9 Open Shop Scheduling

The formulation of an open shop scheduling problem is the same as for the flow shop problem except that the order of processing tasks comprising one job may be arbitrary.

Thus, the open shop scheduling problem (OSP) can be described as follows: a finite set of tasks has to be processed on a given set of machines. Each task has a specific processing time during which it may not be interrupted, i.e. preemption is not allowed. Tasks are grouped to jobs (sets of tasks), so that each task belongs to exactly one job. Furthermore, each task requires exactly one machine for processing. The objective of the OSP is to schedule all tasks, i.e. determine their start times, so as to minimize the maximum completion time (makespan) given the additional constraints that (a) tasks which belong to the same job and (b) tasks which use the same machine cannot be processed simultaneously.

9.1 Complexity Results

Problem $O2||C_{max}$

Let us consider non-preemptive scheduling first. Problem $O2||C_{max}$ can be solved in $O(n)$ time [GS76]. We give here a simplified description of the algorithm presented in [LLRK81]. For convenience let us denote $a_j = p_{1j}$, $b_j = p_{2j}$, $\mathcal{A} = \{J_j \mid a_j \geq b_j\}$, $\mathcal{B} = \{J_j \mid a_j < b_j\}$, $K_1 = \Sigma a_j$ and $K_2 = \Sigma b_j$.

Algorithm 9.1.1 *Gonzalez-Sahni algorithm for* $O2||C_{max}$ [GS76].

begin

Choose any two jobs J_k and J_l for which $a_k \geq \max_{J_j \in \mathcal{A}} \{b_j\}$ and $b_l \geq \max_{J_j \in \mathcal{B}} \{a_j\}$;

Set $\mathcal{A}' := \mathcal{A} - \{J_k\}$;

Set $\mathcal{B}' := \mathcal{B} - \{J_l\}$;

Construct separate schedules for $\mathcal{B}' \cup \{J_l\}$ and $\mathcal{A}' \cup \{J_k\}$ using patterns shown in Figure 9.1.1; -- other tasks from $\mathcal{A}'$ and $\mathcal{B}'$ are scheduled arbitrarily

Join both schedules in the way shown in Figure 9.1.2;

Move tasks from $\mathcal{B}' \cup \{J_l\}$ processed on P_2 to the right;

-- it has been assumed that $K_1 - a_l \geq K_2 - b_k$; the opposite case is symmetric

J. Blazewicz et al., *Handbook on Scheduling*, International Handbooks on Information Systems, https://doi.org/10.1007/978-3-319-99849-7_9

Change the order of processing on P_2 in such a way that T_{2k} is processed first on this machine;
end;

The above problem becomes NP-hard as the number of machines increases to 3. As far as heuristics are concerned we refer to the machine aggregation algorithms introduced in Section 8.3.2 which use Algorithm 9.1.1 in the case of open shop.

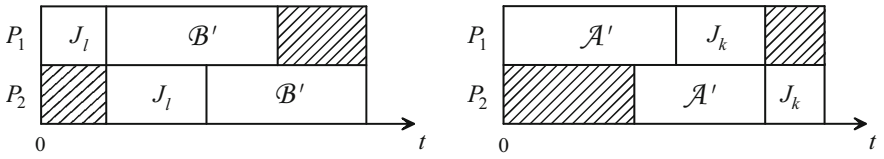

Figure 9.1.1 *A schedule for Algorithm* 9.1.1

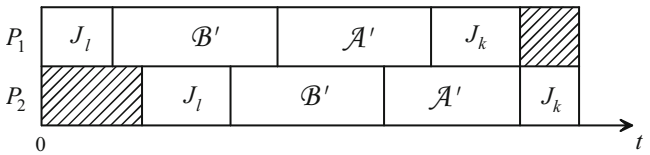

Figure 9.1.2 *A schedule for Algorithm 9.1.1.*

Problem $O\,|\,pmtn\,|\,C_{max}$

Again preemptions result in a polynomial time algorithm. That is, problem $O\,|\,pmtn\,|\,C_{max}$ can be optimally solved by taking

$$C^*_{max} = \max\,\{\max_j\,\{\sum_{i=1}^{m} p_{ij}\},\ \max_i\,\{\sum_{j=1}^{n} p_{ij}\}\}$$

and then by applying Algorithm 5.1.20 [GS76].

Problems $O2\,|\,|\,\Sigma C_j$ and $O2\,|\,|\,L_{max}$

Let us mention here that problems $O2\,|\,|\,\Sigma C_j$ and $O2\,|\,|\,L_{max}$ are *NP*-hard, as proved in [AC82] and [LLRK81], respectively, and problem $O\,|\,pmtn, r_j\,|\,L_{max}$ is solvable via the linear programming approach [CS81].

As far as heuristics are concerned, arbitrary list scheduling and the *SPT* algorithm have been evaluated for $O\,|\,|\,\Sigma C_j$ [AC82]. Their asymptotic performance ratios are $R_L^{\infty} = n$ and $R_{SPT}^{\infty} = m$, respectively. Since the number of tasks is usually much larger than the number of machines, the bounds indicate the advantage of *SPT* schedules over arbitrary ones.

A survey of results in open shop scheduling may be found in [DPP01, KSZ91].

9.2 A Branch and Bound Algorithm for Open Shop Scheduling

Only few exact solution methods are available for the open shop scheduling problem. We describe a branch-and-bound algorithm of Dorndorf et al. [DPP01] for solving this problem which performs better than other existing algorithms. The key to the efficiency of the algorithm lies in the following approach: instead of analyzing and improving the search strategies for finding solutions, the authors focus on constraint propagation based methods for reducing the search space. Extensive computational experiments on several sets of well-known benchmark problem instances are reported. For the first time, many problem instances are solved to optimality in a short amount of computation time.

9.2.1 The Disjunctive Model of the OSP

Studies have shown that within the class of intractable problems the OSP belongs to the especially hard ones [BHJW97, GJP00]. As an example, the famous job shop scheduling problem (JSP) which is a close relative of the OSP is easily solvable by now for problem instances with up to 100 tasks, see e.g. [AC91, CP94, CL95, MS96], while there still remain unsolved instances of the OSP with less than 50 tasks.

In this chapter, we describe a branch-and-bound algorithm for solving the OSP. Instead of analyzing and improving the search strategies, we especially focus on constraint propagation based methods for reducing the search space. As a positive side-effect, the constraint propagation algorithm implicitly calculates strong lower bounds so that an explicit computation is not necessary. Extensive computational experiments on several sets of well-known benchmark problem instances show that this algorithm outperforms other exact solution methods for the OSP. With this algorithm, for the first time, many problem instances were solved to optimality within a very short amount of computation time.

The remainder of this chapter is organized as follows. Next we describe the well-known disjunctive model of the OSP that is due to Roy and Sussmann [RS64] and its extension by Błażewicz et al. [BPS00] and give a short review on solution methods for the OSP. Section 9.2.2 introduces the basic concepts of constraint propagation and presents several consistency tests which are used for the reduction of the search space. These consistency tests are embedded in a branch-and-bound algorithm that uses a branching scheme that is due to Brucker et al. [BHJW97]. A short description of this algorithm is given in section 9.2.3.

Extensive computational results of the branch-and-bound algorithms that use different consistency tests are then presented in the last section.

The disjunctive model that has been introduced by Roy and Sussmann [RS64] for the job shop scheduling problem can be easily adapted to the OSP. Let $\mathcal{T} = \{T_1, \cdots, T_n\}$ be the set of tasks to be scheduled. The processing time of task $T_i \in \mathcal{T}$ is denoted with p_i. Choosing sufficiently small time units, we can always assume that the processing times are positive integer values. Each task is associated a start time variable st_i with domain set $\mathbb{N}_0$. Since we want to minimize the makespan, i.e. the maximum completion time of all tasks, the objective function is $C_{max}(st_1, \cdots, st_n) = \max_{T_i \in \mathcal{T}}\{st_i + p_i\}$.

Let $job(i)$ denote the job associated to a task T_i. Further, let $mach(i)$ be the machine required by task T_i. Obviously, two tasks T_i and T_j cannot be processed simultaneously at any time, if $job(i) = job(j)$ or $mach(i) = mach(j)$. These two tasks (as a pair) will belong to set D of forbidden pairs. However, if T_i and T_j cannot be processed in parallel then *either* T_i must finish before T_j can start *or* T_j must be completed before T_i is started. Thus, given

$$D = \big\{\{i,j\} \mid T_i, T_j \in \mathcal{T},\, i \neq j,\, job(i) = job(j) \vee mach(i) = mach(j)\big\},$$

the OSP can be written as the following model with disjunctive constraints

$$\begin{array}{ll} \min\{C_{max}(st_1, \cdots, st_n)\}, & st_i \in \mathbb{N}_0 \quad T_i \in \mathcal{T}, \\ (st_i + p_i \leq st_j) \;\vee\; (st_j + p_j \leq st_i) & \{i, j\} \in D. \end{array} \tag{9.2.1}$$

A *schedule* is an assignment $S = (st_1, \cdots, st_n) \in \mathbb{N}_0 \times \cdots \times \mathbb{N}_0$ of all start time variables. For the sake of simplicity, we will use the same notation for variables and their assignments. Schedule S is *feasible* if it satisfies all constraints given by (9.2.1). Reformulating the OSP, the goal is to find a feasible schedule with minimal objective function value $C_{max}(S)$.

The significance of the disjunctive scheduling model for the development of efficient solution methods is revealed if we consider its graph theoretical interpretation. The *disjunctive graph* associated to an OSP instance is a weighted graph $G = (\mathcal{T}, D, W)$ with the node set $\mathcal{T}$, arc set D and the weight set $W = \{\, w_{ij} = p_i \mid \{i,j\} \in D \,\}$. D is also called the set of disjunctive arcs. Since D is symmetric, we will represent disjunctive arcs as doubly directed arcs. From now on, we will further use the suggestive notation $i \leftrightarrow j$ for pairs (i,j), (j,i) of disjunctive arcs, and $i \rightarrow j$ to specify one of the arc orientations.

A disjunctive graph is transformed into a directed graph by choosing one arc orientation of each disjunctive arc pair $i \leftrightarrow j \in D$. We obtain a complete (partial) selection if (at most) one arc orientation is chosen from each disjunctive arc pair. The selection is acyclic if after the removal of all remaining undirected disjunctions the resulting directed graph is acyclic.

There exists a simple and well-known many-to-one relationship between

feasible schedules and complete, acyclic selections which allows us to restate the OSP as a graph theoretical problem: find a complete and acyclic selection, so that the longest path in the associated directed graph has a minimum length. Thus, it is sufficient to search through the space of all selections which is of cardinality $2^{|D|}$ instead of the space of all schedules which is of cardinality $|\mathbb{N}_0|^n$.

Most solution methods for the OSP are based on this fundamental observation. However, due to the exceptionally intractable nature of the OSP, mainly heuristic solution methods have been proposed. Simple list scheduling heuristics based on priority dispatching rules have been examined by Guéret and Prins [GP98a]. Matching algorithms are discussed by Bräsel et al. [BTW93] and Guéret and Prins [GP98a, GP98b]. The shifting bottleneck procedure, originally designed for the JSP, has been adapted by Ramudhin and Marier [RM96] to the OSP. Another important class of heuristics are the insertion algorithms which have been introduced by Werner and Winkler [WW95] for the JSP and generalized by Bräsel et al. [BTW93] for the OSP. Local search approaches (tabu search) and genetic algorithms have been examined by Taillard [Tai93], Liaw [Lia98] and Prins [Pri00]. Colak and Agarwal [CA05] developed a neural network based meta-heuristic approach that allows integration of domain specific knowledge. Learning strategies imply improved neighbour solutions. Blum and Sampels [BS04, Blu05] applied ant colony optimization to shop scheduling. Some of these heuristics, especially the genetic algorithm of Prins and the ant colony optimization of Blum and Sampels, show a very good performance, and for specific classes of OSP instances they often are able to find optimal solutions. However, in general, the solutions found for arbitrary OSP instances are of course of a suboptimal nature.

Only few exact solution methods are available for the OSP. A branch-and-bound algorithm which applies a block-oriented branching scheme and some basic constraint propagation methods for reducing the search tree has been proposed by Brucker et al. [BHJW97]. Guéret et al. [GJP00] improved this algorithm by using an intelligent backtracking technique which replaces the simple depth-first search used by the former. They further applied some additional search tree reduction methods in their branch-and-bound algorithm based on *forbidden intervals* (see Chapter 4.1), i.e. time intervals in which no task can start or end in an optimal solution [GP98b]. All these exact solution methods are capable of solving smaller OSP instances for which they naturally show a better performance than the heuristic methods. However, even for simple, but larger OSP instances for which the heuristic methods easily find an optimal solution, the performance of the exact solution methods is rather poor, since the search space reduction methods applied are not sufficient to handle the combinatorial explosion. In the next section, we will therefore examine additional concepts for reducing the search space which have been described in Dorndorf et al. [DPP01]. It will turn out that these constraint propagation based methods are very efficient and allow solving a large number of simple, hard and very hard OSP benchmark instances which up to now have not been solved.

9.2.2 Constraint Propagation and the OSP

Constraint propagation is an elementary method of search space reduction which has become more and more important in the last decades. The basic idea of constraint propagation is to evaluate implicit constraints through the repeated analysis of the variables, domains and constraints that describe a specific problem instance. This analysis makes it possible to detect and remove *inconsistent* variable assignments that cannot participate in any solution by a merely partial problem analysis. A whole theory is devoted to the definition of different concepts of consistency which, roughly speaking, define the maximal search space reduction that is possible regarding some specific criteria and may serve as a theoretical background for propagation techniques. An exhaustive study of the theory of constraint propagation can be found in [Tsa93]. Dorndorf et al. [DPP99, DPP00] examine constraint propagation techniques for disjunctive and cumulative scheduling problems; for the details we refer to Chapter 16.

Removing *all* inconsistent assignments is in general not possible due to an exponentially increasing computational complexity, so we usually have to content ourselves with approximations. The main issue is to describe simple rules which allow efficient search space reductions, but at the same time can be implemented efficiently. These rules are called *consistency tests*. In the disjunctive scheduling community, some of them are also known as *immediate selection* or *edge-finding* rules.

Consistency tests are generally described through a condition and a search space reduction rule. Whenever the condition is satisfied, the reduction rule is executed. In order to describe the basic concepts of constraint propagation more precisely, we will focus on *domain consistency tests* for the time being. Similar results, however, apply for other types of consistency tests.

A domain consistency test is a consistency test which deduces domain reductions. Let Δ_i be the *current domain* of the start time variable st_i. If *UB* is an upper bound of the optimal makespan, then we can initially set $\Delta_i := [0, UB - p_i]$. This is necessary, since most consistency tests can only deduce domain reductions if the current domains are finite. The upper bound *UB* can be found by applying a simple heuristic method or by choosing the trivial value $\sum_{T_i \in \mathcal{T}} p_i$. Given a current domain for each start time variable, a domain consistency test maps a set $\Delta = \{ \Delta_i \mid T_i \in \mathcal{T} \}$ of current domains into a set $\Delta' = \{ \Delta_i' \mid T_i \in \mathcal{T} \}$ of hopefully, but not necessarily reduced current domains. Of course, a domain consistency test only removes values, for which provably no feasible schedule *S* exists that could be developed from Δ.

In order to obtain the maximal domain reduction possible, it is not sufficient to apply each of these tests only once. The reason for this is that after the reduction of several domains, additional domain adjustments could possibly be derived using some of the tests which previously have failed in deducing any reductions.

Thus, all consistency tests have to be applied in an iterative fashion rather than only once until no more updates are possible. This is equivalent to the computation of a fixed point. Notice that this fixed point does not have to be unique and in general depends upon the order of the application of the consistency tests. Thus, for some application orders the domain reductions obtained may be stronger than for others. Fortunately, it is possible to show that for consistency tests which satisfy a quite natural monotony property, the fixed point computed is always unique [DPP00, Chapter 16]. Since the consistency tests studied are all monotonous in this sense, the application order is irrelevant regarding the extent of the domain reduction. Regarding the complexity of the fixed point computation, however, the application order does play a very crucial role. Notice that the revision of a single domain already forces all consistency tests to be reapplied in the next iteration even though only a small number of constraints and variables are possibly affected by this reduction. Thus, choosing an intelligent order can decrease the computation time to a large extent. However, we will not deal with this issue more closely, but choose a quite naive propagation order.

In the next subsections, we will describe the set of consistency tests used in the algorithm. In addition to domain consistency tests, the disjunctive scheduling model and its graph theoretical interpretation allow the definition of consistency tests which operate on the set of complete selections. These consistency tests reduce the set of complete selections by detecting sequences of tasks which must occur in every optimal solution. Since this is done by selecting disjunctive arc orientations, the latter approach has been often labeled *immediate selection* (see e.g. [CP89, BJK94]) or *edge-finding* (see e.g. [AC91]). We will use the term *sequence consistency test* as opposed to domain consistency tests and as used in [DPP99,DPP00]. Domain and sequence consistency tests are two different concepts which complement each other. Often, a situation occurs in which either only reductions of the current domains or only arc orientations are deducible. The best results, in fact, are obtained by applying both types of consistency tests, as fixing disjunctive arcs may initiate additional domain reductions and vice versa, cf. Chapter 16.

Input/Output Consistency Tests

Quite important for the development of efficient consistency tests for the OSP is the concept of *disjunctive cliques* or *cliques* for short. We will say that $O_c \subseteq \mathcal{T}$ is a clique if any pair of tasks in O_c cannot be processed in parallel, i.e. if all tasks in O_c either belong to the same job or require the same machine. A clique O_c is said to be maximal, if no true superset of O_c is a clique. Therefore, there exist $|\mathcal{J}|$ maximal job cliques, where $\mathcal{J}$ denotes the set of jobs, and $|\mathcal{P}|$ maximal machine cliques, where $\mathcal{P}$ denotes the set of machines (processors).

For the rest of this section, we will assume that $O_c \subseteq \mathcal{T}$ is a maximal clique

and that *all* subsets A (tasks T_i) are subsets (elements) of this clique. Without loss of generality we will number the indices of the elements of O_c by $1,2,\cdots,|O_c|$. Let further $est_i := \min \Delta_i$ and $lst_i := \max \Delta_i$ denote the earliest and latest start time of task T_i, and let $ect_i := est_i + p_i$ and $lct_i := lst_i + p_i$ denote the earliest and latest completion time of task T_i. Finally, for a subset $A \subset O_c$ of tasks, let $EST_{min}(A) := \min_{T_i \in A} est_i$, $LCT_{max}(A) := \max_{T_i \in A} lct_i$, and $p(A) := \sum_{T_i \in A} p_i$.

Given a clique of tasks $A \subset O_c$ and an additional task $T_i \in O_c \setminus A$, Carlier and Pinson [CP89] were the first to derive conditions which imply that T_i has to be processed *before* or *after* all tasks $T_j \in A$. In the first case, they called T_i the *input* of A, in the second case, the *output* of A, and so Dorndorf et al. [DPP00] have chosen the name *input/output conditions*.

Theorem 9.2.1 *(Input/Output Sequence Consistency Tests). Let $A \subset O_c$ and $T_i \in O_c \setminus A$. If the input condition*

$$LCT_{max}(A \cup \{T_i\}) - EST_{min}(A) < p(A \cup \{T_i\}) \tag{9.2.2}$$

is satisfied then task T_i has to be processed before all tasks in A, for short, $T_i \rightarrow A$. Likewise, if the output condition

$$LCT_{max}(A) - EST_{min}(A \cup \{T_i\}) < p(A \cup \{T_i\}) \tag{9.2.3}$$

is satisfied then task T_i has to be processed after all tasks in A, $A \rightarrow T_i$. □

Domain consistency tests that are based on the input/output conditions can now be simply derived. We will only examine the adjustment of the earliest start times, as the adjustment of the latest start times can be handled analogously. Obviously, if task T_i is the output of a clique A then T_i can only start if all tasks in A have finished. Thus, the earliest start time of T_i is at least the maximum completion time of all tasks in A being scheduled without preemption. Unfortunately, however, the computation of this makespan requires the solution of an NP-hard single-machine scheduling problem. Therefore, if the current domains are to be updated efficiently, we have to content ourselves with approximations of this bound. The following theorem is due to Carlier and Pinson [CP89, CP90].

Theorem 9.2.2 *(Output Domain Consistency Tests, part 1). If the output condition is satisfied for $A \subset O_c$ and $T_i \in O_c \setminus A$ then the earliest start time of T_i can be adjusted to $est_i := \max\{est_i, C^{pr}_{max}(A)\}$, where $C^{pr}_{max}(A)$ is the maximum completion time of all tasks in A being scheduled with preemption allowed.* □

Notice that the computation of $C^{pr}_{max}(A)$ has time complexity $O(|A|\log|A|)$ [Jac56].

It has already been mentioned that applying both sequence and domain consistency tests together can lead to better search space reductions. Quite evidently, any domain reductions deduced by Theorem 9.2.2 can lead to additional arc orientations deduced by Theorem 9.2.1. We will now discuss the case in which the inverse is also true. Imagine a situation in which $A \rightarrow T_i$ can be deduced for a subset of tasks, but in which the output condition does not hold for the couple (A, T_i). Such a situation can actually occur as can be seen in the following example.

In Figure 9.2.1, an example with three tasks is shown. The earliest start time of T_i is $est_i = 4$, while its latest completion time is $lct_i = 9$. The earliest start and latest completion times of T_j and T_k are $est_j = est_k = 0$ and $lct_j = lct_k = 9$, respectively. The processing times of T_i, T_j and T_k are $p_i = p_j = p_k = 3$. Notice that we can both deduce $T_j \rightarrow T_i$ and $T_k \rightarrow T_i$ using the input conditions for the couple $(\{T_i\}, T_j)$ and $(\{T_i\}, T_k)$, since e.g. $LCT_{max}(\{T_i, T_j\}) - est_i = 5 < 6 = p_i + p_j$. Thus, we know that $\{T_j, T_k\} \rightarrow T_i$. However, the output condition is not satisfied for the couple $(\{T_j, T_k\}, T_i)$ because $LCT_{max}(\{T_j, T_k\}) - EST_{min}(T_i, T_j, T_k) = 9 = p_i + p_j + p_k$.

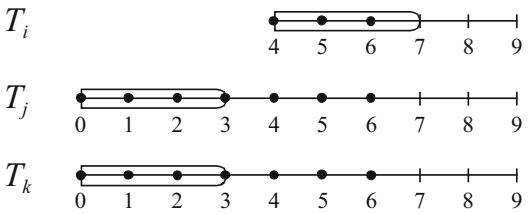

Figure 9.2.1 *An example with three tasks.*

This example motivates the following theorem as an extension of Theorem 9.2.2.

Theorem 9.2.3 (*Input/Output Domain Consistency Tests, part 2*). *Let $A \subset O_c$ and $T_i \in O_c \setminus A$. If $A \rightarrow T_i$ then the earliest start time of T_i can be adjusted to*

$$est_i := \max\{est_i, C^{pr}_{max}(A)\} .$$ □

Here, the reader should recall once more that the subset A mentioned in the last theorem does not have to coincide with the subset for which the input or the output condition is satisfied.

An important question to answer now is whether there exist efficient algorithms that implement the input/output consistency tests. An efficient implementation is obviously not possible if all pairs (A, T_i) of subsets $A \subset O_c$ and tasks $T_i \in O_c \setminus A$ are to be tested separately. Fortunately, it is not necessary to do so as has been first shown by Carlier and Pinson [CP90] who have developed an $O(|O_c|^2)$

algorithm for applying the input/output consistency tests described in Theorem 9.2.1 and Theorem 9.2.2 (It is common practice to only report the time complexity for applying the consistency tests *once* for all couples (A, T_i). In general, however, the number of iterations necessary for computing the fixed point of current domains has to be considered as well. This accounts for an additional factor c which depends upon the size of the current domains, but is omitted here.) Several years later, $O(|O_c| \log |O_c|)$ algorithms have been proposed by Carlier and Pinson [CP94] and Brucker et al. [BJK94] which until now have the best asymptotic performance, but are less efficient for smaller problem instances and require quite complex data structures.

Nuijten [Nui94], Caseau and Laburthe [CL95] and Martin and Shmoys [MS96] have chosen a solely domain oriented approach and have derived different algorithms for implementing the input/output consistency tests that are based on Theorem 9.2.2. Nuijten developed an algorithm with time complexity $O(|O_c|^2)$ which can be generalized to scheduling problems with discrete resource capacity. Caseau and Laburthe presented an $O(|O_c|^3)$ algorithm which works in an incremental fashion, so that $O(|O_c|^3)$ is a worst case, since not all consistency tests are applied within an iteration of the fixed point computation. The algorithm proposed by Martin and Shmoys [MS96] also has a time complexity of $O(|O_c|^2)$.

Dorndorf et al. [DPP01] have implemented the input/output tests described in Theorems 9.2.1 and 9.2.2. They could have used the $O(|O_c|^2)$ algorithm of Carlier and Pinson for implementing Theorem 9.2.1 and then adjusted the current domains according to Theorem 9.2.3. However, the algorithm of Carlier and Pinson already requires the adjustment of some of the domains and, in fact, is a combination of the consistency tests described in Theorems 9.2.1 and 9.2.2. Thus, many of these domain adjustments would be recomputed if Dorndorf et al. afterwards applied the consistency tests described in Theorem 9.2.3. They have therefore developed two algorithms which work in a purely sequential fashion, one of which has a time complexity of $O(|O_c|^3)$, while the other has a time-complexity of $O(|O_c|^2)$. These algorithms are based on the definition of *task sets* as introduced by Caseau and Laburthe [CL95]. A detailed description of the algorithms is given in [Pha00].

Given the arc orientations derived, the domain adjustments of Theorem 9.2.3 can then be applied with effort $O(|O_c|^2 \log |O_c|)$ using Jackson's famous algorithm [Jac56].

Note that the approach by Dorndorf et al. of first deducing the arc orientations and then applying the domain adjustments implies a higher time complexity than for algorithms based on the purely domain oriented approach. However, stronger domain reductions may be achieved, as demonstrated by the previous example.

Input/Output Negation Consistency Tests

In the last subsection, a condition has been described which implies that a task has to be processed before (after) another set of tasks. In this subsection, the inverse situation, that a task *cannot* be processed first (last), is studied. The following theorem is due to Carlier and Pinson [CP89, CP90]. For reasons near at hand, Dorndorf et al. have chosen the name input/output negation for the conditions described in this theorem.

Theorem 9.2.4 *(Input/Output Negation Sequence Consistency Tests). Let* $A \subset O_c$ *and* $T_i \in O_c \setminus A$. *If the input negation condition*

$$LCT_{max}(A) - est_i < p(A \cup \{T_i\}) \tag{9.2.4}$$

is satisfied then task T_i *cannot be processed before all tasks in* A. *Likewise, if the output negation condition*

$$LCT_{max} - EST_{min}(A) < p(A \cup \{T_i\}) \tag{9.2.5}$$

is satisfied then task T_i *cannot be processed after all other tasks in* A. □

This theorem allows a reduction of the current domains which, in general, is weaker than the one that has been described in Theorem 9.2.3. However, since the input/output negation conditions are more often satisfied than the input/output conditions, they will turn out to be quite important for solving the OSP efficiently.

Let us study the input negation condition and the adjustments of earliest start times. If (9.2.4) is satisfied for $A \subset O_c$ and $T_i \in O_c \setminus A$, there must be a task in A which starts and finishes before T_i, although we generally do not know which one. This proves the following theorem [CP89, CP90].

Theorem 9.2.5 *(Input/Output Negation Domain Consistency Tests). If the input negation condition is satisfied for* $A \subset O_c$ *and* $T_i \in O_c \setminus A$ *then the earliest start time of task* T_i *can be adjusted to* $est_i := \max\{est_i, \min_{T_u \in A} ect_u\}$. □

Input/output negation consistency tests have been applied by Nuijten [Nui94], Baptiste and Le Pape [BL95] and Caseau and Laburthe [CL95] for the JSP. All these algorithms only test some, but not all interesting couples (A, T_i). An algorithm which deduces all domain reductions with a time complexity of $O(|O_c|^2)$ has only recently been developed by Baptiste and Le Pape [BL96]. Dorndorf et al. [DPP01] have developed another algorithm which also performs all possible domain adjustments in $O(|O_c|^2)$. This algorithm uses some main ideas of Baptiste and Le Pape, but can be better combined with the algorithms that Dorndorf et al. have developed for the other consistency tests, since some computations can be

reused, see again [Pha00] for the details.

Shaving

A closer look at the consistency tests presented so far reveals that they all share the following common and simple idea: a hypothesis (e.g. task T_i starts at time st_i) can be refuted, if there exists no schedule so that this hypothesis is satisfied. Consistency tests only differ in the kind of hypotheses that are made and the proof for showing that no schedule can exist under these hypotheses. The input negation consistency test, for instance, verifies for a given clique A of tasks whether there exists a schedule in which some task T_i is started within the time interval $[est_i , \min_{T_u \in A} ect_u - 1]$. This verification is carried out through a simple test which compares the length of the time interval $[est_i, LCT_{max}(A)]$ with the sum of processing times $p(A \cup \{T_i\})$. Replacing this simple test with other and possibly more sophisticated tests leads to different and probably more powerful consistency tests.

A general approach in which all hypotheses are of the kind: "task T_i starts at its earliest start time" or "task T_i starts at its latest start time" has been proposed by Martin and Shmoys under the name *shaving* [MS96]. In *exact one-machine shave* the verification is carried out by solving an instance of a one-machine scheduling problem in which $st_i := est_i$ or, alternatively, $st_i := lst_i$. *One-machine shave* relaxes the non-preemption requirement and tests whether a possibly preemptive schedule exists under the aforementioned hypothesis. Carlier and Pinson [CP94] and Martin and Shmoys [MS96] both proposed the computation of fixed points as a method for proving that a feasible schedule cannot exist under a certain hypothesis. More precisely, the hypothesis is falsified if a current domain becomes empty during the fixed point computation.

Dorndorf et al, [DPP01] apply shaving by testing the hypotheses $st_i > t \in \Delta_i$ and $st_i < t \in \Delta_i$. Test values for t are chosen during a combination of bisection and incremental search. Apparently, the application of shaving techniques can be very costly. However, the search space reduction obtained by shaving by far offsets these costs.

9.2.3 The Algorithm and Its Performance

In general, the single application of constraint propagation is not sufficient for solving the OSP. Although for certain problem instances the search space reduction may be of a considerable size, a branch-and-bound search is usually still necessary for finding an optimal solution. In this section, we give a short description of the block branching scheme which has been described by Brucker et al. [BJS94] for the JSP and, for instance, by [BHJW97] for the OSP and which we

have used as well in our branch-and-bound algorithm. A deeper insight into the nature of the block branching scheme is given by Phan Huy [Pha00], who discusses a generalization for shop scheduling problems with arbitrary disjunctive constraints.

The block branching scheme requires the computation of a heuristic solution (complete and acyclic selection) in each node of the branching tree that is compatible with the arc orientations already chosen. Dorndorf et al. [DPP01] chose as a heuristic solution method the priority rule based dispatching heuristic that has been described by Brucker et al. [BHJW97] in their algorithm B&B1. Given a complete and acyclic selection, a critical path B, i.e. a path of maximal length is chosen within the associated directed graph. This critical path is then decomposed into so-called maximal blocks, the definition of which is given in the following. A subpath $B' = u_1 \rightarrow \cdots \rightarrow u_l$ of B of length $l \geq 2$ is a *block* iff, for all $i \neq j$, we have $u_i \leftrightarrow u_j \in D$, i.e. iff two pairwise different tasks in B' are always in disjunction, since they belong to the same job or require the same machine. A block B' is said to be *maximal*, iff extending B' by even only one node (task) already violates the block condition. Obviously, given a critical path, there always exists a unique decomposition into maximal blocks. Given this block decomposition, the block branching scheme as described by Brucker et al. [BJS94] is based on the following observation:

Let $\mathtt{S}$ be a complete and acyclic selection and B a critical path in the corresponding directed graph. If $\mathtt{S}$ is not optimal, i.e. there exists a selection $\mathtt{S}'$ with a smaller makespan, then there is a maximal block in B so that in $\mathtt{S}'$ a task within this block is processed before the first or after the last task of this block.

Thus, child nodes are created by moving tasks of a block to the beginning or end of the block. Consequently, $2\cdot(l-1)$ child nodes are generated for each block of length $l > 2$, while for blocks of length 2 obviously only one child node is generated. Improving this branching scheme, Brucker et al. [BJS94] described how to fix additional arcs depending on the search nodes that have been already visited prior to the generation of the actual search node. Further they described the particular role played by the first and the last block of the maximal block decomposition, since the number of tasks to be moved to the beginning or end of these blocks can be reduced. The search strategy of their branching algorithm has been organized in a depth-first manner. For further details on the block branching scheme we refer the reader to the work of Brucker et al. [BJS94, BHJW97]. Dorndorf et al. [DPP01] have used this branching scheme except for some minor modifications regarding the branching order, i.e. the sequence in which the child nodes are generated, see [Pha00] for the technical details.

Upon finding an improved solution in a node (initial solution in the root node) of the branching tree, the makespan of this solution is, of course, used as upper bound UB. The lower bounds used within the branch-and-bound algorithm are the preemptive one-machine (one-job) lower bounds which are computed using Jackson's algorithm [Jac56]. Notice, however, that stronger bounds are

calculated in an implicit manner by the application of constraint propagation: whenever an inconsistency is detected, for instance, if a current domain becomes empty, we know that no solution can be generated from the actual search tree node with a makespan of *UB* and, therefore, *UB* is indeed a lower bound for this search tree node.

Dorndorf et al. [DPP01] have implemented the branch-and-bound algorithm together with the constraint propagation techniques in *C* on Pentium II (333 MHz) in MSDOS environment. They have tested the algorithm on a large set of benchmark problems that have been generated by Taillard [Tai93] (Tai-*n*-*) and Brucker et al. [BHJW97] (Hur-*n*-*). All test instances are quadratic of size *n* jobs and *n* machines, with *n* ranging from 6 to 20. We will see below that, on the one hand, most of the quite large instances of Taillard are easily solved by Dorndorf et al.'s algorithm. They have solved all the 10×10 instances, something which none of the current exact algorithms is capable of, and even do so with an average run time of less than a minute. Further, they have solved most of the 15×15 instances in several minutes and most of the 20×20 instances in less than an hour. Among these instances, three instances (Tai-15-5, Tai-15-9, Tai-20-6) have not been solved prior the start of their experiments. On the other hand, the rather small instance Hur-7-1 of size 7×7 still remains open, although they have been able to improve the current best lower bound from 1000 to 1021.

Brucker et al. [BHJW97] have proposed an explanation for this phenomenon which is based on the *work load* of a problem instance. The work load of an OSP instance is defined as follows: given a set of jobs $\mathcal{J}$ and a set of machines $\mathcal{P}$. Let O^{ι} be the maximal clique of tasks belonging to job J_{ι} and O_{μ} be the maximal clique of tasks requiring machine P_{μ}. Let, further, $LB := \max\{ \max\{ p(O^{\iota}) \mid J_{\iota} \in \mathcal{J} \}, \max\{ \{ p(O_{\mu}) \mid P_{\mu} \in \mathcal{P} \} \}$ define the trivial lower bound which is the maximum of the job and machine bounds (the sum of processing times of tasks belonging to a job or machine clique). The *average work load WL* is then defined as

$$WL = \frac{\sum_{J_{\iota} \in \mathcal{J}} p(O^{\iota}) + \sum_{P_{\mu} \in \mathcal{P}} p(O_{\mu})}{(|\mathcal{J}| + |\mathcal{P}|) \cdot LB}$$

If the work load of an OSP instance is close to 1 then all job and machine bounds are not much smaller than *LB*, so that finding a solution with a makespan close to *LB* is not very probable. On the contrary, an OSP instance with a low work load tends to have an optimal solution with a makespan equal to the lower bound *LB*. These problem instances are less hard to solve, since an optimal solution can be more easily verified.

Considering this intuitive interpretation, it is possible to use the work load to guide the choice of an appropriate solution strategy. The alternatives that are given are a *top-down* and a *bottom-up* strategy. Both strategies use the branch-and-bound algorithm and only differ in the way of choosing the initial upper

bound(s). The top-down strategy starts with a *real* upper bound which, in [DPP01], is determined by the heuristic solution method and tries to improve (decrease) this upper bound by applying the branch-and-bound algorithm. The bottom-up approach uses a lower bound as a hypothetical upper bound and, whenever the branch-and-bound algorithm does not find a solution which is consistent with this upper bound, increases it by one time unit. This process is repeated until a solution is found.

Notice, that the top-down approach only applies the branch-and-bound algorithm once, but that constraint propagation is less effective since the current domains are less tight due to the high initial upper bound. Hence, searching the whole search tree may require a higher computation time. The bottom-up approach, on the contrary, reinitializes the branch-and-bound algorithm several times, but allows more constraint propagation since the current domains are smaller. Therefore, the search trees that are created are smaller. Altogether, the top-down approach seems to be more suited, if the optimal makespan is far from the lower bound *LB*, since the multiple application of the branch-and-bound algorithm within a bottom-up approach would offset its propagation advantages. Also, according to this logic, the bottom-up approach is to be preferred if the optimal makespan is near to the lower bound *LB*. Thus, it is straightforward to choose the top-down approach whenever the work load of an OSP instance is high and the bottom-up approach whenever the work load is low.

At first, however, we will only evaluate the top-down approach since this allows to analyze better the impact of the different consistency tests. It seems justified to say that instances of the OSP, especially those with a high work load, are generally more difficult to solve than instances of the JSP with the same number of tasks, jobs and machines. To one part, this is due to the larger solution space: not only machine sequences, but also job sequences have to be determined. Thus, Dorndorf et al. [DPP01] have often encountered a situation in which the search process was trapped in an unfavorable region of the search space from which it could not escape within a reasonable amount of time. Another reason for the intractability of the OSP, however, is the lack of strong lower bounds. In fact, if no search is carried out, the lower bound *LB* is already the best bound one is able to find. Thus, constraint propagation plays a more important role in reducing the search space.

In the beginning, the experiments for the two different classes of consistency tests (input/output and input/output negation consistency tests), have been carried out for a set of smaller instances, namely, the instances Tai-7-* and Hur-6-*. The results are shown in Table 9.2.1. CP1 applies the input/output tests as described in Theorem 9.2.1 and Theorem 9.2.3, while CP2 applies both the input/output tests and the input/output negation tests. Since [DPP01] have applied a top-down strategy for CP1 and CP2 , they report for each problem instance the initial upper bound found by the heuristic solution method (UB_{init}) in addition to the optimal makespan (UB_{best}). They further report the number of search tree nodes generated by each of the algorithms and the total run time. All of the instances

have naturally been solved to optimality.

problem	UB_{best}	UB_{init}	CP1		CP2	
			nodes	time	nodes	time
Hur-6-1	1056	1528	55634	149.7 s	36876	133.0 s
Hur-6-2	1045	1377	3291	7.3 s	1711	5.2 s
Hur-6-3	1063	1536	9737	23.9 s	5401	18.0 s
Hur-6-4	1005	1481	8553	20.3 s	4356	14.4 s
Hur-6-5	1021	1647	2983	6.4 s	1562	4.6 s
Hur-6-6	1012	1276	8406	19.7 s	4263	13.8 s
Hur-6-7	1000	1454	4557	11.5 s	3205	10.7 s
Hur-6-8	1000	1636	169	0.4 s	132	0.4 s
Hur-6-9	1000	1524	525	1.2 s	326	1.0 s
Tai-7-1	435	609	147	0.4 s	130	0.4 s
Tai-7-2	443	614	309	1.1 s	225	0.9 s
Tai-7-3	468	632	8789	36.6 s	5661	30.9 s
Tai-7-4	463	664	1892	7.5 s	1040	5.3 s
Tai-7-5	416	551	521	2.0 s	409	2.0 s
Tai-7-6	451	581	28347	124.5 s	16464	95.8 s
Tai-7-7	422	693	61609	254.5 s	30101	167.7 s
Tai-7-8	424	637	1467	5.9 s	961	5.0 s
Tai-7-9	458	551	237	0.8 s	194	0.8 s
Tai-7-10	398	576	25837	107.2 s	9427	53.2 s

Table 9.2.1 *Results for some smaller instances (top-down).*

Obviously, CP2 generates less search tree nodes and has a lower total run time than CP1, although more constraint propagation is applied in each of the single nodes. On average, CP2 generates approximately 40 % less search tree nodes than CP1 and has a run time which is lower by about 25 %. Note, that a different observation has been made for the JSP, see [Pha00]: although the number of search tree nodes decreases as well, the total run time increases (for smaller instances) due to the additional propagation effort. Thus, the additional application of the input/output negation tests is more efficient for the OSP than for the JSP. This can be explained as follows: the input/output tests, if applied on their own, deduce only few arc orientations for the OSP in the beginning of the branch-and-bound process, because at that time most of the current domains are just too large and coincide with the trivial interval $[0, UB - p_i]$. Only at a certain depth of the search tree, more arc orientations are deduced, however, the portion of the search tree that can be pruned by that time is rather small. Consequently, the additional application of the input/output negation tests improves the efficiency of the input/output tests since the former are a relaxation of the latter and so are capable of deducing domain reductions at an earlier stage of the branching process.

Next Dorndorf et al. [DPP01] tested the better algorithm CP2 on the larger OSP instances Tai-10-* and the harder instances Hur-7-*. They further tested a shaving variant of CP2, i.e. in each of the search tree nodes they applied the shaving procedure described in Section 9.2.2 and used the input/output and in-

put/output negation tests for detecting inconsistencies. The results for CP2 are shown in Table 9.2.2 and those for the branch-and-bound algorithm with shaving CPS2 in Table 9.2.3. In addition to the usual information listed further above, they report for each problem instance the best upper bound found (UB_{found}) within a time limit of 5 hours. Upper bounds shown in parentheses are either non optimal or optimal, but could not be verified. As an example, 1048 is the best upper bound known for the instance Hur-7-1 and 1052 is the best bound found by CP2 within 5 hours of computation time.

Problem	UB_{best}	UB_{init}	CP2		
			UB_{found}	nodes	time
Hur-7-1	(1048)	1487	(1052)	2677448	18000.0 s
Hur-7-2	1055	1839	(1055)	2916573	18000.0 s
Hur-7-3	1056	1839	(1056)	2993406	18000.0 s
Hur-7-4	1013	1418	1013	960092	5796.4 s
Hur-7-5	1000	1188	1000	775960	4420.6 s
Hur-7-6	1011	1545	(1011)	2897640	18000.0 s
Hur-7-7	1000	1419	1000	1628	8.8 s
Hur-7-8	1005	1510	1005	208340	1197.6 s
Hur-7-9	1003	1435	1003	1807635	10797.5 s
Tai-10-1	637	949	637	418594	5455.7 s
Tai-10-2	588	751	588	123104	2219.9 s
Tai-10-3	598	854	(607)	1025042	18000.0 s
Tai-10-4	577	856	577	64244	1175.8 s
Tai-10-5	640	1057	640	8173	126.0 s
Tai-10-6	538	770	(555)	1012887	18000.0 s
Tai-10-7	616	904	(827)	2136045	18000.0 s
Tai-10-8	595	853	595	164977	2255.7 s
Tai-10-9	595	880	595	6036	98.1 s
Tai-10-10	596	894	(639)	1162480	18000.0 s

Table 9.2.2 *Results for some larger instances (top-down).*

Regarding the instances of Taillard, CP2 solves 6 of them. The run times for all the instances that have been solved have a high standard deviation and vary from 2 minutes to 2 hours. This is because the optimal makespan may be hard to find, but once found it is easily verified and in all cases coincides with the trivial lower bound *LB*. Regarding the instances of Brucker et al., CP2 solves 5 instances, but none of the very hard instances Hur-7-1, Hur-7-2 and Hur-7-3. It finds, however, the optimal makespans of Hur-7-2 and Hur-7-3 without proof of optimality.

The results for CPS2 are much better. To the best of our knowledge, it is the first exact algorithm which solves all 10×10 OSP instances of Taillard. It even does so with an average run time of slightly above 10 minutes starting with a rather high upper bound. Further, CPS2 solves nearly all instances of Brucker et al. except the instance Hur7-1 which is still unsolved. The quality of CPS2 relies on the fact that the extensive application of constraint propagation results in a drastic reduction of the search tree. Quite impressively, the number of search tree

nodes generated by CPS2 on average only amounts to 0.1% of the number of nodes generated by CP2 . Therefore, the probability of getting lost in unfavourable regions of the search tree is significantly cut down. This underlines the importance and effectiveness of enhanced constraint propagation techniques for solving the OSP.

Problem	UB_{best}	UB_{init}	CPS2		
			UB_{found}	nodes	time
Hur-7-1	(1048)	1487	(1058)	4575	18000.0
Hur-7-2	1055	1839	1055	3364	9421.8
Hur-7-3	1056	1839	1056	3860	9273.5
Hur-7-4	1013	1418	1013	1123	2781.9
Hur-7-5	1000	1188	1000	742	1563.0
Hur-7-6	1011	1545	1011	5195	15625.1
Hur-7-7	1000	1419	1000	88	48.8
Hur-7-8	1005	1510	1005	209	318.8
Hur-7-9	1003	1435	1003	788	2184.9
Tai-10-1	637	949	637	612	1398.6
Tai-10-2	588	751	588	396	981.7
Tai-10-3	598	854	598	520	2664.3
Tai-10-4	577	856	577	496	847.1
Tai-10-5	640	1057	640	392	724.5
Tai-10-6	538	770	538	415	1101.5
Tai-10-7	616	904	616	565	982.8
Tai-10-8	595	853	595	461	837.3
Tai-10-9	595	880	595	222	655.1
Tai-10-10	596	894	596	562	993.8

Table 9.2.3 *Results for some larger instances using shaving (top-down).*

Up to now, we have applied a top-down solution approach which starts with an initial upper bound and tries to improve, i.e decrease this upper bound. As an alternative, we will now consider a bottom-up approach which starts with a lower bound as hypothetical upper bound and increases this bound by one time unit until a solution is found. The trivial job and machine based lower bound LB is chosen as an initial lower bound. For the computation of more sophisticated lower bounds which involves some search, we refer the reader to the work of Guéret and Prins [GP99].

The results for this approach are shown in Table 9.2.4. There are only the results for the best algorithm, namely CPS2. UB_{best} denotes the best lower bound found within a maximum run time of 5 hours. If LB_{best} is not written in parentheses then it also has been verified to be an upper bound. Again, all 10×10 instances of Taillard are solved, however, this time with an average run time of less than a minute. The results for the instances of Brucker are less impressive if compared with the top-down approach. Studying the work load WL of each instance, we can observe that the bottom-up approach is more efficient for instances with a lower work load, while the top-down approach shows better results for

those with a higher work load. This perfectly fits with the intuitive remarks made at the beginning of this section: instances with a lower work load tend to have an optimal makespan close or equal to LB, so that only a few lower bounds have to be tested in a bottom-up approach. On the contrary, instances with a higher work load tend to have an optimal makespan which is far from the initial lower bound. For these instances, the top-down approach is more efficient. Dorndorf et al. propose that the bottom-up approach is the favourite choice for problem instances with a work load of less than 0.9, while the top-down approach is to be preferred for instances with a work load greater than 0.95. For problem instances with a work load between 0.9 and 0.95, the situation is less clear, see e.g. the problem instance Hur-7-5 with a work load of 0.944 (bottom-up performs better) and Hur-7-9 with a work load of 0.925 (top-down performs better).

Problem	UB_{best}	LB	WL	CPS2		
				LB_{best}	nodes	time
Hur-7-1	(1048)	1000	1.000	(1021)	3974	18000.0
Hur-7-2	1055	1000	1.000	(1045)	5988	18000.0
Hur-7-3	1056	1000	1.000	(1042)	7057	18000.0
Hur-7-4	1013	1000	0.958	1013	5692	15178.1
Hur-7-5	1000	1000	0.944	1000	146	314.7
Hur-7-6	1011	1000	0.951	(1006)	5797	18000.0
Hur-7-7	1000	1000	0.879	1000	10	5.0
Hur-7-8	1005	1000	0.931	1005	194	625.5
Hur-7-9	1003	1000	0.925	1003	1376	4073.0
Tai-10-1	637	637	0.861	637	12	30.2
Tai-10-2	588	588	0.834	588	22	70.6
Tai-10-3	598	598	0.850	598	23	185.5
Tai-10-4	577	577	0.828	577	21	29.7
Tai-10-5	640	640	0.834	640	17	32.0
Tai-10-6	538	538	0.857	538	17	32.7
Tai-10-7	616	616	0.838	616	18	30.9
Tai-10-8	595	595	0.823	595	17	44.1
Tai-10-9	595	595	0.846	595	14	39.8
Tai-10-10	596	596	0.834	596	14	29.1

Table 9.2.4 *Results for some larger instances using shaving (bottom-up).*

Dorndorf et al. have also tested the bottom-up variant of CPS2 on the remaining 15×15 and 20×20 instances of Taillard for which no other results of exact solution approaches have been reported in literature. These results are shown in Table 9.2.5. Again, the bottom-up approach shows very good results. All 15×15 instances except one and 7 of the 20×20 instances have been solved, among others the instances Tai-15-5, Tai-15-9 and Tai-20-6 that have not been solved before. Except for the unsolved instances, the run time is always less than 12 minutes for the 15×15 instances and about an hour for the 20×20 instances.

Let us finally compare the results of the algorithms from [DPP01] with those of some other branch-and-bound algorithms for the OSP. B&B1 of Brucker et al.

[BHJW97] is a typical representative of their 6 slightly different algorithms and the algorithm B&Bi of Guéret et al. [GJP00], where 'i' stands for intelligent backtracking. These algorithms are compared with Dorndorf et al.'s combined top-down/bottom-up approach which works as follows: whenever the work load of an instance is at most 0.9, the bottom-up version of CPS2 is applied; for instances with a work load greater than 0.9, on the contrary, the top-down version of CPS2 is used.

Problem	UB_{best}	LB	WL	CPS2		
				LB_{best}	nodes	time
Tai-15-1	937	937	0.800	937	42	481.4 s
Tai-15-2	918	918	0.834	(918)	193	18000.0 s
Tai-15-3	871	871	0.824	871	44	611.6 s
Tai-15-4	934	934	0.794	934	45	570.1 s
Tai-15-5	946	946	0.842	946	34	556.3 s
Tai-15-6	933	933	0.795	933	51	574.5 s
Tai-15-7	891	891	0.828	891	52	724.6 s
Tai-15-8	893	893	0.813	893	46	614.0 s
Tai-15-9	899	899	0.830	899	36	646.9 s
Tai-15-10	902	902	0.824	902	34	720.1 s
Tai-20-1	1155	1155	0.820	1155	59	3519.8 s
Tai-20-2	1241	1241	0.838	(1241)	69	18000.0 s
Tai-20-3	1257	1257	0.803	1257	77	4126.3 s
Tai-20-4	1248	1248	0.825	(1248)	92	18000.0 s
Tai-20-5	1256	1256	0.809	1256	56	3247.3 s
Tai-20-6	1204	1204	0.810	1204	65	3393.0 s
Tai-20-7	1294	1294	0.807	1294	48	2954.8 s
Tai-20-8	(1171)	1169	0.854	(1169)	69	18000.0 s
Tai-20-9	1289	1289	0.800	1289	69	3593.8 s
Tai-20-10	1241	1241	0.817	1241	65	4936.2 s

Table 9.2.5 *Results for even larger instances using shaving (bottom-up).*

The results have been summarized in Table 9.2.6. A dash indicates that the corresponding data have not been available. Brucker et al. chose a time limit of 50 hours on a Sun 4/20 workstation, whereas Guéret et al. stopped the search after 250000 backtracks which according to their time measurements corresponds to approximately 3 hours on a Pentium PC with a clock pulse of 133 MHz. As the results have been established on different platforms, they have to be interpreted with care. However, especially regarding the Taillard instances, it seems fair to say that the algorithm of Dorndorf et al. has a much better performance. While neither B&B1 and B&Bi solve more than 3 of the 10×10 instances of Taillard to optimality, they solve all 10 instances in an average run time of less than a minute. Notice further that even the version of CPS2 which works in a purely top-down fashion as well solves all ten instances and that CP2 which does not use shaving all the same solves 6 instances to optimality. Since the branching schemes employed by all these exact algorithms are basically the same (except

for the branching order in the algorithm of Dorndorf et al. and the intelligent backtracking component in the algorithm of Guéret et al.), we can conclude that the application of strong constraint propagation techniques sheds a new light on the solvability of the OSP and allows to cope with instances of the OSP that formerly seemed intractable. Computational experiments on some famous test sets of benchmark problem instances taken from literature demonstrate the efficiency of this approach. For the first time, many problem instances are solved in a short amount of computation time.

Problem	UB_{best}	B&B1[a]		B&B1[b]		CPS2[c]	
		nodes	time	nodes	time	nodes	time
Hur-7-1	(1048)	-	>50 h	-	-	4575	>5 h
Hur-7-2	1055	-	35451.5 s	-	-	3364	9421.8 s
Hur-7-3	1056	-	176711.1 s	-	-	3860	9273.5 s
Hur-7-4	1013	-	77729.2 s	-	-	1123	2781.9 s
Hur-7-5	1000	-	6401.6 s	-	-	742	1563.0 s
Hur-7-6	1011	-	277271.1 s	-	-	5195	15625.1 s
Tai-10-1	637	-	>50 h	>250000	>3 h	12	30.2 s
Tai-10-2	588	44332	10671.5 s	>250000	>3 h	22	70.6 s
Tai-10-3	598	-	>50 h	>250000	>3 h	23	185.5 s
Tai-10-4	577	163671	40149.4 s	26777	-	21	29.7 s
Tai-10-5	640	-	>50 h	>250000	>3 h	17	32.0 s
Tai-10-6	538	-	>50 h	>250000	>3 h	17	32.7 s
Tai-10-7	616	-	>50 h	4843	-	18	30.9 s
Tai-10-8	595	-	>50 h	>250000	>3 h	17	44.1 s
Tai-10-9	595	97981	24957.0 s	245100	-	14	39.8 s
Tai-10-10	596	-	>50 h	>250000	>3 h	14	29.1 s

[a] Run time on a Sun 4/20 Workstation
[b,c] Run time on a Pentium II/133

Table 9.2.6 *A comparison of computational results.*

References

AC82 J. O. Achugbue, F. Y. Chin, Scheduling the open shop to minimize mean flow time, *SIAM J. Comput.* 11, 1982, 709-720.

AC91 D. Applegate, W. Cook, A computational study of the job shop scheduling problem, *ORSA Journal on Computing* 3, 1991, 149-156.

BHJW97 P. Brucker, J. Hurink, B. Jurisch, B. Wöstmann, A branch and bound algorithm for the open shop problem, *Discret Appl. Math.* 76, 1997, 43-59.

BJK94 P. Brucker, B. Jurisch, A. Krämer, The job shop problem and immediate selection, *Ann. Oper. Res.* 50, 1994, 73-114.

BJS94 P. Brucker, B. Jurisch, B. Sievers, A fast branch and bound algorithm for the job shop scheduling problem, *Discret Appl. Math.* 49, 1994, 107-127.

BL95 P. Baptiste, C. Le Pape, A theoretical and experimental comparison of constraint propagation techniques for disjunctive scheduling, *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Montreal, 1995, 136-140.

BL96 P. Baptiste, C. Le Pape, Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling, *Proceedings of the 15th Workshop of the U. K. Planning Special Interest Group*, Liverpool, 1996.

Blu05 C. Blum, Beam-ACO – Hybridizing ant colony optimization with beam search: An application to open shop scheduling, *Comput. Oper. Res.* 32, 2005, 1565-1591.

BPS00 J. Błażewicz, E. Pesch, M. Sterna, The disjunctive graph machine representation of the job shop scheduling problem, *Eur. J. Oper. Res*. 127, 2000, 317-331.

BS04 C. Blum, M. Sampels, An ant colony optimization algorithm for shop scheduling problems, *Journal of Mathematical Modelling and Algorithms* 3, 2004, 285-308.

BTW93 H. Bräsel, T. Tautenhahn and F. Werner, Constructive heuristic algorithms for the open shop problem, *Computing* 51, 1993, 95-110.

CA05 S. Colak, A. Agarwal, Non-greedy heuristics and augmented neural networks for the open-shop scheduling problem, *Nav. Res. Logist.* 52, 2005, 631-644.

CL95 Y. Caseau, F. Laburthe, Disjunctive scheduling with task intervals, Technical report 95-25, Laboratoire d'Informatique de l'Ecole Normale Superieure, Paris, 1995.

CP89 J. Carlier, E. Pinson, An algorithm for solving the job shop problem, *Manage. Sci.* 35, 1989, 164-176.

CP90 J. Carlier, E. Pinson, A practical use of Jackson's preemptive schedule for solving the job shop problem, *Ann. Oper. Res.* 26, 1990, 269-287, 1990.

CP94 J. Carlier, E. Pinson, Adjustments of heads and tails for the job shop problem, *Eur. J. Oper. Res.* 78, 1994, 146-161.

CS81 Y. Cho, S. Sahni, Preemptive scheduling of independent jobs with release and due times on open, flow and job shops, *Oper. Res.* 29, 1981, 511-522.

DPP99 U. Dorndorf, T. Phan-Huy, E. Pesch, A survey of interval capacity consistency tests for time and resource constrained scheduling, in: J. Weglarz (ed.), *Project Scheduling - Recent Models, Algorithms and Applications*, Kluwer Academic Publishers, Boston, 1999, 213-238.

DPP00 U. Dorndorf, E. Pesch, T. Phan-Huy, Constraint propagation techniques for disjunctive scheduling problems, *Artif. Intell.* 122, 2000, 189-240.

DPP01 U. Dorndorf, E. Pesch, T. Phan-Huy, Solving the open shop scheduling problem, *J. Sched.* 4, 2001, 157-174.

GJP00 C. Gueret, N. Jussien, C. Prins, Using intelligent backtracking to improve branch and bound methods: an application to open shop problems, *Eur. J. Oper. Res*. 127, 2000, 344-354.

GP98a C. Gueret, C. Prins. Classical and new heuristics for the open shop problem: a computational evaluation, *Eur. J. Oper. Res.* 107, 1998, 306-314, 1998.

GP98b C. Gueret, C. Prins. Forbidden intervals for open-shop problems, Research report 98/10/AUTO, Ecole de Mines de Nantes, Nantes, 1998.

GP99 C. Gueret, C. Prins, A new lower bound for the open-shop problem, *Ann. Oper. Res.* 92, 1999, 165-183.

GS76 T. Gonzalez, S. Sahni, Open shop scheduling to minimize finish time, *J. ACM* 23, 1976, 665-679.

Jac56 J. Jackson, An extension on Johnson's results on job lot scheduling, *Nav. Res. Logist. Quart.* 3, 1956, 201-203.

KSZ91 W. Kubiak, C. Srishkandarajah, K. Zaras, A note on the complexity of open shop scheduling problems, *Infor* 29, 1991, 284-294.

Lia98 C. F. Liaw, An iterative improvement approach for nonpreemptive open shop scheduling problem, *Eur. J. Oper. Res.* 111, 1998, 509-517.

LLRK81 E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, Minimizing maximum lateness in a two-machine open shop, *Math. Oper. Res.* 6, 1981, 153-158 (Erratum: *Math. Oper. Res.* 7, 1982, 635).

MS96 P. Martin, D. B. Shmoys, A new approach to computing optimal schedules for the job shop scheduling problem, *Proceedings of the 5th International Conference on Integer Programming and Combinatorial Optimization,* 1996.

Nui94 W. P. M. Nuijten, *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach*, Ph.D. thesis, Eindhoven University of Technology, 1994.

Pha00 T. Phan-Huy, *Constraint Propagation in Flexible Man*ufacturing, Springer, Heidelberg, 2000.

Pri00 C. Prins, Competitive genetic algorithms for the open-shop scheduling problem *Math. Meth. Oper. Res.* 52, 2000, 389-411.

RM96 A. Ramudhin, P. Marier, The generalized shifting bottleneck procedure, *Eur. J. Oper. Res.* 93, 1996, 34-48.

RS64 B. Roy, B. Sussmann, Les problemes d`ordonnancement avec contraintes disjonctives, Note D. S. 9, SEMA, Paris, 1964.

Tai93 E. Taillard, Benchmarks for basic scheduling problems, *Eur. J. Oper. Res.*, 64, 1993, 278-285.

Tsa93 E. Tsang, *Foundations of Constraint Satisfaction*. Academic Press, Essex, 1993.

WW95 F. Werner, A. Winkler, Insertion techniques for the heuristic solution of the job shop problem, *Discret Appl. Math.* 58, 1995, 191-211.