A Computational Study of Constraint Satisfaction for Multiple Capacitated Job Shop Scheduling*

W.P.M. Nuijten[1] E.H.L. Aarts[2,3]

[1] ILOG S.A., 2 Avenue Galliéni, BP 85, F-94253 Gentilly Cedex
[2] Eindhoven University of Technology, Department of Mathematics and Computing Science, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
[3] Philips Research Laboratories, P.O. Box 80000, 5600 JA Eindhoven, The Netherlands

Abstract

We introduce the multiple capacitated job shop scheduling problem as a generalization of the job shop scheduling problem. In this problem machines may process several operations simultaneously. We present an algorithm based on constraint satisfaction techniques to handle the problem effectively. The most important novel feature of our algorithm is the consistency checking. An empirical performance analysis is performed using a well-known set of instances of the job shop scheduling problem and a newly constructed set of instances of the multiple capacitated job shop scheduling problem. We show that our algorithm performs well for both sets of instances.

1 Introduction

We are concerned with a generalization of the Job Shop Scheduling Problem (JSSP) [French, 1982] which we call the Multiple Capacitated Job Shop Scheduling Problem (MCJSSP). Informally, the problem can be stated as follows. Given are a set of *jobs* and a set of *machines*. Each job consists of a set of *operations* that must be processed in a given order. Furthermore, each operation is given an integer *processing time* and a machine by which it has to be processed. Once an operation is started, it is processed without interruption. Machines may process several operations simultaneously. To this end, each machine is given an integer *capacity*, and each operation is given an integer *size*. A machine can simultaneously process only those sets of operations whose sizes do not exceed the capacity of the machine. Finally, a fixed

*Accepted for publication in the European Journal of Operational Research

overall deadline is given. A *schedule* assigns a start time to each operation. One is asked to find a schedule, if it exists, in which the capacity of none of the machines is exceeded and which meets the overall deadline and the order in which the operations must be processed.

Over the years, the development of scheduling algorithms has been the subject of extensive research. Much of this research is focused on the JSSP in which each machine can only process one operation at a time and each operation is processed by a predetermined machine. However, in many practical situations (1) machines are capable of processing more than one operation at a time and (2) operations can be processed by alternative machines. Obviously, problems having property (1) are included in the MCJSSP. We remark that a subset of scheduling problems having property (2) is also included in the MCJSSP. Suppose we are given a set of operations, all of size 1, and a set of machines, all with capacity 1, that can be partitioned into subsets $O_1, \ldots, O_n$ and $M_1, \ldots, M_n$, respectively, such that all operations in O_j can be processed by any of the machines in M_j, for $1 \leq j \leq n$. This can be translated into an instance of the MCJSSP with n machines m_j, each with capacity $|M_j|$, in which all operations in O_j have size 1 and must be processed by m_j.

The MCJSSP can be seen as a special case of the Constraint Satisfaction Problem (CSP) [Montanari, 1974] as is shown in Section 2. An instance of the CSP involves a set of variables, a domain for each variable specifying the values to which it may be assigned, and a set of constraints on the variables which define allowed domain value combinations. One is asked to assign values to variables such that all constraints are simultaneously satisfied.

This paper belongs to a series of papers in which we report on our investigations of the potentials of constraint satisfaction techniques for scheduling. In this paper we concentrate on the MCJSSP. We extend [Nuijten & Aarts, 1994] both on the consistency checking techniques and on the computational results that are presented. Results for the JSSP can be found in [Nuijten, Aarts, Van Erp Taalman Kip & Van Hee, 1993]. Nuijten, Kunnen, Aarts & Dignum [1994] report on results for a practical time tabling problem. The organization of the paper is as follows. Section 2 defines the MCJSSP and the CSP and their relation. Section 3 discusses our constraint satisfaction approach to the MCJSSP. Finally, Section 5 presents the computational results.

2 The Multiple Capacitated Job Shop Scheduling Problem

Definition 1. An instance of the *Multiple Capacitated Job Shop Scheduling Problem* consists of a set $\mathcal{O}$ of N operations, a set $\mathcal{J}$ of n jobs, a set $\mathcal{M}$ of m machines, and an overall deadline $D \in \mathbb{Z}^+$. Furthermore, for each operation

$o \in \mathcal{O}$, $J(o)$ denotes the job to which o belongs, $M(o)$ is the machine by which o must be processed, $pt(o) \in \mathbb{Z}^+$ is the processing time of o, and $sz(o) \in \mathbb{Z}_0^+$ is the size of o. The capacity of a machine $\mu \in \mathcal{M}$ is given by $cp(\mu) \in \mathbb{Z}_0^+$ and a binary relation $\prec$ is given that decomposes $\mathcal{O}$ into chains, such that every chain corresponds to a job. A *schedule* is a function $s : \mathcal{O} \rightarrow \mathbb{Z}_0^+$ giving the start times of the operations. The problem is to find a schedule s such that for all $o, o' \in \mathcal{O}$, $s(o) \geq 0$, $s(o) + pt(o) \leq D$, and $s(o) + pt(o) \leq s(o')$ if $o \prec o'$, and such that for all $\mu \in \mathcal{M}$ and all $t \in [0, D]$:

$$\sum_{o \in \mathcal{O}_\mu | s(o) \leq t < s(o) + pt(o)} sz(o) \leq cp(\mu),$$

where $\mathcal{O}_\mu = \{o \in \mathcal{O} \mid M(o) = \mu\}$. The set of all schedules is denoted by $\mathcal{S}$ □

We refer to constraints defined on operations that are related by the binary relation as *precedence constraints*, and to constraints defined on operations that must be scheduled on the same machine as *capacity constraints*. Remark that the decision variant of the MCJSSP is NP-complete, as (i) it is in NP, because for a given schedule s all constraints can be checked in polynomial time, and (ii) the decision variant of the JSSP is an NP-complete special case of the decision variant of MCJSSP [Garey, Johnson & Sethi, 1976]. Next, we need the definition of the CSP; see also [Nuijten, 1994].

Definition 2. An instance of the *Constraint Satisfaction Problem* consists of a set $X = \{x_1, \ldots, x_p\}$ of variables, a domain $D(x_i)$ specifying for each variable x_i the values to which it may be assigned, and a set $C = \{c_1, \ldots, c_q\}$ of constraints on the variables, where $c_i : D(x_1) \times \ldots \times D(x_p) \rightarrow \{true, false\}$, for $i = 1, \ldots, q$. The problem is to find an *assignment* $a \in D(x_1) \times \ldots \times D(x_p)$ such that $\forall_{1 \leq i \leq q}\ c_i(a) = true$. □

Below, we treat the constraints of Definition 2 as predicates, e.g., $c_i(a)$ is used instead of $c_i(a) = true$ and $\neg c_i(a)$ instead of $c_i(a) = false$. Although in Definition 2 constraints are defined involving all variables, a constraint is often only concerned with two variables. Such a *binary* constraint, relating variable x to variable x', is denoted by $c_{\{x,x'\}}$.

The MCJSSP is a special case of the CSP, as can easily be seen as follows. For each operation $o \in \mathcal{O}$, a variable is introduced, i.e., $\mathcal{O}$ specifies the set of variables. Furthermore, for each operation a domain $D(o) = [0, D \Leftrightarrow pt(o)]$ is defined, specifying its start times. Note that with $\mathcal{O} = \{o_1, \ldots, o_N\}$, an assignment $(a_1, \ldots, a_N) \in D(o_1) \times \ldots \times D(o_N)$ corresponds to a schedule $s : \mathcal{O} \rightarrow \mathbb{Z}_0^+$ for which $s(o_i) = a_i$, $1 \leq i \leq N$. In the sequel, we therefore

use constraints that are defined on schedules, i.e., a constraint c is a function $c : \mathcal{S} \rightarrow \{true, false\}$. We denote a binary constraint defined on the variables of operations o and o' by $c_{\{o,o'\}}$. For the MCJSSP we have the following set of constraints. For each pair of operations $o, o' \in \mathcal{O}$ such that $o \prec o'$, we define a precedence constraint

$$c_{\{o,o'\}}(s) \Leftrightarrow s(o) + pt(o) \leq s(o'),$$

where $s \in \mathcal{S}$ is a schedule. Furthermore, for each machine $\mu \in \mathcal{M}$, we define a capacity constraint c_μ such that

$$c_\mu(s) \Leftrightarrow \forall_{t \in [0,D]} \sum_{o \in \mathcal{O}_\mu | s(o) \leq t < s(o) + pt(o)} sz(o) \leq cp(\mu).$$

3 A constraint satisfaction approach

We use a constraint satisfaction approach that is based on tree search algorithms. In general, constraint satisfaction tree search algorithms can be described by the framework of Figure 1.

```
while not solved and not infeasible do
    check consistency
    if a dead end is detected then
        try to escape from dead end
    else
        select variable
        select value for variable
    endif
endwhile
```

Figure 1: Framework of constraint satisfaction tree search algorithms.

Each node in the search tree corresponds to a partial solution and going from one node to another is done by assigning a value to a variable. The selection of a next variable and its value is done by *variable* and *value selection heuristics*, here referred to as operation and start time selection heuristics, respectively. After a value is assigned to a variable, *inconsistent* values of unassigned variables are removed. A value $v \in D(x)$ of a variable x is inconsistent if no solution exists with the assignment of v to x, in addition to the assignments made so far. We call the process of removing inconsistent values *consistency checking*. For a variable x, the *current domain* $\delta(x)$ is the set of values of which we cannot prove that they are inconsistent, using the consistency checking we have available. We remark that, in general, it is impossible

to efficiently remove all inconsistent values. If, by removing inconsistent values from the current domains, a current domain becomes empty, we say a *dead end* occurs. Then, one or more assignments must be undone to try alternatives. An instance is solved if every variable is assigned a value, and an instance is proven to be infeasible if for a variable in the root of the tree no values are left to be tried.

Our research focuses on finding efficient ways to effectively identify inconsistent values. In [Nuijten, Aarts, Van Erp Taalman Kip & Van Hee, 1993] it is shown that for the JSSP this leads to a high quality scheduling algorithm. The consistency checking we use for the MCJSSP is discussed in Section 4. The relative simple operation and start time selection heuristics and dead end handling are described below.

Operation and start time selection. Based on the fact that every schedule can be transformed into a left justified schedule in which the operations are scheduled as early as possible while preserving the precedence constraints and the machine orderings, we use operation and start time selection that construct left justified schedules. Furthermore, for reasons explained below, we introduce randomization. For operation selection, we determine the earliest minimal completion time of any unscheduled operation and then randomly select one operation that can be started before this completion time. The start time is chosen as the earliest possible start time of this operation.

Dead end handling. To escape from dead ends, we employ an approach that consists of two parts. First we try to solve the instance at hand by using *chronological backtracking*, i.e., undoing the last assignment and trying an alternative. However, if this does not lead to a solution after a reasonable number of backtrack steps, we use the most rigorous escape facility that is conceivable, namely the complete restart of the search. A problem then is to direct the search along a path different from the ones followed previously. Inspired by recent successes of randomized search methods, we combine restarting the search with the randomized operation and start time selection described above. In this way, the probability of following the same search path more than once is very small since the number of possible paths is usually very large.

4 Consistency checking

Before we discuss the consistency checking for the MCJSSP, we introduce the following notations. For each operation $o \in \mathcal{O}$, $\delta(o)$ is the current domain of the variable of o, $est(o) = \min(\delta(o))$ is the *earliest start time*, $ect(o) = est(o) + pt(o)$ is the *earliest completion time*, $lst(o) = \max(\delta(o))$ is the *latest*

start time, and $lct(o) = lst(o) + pt(o)$ is the *latest completion time* of o. Below, we briefly elaborate on *forward checking* [Haralick & Elliott, 1980] and on *arc consistency* [Mackworth, 1977].

4.1 Forward Checking

This section is concerned with achieving consistency of the domains of all unassigned operations with the domains of all assigned operations. This form of consistency checking is known as forward checking. We refer to the set of assigned operations as $\mathcal{AO}$ and to the set of unassigned operations as $\mathcal{UO}$. Performing forward checking for precedence constraints is rather trivial and is described by Nuijten & Aarts [1994]. Here, we concentrate on the capacity constraints. Recall the capacity constraints which require that for all $\mu \in \mathcal{M}$ and $t \in [0, D]$,

$$\sum_{o \in \mathcal{O}_\mu | s(o) \leq t < s(o)+pt(o)} sz(o) \leq cp(\mu).$$

For each machine $\mu \in \mathcal{M}$ we introduce a function $rcp_\mu : \mathbb{N} \rightarrow \mathbb{N}$ that gives the remaining capacity of μ as a function of time. Let $t \in \mathbb{N}$, then we define

$$rcp_\mu(t) = cp(\mu) \Leftrightarrow \sum_{o \in \mathcal{O}_\mu | o \in \mathcal{AO} \wedge s(o) \leq t < s(o)+pt(o)} sz(o).$$

Obviously, if for an unassigned operation the remaining capacity of a machine at a point in time is smaller than the size of the operation, then the operation cannot be scheduled using that point in time.

Theorem 1. *Let $\mu \in \mathcal{M}$ and $o \in \mathcal{UO}$. Then, with*

$$\{t \in [0, D) \mid rcp_\mu(t) < sz(o)\} = [a_1, b_1] \cup \ldots \cup [a_x, b_x],$$

all start times in $(a_1 \Leftrightarrow pt(o), b_1] \cup \ldots \cup (a_x \Leftrightarrow pt(o), b_x]$ are inconsistent for o.

Proof. Follows directly from the definition of rcp_μ. □

We refer to the consistency checking algorithm that performs forward checking as ForwardCheck.

4.2 Arc consistency.

Arc consistency can be defined for the MCJSSP as follows.

Definition 3. Let $o, o' \in \mathcal{O}$, with current domains $\delta(o)$ and $\delta(o')$ respectively, and let $c_{\{o,o'\}}$ be a binary constraint. Then, $\delta(o)$ is *arc consistent* with $\delta(o')$ for $c_{\{o,o'\}}$, if and only if for all $t \in \delta(o)$, there exists a $t' \in \delta(o')$, such that for a schedule $s \in \mathcal{S}$ with $s(o) = t$ and $s(o') = t'$, $c_{\{o,o'\}}(s)$ holds. □

Achieving arc consistency for precedence constraints is rather trivial and is described by Nuijten & Aarts [1994]. Here ,we concentrate on arc consis-

tency of capacity constraints. For $o, o' \in \mathcal{O}$, with $M(o) = M(o') = \mu$, the corresponding capacity constraint implies the following binary constraint:

$$c_{\{o,o'\}}(s) \Leftrightarrow \forall_{t \in [0,D]} \sum_{o'' \in \{o,o'\} | s(o'') \leq t < s(o'') + pt(o'')} sz(o'') \leq cp(\mu).$$

Observe that $c_{\{o,o'\}}$ always holds if $sz(o) + sz(o') \leq cp(\mu)$, i.e., if there is enough capacity to process o and o' simultaneously. Otherwise, $c_{\{o,o'\}}$ can be redefined as

$$c_{\{o,o'\}}(s) \Leftrightarrow s(o) + pt(o) \leq s(o') \ \vee \ s(o') + pt(o') \leq s(o).$$

These constraints correspond to the classical disjunctive constraints found in the JSSP. The following theorem is concerned with the general conditions under which these constraints are arc consistent.

Theorem 2. *Let $o, o' \in \mathcal{O}$, $s \in \mathcal{S}$, and $c_{\{o,o'\}}(s) \Leftrightarrow s(o) + pt(o) \leq s(o') \vee s(o') + pt(o') \leq s(o)$. Then, $\delta(o)$ is arc consistent with $\delta(o')$ for $c_{\{o,o'\}}$, if and only if $\delta(o) \cap (lst(o') \Leftrightarrow pt(o), ect(o')) = \emptyset$.* □

Arc consistency of $c_{\{o,o'\}}$ of Theorem 2 can be achieved by deleting the start times in $(lst(o') \Leftrightarrow pt(o), ect(o'))$ from $\delta(o)$. For a proof of Theorem 2 we refer to [Nuijten, Aarts, Van Erp Taalman Kip & Van Hee, 1993].

We refer to the consistency checking algorithm that achieves arc consistency for both the precedence constraints and the capacity constraints as ArcConsCheck.

4.3 Sequencing checking.

This section presents three different lower bounds on the earliest start time and three different upper bounds on the latest completion time of operations. These bounds are used to identify inconsistent assignments for operations. We refer to this way of consistency checking as *sequencing checking*, as we sequence basically some of the operations, i.e., we determine some necessary orderings of operations.

LB_{est} and UB_{lct}. Nuijten, Aarts, Van Erp Taalman Kip & Van Hee [1993] identify conditions for which it can be proven that an operation must be scheduled before or after a specific subset of operations on the same machine for machines with capacity 1. Here, we extend this to machines with arbitrary capacity. The idea of this extension is the following. For machines with capacity 1, only one dimension is regarded, namely the time. In case of arbitrary capacities, we add the capacity as a second dimension. Instead of the processing time $pt(o)$ of an operation, we consider the *area* an operation uses, defined as $pt(o) \cdot sz(o)$. Before continuing, we introduce the following notations,

for each $o \in \mathcal{O}$, $a(o) = pt(o) \cdot sz(o)$, and for each $S \subseteq \mathcal{O}$, $A(S) = \sum_{o \in S} a(o)$, $e(S) = \min_{o \in S} est(o)$, and $C(S) = \max_{o \in S} lct(o)$.

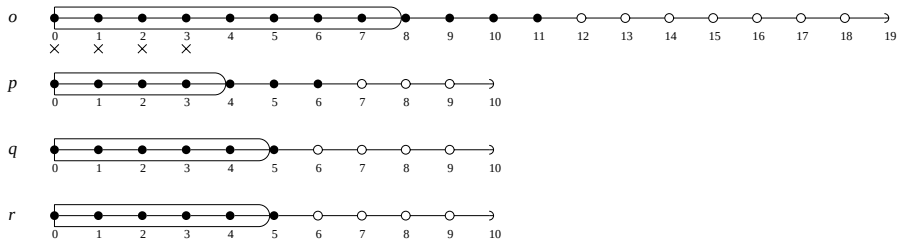

Figure 2: Four operations on the same machine with capacity 2.

Figure 2 shows four operations each of size 1, that must be scheduled on a machine with capacity 2. Possible start times are denoted by a '$\bullet$' and times at which an operation cannot be started although the operation may be processed at that time are denoted by a '$\circ$'. Furthermore the processing time is indicated by a bar starting at the earliest start time. The '$\times$'s indicate that these start times are inconsistent.

Observe that the sum of the areas of the operations is 22, whereas the available area between times 0 and 10 is 20. This implies that o cannot be started at time 0. The same holds for times 1 to 3. What we can actually deduce is that no operation in $S = \{p, q, r\}$ can succeed o.

The following theorem provides conditions for which an operation o either cannot precede or cannot succeed any of the operations of a set S that are all to be scheduled on the same machine with arbitrary capacity.

Theorem 3. *Let $\mu \in \mathcal{M}$, $o \in \mathcal{O}_\mu$, and $S \subset \mathcal{O}_\mu$, be such that $o \notin S$. Then, (i) if*

$$A(S \cup \{o\}) > (C(S) \Leftrightarrow e(S \cup \{o\})) \cdot cp(\mu), \tag{1}$$

then no schedule exists in which o precedes any of the operations in S, and (ii) if

$$A(S \cup \{o\}) > (C(S \cup \{o\}) \Leftrightarrow e(S)) \cdot cp(\mu),$$

then no schedule exists in which o succeeds any of the operations in S.

Proof. (i) Suppose all operations are scheduled and o precedes an operation in S. Then $s(o) + pt(o) < C(S)$ and thus the area that is needed between $e(S \cup \{o\})$ and $C(S)$ is at least $A(S \cup \{o\})$ which, according to (1), is larger than the available area $(C(S) \Leftrightarrow e(S \cup \{o\})) \cdot cp(\mu)$, which leads to a contradiction. The proof of (ii) is similar to the proof of (i) □

If no schedule exists in which an operation o precedes any of the operations in $S \subset \mathcal{O}_\mu$, we can adjust the earliest start time of o as follows. Let σ be a search state, we then define for each schedule $s \in \mathcal{F}(\sigma)$,

$$ct(S, s, j) = \min\{t \geq e(S) \mid \forall_{t' \geq t} \sum_{o' \in S \,\wedge\, s(o') \leq t' < s(o') + pt(o')} sz(o') \leq cp(\mu) \Leftrightarrow j\}$$

as the earliest time after which, in schedule s, at least a capacity of j is available on machine μ. A lower bound on $est(o)$ is

$$\min_{s \in \mathcal{F}(\sigma)} ct(S, s, sz(o)).$$

The problem of calculating this lower bound is NP-hard as is shown in the following theorem.

Theorem 4. *Let $\mu \in \mathcal{M}$, $o \in \mathcal{O}_\mu$, and $S \subset \mathcal{O}_\mu$, be such that $o \notin S$. The problem of calculating*

$$\min_{s \in \mathcal{F}(\sigma)} ct(S, s, sz(o))$$

is NP-hard.

Proof. It can easily be seen that the optimization variant of the Sequencing Problem with Release Times and Deadlines [Garey & Johnson, 1979], is a special case of this problem by defining $sz(o) = cp(\mu)$ and $sz(o') = cp(\mu)$, for all $o' \in S$. □

Next, we introduce the following lower bound which can be computed efficiently. Let $S \subset \mathcal{O}$ be given, then the total area to be scheduled in $[e(S), C(S))$ is $A(S)$. We can, at best, schedule an area of $(C(S) \Leftrightarrow e(S)) \cdot (cp(\mu) \Leftrightarrow sz(o))$, without disabling any start time of o. The rest of $A(S)$ is then scheduled as early as possible. All this corresponds to preemptively scheduling the operations in S as early as possible. With

$$rest(S, j) = A(S) \Leftrightarrow (C(S) \Leftrightarrow e(S)) \cdot (cp(\mu) \Leftrightarrow j),$$

we derive that if $rest(S, sz(o)) > 0$, a lower bound on the earliest completion time of all operations in S, and thus a lower bound on $est(o)$, equals

$$e(S) + \lceil rest(S, sz(o))/sz(o) \rceil.$$

The condition $rest(S, sz(o)) > 0$ states that the total area $A(S)$ that needs to be scheduled in $[e(S), C(S))$ is strictly larger than the area $(C(S) \Leftrightarrow e(S)) \cdot (cp(\mu) \Leftrightarrow sz(o))$, which is the area that can be scheduled without disabling any start time of o.

Evidently, if for a machine $\mu \in \mathcal{M}$, no schedule exists in which an operation $o \in \mathcal{O}_\mu$ precedes any of the operations in a set $S \subset \mathcal{O}_\mu$, then also no schedule exists in which o precedes any of the operations in any subset $S' \subseteq S$. This

leads to the observation that all start times in $\delta(o)$ smaller than

$$\max_{S' \subseteq S | S' \neq \emptyset \ \wedge \ rest(S', sz(o)) > 0} e(S') + \lceil rest(S', sz(o))/sz(o) \rceil,$$

are inconsistent for o. If we do this for all $S \subseteq \mathcal{O}_\mu$, we obtain

$$LB_{est}(o) = \max_{S \subseteq \mathcal{O}_\mu | \alpha} \ \max_{S' \subseteq S | S' \neq \emptyset \ \wedge \ rest(S', sz(o)) > 0} e(S') + \lceil rest(S', sz(o))/sz(o) \rceil,$$

with

$$\alpha \Leftrightarrow o \notin S \ \wedge \ A(S \cup \{o\}) > (C(S) \Leftrightarrow e(S \cup \{o\})) \cdot cp(\mu),$$

as a lower bound on the earliest start time of operation o.

Similarly, if for a machine $\mu \in \mathcal{M}$ no schedule exists in which an operation $o \in \mathcal{O}_\mu$ succeeds any of the operations in a set $S \subset \mathcal{O}_\mu$, then all start times in $\delta(o)$ larger than

$$\min_{S' \subseteq S | S' \neq \emptyset \ \wedge \ rest(S', sz(o)) > 0} C(S') \Leftrightarrow \lceil rest(S', sz(o))/sz(o) \rceil \Leftrightarrow pt(o),$$

are inconsistent for o. If we do this for all $S \subseteq \mathcal{O}_\mu$, we obtain

$$UB_{lct}(o) = \min_{S \subseteq \mathcal{O}_\mu | \alpha} \ \min_{S' \subseteq S | S' \neq \emptyset \ \wedge \ rest(S', sz(o)) > 0} C(S') \Leftrightarrow \lceil rest(S', sz(o))/sz(o) \rceil,$$

with

$$\alpha \Leftrightarrow o \notin S \ \wedge \ A(S \cup \{o\}) > (C(S) \Leftrightarrow e(S)) \cdot cp(\mu)$$

as an upper bound on the latest completion time of operation o. We remark that the lower bounds on earliest start times and upper bounds on latest completion times given by Carlier & Pinson [1990] for the JSSP, are special cases of LB_{est} and UB_{lct}.

Nuijten [1994] presents an algorithm for calculating LB_{est} and UB_{lct} with a time complexity equal to $O(|SzSet_\mu| \cdot |\mathcal{O}_\mu|^2)$, where $SzSet_\mu = \{sz(o) \mid o \in \mathcal{O}_\mu\}$. [Nuijten, 1995] shows how to change this algorithm into an equivalent algorithm with a time complexity equal to $O(|\mathcal{O}_\mu|^2)$.

$LB2_{est}$ and $UB2_{lct}$. The following theorem provides conditions for which an operation o must be scheduled either after at least one operation or before at least one operation of a set S of operations that are all to be scheduled on the same machine with arbitrary capacity.

Theorem 5. *Let $\mu \in \mathcal{M}$, $o \in \mathcal{O}_\mu$, and $S \subset \mathcal{O}_\mu$, be such that $o \notin S$. Then, (i) if*

$$e(S) < est(o) < \min_{o' \in S} ect(o'), \text{ and}$$

$$A(S) + (\min(ect(o), C(S)) \Leftrightarrow e(S)) \cdot sz(o) > (C(S) \Leftrightarrow e(S)) \cdot cp(\mu), \qquad (2)$$

then all $t \in \delta(o)$, for which $t < \min_{o' \in S} ect(o')$, are inconsistent, and (ii) if

$$\max_{o' \in S} lst(o') < lct(o) < C(S), \text{ and}$$

$$A(S) + (C(S) \Leftrightarrow \max(lst(o), e(S))) \cdot sz(o) > (C(S) \Leftrightarrow e(S)) \cdot cp(\mu),$$

then all $t \in \delta(o)$*, for which* $t > \max_{o' \in S} lst(o') \Leftrightarrow pt(o)$*, are inconsistent.*

Proof. (i) If o is started at time $est(o)$, then no operation of S can be scheduled to be completed before o as $est(o) < \min_{o' \in S} ect(o')$. Thus an area $(est(o) \Leftrightarrow e(S)) \cdot sz(o)$ is occupied by no operation. Then the area the operations in $S \cup \{o\}$ occupy in $[e(S), C(S))$ is

$$A(S) + (\min(ect(o), C(S)) \Leftrightarrow e(S)) \cdot sz(o),$$

which, with (2) is larger than the available area $(C(S) \Leftrightarrow e(S)) \cdot cp(\mu)$. Thus, $est(o)$ is inconsistent for o, and the same holds for all start times in $\delta(o)$ smaller than $\min_{o' \in S} ect(o')$. The proof of (ii) is similar to the proof of (i) □

From Theorem 5 we derive

$$LB2_{est}(o) = \max_{S \subset \mathcal{O}_\mu | \alpha} \min_{o' \in S} ect(o'),$$

with

$$\alpha \Leftrightarrow o \notin S \ \wedge \ e(S) < est(o) < \min_{o' \in S} ect(o) \ \wedge$$
$$A(S) + (\min(ect(o), C(S)) \Leftrightarrow e(S)) \cdot sz(o) > (C(S) \Leftrightarrow e(S)) \cdot cp(\mu),$$

as a lower bound on the earliest start time of operation o. Similarly, we derive

$$UB2_{lct}(o) = \min_{S \subset \mathcal{O}_\mu | \alpha} \max_{o' \in S} lst(o'),$$

with

$$\alpha \Leftrightarrow o \notin S \ \wedge \ \max_{o' \in S} lst(o') < lct(o) < C(S) \ \wedge$$
$$A(S) + (C(S) \Leftrightarrow \max(lst(o), e(S))) \cdot sz(o) > (C(S) \Leftrightarrow e(S)) \cdot cp(\mu),$$

as an upper bound on the latest completion time of operation o.

Nuijten [1995] presents an algorithm for calculating $LB2_{est}$ and $UB2_{lct}$ with a time complexity equal to $O(|\mathcal{O}_\mu|^3)$.

$LB3_{est}$ and $UB3_{lct}$. The following example introduces another lower bound on the earliest start time of an operation.

Figure 3 shows four operations of size 1 which must be scheduled on a machine μ with capacity 2. Observe that condition (1) of Theorem 3 does not hold for o and $S = \{p, q, r\}$, as the sum of the areas of the operations between times 0 and 11 is 22, whereas the available area between times 0 and 11 is also 22. Therefore, $A(S \cup \{o\}) = (C(S) \Leftrightarrow e(S \cup \{o\})) \cdot cp(\mu)$, and no deduction can be made by using condition (1) of Theorem 3. However, if we focus on the interval $[e(S), C(S))$, we observe that when o is scheduled as early as possible,

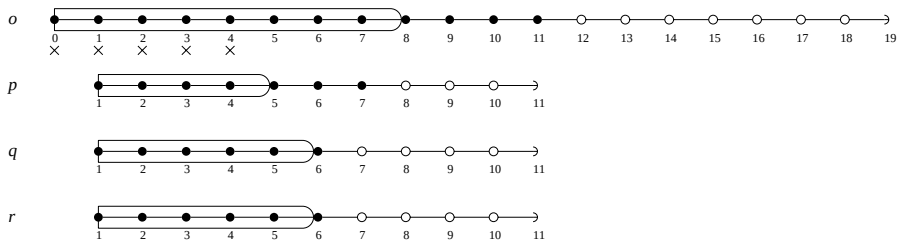

Figure 3: Four operations on the same machine with capacity 2.

the sum of the areas of the operations between 1 and 11 is 21, whereas the available area is 20. Again this implies that no operation in $S = \{p, q, r\}$ can succeed o.

The following theorem provides conditions for which an operation o either cannot precede or cannot succeed any of the operations of a set S that are all to be scheduled on the same machine with arbitrary capacity.

Theorem 6. *Let $\mu \in \mathcal{M}$, $o \in \mathcal{O}_\mu$, and $S \subset \mathcal{O}_\mu$, be such that $o \notin S$. Then (i) if*

$$est(o) < e(S) < ect(o), \text{ and}$$
$$A(S) + (ect(o) \Leftrightarrow e(S)) \cdot sz(o) > (C(S) \Leftrightarrow e(S)) \cdot cp(\mu), \tag{3}$$

then no schedule exists in which o precedes any of the operations in S, and (ii) if

$$lst(o) < C(S) < lct(o), \text{ and}$$
$$A(S) \Leftrightarrow (C(S) \Leftrightarrow lst(o)) \cdot sz(o) > (C(S) \Leftrightarrow e(S)) \cdot cp(\mu),$$

then no schedule exists in which o succeeds any of the operations in S.

Proof. (i) Suppose all operations are scheduled and o precedes an operation in S. Then $s(o) + pt(o) < C(S)$ and thus the area that is occupied between $e(S)$ and $C(S)$ is at least $A(S) + (ect(o) \Leftrightarrow e(S)) \cdot sz(o)$ which, according to (3), is larger than the available area $(C(S) \Leftrightarrow e(S)) \cdot cp(\mu)$, which leads to a contradiction. The proof of (ii) is similar to the proof of (i) □

If for a machine $\mu \in \mathcal{M}$, no schedule exists in which an operation $o \in \mathcal{O}_\mu$ precedes any of the operations in a set $S \subset \mathcal{O}_\mu$, all start times in $\delta(o)$ smaller than

$$\max_{S' \subseteq S | S' \neq \emptyset \ \wedge \ rest(S', sz(o)) > 0} e(S') + \lceil rest(S', sz(o))/sz(o) \rceil,$$

are inconsistent for o. If we do this for all $S \subseteq \mathcal{O}_\mu$, we obtain

$$LB3_{est} = \max_{S \subseteq \mathcal{O}_\mu | \alpha} \ \max_{S' \subseteq S | S' \neq \emptyset \ \wedge \ rest(S', sz(o)) > 0} e(S') + \lceil rest(S', sz(o))/sz(o) \rceil,$$

with

$$\alpha \quad \Leftrightarrow \quad o \notin S \ \wedge \ est(o) < e(S) < ect(o) \ \wedge \\ A(S) + (ect(o) \Leftrightarrow e(S)) \cdot sz(o) > (C(S) \Leftrightarrow e(S)) \cdot cp(\mu),$$

as a lower bound on the earliest start time of operation o.

Similarly, if for a machine $\mu \in \mathcal{M}$ no schedule exists in which an operation $o \in \mathcal{O}_\mu$ succeeds any of the operations in a set $S \subset \mathcal{O}_\mu$, all start times in $\delta(o)$ larger than

$$\min_{S' \subseteq S | S' \neq \emptyset \ \wedge \ rest(S', sz(o)) > 0} C(S') \Leftrightarrow \lceil rest(S', sz(o))/sz(o) \rceil \Leftrightarrow pt(o),$$

are inconsistent for o. If we do this for all $S \subseteq \mathcal{O}_\mu$, we obtain

$$UB3_{lct} = \min_{S \subseteq \mathcal{O}_\mu | \alpha} \ \min_{S' \subseteq S | S' \neq \emptyset \ \wedge \ rest(S', sz(o)) > 0} C(S') \Leftrightarrow \lceil rest(S', sz(o))/sz(o) \rceil,$$

with

$$\alpha \quad \Leftrightarrow \quad o \notin S \ \wedge \ lst(o) < C(S) < lct(o) \ \wedge \\ A(S) \Leftrightarrow (C(S) \Leftrightarrow lst(o)) \cdot sz(o) > (C(S) \Leftrightarrow e(S)) \cdot cp(\mu),$$

as an upper bound on the latest completion time of operation o.

Nuijten [1995] presents an algorithm for calculating $LB3_{est}$ and $UB3_{lct}$ with a time complexity equal to $O(|\mathcal{O}_\mu|^3)$.

The algorithm SequencingCheck**.** We refer to the consistency checking algorithm using the bounds given above by SequencingCheck. This algorithm can be used in two ways, viz., either by only using LB_{est} and UB_{lct} to update the earliest start times and latest end times, or by also using $LB2_{est}$, $UB2_{lct}$, $LB3_{est}$, and $UB3_{lct}$. The latter is referred to as the algorithm SequencingCheck2. The reason for this separation is that the time complexity of calculating $LB_{est}(o)$ and $UB_{lct}(o)$ for all operations $o \in \mathcal{O}_\mu$ for machines $\mu \in \mathcal{M}$ is $O(|\mathcal{O}_\mu|^2)$, whereas the time complexity of calculating $LB2_{est}(o)$, $UB2_{lct}(o)$, $LB3_{est}(o)$, and $UB3_{lct}(o)$ for all operations $o \in \mathcal{O}_\mu$ for machines $\mu \in \mathcal{M}$ is $O(|\mathcal{O}_\mu|^3)$.

4.4 Checking Remaining Capacity

Another way of identifying inconsistent start times is based upon the observation that, if for an operation o, $lst(o) < ect(o)$, o uses the interval $[lst(o), ect(o))$, and thus the remaining capacity for the other operations on $M(o)$ in $[lst(o), ect(o))$ is at most $cp(M(o)) \Leftrightarrow sz(o)$. If for a point in time the remaining capacity is 0 due to a set S of operations, no operation $o' \notin S$ can be scheduled using that point in time. To formalize this, we define for each

operation $o \in \mathcal{O}$, a function $ucp_o : \mathbb{N} \rightarrow \mathbb{N}$ denoting the capacity o uses in time, as follows.

$$ucp_o(t) = \begin{cases} sz(o) & \text{for } lst(o) \leq t < ect(o), \\ 0 & \text{otherwise.} \end{cases}$$

Theorem 7. *Let $\mu \in \mathcal{M}$, $o \in \mathcal{O}_\mu$, and $V = [a_1, b_1] \cup \ldots \cup [a_n, b_n] \subset \mathbb{N}$ be such that*

$$\{t \in [0, D] \mid \sum_{o' \in \mathcal{O}_\mu \setminus \{o\}} ucp_{o'}(t) > cp(\mu) \Leftrightarrow sz(o)\} = V,$$

then all start times in $(a_1 \Leftrightarrow pt(o), b_1] \cup \ldots \cup (a_n \Leftrightarrow pt(o), b_n]$ are inconsistent for o.

Proof. Follows directly from the definition of *ucp*. □

We refer to consistency checking algorithm that removes the inconsistent start times identified in Theorem 7 as RCPCheck.

4.5 The algorithm CheckConsistency

The results presented in the previous sections lead to the combined consistency checking algorithm presented in Figure 4.

```
proc CheckConsistency
    ForwardCheck;
    while domains have changed do
        2-ConsCheck;
        SequencingCheck;
        RCPCheck;
    endwhile
endproc
```

Figure 4: The algorithm CheckConsistency.

We remark that this consistency checking is incorporated in Ilog Schedule [Le Pape, 1994], an add-on to Ilog Solver, a C++ library for constraint programming [Puget, 1994].

4.6 Other ways of consistency checking

Other techniques that could be used in the process of consistency checking include the techniques described by Erschler, Lopez & Thuriot [1991] and Beck [1992]. In [Baptiste & Le Pape, 1995] the techniques of Erschler, Lopez & Thuriot [1991] are compared to our algorithm SequencingCheck for a number of scheduling problems, including the JSSP. It is concluded that SequencingCheck outperforms the techniques of Erschler, Lopez & Thuriot

[1991] both in effectivity and efficiency. The reason for the latter is mainly that the time complexity of SEQUENCINGCHECK is $O(|\mathcal{O}_\mu|^2)$, whereas the time complexity of the techniques of Erschler, Lopez & Thuriot [1991] is $O(|\mathcal{O}_\mu|^3)$.

5 Computational results

This section presents an empirical performance analysis which investigates whether our approach is capable of finding both good upper bounds and lower bounds on the minimum makespan for instances of the MCJSSP.

Note that our approach does not handle optimization problems since it merely searches for a schedule satisfying the constraints. We can, however, calculate upper bounds and lower bounds on the minimum makespan by iteratively redefining the scheduling horizon D. Calculating a lower bound on the minimum makespan of an instance of the MCJSSP is done as follows. First, a trivial lower bound on the minimum makespan of an instance is determined, e.g., 0. Second, a trivial upper bound is determined, e.g., the total sum of the processing times. Next, a binary search on the scheduling horizon D is performed by iteratively checking whether the instance with the scheduling horizon at hand is infeasible, by only using the algorithm CHECK-CONSISTENCY in the root of the search tree. If an instance is infeasible with a scheduling horizon D, $D + 1$ is a lower bound on the minimum makespan.

Upper bounds on the minimum makespan are calculated as follows. We start by solving the instance with a scheduling horizon D equal to the same trivial upper bound as mentioned above, and we determine the makespan of the solution found. We, then, iteratively try to solve the instance with a scheduling horizon that is less than the best found makespan so far. The value with which the best found makespan is decreased to obtain a new scheduling horizon, depends on the number of restarts done to solve the instances with the previous scheduling horizon, i.e., the smaller the number of restarts the larger the value of the decrement. We allow an overall maximum number of restarts, and we repeat lowering the scheduling horizon as long as this overall maximum number of restarts is not exceeded.

We refer to the algorithm calculating lower bounds and upper bounds in the way described above, as Randomized Constraint Satisfaction (RCS). The lower and upper bound on the minimum makespan that are calculated by RCS are referred to by $\mathrm{LB}_{\mathrm{RCS}}$ and $\mathrm{UB}_{\mathrm{RCS}}$, respectively.

Job Shop Scheduling. To test the performance of RCS on instances of the JSSP, we used 40 instances of the JSSP from Lawrence [1984] and three instances from Fisher & Thompson [1963]. We use the algorithms FORWARD-

	LB/UB	n	m	LB_{RCS}	UB_{RCS}	Δ	avg	t (sec)
F1	666	10	5	666	666	0	666	2
F2	655	10	5	655	655	0	655	22
F3	597	10	5	597	597	0	597	0.4
F4	590	10	5	583	590	0	590	20
F5	593	10	5	593	593	0	593	0.4
G1	926	15	5	926	926	0	926	0.9
G2	890	15	5	890	890	0	890	4
G3	863	15	5	863	863	0	863	5
G4	951	15	5	951	951	0	951	2
G5	958	15	5	958	958	0	958	2
H1	1222	20	5	1222	1222	0	1222	5
H2	1039	20	5	1039	1039	0	1039	11
H3	1150	20	5	1150	1150	0	1150	4
H4	1292	20	5	1292	1292	0	1292	2
H5	1207	20	5	1207	1207	0	1207	17
A1	945	10	10	901	945	0	948.8	369
A2	784	10	10	777	784	0	784	143
A3	848	10	10	803	848	0	851.2	466
A4	842	10	10	756	848	0.71	848.6	291
A5	902	10	10	842	907	0.55	909.6	202
B1	1040/1046	15	10	1033	1069	2.79	1081.2	1520
B2	927	15	10	913	937	1.08	942.0	1367
B3	1032	15	10	1032	1032	0	1032	103
B4	935	15	10	889	942	0.75	946.2	1477
B5	977	15	10	919	981	0.41	989.2	1472
C1	1218	20	10	1218	1218	0	1218	2691
C2	1235/1236	20	10	1235	1285	4.05	1294.2	2790
C3	1216	20	10	1216	1216	0	1216	2469
C4	1120/1160	20	10	1119	1208	7.86	1238.6	1132
C5	1355	20	10	1355	1355	0	1355	56
D1	1784	30	10	1784	1784	0	1784	26
D2	1850	30	10	1850	1850	0	1850	60
D3	1719	30	10	1719	1719	0	1719	103
D4	1721	30	10	1721	1721	0	1721	187
D5	1888	30	10	1888	1888	0	1888	145
I1	1268	15	15	1233	1292	1.89	1305.8	2233
I2	1397	15	15	1397	1411	1.00	1421.6	2552
I3	1184/1196	15	15	1106	1278	7.94	1292.4	2815
I4	1233	15	15	1221	1233	0	1243.4	2724
I5	1222	15	15	1192	1247	2.05	1256.2	2575
FI06	55	6	6	54	55	0	55	0.8
FI10	930	10	10	858	930	0	946.6	591
FI20	1165	20	5	1165	1165	0	1165	21

Table 1: Results obtained with RCS for 43 instances of the JSSP.

CHECK, 2-CONSCHECK, and SEQUENCINGCHECK2 in the consistency checking, a *backtracking factor* of 0.15 and an overall maximum of 500 restarts. The backtracking factor is defined as follows. If the backtracking factor is B, then with N the number of operations of an instance, RCS restarts if the number of chronological backtrack steps exceeds $B \cdot N$. For each instance, we performed five runs using different seeds for the random number generator. All tests in this article are performed on a SPARC-station ELC.

The results of the tests are given in Table 1. The column "LB/UB" of Table 1 gives the best known lower and upper bound on the minimum makespan. If only a single number is given, both bounds coincide, resulting in an optimal value of the makespan. The columns "n" and "m" give the number of jobs and the number of machines, respectively. For all instances, the number of operations of each job equals the number of machines. The column "LB_{RCS}" gives the lower bound found by RCS. The column "UB_{RCS}" gives the minimum makespan found over five runs and the column "Δ" gives the deviation of the minimum found makespan from the best known lower bound in terms of percentage of the best known lower bound. In the sequel we refer to the deviation of a value a from a value b in terms of percentage of b simply by the deviation of a from b. Column "avg" gives the average minimum makespan over five runs and column "t" gives the average running time over five runs.

31 instances are solved to optimality, including the notorious FI10 instance, and the deviation of the minimum found makespan from the best known lower bound is on average 0.72%. The deviation of our lower bound from the best known lower bound is on average 1.50%. Vaessens, Aarts & Lenstra [1994] compare 20 of the best approaches to the JSSP for 13 instances of the aforementioned set of 43 instances. Ranking the approaches that were used according to effectiveness, our approach comes in the sixth place with an average deviation of 2.26%, where the *tabu search* approach of Nowicki & Smutnicki [1993] performs best with an average deviation of 0.54%. This leads to the conclusion that our approach performs well.

Multiple Capacitated Job Shop Scheduling. To test the performance of RCS on instances of the MCJSSP, we first used 30 randomly generated instances with 5 - 10 machines and 100 - 200 operations, for which the number of operations of each job equals the number of machines, each operation is given a random machine by which it must be processed, the processing times are randomly taken from the interval $[1, 99]$, the sizes are all 1, and the capacity of the machines is set to 2. We use the algorithms FORWARDCHECK, RCPCHECK, and SEQUENCINGCHECK2 in the consistency checking, a back-

tracking factor of 0.15 and an overall maximum of 100 restarts. For each instance we performed five runs using different seeds for the random number generator. For these instances, RCS performs extremely well. Out of 30 instances, 22 are solved to optimality and the deviation of our upper bounds from our lower bounds is on average 0.52%. The average running time for calculating upper bounds is 211 and for calculating lower bounds is 7 seconds. We remark that experiments with machines with capacity larger than 2 gave similar results.

	LB/UB	n	m	c	UBT	UB_{RCS}	Δ	avg	t (sec)
F1D	666	20	5	2	666	666	0	667.8	113
F2D	655	20	5	2	655	683	4.27	688.0	429
F3D	593/597	20	5	2	597	638	7.59	640.2	270
F4D	572/590	20	5	2	590	590	3.15	600.6	266
F5D	593	20	5	2	593	593	0	593	2
F1T	666	30	5	3	666	671	0.75	676.0	356
F2T	655	30	5	3	655	704	7.48	718.0	890
F3T	590/597	30	5	3	597	647	9.66	655.4	788
F4T	570/590	30	5	3	590	592	3.86	617.6	201
F5T	593	30	5	3	593	593	0	593	3
G1D	926	30	5	2	926	926	0	926	5
G2D	890	30	5	2	890	890	0	894.4	12
G3D	863	30	5	2	863	863	0	863.4	94
G4D	951	30	5	2	951	951	0	951	112
G5D	958	30	5	2	958	958	0	958	9
G1T	926	45	5	3	926	926	0	926	11
G2T	890	45	5	3	890	913	2.58	913.0	782
G3T	863	45	5	3	863	866	0.35	869.6	895
G4T	951	45	5	3	951	952	0.11	953.6	843
G5T	958	45	5	3	958	960	0.21	960.4	1298
A1D	888/935	20	10	2	945	935	5.29	941.0	636
A2D	750/765	20	10	2	784	765	2.00	770.8	719
A3D	783/844	20	10	2	848	844	7.79	854.8	567
A4D	730/840	20	10	2	842	840	15.07	843.6	451
A5D	829/902	20	10	2	902	902	8.81	908.8	584
FI06D	53/55	12	6	2	55	55	3.77	55.0	70.2
FI10D	835/930	20	10	2	930	963	15.33	979.6	863
FI20D	1165	40	5	2	1165	1319	13.22	1339.4	1340

Table 2: Results for 28 modified JSSP instances with machines with capacity 2 and 3.

The 28 instances of Table 2 were devised to provide more challenging instances. We, therefore, modified 15 JSSP instances from Lawrence [1984] and three JSSP instances from Fisher & Thompson [1963] by either duplicating

or triplicating them as follows. We first explain the duplication of a JSSP instance. Each operation of the instance is duplicated and all operations are given size 1. Furthermore, the duplicated operations have the same binary relation and machine assignment as the original operations. Finally, the capacity of each machine is set to 2. Triplicating an instance is done in a similar way, with the difference that the capacity of the machines are set to 3. This way of constructing instances implies that the upper bounds on the minimum makespan of the JSSP instances are also upper bounds on the minimum makespan of the MCJSSP instances, as a schedule of a MCJSSP instance can be obtained from a schedule of the corresponding JSSP instance by setting the start times of the original and extra operations in the MCJSSP equal to the start times of the original operations of the schedule for the JSSP. This upper bound is given in column "UBT" of Table 2. Column "c" gives the capacity of the machines. The other columns of Table 2 correspond to the columns of Table 1. We use the algorithms ForwardCheck, RCPCheck, and SequencingCheck2 in the consistency checking, a backtracking factor of 0.15 and an overall maximum of 100 restarts. For each instance we performed five runs using different seeds for the random number generator. Note that our lower bounds are the best known lower bounds for these instances. Therefore, they are not given in a separate column.

The deviation of our upper bounds from our lower bounds is on average 3.97%. The deviation of the best known upper bound, either our upper bound or UBT, from our lower bounds is on average 2.12 %. The average running time for calculating lower bounds is 8 seconds. A remarkable instance is FI20D. Although we know that 1165 is the minimum makespan, our upper bound is not at all close to 1165. This is even more remarkable as the corresponding JSSP instance is easily solved to optimality by RCS.

Varying the consistency checking. In the following experiments we vary upon the consistency checking that is used. In Table 3 results are given for calculating a lower bound on the minimum makespan when using different consistency checking. We calculated lower bounds for all 28 instances of Table 2. Column "Δ" gives the average deviation of the lower bound from the best known lower bound.
"FC" refers to the algorithm ForwardCheck, "+ SEQ" refers to a combination of ForwardCheck and SequencingCheck, "RCP + SEQ" refers to a combination of ForwardCheck, SequencingCheck and RCPCheck, and "RCP + SEQ2" refers to a combination of ForwardCheck, SequencingCheck2, and RCPCheck.

checking	Δ
FC	36.84 %
+ SEQ	0.45 %
RCP + SEQ	0.29 %
RCP + SEQ2	0 %

Table 3: Average deviation of the lower bounds from the best known lower bounds calculated using different consistency checking algorithms.

The table shows that substantial performance improvement can be observed when using more extensive consistency checking. Using only ForwardCheck is strongly outperformed by also using SequencingCheck. Adding RCPCheck and using SequencingCheck2 instead of SequencingCheck slightly improve performance. We remark that there are only slight differences in running time.

A similar behavior can be observed in Table 4, in which results are given for calculating an upper bound on the minimum makespan for different consistency checking. We did time equivalent tests for calculating upper bounds for all 28 instances of Table 2. Column "Δ" gives the average deviation of the found upper bound from the best known lower bound.

checking	Δ
FC	10.92 %
+ SEQ	6.89 %
RCP + SEQ	5.73 %
RCP + SEQ2	5.46 %

Table 4: Average deviation of the upper bounds from the best known lower bounds calculated using different consistency checking algorithms.

Again, using SequencingCheck improves the performance considerably when compared to only using ForwardCheck. We remark that due to the nature of the instances, using the algorithm 2-ConsCheck is useless. It is remarkable that the performance improvement when also using RCPCheck and using SequencingCheck2 instead of SequencingCheck is more pronounced here than it is for the calculation of lower bounds. Again, we conclude that doing more extensive consistency checking pays off.

6 Conclusions

We have presented an algorithm based on constraint satisfaction techniques to handle the MCJSSP. We have shown that this algorithm performs well both on instances of the JSSP and on instances of the MCJSSP. We have also shown that the extensive consistency checking presented in this article plays a dominant role in achieving this performance.

References

Baptiste, P., and C. Le Pape [1995], A theoretical and experimental comparison of constraint propagation techniques for disjunctive scheduling, *Proc. 14th International Joint Conference on Artificial Intelligence.*

Beck, H. [1992], Constraint monitoring in TOSCA, *Working Papers of the AAAI Spring Symposium on Practical Approaches to Planning and Scheduling.*

Carlier, J., and E. Pinson [1990], A practical use of Jackson's preemptive schedule for solving the job shop problem, *Annals of Operations Research* **26**, 269–287.

Erschler, J., P. Lopez, and C. Thuriot [1991], Raisonnement temporel sous contraintes de ressource et problèmes d'ordonnacement, *Revue d'Intelligence Artificielle* **5**, 7–32, in French.

Fisher, H., and G.L. Thompson [1963], Probabilistic learning combinations of local job-shop scheduling rules, in: J.F. Muth and G.L. Thompson (eds.), *Industrial Scheduling*, Prentice Hall.

French, S. [1982], *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*, Wiley & Sons.

Garey, M.R., and D.S. Johnson [1979], *Computers and Intractability; A Guide to the Theory of NP-completeness*, W.H. Freeman and Company, New York.

Garey, M.R., D.S. Johnson, and R. Sethi [1976], The complexity of flowshop and jobshop scheduling, *Mathematics of Operations Research* **1**, 117–129.

Haralick, R.M., and G.L. Elliott [1980], Increasing tree search efficiency for constraint satisfaction problems, *Artificial Intelligence* **14**, 263–313.

Lawrence, S. [1984], *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement)*, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh.

Le Pape, C. [1994], Implementation of resource constraints in ilog schedule: A library for the development of constraint-based scheduling systems, *Intelligent Systems Engineering* **3**, 55–66.

Mackworth, A.K. [1977], Consistency in networks of relations, *Artificial Intelligence* **8**, 99–118.

Montanari, U. [1974], Networks of constraints: Fundamental properties and applications to picture processing, *Information Sciences* **7**, 95 – 132.

Nowicki, E., and C. Smutnicki [1993], *A Fast Taboo Search Algorithm for the Job Shop Problem*, Preprinty nr. 8/93, Instytut Cybernetyki Technicznej, Po-

litechnicki Wroclawskiej, Poland.

NUIJTEN, W.P.M., AND E.H.L. AARTS [1994], Constraint satisfaction for multiple capacitated job shop scheduling, in: A. Cohn (ed.), *Proc. 11th European Conference on Artificial Intelligence*, John Wiley & Sons, 635–639.

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

NUIJTEN, W.P.M. [1995], Improving efficiency of constraint propagation algorithms for multiple capacitated job shop scheduling, Private Note.

NUIJTEN, W.P.M., E.H.L. AARTS, D.A.A. VAN ERP TAALMAN KIP, AND K.M. VAN HEE [1993], Randomized constraint satisfaction for job shop scheduling, *Proc. IJCAI '93 Workshop on Knowledge-Based Production Planning, Scheduling and Control*, 251–262.

NUIJTEN, W.P.M., G.M. KUNNEN, E.H.L. AARTS, AND F.P.M. DIGNUM [1994], Examination time tabling: A case study for constraint satisfaction, *Proc. ECAI '94 Workshop on Constraint Satisfaction Issues Raised by Practical Applications*, 11–19.

PUGET, J.-F. [1994], *A C++ Implementation of CLP*, Technical Report 94-01, ILOG S.A., Gentilly, France.

VAESSENS, R.J.M., E.H.L. AARTS, AND J.K. LENSTRA [1994], *Job Shop Scheduling by Local Search*, COSOR Memorandum 94-05, Eindhoven University of Technology.