^d) + 1, \cdots, Q$) is unchanged, because the completion times of operations $O(1), O(2), \cdots, O(\ell - 1)$ are unchanged. □

Theorem 1 shows that different operation sequences may be decoded to the same schedule. The operation sequence is redundant in this sense.

3. 2 Neighborhood Structure

This subsection describes the procedure to generate a new operation sequence from the incumbent one. Basically, a single operation O_{ij} is selected at random, and the new operation sequence is generated by shifting it forwardly or backwardly. The former shift is called forward shifting, and the latter is called backward shifting. If O_{ij} is shifted before O_{ij-1} or after O_{ij+1}, the precedence constraint is violated. To prevent this violation, O_{ij} is shifted between O_{ij-1} and O_{ij+1}. Since the range between O_{ij-1} and O_{ij+1} is limited in this shift, a procedure is introduced to increase the range. In this procedure, operation $O_{ij^*} \in \{O_{i1}, O_{i2}, \cdots, O_{ij-1}\}$ ($\{O_{iK}, O_{iK-1}, \cdots, O_{ij+1}\}$) is shifted forwardly (backwardly) in such a way that the schedule is unchanged. This procedure is called the preliminary procedure, and is explained as follows:

[Preliminary procedure for forward shifting]

Step 1 Set $j^* \leftarrow 1$.

Step 2 Set $\ell^{AB} \leftarrow \max(\ell^A, \ell^B)$, where
- ℓ^A : Position of O_{ij^*-1}.
- ℓ^B : Position of operation executed just before O_{ij^*} on machine $M_{R_{ij^*}}$.

Step 3 Shift operation O_{ij^*} just after operation $O(\ell^{AB})$.

Step 4 If $j^* = j-1$, terminate the procedure. Otherwise, set $j^* \leftarrow j^* + 1$ and return to Step 2.

[Preliminary procedure for backward shifting]

Step 1 Set $j^* \leftarrow K$.

Step 2 Set $\ell^{CD} \leftarrow \min(\ell^C, \ell^D)$, where
- ℓ^C : Position of O_{ij^*+1}.
- ℓ^D : Position of operation executed just after O_{ij^*} on machine $M_{R_{ij^*}}$.

Step 3 Shift operation O_{ij^*} just before operation $O(\ell^{CD})$.

Step 4 If $j^* = j+1$, terminate the procedure. Otherwise, set $j^* \leftarrow j^* - 1$ and return to Step 2.

Even if the preliminary procedure is applied, the schedule is unchanged because of Theorem 1.

Example 2 . For instance EX in **Table** 1, consider operation sequence

O_{ij}	O_{21}	O_{11}	O_{22}	O_{31}	O_{32}	O_{23}	O_{12}	O_{33}	O_{13}
R_{ij}	2	1	1	3	1	3	2	2	3

in Example 1. After the preliminary procedure for shifting operation O_{13} forwardly, the operation sequence becomes

O_{ij}	O_{11}	O_{21}	O_{12}	O_{22}	O_{31}	O_{32}	O_{23}	O_{33}	O_{13}
R_{ij}	1	2	2	1	3	1	3	2	3

The schedule after the preliminary procedure is unchanged, and is the same as that shown in **Fig.** 1.

After the preliminary procedure, operation O_{ij} is shifted. To explain the procedure to shift it, the following variables are defined for forward (backward) shifting in the operation sequence:

n_1 : Number of operations between O_{ij-1} (O_{ij+1}) and O_{ij}.

n_2 : Number of operations executed on machine $M_{R_{ij}}$ in operations between O_{ij-1} (O_{ij+1}) and O_{ij}.

$o_m (\in \{O_{11}, O_{12}, \cdots, O_{IK}\})$ $(m = 1, 2, \cdots, n_2)$:
Operation executed on machine M_{ij} in operations between O_{ij-1} (O_{ij+1}) and O_{ij}. These operations are numbered in the order of the operation sequence.

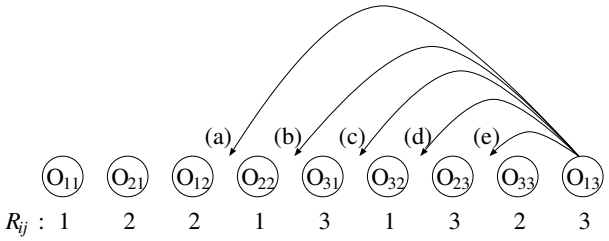

Fig. 2 Example of shifting O_{13}

There exist n_1 kinds of shifting O_{ij} in such a way that the precedence constraint is satisfied. Even if O_{ij} is shifted to any position between o_m and o_{m+1}, the schedules generated are the same because of the redundancy of operation sequence shown in Theorem 1. Therefore, only n_2 schedules are different from the incumbent one. Thus, O_{ij} is shifted just before o_m in forward shifting so that duplicate solutions are not generated in the proposed method. Similarly, O_{ij} is shifted just after o_m in backward shifting.

Example 3 . For instance EX in **Table** 1, consider operation sequence

O_{ij}	O_{11}	O_{21}	O_{12}	O_{22}	O_{31}	O_{32}	O_{23}	O_{33}	O_{13}
R_{ij}	1	2	2	1	3	1	3	2	3

generated by applying the preliminary procedure in Example 2. In the case where operation O_{13} is shifted forwardly, there exist five kinds of shifting (a)–(e) shown in **Fig. 2**. (i.e. n_1=5.) The schedule generated by (a) is the same as that by (b). Similarly, the schedule generated by (c) is the same as that by (d). Moreover, the schedule generated by (e) is the same as the incumbent one. Therefore, the available shifts are only (b) and (d). (n_2=2, o_1=O_{31}, o_2=O_{23}.)

After the n_2 shifts are selected, they are tried in the order of m. If an operation sequence generated is accepted, or all the shifts have been tried, then the procedure to shift O_{ij} ends, and another operation to be shifted is selected. The acceptance condition is described in the next subsection.

3. 3 Flow of Algorithm

An initial operation sequence is generated at random in such a way that it is decoded to a feasible solution. From this operation sequence, a new one is generated by means of the procedure described in **3. 2**. If the objective value is equal or better, then the new operation sequence is accepted, and the incumbent one is replaced with it. This procedure is iterated T times, and the schedule obtained from the last operation sequence is adopted as the suboptimal solution. The total iteration number T is given in advance.

As mentioned in **3. 2**, there are two neighborhood structures: forward shifting and backward shifting. In the former neighborhood structure, a single operation is shifted forwardly. Since the completion time of this operation becomes earlier, it is expected that the objective value becomes better. On the other hand, in the latter neighborhood structure, an operation is shifted backwardly. Although the completion time of this operation becomes later, those of other some operations may become earlier. As a result, it is expected that the objective value becomes better. These neighborhood structures have different influence in this sense. Thus, they are used alternately in the proposed method. A neighborhood structure is changed, if the search process converges to a local optimal solution. It is judged that the search process converges in the case where operation sequences generated are rejected $0.05T$ times continuously.

The detail algorithm is described as follows:

[The whole algorithm]

Step 1 Generate an initial operation sequence x at random, and calculate the objective value $Z(x)$. Set $t \leftarrow 1$, $m^* \leftarrow 0$, $w \leftarrow 0$ and $ne \leftarrow 0$, where
- t : Iteration step.
- m^* : Operation number to which operation O_{ij} is shifted.
- w : Number of times which the operation sequence generated is rejected.
- ne : Neighborhood structure used. $ne = 0$ (1) means that forward (backward) shifting is used.

Step 2 If $m^* > 0$, go to Step 6.

Step 3 Select a single operation O_{ij} at random.

Step 4 If $ne = 0$, apply the preliminary procedure for forward shifting. If $ne = 1$, apply the preliminary procedure for backward shifting. Set x to the operation sequence changed.

Step 5 Find the n_2 operations $\{o_m\}$ $(m = 1, 2, \cdots, n_2)$ to which O_{ij} is shifted. Set $m^* \leftarrow 1$.

Step 6 If $ne = 0$, generate a new operation sequence x^* by shifting O_{ij} just before operation o_{m^*}. If $ne = 1$, generate a new operation sequence x^* by shifting O_{ij} just after operation o_{m^*}.

Step 7 Calculate the objective value of the new operation sequence x^*. If $Z(x) \geq Z(x^*)$, accept x^* and set $x \leftarrow x^*$, $Z(x) \leftarrow Z(x^*)$, $m^* \leftarrow 0$ and $w \leftarrow 0$. If $Z(x) < Z(x^*)$, reject x^* and set $m^* \leftarrow m^* + 1$ and $w \leftarrow w + 1$.

Step 8 If $m^* = n_2 + 1$, set $m^* \leftarrow 0$.

Step 9 If $w \geq 0.05T$ and $ne = 0$, set $w \leftarrow 0$ and $ne \leftarrow 1$. If $w \geq 0.05T$ and $ne = 1$, set $w \leftarrow 0$ and $ne \leftarrow 0$.

Step 10 If $t = T$, terminate this algorithm and output the schedule obtained from x as the suboptimal solution. If $t < T$, set $t \leftarrow t + 1$ and return to Step 2.

4. Computational Experiment

In order to examine the performance of the proposed method, it is applied to several benchmark instances, and is compared with other methods. Moreover, the objective value obtained is compared with the optimal value or the upper bound.

4. 1 Computational Condition

The proposed method (called LS-I) is compared with Nakano's method [6] (LS-N) and a local search method (LS-T) with Taillard's neighborhood structure [7]. In this neighborhood structure, a schedule is represented by a graph. In this graph, a node and an arrow correspond to an operation and a precedence relation between operations, respectively. A new schedule is generated by applying the following steps:

[Taillard's neighborhood structure]

Step 1 On the critical path, select randomly two neighboring nodes (operations) executed on one machine.

Step 2 Reverse the direction of arrow between these nodes.

The objective value of new schedule does not become better for reversing the direction of arrow between nodes each of which is not on the critical path. Moreover, an infeasible schedule may be generated in this case.

Taillard's neighborhood structure and neighborhood structures similar to it are often used in local search based algorithms, and good results are obtained [8]. Thus, Taillard's neighborhood structure is used as a typical one in this paper. Taillard uses this neighborhood structure in the tabu search method. In this paper, a local search method is used with this neighborhood structure, because we intend to compare the neighborhood structures in the local search method. LS-T is designed in the following way:

[LS-T]

Step 1 Generate an initial graph x_g at random, and calculate the objective value $Z(x_g)$. Set the iteration step $t \leftarrow 1$.

Step 2 Generate a new graph x_g^* according to Taillard's neighborhood structure.

Step 3 Calculate the objective value $Z(x_g^*)$. If $Z(x_g) \geq Z(x_g^*)$, accept x_g^* and set $x_g \leftarrow x_g^*$ and $Z(x_g) \leftarrow Z(x_g^*)$.

Table 2 Benchmark instances

Instance	I	K
abz6	10	10
la16	10	10
la23	15	10
la24	15	10
abz7	20	15
abz8	20	15
yam2	20	20
yam3	20	20
ta41	30	20
ta43	30	20
ta64	50	20
ta65	50	20

Table 3 Total iteration number

Instance	T
abz6	10000
la16	10000
la23	45000
la24	45000
abz7	200000
abz8	200000
yam2	640000
yam3	640000
ta41	1500000
ta43	1500000
ta64	6000000
ta65	6000000

Step 4 If $t = T$, terminate this algorithm and output the schedule obtained from x_g as the suboptimal solution. If $t < T$, set $t \leftarrow t + 1$ and return to Step 2.

As for the instance data, twelve benchmarks [9] are examined. **Table 2** shows the size of the instances, i.e. the number I of jobs and the number K of machines. Of these instances, those with $I \leq 20$ are also examined in Nakano's study. Although the remaining instances are not examined in Nakano's study, they are examined in this paper in order to examine the performance for larger-scale instances.

In LS-I the value of total iteration number T must be given in advance. **Table 3** shows the value of T given for each instance. The values for the instances with $I \leq 20$ are decided in such a way that the number of schedules generated in LS-I is the same as that in LS-N. This is because both methods are compared fairly. The values for the remaining instances are decided through a preliminary calculation in such a way that the process of LS-I converges to a suboptimal solution. The algorithm is coded in C and is performed on a 2GHz, Pentium IV PC, running under Linux.

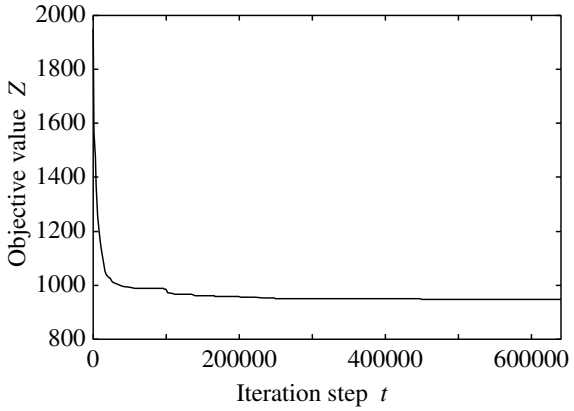

Fig. 3 Convergence process of LS-I (yam2)

4. 2 Computational Result

Fig. 3 shows the convergence process of LS-I for a certain trial in yam2. It is confirmed from this figure that the process of LS-I converges within the total iteration number T given in advance. Since the initial solution is generated at random, it is not good. Hence, the solution is frequently updated, and a better schedule is gradually obtained. At $t \approx 80000$, the neighborhood structure is changed in order to escape from a local optimal solution. Consequently, a better solution is obtained at $t \approx 100000$. Then, the neighborhood structure is changed again at $t \approx 200000$, and a slightly better solution is obtained after this change. Although the neighborhood structure is changed several times in $t > 450000$, the solution is updated no longer. It is concluded from this result that a better solution is obtained by using the two neighborhood structures alternately.

Table 4 and **Table 5** show the average and the best objective value of solutions obtained by each method, respectively. The upper bound of the objective value is also shown in Table 5. The result of LS-N in the instances used by Nakano et al. cites their paper, and is the result for ten trials with different random seeds. The others are obtained by running each program in this paper, and are the results for one hundred trials with different random seeds. It is needless to say that the best objective value for the one hundred trials tends to be better than that for the ten trials.

It is confirmed from these tables that LS-T is worse than LS-I and LS-N in all the instances. Therefore, the operation sequence is effective as the solution representation. In LS-T, the number of neighboring solutions is relatively small, because a neighboring solution is generated by reversing only the processing order of two successive operations on the critical path. Hence, the process of LS-T converges to a local optimal solution as soon as it starts.

Table 4 Comparison of average objective value

Instance	LS-I	LS-N	LS-T
abz6	985	988	1112
la16	997	992	1155
la23	1042	1047	1288
la24	988	997	1278
abz7	699	714	864
abz8	710	734	900
yam2	953	982	1250
yam3	942	971	1210
ta41	2180	2277	2823
ta43	2011	2110	2653
ta64	2709	2866	3527
ta65	2765	2986	3615

Table 5 Comparison of best objective value

Instance	LS-I	LS-N	LS-T	Upper bound
abz6	945	958	1013	943*
la16	959	959	1074	945*
la23	1032	1032	1196	1032*
la24	943	976	1164	935*
abz7	674	702	821	655
abz8	693	718	849	638
yam2	932	961	1172	861
yam3	913	945	1100	827
ta41	2120	2201	2668	2026
ta43	1933	2043	2441	1886
ta64	2702	2830	3498	2702*
ta65	2726	2926	3461	2725*

∗: Optimal value

The average objective value for LS-I is compared with that for LS-N. In small-scale instances abz6, la16, la23 and la24, LS-I is almost as good as LS-N. On the other hand, LS-I is better than LS-N in middle-scale instances abz7, abz8, yam2 and yam3. Similarly, the average objective value for LS-I is much smaller than that for LS-N in large-scale instances ta41, ta43, ta64 and ta65.

The best objective value for LS-I is compared with the optimal value or the upper bound. In la23 and ta64, the optimal solutions are obtained. Moreover, the solutions obtained in abz6 and ta65 are almost as good as the respective optimal solutions. Judging from these results, the proposed method is effective and efficient.

Finally, the average computation time in LS-I is shown in **Table 6**. For small-scale instances, the solution is obtained in a moment. Furthermore, the solution is obtained in short time for large-scale instances. Therefore, the proposed method is desirable so as to obtain a suboptimal solution in short time.

Table 6 Average computation time

Instance	Time[s]
abz6	0.37
la16	0.38
la23	2.17
la24	2.35
abz7	22.32
abz8	23.04
yam2	117.06
yam3	115.09
ta41	326.07
ta43	315.58
ta64	830.90
ta65	859.36

5. Conclusion

This paper has dealt with a job shop scheduling problem of minimizing makespan in the case where the computation time is limited. For solving this problem, new neighborhood structures have been proposed for a local search method in which a solution is represented by an operation sequence. Since this operation sequence is redundant, useless operation sequences decoded to the same schedule are generated in the method proposed by Nakano et al.. This paper has shown a condition of operation sequences decoded to the same schedule, and has presented an effective procedure to generate necessarily an operation sequence which is different from the incumbent one. In addition, two neighborhood structures based on this idea have been used alternately for escaping from local optimal solutions.

The proposed method has been compared with local search methods with another neighborhood structure. It is confirmed from the computational result that the proposed method outperforms the other methods in almost all the instances. Moreover, the solution is obtained in short time by the proposed method.

References

1) P. Brucker: Scheduling Algorithms, Springer (1995)
2) M. Pinedo: Scheduling — Theory, Algorithms, and Systems, 2nd Edition, Prentice Hall (2002)
3) M. Gen and R. Cheng: Genetic Algorithms & Engineering Design, John Wiley & Sons (1997)
4) I. Ono, M. Yamamura and S. Kobayashi: A Genetic Algorithm for Job-Shop Scheduling Problems Using Job-Based Order Crossover, Proceedings of 1996 IEEE International Conference on Evolutionary Computation, 547/552 (1996)
5) G. Shi, H. Iima and N. Sannomiya: A New Encoding Scheme for Solving Job Shop Problems by Genetic Algorithm, Proceedings of 35th IEEE Conference on Decision and Control, 4395/4400 (1996)
6) T. Nakano, Y.J. Tian and N. Sannomiya: An Improved Local Search Method for Solving Job Shop Scheduling Problems, The Transactions of the Institute of Electrical Engineers of Japan, **121C**–10, 1627/1633 (2001) (in Japanese)
7) E.D. Taillard: Parallel Taboo Search Techniques for the Job Shop Scheduling Problem, ORSA Journal on Computing, **6**–2, 108/117 (1994)
8) R.J.M. Vaessens, E.H.L. Aarts and J.K. Lenstra: Job Shop Scheduling by Local Search, INFORMS Journal on Computing, **8**–3, 302/317 (1996)
9) http://www.ms.ic.ac.uk/info.html

Hitoshi Iima (Member)

He received the B.E., M.E. and Dr. Eng. degrees from Kyoto Institute of Technology in 1991, 1993 and 1999, respectively. Since 1995, he has been a Research Associate at the Faculty of Engineering and Design in Kyoto Institute of Technology. His research interests include combinatorial optimization and manufacturing systems.

Nobuo Sannomiya (Member)

He received Dr. Eng. from Kyoto University in 1969. He was a Professor at the Faculty of Engineering and Design in Kyoto Institute of Technology from 1986 to 2003. He is now the Professor Emeritus of Kyoto Institute of Technology and the President of Okayama Prefectural University. His research interests include system modeling and optimization.