OR Spektrum (1998) 20: 21–28

OR Spektrum

Tabu-search for the multi-mode job-shop problem

Peter Brucker, Jürgen Neyer

Fachbereich Mathematik/Informatik, Universität Osnabrück, Albrechtstrasse 28, D-49076 Osnabrück, Germany

Received: 21 November 1996 / Accepted: 18 July 1997

Abstract. In a multi-processor-tasks job-shop problem (MPTJSP) there is a machine set associated with each operation. All machines are needed for the whole processing period to process the operation. The objective is to find a schedule which minimizes the makespan. In a multi-mode job-shop problem (MMJSP) there is a set of machine sets associated with each operation. One has to assign a machine set to each operation and to solve the resulting MPTJSP such that the resulting makespan is minimized. For the MMJSP a tabu-search algorithm is presented. Computational results are reported.

Zusammenfassung. In einem Multi-Processor-Task Job-Shop Problem (MPTJSP) wird jeder Operation eine Maschinenmenge zugeordnet. Für die Bearbeitung einer Operation werden dabei während des gesamten Bearbeitungszeitraums alle Maschinen benötigt. Ziel ist es nun, einen Bearbeitungsplan zu bestimmen, in dem die Gesamtbearbeitungsdauer minimal ist. In einem Multi-Mode Job-Shop Problem (MMJSP) wird jeder Operation eine Menge von Maschinenmengen zugeordnet. Hierbei muß jeder Operation eine Maschinenmenge zugewiesen werden und das sich daraus ergebene MPTJSP mit dem Ziel der Minimierung der Gesamtbearbeitungsdauer gelöst werden. Für das MMJSP wird ein Tabu-Suche Algorithmus vorgestellt. Außerdem werden die erhaltenen Rechenergebnisse aufgeführt.

Key words: Tabu-search, multi-mode job-shop, multi-processor-task job-shop, multi-purpose-machine job-shop

Schlüsselwörter: Tabu-Suche, Mehrmodus-Job-Shop, Mehrprozessoroperationen-Job-Shop, Mehrzweckmaschinen-Job-Shop

1 Introduction

In a job-shop problem, n jobs $J_1, \dots, J_n$ have to be processed on m machines $M_1, \dots, M_m$. Job J_i consists of n_i operations $O_{i1}, \dots, O_{in_i}$ which have to be processed in this order, i.e. operation $O_{i,j+1}$ has to be processed after operation O_{ij} for each stage $j = 1, \dots, n_i - 1$.

At any time each machine can process at the most one operation, and for each operation O_{ij} a processing time $p_{ij} > 0$ and a machine μ_{ij}, on which operation O_{ij} must be processed, are known in advance.

The objective is to find a schedule which minimizes the makespan $C_{\max} = \max_{i=1}^{n} C_i$ where C_i denotes the finishing time of the last operation of job J_i.

In a multi-processor-task (operation) job-shop problem (MPT job-shop problem) there is a set $A_{ij} \subseteq \{M_1, \dots, M_m\}$ of machines associated with each operation O_{ij}. Operation O_{ij} occupies all machines in this set A_{ij} during its processing time. Thus, two operations O_{ij} and $O_{i'j'}$ with $(i, j) \neq (i', j')$ can be processed at the same time only if $A_{ij} \cap A_{i'j'} = \emptyset$. MPT job-shop problems have been investigated in Krämer (1995) and Brucker and Krämer (1995).

In a multi-purpose-machine job-shop problem (MPM job-shop problem) there is again a set A_{ij} of machines associated with each operation O_{ij}. Here we have to assign a machine $\mu_{ij} \in A_{ij}$ to each operation O_{ij} and to schedule the operations on the assigned machines such that the corresponding makespan is minimized. MPM job-shop problems are discussed in Jurisch (1992), Dauzère-Pérès and Paulli (1995), Hurink et al. (1994), and Brucker and Schlie (1990).

The multi-mode job-shop problem (MMJSP) is a combination of both the MPT job-shop problem and the MPM job-shop problem. Associated with each operation O_{ij} there is a set $\mathcal{A}_{ij} = \{A_{ij}^1, \dots, A_{ij}^{m_{ij}}\}$ of machine sets $A_{ij}^k \subseteq \{M_1, \dots, M_m\}$ and processing times $p_{ij}^k > 0$, $k = 1, \dots, m_{ij}$. We have to assign a machine set $A_{ij}^k \in \mathcal{A}_{ij}$ to each operation O_{ij} on which O_{ij} has to be processed. If A_{ij}^k is assigned to O_{ij}, then O_{ij} occupies all machines in A_{ij}^k for p_{ij}^k time units.

Sprecher and Drexl (1996a,b) developed a branch-and-bound algorithm for the multi-mode resource-constrained project scheduling problem which is a generalization of the MMJSP.

MMJSP is a very difficult problem because the job-shop problem which is a special case of MMJSP is strongly NP-

Supported by the Deutsche Forschungsgemeinschaft, Project Br389/15-1
Correspondence to: P. Brucker

hard. We present a tabu-search heuristic for the MMJSP. The quality of tabu-search is influenced by the quality of the underlying neighborhood. In Sect. 2 we introduce neighborhood structures which exploit structural properties of the problem. Corresponding tabu-search procedures are described, and computational results with these procedures are reported in Sect. 3.

2 Neighborhood structures

An illuminating representation for the job-shop problem is provided by the disjunctive graph model due to Roy and Sussmann (1964). We use this representation to derive neighborhoods for our tabu-search heuristic. In Sect. 2.1 the disjunctive graph model is briefly described. Based on this description and possible ways to improve nonoptimal schedules, neighborhoods for the MMJSP are derived in Sect. 2.2. Finally, in Sect. 2.3 efficient methods are presented for calculating neighbors for a given solution.

2.1 The disjunctive graph model

Given an assignment μ which associates with each operation O_{ij} a machine set $\mu(O_{ij}) \in \mathcal{M}_{ij}$, corresponding schedules can be represented using the disjunctive graph model. A disjunctive graph $G = (V, C \cup D)$ is defined as follows.

V is the set of nodes representing the operations of all jobs. In addition, there are two special nodes, a source 0 and a sink $*$. There is a weight associated with each node. The weights of 0 and $*$ are zero while the weights of the other nodes are the processing times of the corresponding operations.

C is the set of directed **conjunctive arcs**. These arcs reflect the job orders of the operations. Additionally, there are conjunctive arcs between the source and the first operations of all jobs and between the last operations of all jobs and the sink. More precisely, we have

$$\begin{aligned} C = & \{O_{ij} \to O_{i,j+1} \mid i = 1, \dots, n;\ j = 1, \dots, n_i - 1\} \\ & \cup \{0 \to O_{i1} \mid i = 1, \dots, n\} \\ & \cup \{O_{in_i} \to * \mid i = 1, \dots, n\}. \end{aligned}$$

D is the set of undirected **disjunctive arcs**. Such an arc exists for each pair of operations which are incompatible with respect to μ, i.e. operations O_{ij} and $O_{i'j'}$ with $\mu(O_{ij}) \cap \mu(O_{i'j'}) \neq \emptyset$.

The basic scheduling decision is to define an ordering between those operations which are incompatible. This can be done by turning undirected disjunctive arcs into directed ones. A set S of directed disjunctive arcs is called a **selection**. Feasible schedules are represented by complete selections. A selection S is **complete** if

- each disjunctive arc is directed,
- the resulting directed graph $G(S) = (V, C \cup S)$ has no cycles.

The schedule represented by a complete selection S is constructed as follows. Start each operation O_{ij} at time r_{ij} where r_{ij} is the length of a longest path in $G(S)$ from 0 to O_{ij}. The length of a path in $G(S)$ is the sum of the weights of all nodes belonging to the path, the last node excluded. A longest path from 0 to the sink $*$ is called a **critical path**. The **length** $L(P)$ of a critical path P is equal to the makespan of the schedule.

We call r_{ij} the **head** of operation O_{ij}. Symmetrically, the **tail** of operation O_{ij} is the length of a longest path from O_{ij} to $*$ in $G(S)$.

2.2 Neighborhood

In this section we introduce operators which are used to define neighborhoods for the tabu-search. These operators generalize operators used by Hurink et al. (1994) in connection with the multi-purpose machine job-shop problem. Again, we consider a given assignment μ. Let S be a complete selection with respect to μ and let $G(S)$ be the resulting directed graph. We have shown that S defines a feasible schedule. On the other hand, if we have a feasible schedule, then this schedule induces a complete selection. Therefore we identify feasible schedules with corresponding complete selections.

Our search space will be the set of all pairs (μ, S) where S is a complete selection with respect to μ. (μ, S) is called **feasible solution**.

The definition of a neighborhood for this search space is based on necessary conditions for improving a current solution S. These conditions use the concept of blocks.

Let $U = (u_1, \dots, u_l)$ be a path in $G(S)$. Then **blocks** of U can be defined recursively as follows:

- if $u_1 \to u_2 \in C$, then the set of blocks of U is equal to the set of blocks of $(u_2, \dots, u_l)$.
- if $u_1 \to u_2 \notin C$, then a longest subpath $U' = (u_1, \dots, u_i)$ satisfying the properties
  (i) the vertices of U' are a clique, i.e. $u_r \to u_s \in C \cup S$ for all $1 \leq r < s \leq i$,
  (ii) $u_r \to u_{r+1} \notin C$ for $r = 1, \dots, i-1$,
  is a block of U. Furthermore, the blocks of $(u_i, \dots, u_l)$ are blocks of U.

Note that a block contains at least two vertices.

Theorem 1. Let y be a feasible solution for a given MMJSP and let S be the corresponding complete selection. Let y' be a feasible solution which improves y. Then there exists a block B of a critical path P in $G(S)$ such that one of the following properties holds.

(i) In y' one operation O_{ij} of B is processed on a machine set which is different from the machine set for O_{ij} in y.
(ii) In y' one operation of B different from the first operation of B is processed before all operations of B.
(iii) In y' one operation of B different from the last operation of B is processed after all operations of B.

The proof of this theorem is similar to the proof of a corresponding theorem in Jurisch (1992) or Krämer (1995).

We obtain neighbors of a feasible solution y with the corresponding graph $G(S)$ by applying to y one of the three operators change-assignment (O_{ij}), move-before (O_{ij}), move-after (O_{ij}) where O_{ij} is a suitable operation on a critical path in $G(S)$.

Change-assignment (O_{ij}), which is defined for any operation on a critical path in $G(S)$, can be specified by the following steps.

Change-assignment (O_{ij})

1. Eliminate all (directed) disjunctive arcs which are incident with O_{ij}.
2. Add all disjunctive arcs according to the new machine set assigned to O_{ij}.
3. Turn the new disjunctive arcs into directed ones such that the resulting directed graph is acyclic.

The operator move-before is more complicated. Let O_{ij} be an operation of a block B belonging to a critical path of $G(S)$, which is different from the first operation in B. If in $G(S)$ there exists a path from the first operation in B to $O_{i,j-1}$, then it is not possible to move O_{ij} before the first operation in B without creating a cycle. In this case the operator move-before (O_{ij}) is not defined.

Move-before (O_{ij})

1. IF no path exists from the first operation in B to $O_{i,j-1}$ THEN
   BEGIN
2. Eliminate the directions of all disjunctive arcs incident with O_{ij};
3. Add a directed disjunctive arc from O_{ij} to the first operation in B;
4. Turn the remaining disjunctive arcs into directed ones such that the resulting directed graph is acyclic
   END

The operator move-after is defined similarly.

In Step 3 of change-assignment (O_{ij}) and Step 4 of move-before (O_{ij}) there are several possibilities to turn disjunctive arcs into directed ones such that the resulting directed graph is acyclic. In the next section we will present a procedure which chooses from all possible orientations one which minimizes the makespan.

2.3 Reorientation of disjunctive arcs

In this section we will describe a procedure for the reorientation of disjunctive arcs incident with O_{ij} in Step 3 of change-assignment (O_{ij}) or Step 4 of move-before (O_{ij}). The reorientation is undertaken such that

- the resulting network has no cycles,
- the makespan of a corresponding schedule is minimal.

Such a reorientation is called **optimal**.

We start with the reorientation for change-assignment (O_{ij}). A reorientation procedure for move-before (after) (O_{ij}) can be derived by an easy modification.

Let I be the set of operations which are incompatible with O_{ij} after a new machine set is assigned to O_{ij}. Assume that $I \neq \emptyset$ (otherwise there is nothing to do) and denote by I_P and I_S the sets of disjunctive predecessors and disjunctive successors, respectively, of O_{ij} after a reorientation of the arcs connecting O_{ij} with the operations in I. Clearly, any partition of I into disjoint sets I_P and I_S defines a reorientation.

We now study a possible structure of an optimal reorientation I_P, I_S. The corresponding heads and tails after this reorientation are denoted by $\widetilde{r}_{kl}$ and $\widetilde{q}_{kl}$.

If in the network $\bar{N}$, which is derived from the original network by the elimination of O_{ij}, there exists a path from $O_{i,j+1}$ to an operation O_{kl}, then O_{kl} must belong to I_S. Otherwise the reorientation would create a cycle. Similarly, if in $\bar{N}$ there exists a path from some operation O_{kl} to $O_{i,j-1}$, then O_{kl} must belong to I_P. Note that there is no operation O_{kl} satisfying both properties because otherwise in the original network we would have a cycle

$$O_{kl} \to \ldots \to O_{i,j-1} \to O_{ij} \to O_{i,j+1} \to \ldots \to O_{kl}.$$

Let I_P^f (I_S^f) be the set of operations $O_{kl} \in I$ such that there exists a path from O_{kl} $(O_{i,j+1})$ to $O_{i,j-1}$ (O_{kl}). We must have $I_P^f \subseteq I_P$ and $I_S^f \subseteq I_S$.

To study how the remaining operations of I split into

$$I_P^r \cup I_S^r = I \setminus (I_P^f \cup I_S^f)$$

we consider the length $L(P)$ of a longest path P from 0 to $*$ containing O_{ij} in the network induced by I_P, I_S:

$$L(P) = \max\{\max_{O_{kl}\in I_P}(\widetilde{r}_{kl} + p_{kl}), \widetilde{r}_{i,j-1} + p_{i,j-1}\} + p_{ij}$$

$$+\max\{\max_{O_{kl}\in I_S}(p_{kl} + \widetilde{q}_{kl}), p_{i,j+1} + \widetilde{q}_{i,j+1}\}. \quad (1)$$

Let $h = \max_{O_{kl}\in I_P}(\widetilde{r}_{kl} + p_{kl})$. Then all operations $O_{kl} \in I_S \setminus I_S^f$ with $\widetilde{r}_{kl} + p_{kl} \leq h$ can be moved from I_S to I_P without increasing the longest path length (1). We also know that $h_f := \max_{O_{kl}\in I_P^f}(\widetilde{r}_{kl} + p_{kl}) \leq h$.

We conclude that an optimal partition of I has the form

$$I_P^f \cup I_P^h, I_S^f \cup I_S^h \quad (2)$$

where

$$I_P^h = \{O_{kl} \in I \setminus (I_P^f \cup I_S^f) | \widetilde{r}_{kl} + p_{kl} \leq h\}$$

$$I_S^h = \{O_{kl} \in I \setminus (I_P^f \cup I_S^f) | \widetilde{r}_{kl} + p_{kl} > h\}$$

with $h \in H := \{h_f\} \cup \{\widetilde{r}_{kl} + p_{kl} | O_{kl} \in I \setminus (I_P^f \cup I_S^f); \widetilde{r}_{kl} + p_{kl} > h_f\}$.

To find an optimal reorientation we have to evaluate partition (2) for each $h \in H$ and to choose the best one. All relevant values h and the corresponding $L(P)$-value can be computed easily if we sort the operations $O_{kl} \in I \setminus (I_P^f \cup I_S^f)$ according to their $(\widetilde{r}_{kl} + p_{kl})$ - values.

Next we will show that the network resulting from $I_P = I_P^f \cup I_P^h$ and $I_S = I_S^f \cup I_S^h$ has no cycles. To prove this it is sufficient to show that there is no path from an operation $O_{kl} \in I_S$ to an operation $O_{k'l'} \in I_P$. This claim holds for

- $O_{kl} \in I_S^h$, $O_{k'l'} \in I_P^h$, because $\widetilde{r}_{kl} + p_{kl} > \widetilde{r}_{k'l'} + p_{k'l'} \geq \widetilde{r}_{k'l'}$,
- $O_{kl} \in I_S^f$, $O_{k'l'} \in I_P^h$, because otherwise a path

  $$O_{i,j+1} \to \ldots \to O_{kl} \to \ldots \to O_{k'l'}$$

  would exist, which would imply $O_{k'l'} \in I_S^f$,

- $O_{kl} \in I_S^h$, $O_{k'l'} \in I_P^f$, because otherwise a path

$$O_{kl} \to \ldots \to O_{k'l'} \to \ldots \to O_{i,j-1}$$

would exist, which would imply $O_{kl} \in I_P^f$,
- $O_{kl} \in I_S^f$, $O_{k'l'} \in I_P^f$, because otherwise we would have the following cycle in the original graph:

$$O_{kl} \to \ldots \to O_{k'l'} \to \ldots \to O_{i,j-1} \to O_{ij} \to O_{i,j+1} \\ \to \ldots \to O_{kl}.$$

To calculate an optimal partition we have to evaluate (1) for the partitions (2). This means that we have to calculate the heads $\widetilde{r}_{kl}$ and tails $\widetilde{q}_{kl}$ in (1) in advance.

For $O_{i,j-1}$ we have $\widetilde{r}_{i,j-1} = r_{i,j-1}$ and for $O_{i,j+1}$ we have $\widetilde{q}_{i,j+1} = q_{i,j+1}$. This follows from the fact that the head of $O_{i,j-1}$ (tail of $O_{i,j+1}$) changes only if, before the new orientation, O_{ij} belonged to a longest path from 0 to $O_{i,j-1}$ (from $O_{i,j+1}$ to $*$). This is impossible because O_{ij} is a conjunctive successor of $O_{i,j-1}$ (predecessor of $O_{i,j+1}$).

After reorientation, for an operation $O_{kl} \in I_P$ ($O_{kl} \in I_S$) it is not possible for O_{ij} to belong to a path from 0 to O_{kl} (from O_{kl} to $*$). Otherwise we would have a cycle. Thus, the new heads and tails can be calculated using the following procedure.

1. Eliminate all disjunctive arcs which are incident to O_{ij};
2. Calculate the new heads and tails by applying longest path algorithms;
3. Reinsert the disjunctive arcs eliminated in Step 1.

The computational effort for the whole procedure is bounded by $O(m)$ where m is the number of arcs in the network.

For the procedure move-before (O_{ij}) we already know that the first operation in the block, say O_{kl}, must belong to I_S. Thus, we have to add O_{kl} to I_S^f. Furthermore, all operations $O_{k'l'} \in I$ with the property that a path from O_{kl} to $O_{k'l'}$ exists must be added to I_S^f.

The procedure move-after (O_{ij}) can be implemented in a similar way.

The algorithms presented so far are time-consuming because we have to

- recalculate heads and tails,
- generate the sets I_P^f and I_S^f to check its feasibility.

Next we try to reduce the computational effort by modifying the procedure. A consequence of these modifications is that we can no longer guarantee that the reorientation is optimal. The modifications are as follows.

1. Replace heads and tails $\widetilde{r}_{ij}, \widetilde{q}_{ij}$ by r_{ij}, q_{ij}.
2. Avoid generation of I_P^f and I_S^f by considering partitions satisfying

$$\max_{O_{kl} \in I_P} \{r_{kl} + p_{kl}\} < \min_{O_{kl} \in I_S} \{r_{kl} + p_{kl}\}$$

and

$$\max_{O_{kl} \in I_P} \{r_{kl} + p_{kl}\} \in [a, b[$$

where a, b are defined by

- $a = r_{i,j-1} + p_{i,j-1}$, $b = r_{i,j+1} + p_{i,j+1}$ if another machine set is assigned to O_{ij},
- $a = r_{i,j-1} + p_{i,j-1}$, $b = r_{f(B)} + p_{f(B)}$ if O_{ij} is moved before the first operation $O_{f(B)}$ of the block B containing O_{ij},
- $a = r_{l(B)} + p_{l(B)}$, $b = r_{i,j+1} + p_{i,j+1}$ if O_{ij} is moved after the last operation $O_{l(B)}$ of the block containing O_{ij}.

If $b \leq a$, then the corresponding operation is not applied.

It remains to show that the modified procedure always creates a feasible schedule, i.e. there exists neither a path from I_S to $O_{i,j-1}$ nor from $O_{i,j+1}$ to I_P in the reoriented network. We may consider only the case $a = r_{i,j-1} + p_{i,j-1}$, $b = r_{i,j+1} + p_{i,j+1}$ because $r_{l(B)} + p_{l(B)} > r_{i,j-1} + p_{i,j-1}$ and $r_{f(B)} + p_{f(B)} < r_{i,j+1} + p_{i,j+1}$. Thus, the intervals $[a, b[$ for the move operations are contained in the interval for the reassignment operation. $r_{l(B)} + p_{l(B)} > r_{i,j-1} + p_{i,j-1}$ holds because otherwise $r_{l(B)} < r_{l(B)} + p_{l(B)} \leq r_{i,j-1} + p_{i,j-1} \leq r_{ij}$, which is a contradiction to the fact that O_{ij} is a disjunctive predecessor of $l(B)$. Similarly, $r_{f(B)} + p_{f(B)} < r_{i,j+1} + p_{i,j+1}$ holds.

No path from an operation in I_S to $O_{i,j-1}$ exists because

$$r_{k'l'} + p_{k'l'} \geq \min_{O_{kl} \in I_S} \{r_{kl} + p_{kl}\} > \max_{O_{kl} \in I_P} \{r_{kl} + p_{kl}\} \\ \geq r_{i,j-1} + p_{i,j-1} > r_{i,j-1}$$

for all $O_{k'l'} \in I_S$. Similarly, no path exists from $O_{i,j+1}$ to I_P.

Table 1

	m	n
m06	6	6
m10	10	10
m20	5	20
l01 - l05	5	10
l06 - l10	5	15
l11 - l15	5	20
l16 - l20	10	10
l21 - l25	10	15
l26 - l30	10	20
l31 - l35	10	30
l36 - l40	15	15

Table 2

	$\lvert M_{ij}\rvert$ave	$\lvert M_{ij}\rvert$max
edata:	1.15	2 ($m \leq 6$)
		3 ($m \geq 10$)
rdata:	2	3
vdata:	$\frac{1}{2}m$	$\frac{4}{5}m$

3 Implementation of tabu-search procedures and computational results

In Sect. 3.1 the implemented tabu-search procedures are described. These procedures have been tested on different types of problems. The corresponding computational results are presented in Sect. 3.2.

3.1 Tabu-search procedures

We start the tabu-search with a solution calculated by a simple heuristic. This heuristic works as follows. To each op-

Table 3

Tabu-search edata							
		MMJSP			MPMJSP		
Data	LB	UB-start	UB-TS	CPU	UB-start	UB-TS	CPU
m06	*55	60	55	0.1	57	57	0.3
m10	*871	1314	920	32.5	995	917	7.5
m20	*1088	1699	1210	82.9	1210	1109	69.2
l01	*609	775	609	2.8	688	611	1.9
l02	*655	901	700	1.5	667	655	12.3
l03	*550	840	594	1.5	647	573	1.1
l04	*568	805	607	1.5	613	578	12.3
l05	*503	622	503	1.8	510	503	1.8
l06	*833	968	833	0.7	900	833	33.7
l07	*762	1001	777	90.8	807	765	37.3
l08	*845	991	850	3.6	949	845	35.9
l09	*878	990	878	79.2	912	878	40.5
l10	*866	962	873	2.9	880	866	27.3
l11	1087	1254	1135	109.7	1158	1106	86.3
l12	*960	1211	979	10.4	1039	960	84.7
l13	*1053	1242	1053	4.0	1215	1053	55.1
l14	*1123	1292	1123	123.8	1173	1151	70.7
l15	*1111	1655	1122	134.6	1217	1111	61.2
l16	*892	1230	1013	2.0	961	924	3.9
l17	*707	938	749	10.8	774	757	4.1
l18	*842	1067	894	0.9	864	864	2.3
l19	*796	1063	844	21.7	854	850	3.6
l20	*857	1258	909	9.7	947	919	2.9
l21	895	1430	1077	169.1	1259	1066	47.1
l22	832	1290	929	20.3	1049	919	39.6
l23	950	1409	977	168.6	1122	980	43.6
l24	881	1321	951	138.5	1047	952	43.2
l25	894	1375	973	26.0	1148	970	40.7
l26	1089	1664	1160	248.6	1268	1169	79.4
l27	1181	1732	1327	17.1	1403	1230	81.1
l28	1116	1605	1247	37.4	1335	1204	79.7
l29	1058	1582	1216	109.0	1369	1210	85.9
l30	1147	1712	1272	124.7	1436	1253	85.0
l31	1523	2139	1566	415.6	1797	1596	272.8
l32	1698	2171	1722	405.0	1835	1769	112.6
l33	*1547	2062	1599	397.3	1749	1575	294.8
l34	1592	2310	1645	386.9	1781	1627	255.0
l35	*1736	2504	1760	120.1	1817	1736	236.6
l36	1006	1727	1215	227.0	1355	1247	58.8
l37	1355	2019	1455	56.0	1621	1453	57.9
l38	1019	1881	1304	91.4	1232	1185	55.6
l39	1151	1673	1231	242.5	1390	1226	58.4
l40	1034	1795	1223	289.3	1324	1214	113.3

eration O_{ij} a machine set A_{ij}^k with the smallest processing time p_{ij}^k is assigned. Using these processing times its total processing time is calculated for each job. The scheduling procedure is based on a list which contains all of the jobs ordered according to nonincreasing total processing times. All first operations of the jobs are scheduled in this list order. Then the second operations of all jobs are scheduled in this list order, etc. If a job is completely scheduled, it will be eleminated from the list. The process stops once all jobs are eliminated.

The neighbors of a solution are specified by

- an operation O_{ij},
- the type of operator to be applied to O_{ij}: change-assignment, move-before, move-after,
- the partition of the set I of jobs which are incompatible with O_{ij} into the sets I_P and I_S.

In the tabu-search procedure the neighbors are investigated according to the sequence of the block operations on the critical path. For each block operation those neighbors which can be generated by the operator change-assignment are considered first. After that the neighbors generated by the operator move-before and move-after are considered successively if possible.

There are two strategies for choosing a partition I_S, I_P of I:

S1: Choose the best partition, i.e. a partition of I which minimizes the makespan.

S2: Choose the best partition from the restricted set of partitions described at the end of Sect. 2.3.

The tabu-list is organized as follows. Each tabu-list element contains

- the moved operation O_{ij},
- the old machine set for O_{ij},
- the predecessor and successor set of O_{ij} in the old schedule.

Table 4

Data	Tabu-search rdata							Tabu-search vdata						
		MMJSP			MPMJSP				MMJSP			MPMJSP		
	LB	UB-start	UB-TS	CPU	UB-start	UB-TS	CPU	LB	UB-start	UB-TS	CPU	UB-start	UB-TS	CPU
m06	*47	60	47	1.2	50	47	1.8	*47	100	47	1.2	48	47	0.3
m10	679	1314	701	113.7	803	737	4.1	*655	2237	655	54.3	655	655	2.7
m20	1022	1699	1025	250.3	1072	1028	27.5	*1022	2743	1024	353.2	1038	1023	218.6
l01	570	775	575	65.6	591	574	24.0	*570	1745	572	94.5	595	573	43.6
l02	529	901	534	62.4	580	535	24.9	*529	1610	532	90.2	659	531	41.3
l03	477	840	481	65.1	537	481	26.6	477	1260	480	83.1	498	482	32.0
l04	*502	805	506	64.1	550	509	26.1	*502	1197	504	74.5	517	504	35.9
l05	*457	622	461	68.4	487	460	31.8	457	1357	464	81.3	486	464	34.2
l06	799	968	802	132.7	831	801	61.3	*799	2236	801	189.4	841	802	102.2
l07	749	1001	753	144.6	780	752	66.4	749	2073	752	179.0	774	751	97.7
l08	765	991	766	125.4	788	767	65.8	765	1823	766	214.4	774	766	99.1
l09	853	990	855	132.5	895	859	70.0	853	2321	854	197.5	857	854	111.2
l10	804	962	806	127.8	824	806	69.4	*804	2022	805	204.7	486	805	111.6
l11	*1071	1254	1072	236.9	1093	1073	147.0	*1071	2641	1073	341.6	1079	1073	207.2
l12	936	1211	937	202.4	961	937	157.9	*936	2421	937	281.6	951	940	26.0
l13	*1038	1242	1041	206.9	1046	1039	156.4	*1038	2515	1039	308.4	1052	1040	206.4
l14	*1070	1292	1072	209.5	1086	1071	151.0	1070	2505	1071	285.6	1091	1071	197.0
l15	1089	1655	1092	230.1	1126	1093	60.9	1089	2890	1091	325.3	1096	1091	212.1
l16	*717	1230	725	104.9	835	717	10.1	*717	2561	717	51.0	717	717	2.8
l17	*646	938	646	4.0	898	646	2.3	*646	2247	646	46.1	646	646	2.8
l18	*666	1067	685	22.2	755	647	29.3	*663	2732	663	56.2	663	663	2.8
l19	647	1063	707	100.8	777	725	31.2	*617	2212	617	63.4	648	617	42.5
l20	*756	1258	814	3.9	808	756	3.6	*756	1925	756	34.8	756	756	2.7
l21	808	1430	863	258.1	960	861	92.6	800	3324	820	686.2	844	826	287.8
l22	737	1290	784	233.1	960	790	87.2	733	3154	743	583.6	757	745	267.8
l23	816	1409	863	251.6	961	884	81.1	809	3497	827	623.7	842	826	287.3
l24	775	1321	837	228.9	925	825	93.4	773	3805	787	689.4	817	796	287.1
l25	752	1375	821	233.8	914	823	81.7	751	3562	772	664.5	804	770	286.4
l26	1056	1664	1085	435.2	1148	1086	202.2	1052	4226	1063	1204.8	1073	1058	655.6
l27	1085	1732	1107	397.4	1214	1109	188.5	1084	4418	1094	1271.8	1118	1088	653.3
l28	1075	1605	1092	382.4	1165	1097	190.9	1069	4776	1072	1298.5	1109	1073	655.3
l29	993	1582	1007	424.8	1082	1016	172.0	993	4222	1002	1168.8	1020	995	647.3
l30	1068	1712	1101	408.2	1221	1105	180.6	1068	5659	1074	1295.1	1078	1071	690.8
l31	1520	2139	1528	839.1	1595	1532	552.2	1520	6758	1521	3685.0	1543	1521	1897.7
l32	1657	2171	1665	854.9	1768	1668	556.1	1657	6804	1661	3547.2	1662	1658	1879.7
l33	1497	2062	1500	803.0	1575	1511	604.3	1497	6478	1500	3349.9	1509	1498	1971.5
l34	1535	2310	1538	824.5	1640	1542	562.6	1535	6166	1537	3601.2	1550	1536	2169.4
l35	1549	2504	1553	895.7	1629	1559	549.4	1549	6725	1566	3784.5	1571	1553	2142.3
l36	1016	1727	1052	354.8	1214	1054	98.0	*948	4894	948	1255.5	948	948	22.8
l37	989	2019	1116	375.1	1264	1122	112.3	*986	5785	986	1372.1	993	986	25.5
l38	943	1881	988	359.8	1134	1004	104.8	*943	5181	943	1121.7	943	943	21.1
l39	966	1673	1063	188.3	1169	1041	94.2	*922	5291	922	1272.6	952	922	110.5
l40	955	1795	1018	385.0	1105	1009	108.6	*955	4966	955	1072.2	955	955	21.3

Table 5

	Original data sets	$\lvert A_{ij}^k\rvert$ave	$\lvert A_{ij}^k\rvert$max
erdata		2	3
emdata	edata	$\frac{1}{3}m$	$\frac{3}{5}m$
evdata		$\frac{1}{2}m$	$\frac{4}{5}m$
rrdata		2	3
rmdata	rdata	$\frac{1}{3}m$	$\frac{3}{5}m$
rvdata		$\frac{1}{2}m$	$\frac{4}{5}m$
vrdata		2	3
vmdata	vdata	$\frac{1}{3}m$	$\frac{3}{5}m$
vvdata		$\frac{1}{2}m$	$\frac{4}{5}m$

A move of an operation O_{ij} is tabu if one of the following conditions is fulfilled:

- a tabu-list element exists which contains O_{ij}, the new machine set for O_{ij}, and the new predecessor set of O_{ij},
- a tabu-list element exists which contains O_{ij}, the new machine set for O_{ij}, and the new successor set of O_{ij},
- a tabu-list element exists which contains an operation O_{kl} of the new successor set of O_{ij}, as well as the machine set for O_{kl}, and the new predecessor set of O_{kl},
- a tabu-list element exists which contains an operation O_{kl} of the new predecessor set of O_{ij}, as well as the machine set for O_{kl}, and the new successor set of O_{kl}.

If the best partition is tabu, then the next best partition is chosen. If all partitions are tabu, then O_{ij} and/or the type of the operator is changed. Note that the operators move-before and move-after are only applied to operations O_{ij} which belong to a block.

Other features of the tabu-search procedure are implemented as in the tabu-search algorithm of Hurink et al. (1994).

3.2 Computational results

The multi-purpose machine job-shop problem (MPMJSP) is the special case of the multi-mode job-shop problem

Table 6

Tabu-search e-data						
	erdata		emdata		evdata	
Data	Iter-ave	CPU-ave	Iter-ave	CPU-ave	Iter-ave	CPU-ave
m06	276.0	109.2	149.0	119.1	510.0	146.1
m10	721.0	442.1	396.0	1342.7	892.0	1651.8
m20	960.0	901.7	975.0	372.8	480.0	1291.4
l01-l05	583.0	222.4	557.2	143.6	431.8	266.6
l06-l10	706.2	493.2	685.4	251.9	351.0	706.6
l11-l15	507.6	1056.5	577.6	421.3	246.0	1200.1
l16-l20	659.8	449.5	729.4	1060.2	332.8	1647.3
l21-l25	885.6	926.3	584.0	2880.3	229.8	4594.2
l26-l30	890.8	1599.9	409.4	5720.7	318.6	8930.3
l31-l35	939.0	3952.1	646.6	15075.1	278.2	24495.6
l36-l40	674.6	1496.7	576.0	9881.6	243.2	11299.4
Tabu-search r-data						
	rrdata		rmdata		rvdata	
Data	Iter-ave	CPU-ave	Iter-ave	CPU-ave	Iter-ave	CPU-ave
m06	436.0	138.6	661.0	139.2	203.0	308.0
m10	514.0	604.3	866.0	1327.7	95.0	2497.6
m20	824.0	897.7	565.0	499.3	468.0	1473.3
l01-l05	630.8	289.6	738.0	182.7	581.6	396.3
l06-l10	778.0	639.6	797.4	344.8	512.6	895.6
l11-l15	908.4	1096.4	839.2	617.0	216.4	1799.5
l16-l20	684.6	523.5	912.8	1432.2	168.0	2742.1
l21-l25	653.4	1118.1	694.4	3411.3	489.0	7146.7
l26-l30	887.2	2145.6	899.4	6634.4	259.2	13342.1
l31-l35	845.6	4954.2	830.2	16617.7	492.2	36989.8
l36-l40	827.6	1696.1	692.8	14020.8	368.6	20122.1
Tabu-search v-data						
	vrdata		vmdata		vvdata	
Data	Iter-ave	CPU-ave	Iter-ave	CPU-ave	Iter-ave	CPU-ave
m06	160.0	147.4	389.0	127.6	317.0	299.9
m10	896.0	664.1	940.0	1935.2	198.0	6149.6
m20	923.0	1161.7	528.0	783.3	991.0	2074.8
l01-l05	753.0	277.4	456.2	186.3	761.8	463.1
l06-l10	631.6	731.9	612.4	412.6	530.6	1011.5
l11-l15	867.2	1239.7	704.2	708.3	502.8	1978.4
l16-l20	835.6	739.0	839.4	2396.3	599.6	5250.6
l21-l25	905.8	1755.5	924.6	6064.4	377.8	12518.9
l26-l30	797.0	3609.4	947.0	10070.1	484.2	25458.5
l31-l35	927.8	9461.5	971.8	31178.4	481.0	69776.5
l36-l40	872.8	4191.0	931.8	32271.5	198.8	71068.6

(MMJSP) in which all machine sets A_{ij}^k are one-element sets. Therefore we also used the benchmark problems of Hurink et al. (1994) to test our tabu-search procedures. Furthermore, we extended these benchmark problems to obtain test data for the general MMJSP.

The benchmark problems of Hurink et al. are derived from the job-shop benchmark problems m06, m10, m20 of Fisher and Thompson (1966) and l01–l40 of Adams et al. (1988). The sizes of these problems are listed in Table 1. Here, m and n denote the number of machines and jobs, respectively. For all these problems the number of operations of each job is equal to the corresponding number of machines.

To generate test problems for the MPMJSP Hurink et al. added alternative machines to the operations with certain probabilities. Depending on these probabilities different test data sets edata, rdata, vdata have been created. The characteristics of these sets are shown in Table 2, where $|M_{ij}|$ave denotes the average number of alternative machines and $|M_{ij}|$max is the maximal number of alternative machines.

To find out how the S2-version of our tabu-search procedure performs on instances of the MPMJSP we applied it to these test data. Like Hurink et al. (1994) we defined our tabu-list length to be equal to 30 and limited the number of iterations by 1000. The results are compared with the results of Hurink et al. in Table 3 (edata) and Table 4 (rdata, vdata). These tables contain the following information:

- LB: best known lower bound for the problem instance. The bounds which are due to Jurisch (1992) are marked with an asterisk if they are equal to the optimal C_{max}-values.
- MMJSP: results for the tabu-search presented in this paper.

- MPMJSP: results from Hurink et al. (1994).
- UB-Start: C_{max}-value provided by the start heuristic.
- UB-TS: C_{max}-value provided by the tabu-search.
- CPU: CPU-time in seconds.

The small computation times are due to the fact that the procedure stopped when all neighbours were tabu.

We implemented the S2-version of our tabu-search procedure on a SUN-SPARC Station 10/40 using the programming language C. Hurink et al. used a slower SUN 4/20 workstation. The average speed-up factor between these two machines is 3.2.

Note that Hurink et al. used a better start heuristic than ours. Nevertheless, our tabu-search results are comparable with those of Hurink et al.

To create MMJSP test problems we randomly added other machine sets to the one-element machine sets of the test data in the sets edata, rdata, and vdata. Table 5 shows the different average sizes $|A_{ij}^k|$ave and maximal sizes $|A_{ij}^k|$max of the machine sets created in connection with edata, rdata, and vdata.

We tested the S1-version and S2-version of the tabu-search procedure on these 9 data sets. The C_{max}-values calculated by both versions are nearly identical. However, the S1-version which considers all partitions of the set I was 15.7% slower than the other version. Therefore, in Table 6 we only present the test results of the S2-version for the 9 different test sets described in Table 5. In this table the results for the test problems of the same size are summarized by the average value. Table 6 contains the following information:

- Iter-ave: The average number of iterations after which the best C_{max}-value (of 1000 iterations) was found.
- CPU: The average CPU-time in seconds.

The other figures for these test problems can be accessed via
ftp://ftp.mathematik.Uni-Osnabrueck.DE/pub/osm/preprints

The main results can be summarized as follows:

- The tabu-search procedure improves the C_{max}-values provided by the start heuristic considerably.
- Except for problem instance 110 of emdata and some problems for the MPMJSP the tabu-search never terminated before reaching the maximal iteration count 1000.
- In many cases Iter-ave is close to 1000. Thus, it seems that the C_{max}-value can be further improved by increasing the bound on the maximal number of iterations.
- The computational times can be high. For example, the CPU-time for problem 138 of vvdata was 21.3 hours. Generally, the CPU-time increases with the number of operations as well as with the number and size of alternative machine sets. It decreases with the number of machines.
- No good lower bounds are available for the MMJSP. Thus, we cannot estimate the quality of the solutions.

To test the influence of the number of iterations on the quality of the C_{max}-value we increased the maximum number of iterations to 2000 when running the test problems m06, m10, m20, and 101 - 120 of all problem sets. For 57% of these instances the C_{max}-value improved. The average improvement was 1.39%. The CPU-time doubled. Again, the tabu-search never stopped before reaching iteration 2000. Therefore, when testing problems mt06, 103, 104, 107, 108, 113, 114, 119, and 120 we increased the maximum number of iterations to 5000. For 52% of these instances there was another improvement of the C_{max}-value which, on average, decreased by an additional 1.35%.

4 Concluding remarks

A tabu-search algorithm for the multi-mode job-shop scheduling problem has been introduced and applied to a large number of test problems. A comparison with a tabu-search procedure which has been especially designed for MPM job-shop problems shows that for this special case of the MMJSP our algorithm provides very good results.

For the general case the tabu-search algorithms provide new benchmark results. A challenging task is to provide good lower bounds and/or a branch-and-bound procedure for the multi-mode job-shop scheduling problem.

References

1. Adams J, Balas E, Zawack D (1988) The shifting bottleneck procedure for job-shop scheduling. Management Science 34:391-401
2. Brucker P, Krämer A (1995) Shop scheduling problems with multiprocessor tasks on dedicated processors. Annals of Operations Research 57:13-27
3. Brucker P, Schlie R (1990) Job-shop scheduling with multi-purpose machines. Computing 45:369-375
4. Dauzère-Pérès S, Paulli J (1995) A global tabu search procedure for the general multiprocessor job-shop scheduling problem. Department of Operations Research, University of Aarhus, No 5/95
5. Fisher H, Thompson GL (1963) Probabilistic learning combinations of local job-shop scheduling rules. In: Muth JF, Thompson GL (eds) Industrial scheduling, pp. 225-251. Englewood Cliffs: Prentice Hall
6. Hurink J, Jurisch B, Thole M (1994) Tabu search for the job-shop scheduling problem with multi-purpose machines. OR-Spektrum 15:205-215
7. Jurisch B (1992) Scheduling jobs in shops with multi-purpose machines. PhD thesis, Department of Mathematics/Informatics, Universität Osnabrück
8. Krämer A (1995) Scheduling multiprocessor tasks on dedicated processors. PhD thesis, Department of Mathematics/Informatics, Universität Osnabrück
9. Roy B, Sussmann B (1964) Les problèmes d'ordonnancement avec constraintes disjonctives. Note DS no 9 bis, SEMA, Paris
10. Sprecher A, Drexl A (1996a) Solving multi-mode resource-constrained project scheduling problems by a simple, general and powerful sequencing algorithm. Part I: Theory. Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No 385
11. Sprecher A, Drexl A (1996b) Solving multi-mode resource-constrained project scheduling problems by a simple, general and powerful sequencing algorithm. Part II: Computation. Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No 386