ELSEVIER

Discrete Applied Mathematics 76 (1997) 43–59

DISCRETE APPLIED MATHEMATICS

A branch & bound algorithm for the open-shop problem

Peter Brucker *,[1], Johann Hurink, Bernd Jurisch [1], Birgit Wöstmann

Fachbereich Mathematik/Informatik, Universität Osnabrück, Post fach 4469, 49069 Osnabruck, Germany

Received 23 January 1995; revised 16 October 1995

Abstract

A fast branch & bound method for the open-shop problem based on a disjunctive graph formulation of the problem is developed. Computational results show that the method yields excellent results. Some benchmark problems from the literature were solved to optimality for the first time.

Keywords: Open-shop scheduling; Branch & bound method

1. Introduction

The open-shop problem may be formulated as follows. We have m machines $M_1, \ldots, M_m$ and n jobs $J_1, \ldots, J_n$. Each job J_i consists of m operations O_{ij} $(j = 1, \ldots, m)$ where O_{ij} has to be processed on machine M_j for p_{ij} time units without preemption. Furthermore, we assume that each machine can process at most one operation at a time and each job can be processed by at most one machine at a time. For each machine the order in which the jobs are processed on the machine (machine orders) and for each job the order in which this job is processed by the machines (job orders) can be chosen arbitrarily. The problem is to determine a feasible combination of the machine and job orders which minimizes a certain objective function.

If the makespan has to be minimized, we have polynomial algorithms for the case $m = 2$ or $n = 2$, and for the open-shop problem with arbitrary number of jobs and machines and allowed preemptions [7]. Moreover, the two-machine problem is solvable in polynomial time even under the consideration of one additional resource [8]. However, most problems with other regular criteria are NP-hard (cf. [9]). For a large class of open-shop problems with $p_{ij} = 1$ for all operations O_{ij} polynomial algorithms have been developed [2, 10].

In this paper we present a branch & bound algorithm for the general open-shop problem with $C_{\max}$-objective $O \,||\, C_{\max}$. The algorithm is based on a disjunctive graph

* Corresponding author. E-mail: Peter.Brucker@mathematik. Uni-Osnabrueck.DE

[1] Supported by Deutsche Forschungsgemeinschaft (Project JoPTAG).

PII S0166-218X(96)00116-3

formulation for open-shop problems in which precedence constraints between operations of the same job and between operations to be processed on the same machine are successively added. Section 2 contains a description of the disjunctive graph model and the algorithm. In the last section computational results are reported. Furthermore, a method for creating hard open-shop problems is introduced. Such a method has been developed because for the benchmark problems in the literature (cf. [14]) it is generally easy to get upper and lower bound which are very close to each other.

2. A branch & bound algorithm

In this section we will develop a branch & bound algorithm based on a disjunctive graph model for the open-shop problem. This disjunctive graph model will be presented in Section 2.1. The general concepts of the branch & bound method are described in Section 2.2. The branching scheme, lower bounds, heuristics for calculating upper bounds, and applications of immediate selection will be discussed in subsequent Sections 2.3–2.6.

2.1. The disjunctive graph model

The idea of the branch & bound method is to construct the machine orders and job orders step by step by introducing precedence relations either between operations of the same job or between operations to be processed on the same machine. This leads to the concept of a disjunctive graph $G = (V, D_M \cup D_J)$, where

- the set of nodes V is the set of all operations and each node is labeled by the processing time of the corresponding operation;
- D_M is the set of *machine disjunctions* consisting of undirected arcs (or edges) connecting all pairs of operations to be processed on the same machine;
- D_J is the set of *job disjunctions* consisting of edges connecting all pairs of operations of the same job.

目的：确定一台机器上操作的顺序以及同一个job各操作的顺序

In Fig. 1(a) the disjunctive graph of an open-shop problem with $n = m = 3$ is shown. The basic scheduling decision is to define an order between all those operations which have to be processed on the same machine and those of the same job. In the disjunctive graph model this is done by turning undirected (disjunctive) arcs into directed ones. A set S of these "fixed" disjunctions is called *selection*. Obviously, a selection S defines a feasible schedule if and only if

- all disjunctive arcs are fixed and
- the resulting graph $G(S) = (V, S)$ is acyclic.

In this case we call the set S a *complete selection*.

A complete selection provides a feasible schedule by defining the completion time of each operation to be equal to the length of the longest path in $G(S)$ ending at that operation. Here, the length of a path is equal to the sum of labels of all vertices on the path. The $C_{\max}$-value $C_{\max}(S)$ corresponding to the schedule is equal to the longest

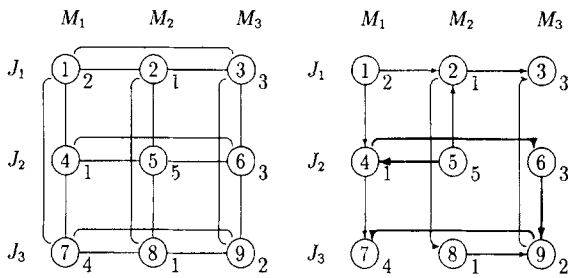

Fig. 1.

(*critical*) path in $G(S)$. In Fig. 1(b) a complete selection and a corresponding critical path are shown (for the sake of clearness, transitive arcs are omitted).

To solve the open-shop problem we have to find a complete selection S such that the critical path in $G(S)$ has minimal length.

2.2. Basic concepts of the branch & bound algorithm

In this section we will give a short description of the branch & bound algorithm. It will be represented by a search tree: Every search tree node r corresponds to a graph $G(F_r) = (V, F_r)$. F_r denotes the set of fixed disjunctive arcs in node r. Node r represents all solutions $Y(r)$ respecting the partial order defined by F_r. Branching is done by dividing $Y(r)$ into disjoint subsets $Y(s_1), \ldots, Y(s_q)$ for some q. Each $Y(s_i)$ is the set of solutions of a problem with a graph $G(F_{s_i})$ where $F_r \subset F_{s_i}$, i.e. $G(F_{s_i})$ is derived from $G(F_r)$ by fixing additional disjunctions. This way of branching creates immediate successors $s_1, \ldots, s_q$ of node r in the branching tree which are treated recursively. For each node r a value $\mathrm{LB}(r)$ bounding the objective values of all solutions in $Y(r)$ from below is calculated. We set $\mathrm{LB}(r) = \infty$ if the corresponding graph $G(F_r)$ contains a cycle. Furthermore, we have an upper bound UB for the solution value of the original problem. UB is updated each time when a new feasible solution is found which improves UB.

To specify the branch & bound procedure in more detail we have to introduce a branching scheme and to discuss methods for calculating bounds. The following sections are devoted to these issues.

2.3. A branching scheme

Brucker et al. [4] have introduced a branching scheme for the job-shop problem that is based on a feasible schedule which corresponds to a complete selection S. We have applied the underlying idea to the open shop problem.

Let P be a critical path in the graph $G(S)$ corresponding to a complete selection S and let $L(S)$ be the length of P. A sequence $u_1,\ldots,u_l$ of $l \geqslant 2$ successive nodes in P is called a *block* on P in $G(S)$ if the following properties are satisfied:

(a) All u_i are either processed on the same machine or belong to the same job.

(b) Extending the sequence from either side results in the violation of (a).

The branching scheme is based on

Theorem 1. *Let S be a complete selection corresponding to some solution of the open-shop problem and let P be a critical path in $G(S)$. If there exists another complete selection S' such that $L(S') < L(S)$, then there is a block $u_1,\ldots,u_l$ on P and an operation u_i in it such that either u_i is before u_1 in S' or u_i is after u_l in S'.*

指的是block

We omit the proof which is similar to proof of a corresponding theorem in Brucker et al. [4].

Now we consider a node r of the search tree and a solution $y \in Y(r)$. This solution y is calculated using some heuristics. Let S be the complete selection corresponding to y. A critical path in $G(S)$ defines blocks $B_1,\ldots,B_k$. For block

$$B_j\colon u_1^j,\ldots,u_{m_j}^j$$

the operations in

$$E_j^{\mathrm{B}} := B_j\backslash\{u_1^j\} \quad \text{and} \quad E_j^{\mathrm{A}} := B_j\backslash\{u_{m_j}^j\}$$

are called *before-candidates* and *after-candidates*, respectively.

According to Theorem 1 in a complete selection S' which improves S, at least one before-candidate l in some block B_j must be a predecessor of all other vertices in B_j or at least one after-candidate l in some block B_j must be a successor of all other vertices in B_j.

To describe the branching scheme we define the following sets of arcs:

$$F_j := \{u_1^j \to l \mid l = u_2^j,\ldots,u_{m_j}^j\}$$
$$L_j := \{l \to u_{m_j}^j \mid l = u_1^j,\ldots,u_{m_j-1}^j\}$$

for $j = 1,\ldots,k$ and consider a permutation $R_1, R_2, \ldots, R_{2k}$ of all sets F_j and L_j.

We get the successor sets $Y(s_1),\ldots,Y(s_q)$ of node r by adding to $Y(r)$ sets of arcs constructed in the following way:

- For each before-candidate $l \in E_j^{\mathrm{B}}$ $(j = 1,\ldots,k)$ find the index m with $R_m = F_j$ and define

$$S_l^{\mathrm{B}} := R_1 \cup R_2 \cup \cdots \cup R_{m-1} \cup \{l \to i \mid i \in B_j\backslash\{l\}\}.$$

- For each after-candidate $l \in E_j^{\mathrm{A}}$ $(j = 1,\ldots,k)$ find the index m with $R_m = L_j$ and define

$$S_l^{\mathrm{A}} := R_1 \cup R_2 \cup \cdots \cup R_{m-1} \cup \{i \to l \mid i \in B_j\backslash\{l\}\}.$$

It can be shown that such sets define a branching with the property that the corresponding solution sets are pairwise disjoint.

So far we have not specified how to choose the permutation $R_1,\ldots,R_{2k}$ of the sets F_j and L_j $(j=1,\ldots,k)$. Our objective is to fix a large number of disjunctive arcs as early as possible. So, we arrange the sets F_i and L_i $(i=1,\ldots,k)$ according to non-increasing cardinality of the corresponding blocks. In addition we always take the set L_i as an immediate successor of the set F_i. More precisely, we choose

$$R_{2i-1} := F_{\pi(i)}, \qquad R_{2i} := L_{\pi(i)} \quad (i=1,\ldots,k)$$

with a permutation π of $1,\ldots,k$ such that $|B_{\pi(i)}| \geqslant |B_{\pi(j)}|$ if $i<j$.

Now we are able to formulate a more specific recursive branch & bound procedure based on the branching rule introduced in this section.

Procedure Branch & Bound (r)

```
BEGIN
  Calculate a solution y corresponding to a selection S ∈ Y(r) using heuristics;
  IF C_max(S) < UB THEN UB := C_max(S);
  Calculate a critical path P;
  Calculate the blocks of P;
  Calculate the sets E_j^B and E_j^A;
  FOR ALL operations i ∈ E_j^μ with j = 1,...,k and μ = A,B DO
      Fix disjunctions for the corresponding successor s;
      Calculate a lower LB(s) for node s;
      IF LB(s) < UB THEN Branch & Bound (s)
  END
END
```

用启发式得到一个可行解

更新上界

算法在某个节点上停止的条件

Notice that the handling of the search tree node stops if

- the lower bound LB(s) is greater than or equal to UB (this is for instance the case if the corresponding disjunctive graph has cycles, i.e. LB(s) $=\infty$) or
- the sets E_j^B and E_j^A are empty for all blocks B_j.

2.4. Heads and tails

With each operation i we associate a head and a tail. Calculations of heads and tails are based on the fixed disjunctive arcs. Thus, they depend on the specific search tree node r.

操作i最早可能开始时间

A *head* r_i of operation i is an earliest possible starting time of i. A *tail* q_i of operation i is a lower bound for the amount of time between the finish time of operation i and the optimal makespan.

在图G(Fr)中到i的最长path

A simple way to get a head r_i is to calculate the length l_i of a longest path ending in i in the disjunctive graph $G(F_r)$. Then $r_i = l_i - p_i$ is a head, where p_i is the

tail：操作i的结束时间和optimal makespan间差值的下界，也就是最晚完成时间

processing time of the operation represented by node i. Similarly, for each operation a tail could be defined by $q_i = \bar{l}_i - p_i$ where $\bar{l}_i$ is the length of a longest path starting in i.

从i开始的最长path的长度

To obtain good lower bounds it is desirable to have large heads and tails. For this purpose we used the following more sophisticated procedures for calculating heads and tails.

Let Q_i (Q_i') be the set of predecessors (successors) of an operation i which belong to the same job as i. Similarly, let R_i (R_i') be the set of predecessor (successors) of operation i which are to be processed on the same machine as i. Clearly, for each subset $J \subseteq Q_i$ ($J \subseteq R_i$) operation i cannot start before time $\min_{j\in J} r_j + \sum_{j\in J} p_j$. Thus, we have the following recursion for calculating heads:

计算head和tail

$$r_i = \max_{J\subseteq Q_i \text{ or } J\subseteq R_i} \left\{ \min_{j\in J} r_j + \sum_{j\in J} p_j \right\}, \tag{2.1}$$

where $r_i = 0$ if i has no predecessors.

Similarly, for tails we have the recursion

$$q_i = \max_{J\subseteq Q_i' \text{ or } J\subseteq R_i'} \left\{ \sum_{j\in J} p_j + \min_{j\in J} q_j \right\}, \tag{2.2}$$

where $q_i = 0$ if i has no successor. Both values (2.1) and (2.2) can be calculated in $O(k \log k)$, where k is defined as $\max\{|Q_i|, |R_i|\}$, or $\max\{|Q_i'|, |R_i'|\}$, respectively, using the algorithm of Carlier [5] to calculate the Jackson's Preemptive Schedule for problem $1|r_i|L_{\max}$.

Heads and tails are important for calculating lower and upper bounds. They are also used in connection with immediate selection.

2.5. Immediate selection

The main idea of the branching scheme was to fix many disjunctions early during the branching process. By applying immediate selection disjunctions can be fixed without branching. This can be accomplished by considering a set I of operations either belonging to the same job or to be processed on the same machine. Under the assumption that a disjunction is fixed in one direction, say $i \to j$, a simple lower bound for the set I is derived. The calculation of the lower bound uses the heads and tails of the operations in I. If this lower bound is greater than or equal to the current upper bound then in all solutions improving the current best solution j must be processed before i. Therefore, the relation $j \to i$ may be fixed. Immediate selection was first introduced by Carlier & Pinson [6]. Further concepts for immediate selection can be found in Brucker et al. [3].

2.6. Lower bounds

Let r be a search tree node with a set F_r of fixed disjunctive arcs. Based on F_r for each operation i a head r_i and a tail q_i is given. A lower bound LB(s) is calculated for each successor s of r. If this value is greater than or equal to the current upper bound UB then an inspection of s is not necessary. However, the calculation of lower bounds will be done not only at one place of the algorithm (as indicated by the algorithmic description in Section 2.3). Whenever we change data that have influence on the lower bounds we recalculated the corresponding lower bounds. More precisely, we calculate lower bounds at the following places in the algorithm:

- Lower bound calculation after the computation of the sets E_l^B and E_l^A: If operation i should be moved before block B_k, all disjunctive arcs $\{i \to j \mid j \in B_k \backslash \{i\}\}$ are fixed. Thus, the value

$$r_i + p_i + \max\left\{\max_{j\in B_k\backslash\{i\}}(p_j+q_j);\ \sum_{j\in B_k\backslash\{i\}} p_j + \min_{j\in B_k\backslash\{i\}} q_j\right\}$$

  is a simple lower bound for the search tree node s.
  Similarly, the value

$$\max\left\{\max_{j\in B_k\backslash\{i\}}(r_j+p_j);\ \min_{j\in B_k\backslash\{i\}} r_j + \sum_{j\in B_k\backslash\{i\}} p_j\right\} + p_i + q_i$$

  is a lower bound for the node s if i should be moved after block B_k.
- Lower bound calculation after the computation of heads and tails: If the value $r_i + p_i + q_i$ of an operation i is not smaller than the current upper bound, then the node does not need to be inspected.
- Lower bound calculation after the computation of heads and tails: We may associate with each machine a corresponding head–tail problem: schedule all operations on this machine in such a way that release times given by the heads of the operations are respected and the value $\max\{C_j+q_j\}$ is minimized, where C_j denotes the completion time of the operation j.
  If we allow preemption this problem can be solved efficiently by constructing Jackson's preemptive schedule (cf. [4]). The corresponding solution value is a lower bound for the corresponding node s. In the same way we may calculate a lower bound by considering all operations of a specific job. We take the maximum of all these $n+m$ bounds.

2.7. Calculation of heuristic solutions

The branching scheme we used is based on the calculation of a heuristic solution that respects the disjunctions fixed in the actual search tree node. Besides heuristic solutions based on priority rules we also experimented with more sophisticated heuristics based

on matching algorithms. We calculate the heuristic solutions by iteratively scheduling the jobs from left to right. In each step we either schedule one operation using a priority rule or we schedule a set of operations using a matching algorithm. More precisely, in each step we firstly calculate the set C of operations for which all predecessors are already scheduled (initially this set contains all operations that have no predecessors for the given fixed disjunctions). Afterwards, we either determine by a priority rule one operation of C or we determine by a matching algorithm a subset A of C such that the operations of A are not in conflict. Finally, the determined operation(s) is (are) scheduled as early as possible.

确定一个操作或者是C的一个子集--相应的操作尽早调度

The priority rule used is an adaptation of a priority rule used by Brucker et al. [4] in connection with the job-shop problem. Sievers [13] has shown that for the job-shop problem this rule was superior to many other priority rules. However, computational experiences have shown, that in all cases the matching heuristics provide better results than the priority heuristic.

For the calculation of the subset A we construct a bipartite graph $BG = (J, M, O)$ as follows. $J = \{J_1, \ldots, J_n\}$ is the set of all jobs and $M = \{M_1, \ldots, M_m\}$ is the set of all machines. Furthermore, $(J_i, M_j) \in O$ if and only if all predecessors of operation O_{ij} are already scheduled. For this graph we calculate a matching A with maximal cardinality and add the operations corresponding with A to the current schedule. Usually, there are many matchings of maximal cardinality. Therefore we add a secondary objective. A possible secondary objective is to find a matching A' of maximal cardinality which minimizes $\sum_{(J_i,M_j)\in A'} p_{ij}$. This and other resulting matching problems which have been used are listed below.

- sum-matching/minimization:

$$\min \left\{ \sum_{(J_i,M_j)\in A} p_{ij} \mid A \text{ is a matching of maximal cardinality} \right\}$$

- sum-matching/maximization:

$$\max \left\{ \sum_{(J_i,M_j)\in A} p_{ij} \mid A \text{ is a matching of maximal cardinality} \right\}$$

- bottleneck-matching/minimization:

$$\min \left\{ \max_{(J_i,M_j)\in A} p_{ij} \mid A \text{ is a matching of maximal cardinality} \right\}$$

- bottleneck-matching/maximization:

$$\max \left\{ \min_{(J_i,M_j)\in A} p_{ij} \mid A \text{ is a matching of maximal cardinality} \right\}$$

- modified bottleneck-matching/minimization:

$$\min \left\{ \max_{(J_i,M_j)\in A} \{r_{ij} + p_{ij}\} \mid A \text{ is a matching of maximal cardinality} \right\}$$

- modified bottleneck-matching/maximization:

$$\max\left\{\min_{(J_i,M_j)\in A}\{r_{ij}+p_{ij}\}\,|\,A \text{ is a matching of maximal cardinality}\right\}$$

Algorithms to calculate such matchings can be found for the sum-criteria in [15] and for the bottleneck-criteria in [12].

3. Computational results

We implemented 6 branch & bound algorithms, which differ in the chosen heuristic (see Section 2.7) on a Sun 4/20 Workstation using the programming language *C*. We tested the algorithms on benchmark problems given in the literature and on some slightly modified versions of these benchmark problems (see Section 3.1). Since these instances turned out to be 'easy', we generated new (harder) instances of open-shop problems. The generation of these instances and the achieved computational results are presented in Section 3.2.

3.1. Benchmark problems

For the first series of computational tests we used benchmark problems from Taillard [14] and some modifications of these instances. In the following the instances tai$x$-y, $x\in\{4,5,7,10\}$, $y\in\{1,\ldots,10\}$, are from Taillard [14] and the instances tai9-y (tai8-y) are obtained from the instances tai10-y by removing the last (and the second-last) jobs and machines, i.e. by removing the last (and the second-last) rows and columns of the processing time matrix of the instances tai10-y, $y\in\{1,\ldots,10\}$. The notation is chosen in such a way, that a problem tai$x$-y is of dimension $n=m=x$ (m denotes the number of jobs, n denotes the number of machines).

In Table 1 we present the results for the 3 most successful versions of the branch & bound algorithm. The table contains the following information:

- (n,m): size of the instances.
- LB: the trivial lower bound

$$\max\left(\left\{\sum_{i=1}^{n}p_{ij}\,\middle|\,j=1,\ldots,m\right\}\cup\left\{\sum_{j=1}^{m}p_{ij}\,\middle|\,i=1,\ldots,n\right\}\right).$$

  If this value is marked with an asterisk, it is equal to the optimal solution of the problem.
- opt: The optimal solution value. We left blank spaces if no solution could be proven to be optimal within the time limit of 50 hours.
- B&B$_i$: results for the branch & bound algorithm based on the following heuristics:
  - B&B$_1$: sum-matching/minimization.
  - B&B$_2$: sum-matching/maximization.

Table 1
Number of search tree nodes and CPU-time

Data	LB	Opt	B&B$_1$ Nodes	CPU	B&B$_2$ Nodes	CPU	B&B$_3$ Nodes	CPU
tai4-1	186	193	30	0.3	25	0.3	18	0.2
tai4-2	229	236	46	0.6	34	0.4	32	0.4
tai4-3	262	271	27	0.3	27	0.3	32	0.3
tai4-4	245	250	38	0.5	29	0.4	40	0.5
tai4-5	287	295	40	0.5	36	0.4	35	0.4
tai4-6	185	189	28	0.4	27	0.3	26	0.3
tai4-7	197	201	32	0.4	27	0.3	23	0.3
tai4-8	212	217	21	0.2	19	0.2	16	0.2
tai4-9	258	261	11	0.1	14	0.1	13	0.1
tai4-10	213	217	50	0.6	23	0.3	27	0.3
tai5-1	295	300	342	8.0	286	6.5	315	7.6
tai5-2	255	262	254	6.3	174	4.0	231	5.5
tai5-3	321	323	660	16.5	883	21.4	851	21.6
tai5-4	306	310	441	10.8	244	5.6	348	8.5
tai5-5	321	326	895	25.8	551	13.5	1256	30.6
tai5-6	307	312	494	14.6	520	12.3	372	8.6
tai5-7	298	303	508	15.1	398	9.7	402	9.5
tai5-8	292	300	743	23.2	541	13.4	762	18.8
tai5-9	349	353	449	13.5	438	10.7	911	22.4
tai5-10	321	326	626	18.4	671	16.9	803	20.0
tai7-1	* 435	435	840	57.6	1183	85.8	203	11.6
tai7-2	* 443	443	1588	112.8	1153	77.1	506	32.1
tai7-3	* 468	468	7943	667.0	4542	342.1	2718	211.2
tai7-4	* 463	463	13629	1082.7	4491	327.2	128	7.6
tai7-5	* 416	416	520	37.6	1985	161.0	2652	202.9
tai7-6	* 451	451	48636	3801.0	92429	6969.8	57958	4499.6
tai7-7	* 422	422	51732	4074.9	2268	162.5	6520	497.6
tai7-8	* 424	424	1896	148.2	1377	91.9	2091	154.9
tai7-9	* 458	458	42044	3389.2	234	15.8	35	1.7
tai7-10	* 398	398	4702	368.7	213	12.3	585	39.6
tai8-1	* 557	557	(569)	–T–	(565)	–T–	157	11.2
tai8-2	* 544	544	189	12.8	110	7.9	(546)	–T–
tai8-3	* 503	503	20323	1956.1	175	11.8	475	32.7
tai8-4	* 462	462	167	12.3	299	20.0	220	15.9
tai8-5	* 525	525	17186	2060.4	295	24.8	745	73.3
tai8-6	* 422	422	20042	1931.7	9862	1111.0	601	56.1
tai8-7	* 500	500	(502)	–T–	186	12.5	134	9.4
tai8-8	* 525	525	74	4.2	63	4.0	(536)	–T–
tai8-9	* 503	503	1	0.1	50	2.9	100	6.8
tai8-10	* 512	512	(514)	–T–	(515)	–T–	392	28.3
tai9-1	* 596	596	530479	96295.5	(601)	–T–	623276	108049.0
tai9-2	* 567	567	(568)	–T–	38406	4916.4	617	62.3
tai9-3	* 574	574	(587)	–T–	2922	474.8	452	46.1
tai9-4	* 518	518	227747	44503.7	22427	3934.6	100437	17330.2
tai9-5	* 609	609	22820	3442.4	3861	607.3	338810	47449.7

Table 1 (Contd.)

tai9-6	* 453	453	83022	15831.5	(454)	–T–	26218	4907.1
tai9-7	* 540	540	(544)	–T–	8111	1270.9	12986	2165.5
tai9-8	* 584	584	88	7.4	194	18.0	393	38.6
tai9-9	* 533	533	(538)	–T–	(538)	–T–	293188	52630.1
tai9-10	* 572	572	(580)	–T–	(579)	–T–	2558	336.9
tai10-1	637		(666)	–T–	(658)	–T–	(648)	–T–
tai10-2	* 588	588	44332	10671.5	(595)	–T–	(591)	–T–
tai10-3	598		(655)	–T–	(609)	–T–	(603)	–T–
tai10-4	* 577	577	163671	40149.4	766842	182096.7	1233	236.7
tai10-5	* 640	640	(658)	–T–	(672)	–T–	742211	176247.7
tai10-6	* 538	538	(565)	–T–	170972	41155.9	(591)	–T–
tai10-7	616		(658)	–T–	(632)	–T–	(670)	–T–
tai10-8	* 595	595	(626)	–T–	(650)	–T–	(621)	–T–
tai10-9	* 595	595	97981	24957.0	9794	2494.7	79597	17510.9
tai10-10	596		(648)	–T–	(613)	–T–	(623)	–T–

Table 2
Average number of search tree nodes and CPU-time

(n,m)		$B\&B_1$	$B\&B_2$	$B\&B_3$
(4,4)	Nodes	33	27	27
	CPU	0.4	0.3	0.3
(5,5)	Nodes	542	471	626
	CPU	15.2	11.4	15.3
(7,7)	Nodes	17353	10988	7340
	CPU	1374.0	824.6	565.9
(8,8)	Nodes	8284	1380	353
	CPU	854.0	149.4	29.2
(9,9)	Nodes	172832	12654	139893
	CPU	32016.1	1870.3	23301.0
(10,10)	Nodes	101995	315869	274347
	CPU	25259.3	75249.1	64665.1

– $B\&B_3$: modified bottleneck-matching/minimization.

- nodes: the number of search tree nodes. If the program has been terminated after reaching the time limit, this column contains (in parenthesis) the value of the best known solution.
- CPU: the CPU-time in seconds. If the program reached the time limit, this column contains a "–T– " (time limit of 50 hours).

In Table 2 we give the average number of search tree nodes and the average CPU-time (in seconds) for the algorithms and different problem sizes. We do not take into consideration the cases in which an algorithm does not terminate within the given time limit.

The results can be summarized as follows.

- For the smaller instances ($n = m \leqslant 7$) all versions of the branch and bound algorithm find the optimal solutions. $B\&B_1$ gives in average the worst results. None of the other two algorithms can be stated as better than the other one, and the performance of each algorithm strictly depends on the instance. Except for the instance tai7-6, both $B\&B_2$ and $B\&B_3$ terminate within 9 minutes.
- For $n = m = 8, 9$, there is always one algorithm which terminates within the given time limit of 50 hours, but no algorithm terminates for all the instances. In the case of termination, the maximum running time is at most 35 minutes for $n = m = 8$ but often several hours for $n = m = 9$.
- For some instances of size $n = m = 10$, not even one algorithm terminates within the given time limit. Mostly, the running times are several hours if the algorithms terminate.
- The two benchmark problems tai10-5 and tai10-8 were solved to optimality for the first time. The latter problem was solved by the branch & bound algorithm based on the heuristic 'bottleneck matching/minimization' that is not included in Table 1 (see [16]).

Next, we will compare our results with a tabu search heuristic given in [14] and a heuristic based on insertion techniques given in [1]. The results are summarized in Table 3, which contains the following information:

- data: the problem.
- LB: the trivial lower bound. If this value is marked with an asterisk, it is equal to the optimal solution of the problem.
- opt: The optimal solution. If no solution could be proven to be optimal within the time limit of 50 hours, the best value found by one of the branch & and bound algorithms is given in parenthesis.
- $UB_{Taillard}$: The best solution given by Taillard [14].
- $UB_{Bräsel}$: The best solution given by Bräsel et al. [1].
- CPU: The CPU-time (in seconds) of the best branch & and bound algorithm. If this entry contains a "–T–", no algorithm terminated.

Since the codes of the heuristics of Taillard and Bräsel et al. were not available to us, we had to restrict the comparison to the benchmark instances given by Taillard [14]. The results can be summarized as follows:

- For $n = m \leqslant 7$, our branch and bound algorithm finds the optimal solution in all but two cases within one minute. For the remaining two problems the computational times are 3, or 20 min, respectively. These times may be considered as acceptable and therefore our branch & bound method may be used to solve instances of this dimension and type.
- For $n = m = 10$, the results of our branch and bound algorithms are mostly as good as the result of the best heuristic, and in four cases our algorithm outperforms the remaining ones. Nevertheless, sometimes our algorithm needs a considerable amount of computational time to obtain good results. In two cases, our branch and bound algorithm is worse than the best heuristic.

Table 3
Comparison of different solution methods

Data	LB	Opt	$UB_{Taillard}$	$UB_{Bräsel}$	CPU
tai4-1	186	193	193	195	0.2
tai4-2	229	236	236	244	0.3
tai4-3	262	271	272	271	0.3
tai4-4	245	250	257	250	0.4
tai4-5	287	295	295	295	0.3
tai4-6	185	189	189	189	0.3
tai4-7	197	201	203	201	0.3
tai4-8	212	217	217	217	0.1
tai4-9	258	261	271	261	0.1
tai4-10	213	217	225	217	0.3
tai5-1	295	300	300	310	6.5
tai5-2	255	262	262	265	4.0
tai5-3	321	323	328	339	16.5
tai5-4	306	310	310	325	5.6
tai5-5	321	326	329	343	13.5
tai5-6	307	312	312	325	8.6
tai5-7	298	303	305	310	9.7
tai5-8	292	300	300	307	13.2
tai5-9	349	353	353	364	10.7
tai5-10	321	326	326	341	14.7
tai7-1	* 435	435	438	442	11.6
tai7-2	* 443	443	449	461	33.3
tai7-3	* 468	468	479	482	63.8
tai7-4	* 463	463	467	473	7.6
tai7-5	* 416	416	419	426	14.5
tai7-6	* 451	451	460	469	1209.4
tai7-7	* 422	422	435	440	162.5
tai7-8	* 424	424	426	431	37.4
tai7-9	* 458	458	460	461	1.7
tai7-10	* 398	398	400	410	12.3
tai10-1	637	(640)	652	645	–T–
tai10-2	* 588	588	596	588	757.9
tai10-3	598	(603)	617	611	–T–
tai10-4	* 577	577	581	577	236.7
tai10-5	* 640	640	657	641	176247.7
tai10-6	* 538	538	545	538	41155.9
tai10-7	616	(629)	623	625	–T–
tai10-8	* 595	595	606	596	76923.5
tai10-9	* 595	595	606	595	2494.7
tai10-10	596	(613)	604	602	–T–

3.2. *Hard instances*

At the first sight it is surprising that the heuristics give such good results for the problems of dimension 7×7 and 10×10 and that the optimal value of these problems is almost always equal to the trivial lower bound LB. But if one starts to analyze the

Table 4
Measures for the hardness of an instance

Data	LB	MIN	DIFF	WORKLOAD
tai7-1	435	249	0.572	0.843
tai7-2	443	225	0.508	0.859
tai7-3	468	351	0.750	0.903
tai7-4	463	271	0.585	0.862
tai7-5	416	283	0.680	0.870
tai7-6	451	311	0.690	0.896
tai7-7	422	309	0.732	0.905
tai7-8	424	255	0.601	0.842
tai7-9	458	301	0.657	0.856
tai7-10	398	250	0.628	0.877
tai10-1	637	353	0.554	0.861
tai10-2	588	326	0.554	0.834
tai10-3	598	326	0.545	0.850
tai10-4	577	312	0.541	0.828
tai10-5	640	281	0.439	0.834
tai10-6	538	368	0.684	0.857
tai10-7	616	376	0.610	0.838
tai10-8	595	250	0.420	0.823
tai10-9	595	354	0.595	0.846
tai10-10	596	331	0.555	0.834

instances in more detail it comes out that these randomly generated instances are 'easy' instances. In Table 4 we give some values for the considered instances that can be used to measure the 'hardness' of instances. The table contains the following information:

- Let P_{J_i} denote the sum of processing times of the operations belonging to job J_i (i.e. $P_{J_i} = \sum_{j=1}^{m} p_{ij}$), $i = 1, \ldots, n$ and let P_{M_j} denote the sum of processing times of the operations which have to be processed on machine M_j (i.e. $P_{M_j} = \sum_{i=1}^{n} p_{ij}$), $j = 1, \ldots, m$.
- LB: the trivial lower bound, i.e. $\mathrm{LB} = \max(\{P_{J_i} \mid i = 1, \ldots, n\} \cup \{P_{M_j} \mid j = 1, \ldots, m\})$.
- $\mathrm{MIN} = \min(\{P_{J_i} \mid i = 1, \ldots, n\} \cup \{P_{M_j} \mid j = 1, \ldots, m\})$.
- DIFF: MIN/LB.
- WORKLOAD: the average workload on the machines for a schedule with $C_{\max}$-value equal to the lower bound LB, i.e.

$$\mathrm{WORKLOAD} = \frac{\text{total processing time}}{m \cdot \mathrm{LB}}.$$

If the WORKLOAD of an instance is close to 1 the processing times P_{J_i} of the jobs and the processing times P_{M_j} on the machines are all within a small range. In this case the chance of finding a solution with $C_{\max}$-value close to LB will be rather small. On the other hand, if the WORKLOAD of an instance is small there are only a few jobs or machines with processing times close to LB and many jobs and machines with processing times much smaller than LB. In this case one can expect to find a schedule

with C_{max}-value close to LB or equal to LB. Furthermore, it will be rather easy to construct a schedule with C_{max} close to LB by scheduling the jobs and machines in order of non-increasing P_{J_i} or P_{M_j} values.

Based on the above considerations we have generated new 'hard' instances. These instances and their characteristics are available on the ftp-site ftp.mathematik.Uni-Osnabrueck.DE under the path /pub/osm/preprints/software/openshop. The random generation is done in such a way that MIN is a given percentage of LB. More precisely, we have generated instances of dimension 5×5, 6×6 and 7×7 with $LB = 1000$ and $DIFF \in \{0.9, 1\}$. We have applied the branch and bound algorithm $B\&B_1$ to all these instances. Since the codes of the heuristics of Taillard [14] and Bräsel et al. [1] were not available to us, we compared our method with a tabu search heuristic from Neumann [11]. This tabu search heuristic was developed for general shop problems, which are a generalization of open-shop problems, and gave similar results as the tabu search heuristic of Taillard for the given open-shop benchmark problems (see [11]). The results are presented in Table 5. The table contains the following information:

- data: *jx-y* denotes an instance with $n = m = x$.
- MIN, WORKLOAD: see Table 4.
- $B\&B_1$, opt: the C_{max}-value obtained by the algorithm $B\&B_1$.
- $B\&B_1$, CPU: the CPU-time of the algorithm $B\&B_1$.
- B-*t*: the C_{max}-value of the algorithm $B\&B_1$ after t seconds.

Table 5
Comparison of different solution methods

Data	MIN	Workload	$B\&B_1$ Opt	CPU	B-1000	B-300	B-60	UB_{Neu}	CPU_{Neu}
j5-1	900	0.948	1004	1.7	1004	1004	1004	1044	281.5
j5-2	900	0.929	1002	3.6	1002	1002	1002	1002	310.2
j5-3	900	0.934	1006	4.8	1006	1006	1006	1028	231.7
j5-4	1000	1.000	1042	7.4	1042	1042	1042	1095	200.6
j5-5	1000	1.000	1054	1.1	1054	1054	1054	1054	166.1
J5-6	1000	1.000	1063	5.8	1063	1063	1063	1089	382.9
j6-1	900	0.944	1005	278.7	1005	1005	1013	1051	679.6
j6-2	900	0.949	1021	57.5	1021	1021	1021	1021	1354.1
j6-3	900	0.946	1012	93.9	1012	1012	1012	1051	1387.9
j6-4	1000	1.000	1056	2090.1	1056	1056	1059	1108	1123.8
j6-5	1000	1.000	1045	26.9	1045	1045	1045	1125	592.0
j6-6	1000	1.000	1063	244.4	1063	1063	1067	1085	1199.8
j7-1	900	0.958	1013	77729.2	1022	1033	1130	1060	5517.0
j7-2	900	0.944	1000	6401.6	1012	1015	1019	1061	4182.9
j7-3	900	0.951	1011	277271.1	1038	1045	1069	1051	5107.9
j7-4	1000	1.000	1048[a]	—	1090	1097	1105	1111	2363.4
j7-5	1000	1.000	1055	35451.5	1075	1081	1093	1103	4734.8
j7-6	1000	1.000	1056	176711.1	1066	1067	1090	1113	5359.8

[a] $B\&B_1$ was aborted after 2 700 000 seconds.

- UB_{Neu}: the solution found by the tabu search heuristic from [11].
- CPU_{Neu}: the CPU-time for the tabu search heuristic.

The results can be summarized as follows.

- The new instances are much harder than the benchmark problems of Taillard [14]. Problems of dimension 7×7 are the largest problems that can be solved to optimality by the branch and bound algorithm if the WORKLOAD is close to 1.
- The WORKLOAD is a good measure for the 'hardness' of an instance since the instances with WORKLOAD equal to 1 need, in average, much more computational time than the instances with smaller WORKLOAD.
- The branch and bound algorithm can also be used as a good heuristic for hard problems (see B-60, B-300 and B-1000). The version B-300 always leads to better results than the tabu search heuristic, although the tabu search heuristic often needs much more computational time.

Summarizing, the new branch and bound method yields excellent results for the tested instances. Two benchmark problems of Taillard [14] (tai10-5, tai10-8) were solved to optimality for the first time. Furthermore, for harder instances the branch and bound method can be used as a heuristic, which outperforms a tabu search method.

4. Concluding remarks

We have presented a branch & bound method for solving the open-shop problem. Computational results show that the method is quite effective. Some benchmark problems of Taillard [14] were solved to optimality for the first time.

However, also some known heuristics [1, 14] give quite good results for the benchmark problems using less computational effort than the branch & bound method. Investigating these instances using the workload of the machines we classified them as rather easy and generated some new instances that were hard according to this criteria. The computational results confirm the hardness of these new instances. Furthermore, for these instances the gap between the quality of the results of the branch & bound method and a tabu search heuristic is much larger than for the benchmark problems of Taillard [14]. Therefore, problems of the new type should be considered as new benchmark problems.

References

[1] H. Bräsel, T. Tautenhahn and F. Werner, Constructive heuristic algorithms for the open-shop problem, Computing 51 (1993) 95–110.
[2] P. Brucker, B. Jurisch and M. Jurisch, Open-shop problems with unit time operations, ZOR 73 (1993) 59–73.
[3] P. Brucker, B. Jurisch and A. Krämer, The job-shop problem and immediate selection, Ann. Oper. Res. 50 (1994) 73–114.
[4] P. Brucker, B. Jurisch and B. Sievers, A fast branch & bound algorithm for the job-shop scheduling problem, Discrete Appl. Math. 49 (1994) 107–127.
[5] J. Carlier, The one machine sequencing problem, Eur. J. Oper. Res. 11 (1982) 42–47.

[6] J. Carlier and E. Pinson, An algorithm for solving the job-shop problem, Management Sci. 35 (1989) 164–176.
[7] T. Gonzales and S. Sahni, Open-shop scheduling to minimize finish time, J. Assoc. Comput. Mach. 23 (1976) 665–679.
[8] B. Jurisch and W. Kubiak, Open-shop problems with resource constraints, Oper. Res., to appear.
[9] W. Kubiak, C. Sriskandarajah and K. Zaras, A note on the complexity of open-shop scheduling problems, INFOR 29 (1991) 284–294.
[10] C.Y. Liu and R.L. Bulfin, Scheduling open-shops with unit execution times to minimize functions of due dates, Oper. Res. 36 (1988) 553–559.
[11] E.-N. Neumann, Heuristische Lösungsverfahren für General Shop Probleme, Diplomarbeit, FB Mathematik/Informatik, Universität Osnabrück, 1993.
[12] C.H. Papadimitriou and K. Steiglitz, Combinatorial optimization: Algorithms and Complexity (Prentice-Hall, Englewood Cliffs, NJ, 1982).
[13] B. Sievers, Ein effizientes Branch & Bound-Verfahren für das Job-Shop Scheduling Problem. Diplomarbeit, FB Mathematik/Informatik, Universität Osnabrück, 1989.
[14] E. Taillard, Benchmarks for basic scheduling problems, ORWP 89/21, Lausanne, 1989.
[15] H. Walther and G. Nägler, Graphen – Algorithmen – Programme (Springer, Wien, 1987).
[16] B. Wöstmann, Ein Branch & Bound–Verfahren für das Open-Shop Problem, Diplomarbeit, FB Mathematik/Informatik, Universität Osnabrück, 1993.