# "EXACT AND APPROXIMATION ALGORITHMS FOR THE TACTICAL FIXED INTERVAL SCHEDULING PROBLEM"

by

**L.G. KROON***
**M. SALOMON****
**and**
**L.N. VAN WASSENHOVE*****

**94/04/TM**

* Erasmus University, P.O. Box 1738, NL-3000 DR Rotterdam, The Netherlands.

** Erasmus University, P.O. Box 1738, NL-3000 DR Rotterdam, The Netherlands.

*** Professor of Operations Management and Operations Research at INSEAD, Boulevard de Constance, 77305 Fontainebleau, France.



Printed at INSEAD, Fontainebleau, France

# Exact and Approximation Algorithms for the Tactical Fixed Interval Scheduling Problem

Leo G. Kroon & Marc Salomon

Erasmus University, P.O. Box 1738

NL-3000 DR Rotterdam, the Netherlands

Luk N. Van Wassenhove

INSEAD, Boulevard de Constance

F-77305, France

**Abstract**

The Tactical Fixed Interval Scheduling Problem (TFISP) is the problem of determining the *minimum number* of parallel non-identical machines, such that a feasible schedule exists for a given set of jobs. In TFISP each job must be carried out in a prespecified time interval and belongs to a specific job class. The problem is complicated by the restrictions that (*i*) each machine can handle one job at a time only, (*ii*) each machine can handle jobs from a subset of the job classes only, and (*iii*) preemption is not allowed.
In this paper we discuss the occurrence of TFISP in practice, we analyze the computational complexity of TFISP, and we present exact and approximation algorithms for solving TFISP. The paper is concluded with a computational study.

**Keywords:** Fixed interval scheduling, heuristics, integer programming, Lagrangean relaxation, non-preemptive scheduling, upper and lower bounds.

## 1 Introduction

The Tactical Fixed Interval Scheduling Problem (TFISP) is the problem of determining the minimum number of parallel non-identical machines, such that a feasible non-preemptive schedule exists for a given set of jobs. Each job belongs to a specific job class, and has a fixed start time, a fixed finish time, and a processing time which equals the length of the time interval between the job's start and finish time. Each machine is allowed to process jobs from a prespecified subset of job classes only, and can process at most one job at the same time.
Practical situations where (variants of) this problem occur include:

- Capacity planning of airport terminals. A problem occurring in this setting is to determine the minimum number of gates at an airport, such that all aircraft within a given time horizon can arrive at and depart from a gate. For technical reasons each gate can handle a limited set of aircraft types only. Furthermore, the fixed arrival and departure time of each aircraft follow from the time-table. Here a gate can be considered as a machine, and an aircraft during its stay at the airport as a job (Hagdorn & Kroon, 1990).

- Capacity planning of aircraft maintenance personnel at an airport. Planes arriving at an airport may require a number of maintenance jobs to be carried out by a number of engineers. The processing times as well as the order in which these jobs have to be carried out are specified by maintenance norms. As a consequence, the time-table and the maintenance norms determine the fixed intervals in which the jobs have to be carried out in order to avoid delays of the airplanes on their next flights. The situation is further complicated by the rule that (for safety reasons) each of the engineers is licensed to carry out jobs on a limited number of aircraft types only. The major problem the management of the maintenance department is faced with is to determine the optimal size and composition of the *teams* in which the engineers operate: i.e. to determine the minimum number of engineers per team and the licenses they should have (Kroon, 1990; Dijkstra et al., 1991; Kolen & Kroon, 1991, 1992, 1993).

This paper is structured as follows. A formulation of TFISP both as an integer program and as a network flow problem with additional side constraints is given in Section 2. In Section 3 we discuss the literature on TFISP. The computational complexity of TFISP and its *preemptive* variant is analyzed in Section 4. Lower and upper bounding procedures are described in the Sections 5 and 6 respectively. In Section 7 the results of a computational study are presented. Finally, Section 8 summarizes the results and conclusions of this paper.

# 2 Model formulations

In this section we give a formulation of TFISP both as an integer program and as a network flow problem with additional side constraints.
Recall that in TFISP the objective is to find the minimum number of parallel non-identical machines required for a feasible non-preemptive schedule for a given set of jobs with fixed start and finish times within a given planning horizon $[0, T]$.
In order to formally define TFISP, we introduce the following notation: suppose there are $J$ jobs to be carried out, where the start and finish time of job $j$ are represented by $s_j$ and $f_j$, and where the job class of job $j$ is represented by $a_j$. The number of different job classes is denoted by $A$. Furthermore, each machine is allowed to handle jobs from a limited number of job classes only. The number of different machine classes is denoted by $C$. Next, we define $\mathcal{A}_c$ as the set of job classes that can be carried out by machines in machine class $c$ ($c = 1, \ldots, C$). Without loss of generality we assume

that no two sets $\mathcal{A}_c$ are equal. The set $\mathcal{J}_c$ consists of all jobs that can be carried out by machines in machine class $c$. Since the objective is to find the *minimum number* of required machines, we assume that no machine class is *dominated*, i.e. there is no machine class $c$ such that $\mathcal{A}_c \subset \mathcal{A}_{c'}$ for some other machine class $c'$. The set $\mathcal{C}_a$ denotes the set of machine classes that can be used for carrying out jobs in job class $a$ $(a = 1, \ldots, A)$. The set $\mathcal{T}_c$ is the set of start times of jobs that can be handled by machine class $c$. Thus $\mathcal{T}_c = \{s_j | j \in \mathcal{J}_c\}$. Furthermore, the set $\mathcal{T}$ denotes the set of all start times of jobs. Thus $\mathcal{T} = \{s_j | j = 1, \ldots, J\}$. The job overlap at time instant $t$, denoted by $L^t$, and the maximum job overlap, denoted by $L$, are defined by

$$L^t = |\{j | s_j \leq t < f_j\}|, \qquad L = \max\{L^t | t \in \mathcal{T}\}.$$

If $\alpha$ is a set of job classes, then the maximum job overlap of the jobs $j$ with $a_j \in \alpha$, which is denoted by $L_\alpha$, is defined in an analogous way. If $a$ is one of the job classes, then $L_{\{a\}}$ is abbreviated to $L_a$. Similarly, if $c$ is a machine class, then $L_{\mathcal{A}_c}$ is abbreviated to $L_c$.

Next, we describe TFISP as an integer program. We define *integer* decision variables $Y_c$ to represent the number of machines in machine class $c$ and *binary* decision variables $x_{j,c}$ to denote whether job $j$ is carried out by a machine in machine class $c$. $Z_{IP}$ is the minimum number of machines. Now, TFISP can be formulated as the following integer program:

*TFISP*:

$$\min Z_{IP} = \sum_{c=1}^{C} Y_c \tag{1}$$

subject to

$$\sum_{\{j | s_j \leq t < f_j \wedge j \in \mathcal{J}_c\}} x_{j,c} \leq Y_c \qquad c = 1, \ldots, C;\ t \in \mathcal{T}_c \tag{2}$$

$$\sum_{c \in \mathcal{C}_{a_j}} x_{j,c} = 1 \qquad j = 1, \ldots, J \tag{3}$$

$$x_{j,c} \in \{0, 1\} \qquad j = 1, \ldots, J;\ c \in \mathcal{C}_{a_j} \tag{4}$$

$$Y_c \in \{0, 1, 2, \ldots\} \qquad c = 1, \ldots, C \tag{5}$$

The objective function (1) ensures that a machine configuration is obtained for which the total number of machines is minimal. The set of constraints (2) guarantees $(i)$ that the number of jobs processed in parallel by the machines in machine class $c$ never exceeds $Y_c$, and $(ii)$ that the jobs can be carried out in a non-preemptive way (Kroon et al., 1992, and Lemma 1). Furthermore, constraints (3) state that each job is processed exactly once. Finally, (4) are the binary constraints on the assignment variables, and (5) requires the number of machines in each machine class to be integer valued.
Note that some constraints of (2) may be redundant. Consider for instance the situation in which $t_1, t_2 \in \mathcal{T}_c$ and $(t_1, t_2] \cap \{f_j | j \in \mathcal{J}_c\} = \emptyset$. Then for machine class $c$ the restriction corresponding to time instant $t_1$ is redundant, since it is dominated by the restriction corresponding to time instant $t_2$.

Next, we describe TFISP as a network flow model with side constraints. The underlying directed graph $G$ contains $C$ subgraphs $G_c$, each one corresponding to one of the machine classes. The nodeset $N_c$ of $G_c$ is in one-to-one correspondence with the set of start and finish times of the jobs that can be handled by machine class $c$. A particular job $j$ with $j \in \mathcal{J}_c$ is represented in $G_c$ by an arc from the node corresponding to $s_j$ to the node corresponding to $f_j$. This arc has upper capacity one. The set $N_c$ is also denoted as $\{n_{c,r} | r = 1, \ldots, p_c\}$, where $p_c = |N_c|$. Here it is assumed that for $r = 2, \ldots, p_c$ the nodes $n_{c,r-1}$ and $n_{c,r}$ correspond to subsequent time instants. In $G_c$ there is for $r = 2, \ldots, p_c$ an arc from $n_{c,r-1}$ to $n_{c,r}$ with unlimited capacity. In order to reduce the number of nodes and arcs of the graph $G_c$, the graph compression procedure of Kroon et al. (1992) can be applied to it.
The graphs $G_c$ are linked together by a super source $P$ and a super sink $Q$. There is an arc from $P$ to each of the nodes $n_{c,1}$ and there is an arc from each of the nodes $n_{c,p_c}$ to the super sink $Q$. Both arcs have unlimited capacity. Figure 1 shows an example of a set of jobs and the corresponding graph $G$.

**Figure 1.** An instance of TFISP and the corresponding graph $G$. Here $A = 3$, $C = 2$, and a machine in machine class $c$ can handle jobs in the job classes $c$ and 3.

The side constraints that must be satisfied specify that all flows must be integer and that for each job the total amount of flow in the corresponding arcs must be exactly one unit. Now, in TFISP the objective is to send an amount of flow from the super source $P$ to the super sink $Q$ in such a way that $(i)$ all capacity constraints are satisfied, $(ii)$ all side constraints are satisfied, and $(iii)$ the total amount of flow is minimal.

# 3 Literature review

TFISP is a generalization of the well known Fixed Job Scheduling Problem (FJSP). In this problem all jobs have a fixed start time and a fixed finish time and belong to the same job class. Furthermore, the machines are identical. FJSP is the problem of determining the minimum number of machines such that a non-preemptive schedule exists for all jobs. This problem was studied by Dantzig & Fulkerson (1954), and by Gertsbakh & Stern (1978) in the context of logistics. It was also studied by Hashimoto & Stevens (1971), and by Gupta et al. (1979) in the context of computer wiring. An optimal solution to this problem is described by Lemma 1.

**Lemma 1** *The minimum number of machines required to carry out all jobs of an instance of FJSP equals the maximum job overlap of the jobs.*

Lemma 1 is a direct consequence of Dilworth's theorem on partially ordered sets, stating that in any partially ordered set the minimum number of chains required for covering all elements is equal to the size of a maximum antichain (Dilworth, 1950). An $\mathcal{O}(J \log J)$ algorithm for determining the maximum job overlap of the jobs is described by Hashimoto & Stevens (1971) and by Gupta et al. (1979).
Figure 2 shows an instance of FJSP. As the maximum job overlap $L$ equals 4, the minimum number of machines required to carry out all jobs equals 4 as well.

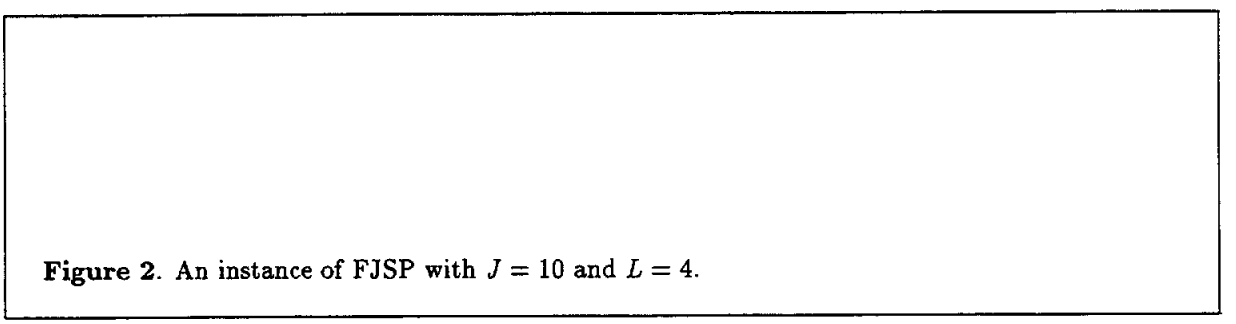

**Figure 2.** An instance of FJSP with $J = 10$ and $L = 4$.

Fischetti et al. (1987, 1989, 1992) describe variants of FJSP with side constraints either on the total workload per machine or on the spread time per machine (i.e. the difference between the finish time of the last assigned job and the start time of the first assigned job). It is shown that these variants of FJSP, which are related to the bus driver scheduling problem, are NP-hard. Several upper and lower bounding procedures together with their corresponding performance guarantees are described.
Also the case with $A = 3$ and $C = 2$, where a machine in machine class $c$ is allowed to carry out jobs in the job classes $c$ and 3, can be solved in polynomial time. One approach for solving this special case of TFISP consists of iteratively solving a minimum cost flow problem, as was proposed by Dondeti & Emmons (1992). Kroon (1990) has shown that this special case of TFISP can also be solved by a polynomial round-off procedure:

**Round-off procedure:**

($i$) Let $Z_{LP}$ denote the value of the optimal solution to the LP-relaxation of TFISP.

($ii$) If $Z_{LP}$ is integer, then the solution to the LP-relaxation is all integer, and hence optimal for TFISP. Otherwise, an optimal solution can be found by solving the extended LP-relaxation, which is obtained by adding the constraint $Y_1 + Y_2 = \lceil Z_{LP} \rceil$.

Dondeti & Emmons (1993) consider the preemptive variant of TFISP. They *conjecture* that this problem is NP-hard. In this paper we show that for a predetermined number of machine classes the problem can be solved in polynomial time. However, if the number of machine classes is *not* predetermined, then the problem is NP-hard in the strong sense.

Another problem closely related to TFISP is the Operational Fixed Interval Scheduling Problem (OFISP), where the machine configuration consisting of a set of parallel non-identical machines is given, where a value is associated with each job, and where the objective is to find a feasible schedule for a subset of jobs of maximum total value. It should be noted that OFISP differs from TFISP in two related aspects. First, the objective in TFISP is to find the *minimum number* of machines required for a feasible schedule for all jobs, while the objective in OFISP is to select a set of jobs with *maximum total value*, if the machine configuration is *given*. As a consequence, the second difference consists herein, that in TFISP all jobs must be processed *exactly* once, while in OFISP a job must be processed *at most* once.
Arkin & Silverberg (1987) show that OFISP can be solved in $\mathcal{O}(J^{M+1})$ by the application of dynamic programming. Here, $M$ denotes the total number of machines. Carter (1989) presents a Lagrangean relaxation algorithm for a variant of OFISP occurring in the context of class room scheduling. Carter & Tovey (1992) present an analysis of the computational complexity of several variants of this class room scheduling problem. Finally, Kroon et al. (1992) present an approximation algorithm for solving OFISP. Their algorithm is based on Lagrangean relaxation and decomposition, and on the application of a straightforward dual-ascent heuristic.

# 4 Complexity results

In this section we consider TFISP as well as a variant of TFISP that allows for preemption of the jobs. Here, preemption is defined as the possibility to split a job $j$ into at least two parts $(s_j, p)$ and $(p, f_j)$, which are carried out by different machines. As already stated, the complexity of the preemptive variant has recently been conjectured by Dondeti & Emmons (1993). Here we will provide a complete overview of the complexity of the preemptive and non-preemptive variant of TFISP. We start with the following auxiliary lemma concerning the preemptive variant of TFISP.

**Lemma 2** *Each preemptive schedule $R$ can be transformed into a preemptive schedule $R'$, in which all preemptions occur at time instants $t \in \mathcal{T}$ only. The number of machines in each machine class is equal for $R$ and $R'$.*

**Proof.** Note first that we can restrict ourselves to schedules with a finite number of preemptions. Now, suppose we have a preemptive schedule $R$, where at least one job is preempted at a time instant $t \notin \mathcal{T}$. Choose $\sigma < t$ such that during the time interval $(\sigma, t)$ the *'status'* of each machine remains unchanged. That is, during this time interval, machine $m$ is either idle, or carrying out one job, say job $j_m$. Next, let $\tau$ be defined as $\tau = \min(\{s_j | j = 1, \ldots, J; s_j > t\} \cup \{T\})$.
Now the schedule during the interval $(t, \tau)$ is modified in the following way. If machine $m$ was idle during the time interval $(\sigma, t)$, then machine $m$ is also idle during the interval $(t, \tau)$. If machine $m$ was carrying out job $j_m$ during the time interval $(\sigma, t)$, then let $e_m$ be defined by $e_m = \min\{f_{j_m}, \tau\}$. Next, machine $m$ is carrying out job $j_m$ during the time interval $(t, e_m)$ and is idle during the time interval $(e_m, \tau)$. By this transformation we

obtain a new preemptive schedule, where the preemption at time instant $t$ does no longer occur. However, another preemption may have been introduced at time instant $\tau \in \mathcal{T}$. By eliminating in this way all preemptions at time instants $t \notin \mathcal{T}$, we ultimately end-up with a schedule $R'$ as specified. Note that the number of required machines does not change under these transformations. □

**Lemma 3** *TFISP is NP-hard in the strong sense, even if preemption is allowed.*

**Proof.** This lemma is proved by a reduction from Three Dimensional Matching (3DM). A definition of 3DM is given in Appendix I. Hence let $I_1$ be an instance of 3DM containing three disjoint sets $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$, each one containing $A$ elements, and a set $\mathcal{W}$ which is a subset of $\mathcal{X} \times \mathcal{Y} \times \mathcal{Z}$. Now an instance $I_2$ of TFISP is constructed as follows. The set of job classes is equal to $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{Z}$. Thus the number of job classes equals $3A$. The following jobs of the form $(s_j, f_j, a_j)$ must be carried out.

$(0, 1, x)$ for all $x \in \mathcal{X}$,
$(1, 2, y)$ for all $y \in \mathcal{Y}$,
$(2, 3, z)$ for all $z \in \mathcal{Z}$.

Each element of $\mathcal{W}$ corresponds to one machine class. A machine in machine class $(x, y, z) \in \mathcal{W}$ is allowed to carry out jobs in the job classes $x$, $y$ and $z$. Now it is evident that the following statement holds: $I_1$ is a 'yes'-instance of 3DM if and only if all jobs in $I_2$ can be carried out by $A$ machines. Note that if all jobs can be carried out by $A$ machines, then all machines must belong to different machine classes. In $I_2$ the set of used machine classes corresponds to the set $\mathcal{W}'$ in $I_1$. The number of required machines will not be reduced by allowing preemption, since each pair of jobs either has identical start and finish times or is non-overlapping. □

**Lemma 4** *If the number of machine classes is predetermined and preemption is allowed, then TFISP is solvable in polynomial time.*

**Proof.** First, note that if the number of machine classes is predetermined, then the maximum number of job classes is predetermined as well, since $A \leq 2^C$.
Next, suppose we have a tentative solution for a given instance of TFISP. Let $Y_1, \ldots, Y_C$ represent the number of machines in each of the machine classes in this solution. According to Lemma 2, preemptions need to occur only at time instants belonging to the set $\mathcal{T}$. Suppose the jobs have been ordered such that $s_{j-1} \leq s_j$ for $j = 2, \ldots, J$. Then all time intervals $(s_{j-1}, s_j)$ can be considered independently of each other in order to check the feasibility of the tentative solution.
Now we calculate for $a = 1, \ldots, A$ and $t \in \mathcal{T}$ the job overlap of the jobs in job class $a$ at time instant $t \in \mathcal{T}$. This job overlap is denoted by $L_a^t$ and can be obtained in $\mathcal{O}(J \log J)$ time, as was discussed in Section 3.
Next, let $x_{c,a}^t$ denote the integer number of machines in machine class $c$ that are used to carry out jobs in job class $a$ at time instant $t \in \mathcal{T}$. In order to check the feasibility of the tentative solution, it has to be decided whether a feasible solution to the following constraints exists:

$$\sum_{c \in \mathcal{C}_a} x_{c,a}^t \geq L_a^t \text{ for } a = 1, \ldots, A \text{ and } t \in \mathcal{T}, \tag{6}$$

$$\sum_{a \in \mathcal{A}_c} x_{c,a}^t \leq Y_c \text{ for } c = 1, \ldots, C \text{ and } t \in \mathcal{T}. \tag{7}$$

For fixed $Y_1, \ldots, Y_C$ this problem is a feasibility check for $\mathcal{O}(|\mathcal{T}|) \sim \mathcal{O}(J)$ independent transportation problems. The latter result was noted already by Dondeti & Emmons (1993). Each transportation problem has $C$ sources and $A$ destinations, and source $c$ and destination $a$ are connected if and only if machine class $c$ is allowed to handle jobs in job class $a$. Each of these transportation problems can be solved in polynomial time. Thus for fixed $Y_1, \ldots, Y_C$ the feasibility of problem (6), (7) can be checked in $\mathcal{O}(J^K)$ time for some fixed $K$.
Furthermore, the numbers $Y_1, \ldots, Y_C$ in an optimal solution to TFISP obviously satisfy the relations

$$Y_c \leq L_c, \qquad L \leq \sum_{c=1}^{C} Y_c \leq J. \tag{8}$$

By verifying each set $\{Y_c | c = 1, \ldots, C\}$ satisfying (8), a feasible preemptive schedule will be obtained in which the number of machines is minimal. As the number of sets $\{Y_c | c = 1, \ldots, C\}$ satisfying (8) is $\mathcal{O}(J^C)$, TFISP can be solved in $\mathcal{O}(J^{K+C})$ time. Since both $C$ and $K$ are fixed, TFISP is solvable in polynomial time. □

If the number of machine classes is predetermined, and preemption is *not* allowed, then the computational complexity of TFISP depends on the number of machine classes. In this case TFISP can be solved in polynomial time if $C \leq 2$, as was discussed in Section 3. If $C > 2$, then TFISP is NP-hard (Kolen & Kroon, 1992). Table 1 summarizes the complexity results obtained for TFISP.

| | $C$ variable | $C$ predetermined | |
|---|---|---|---|
| | | $C \leq 2$ | $C > 2$ |
| preemption allowed | NP-hard (Lemma 3) | polynomially solvable (Section 3) | polynomially solvable (Lemma 4) |
| no preemption allowed | NP-hard (Lemma 3) | polynomially solvable (Section 3) | NP-hard (Kolen & Kroon, 1992) |

**Table 1.** Computational complexity of TFISP.

# 5 Lower bounds

Lower bounds to $Z_{IP}$ are obtained upon relaxation of some of the constraints. We consider the following relaxations:

- Relaxation of the constraints that all jobs should be processed by machines that are allowed to process jobs of that particular job class. In the remaining problem all machines are allowed to process all jobs. We denote this relaxation by Class Relaxation (CR). According to Lemma 1, the corresponding lower bound $Z_{CR}$ is obtained in polynomial time by computing the maximum job overlap.

- Relaxation of the constraints that all jobs must be processed in a non-preemptive way. This relaxation is called Non-preemptive Relaxation (NR). For fixed $C$ the corresponding lower bound $Z_{NR}$ can be obtained in polynomial time by applying the algorithm specified in the proof of Lemma 4. However, for computational efficiency we decided to compute this bound by solving the integer program ($\min \sum_c Y_c$ subject to (6)–(7), and $Y_c$ integer for all $c$) using a standard Branch & Bound algorithm.

- Relaxation of the integrality constraints (4) and (5). The remaining Linear Programming relaxation (LP) can be solved in polynomial time. The corresponding lower bound is denoted by $Z_{LP}$.

- Lagrangean relaxation of the constraints (3), ensuring that each job is carried out exactly once. The resulting Lagrangean Relaxation (LR) is written as:

  *LR*:

$$Z_{LR}(v) = \min \sum_{c=1}^{C} Y_c + \sum_{j=1}^{J} (v_j - \sum_{c \in \mathcal{C}_{a_j}} v_j x_{j,c}) \qquad (1')$$

$$= \min \sum_{c=1}^{C} (Y_c - \sum_{\{j | j \in \mathcal{J}_c\}} v_j x_{j,c}) + \sum_{j=1}^{J} v_j \qquad (1'')$$

  subject to (2), (4), and (5)

  where $v_j$ is the (unconstrained) Lagrangean multiplier corresponding to job $j$.

Obviously, LR decomposes into $C$ subproblems. Each subproblem corresponds to a machine class $c$, and can be represented by the graph $G_c(v)$. This graph $G_c(v)$ is obtained from the graph $G_c$ by changing the cost coefficient of each arc corresponding to job $j$ into $-v_j$. Note that in calculating $Z_{LR}(v)$ the amount of flow that must be transported in $G_c$ is not known in advance, as the amount of flow in this graph represents $Y_c$. Therefore, $Z_{LR}(v)$ is calculated by the following incrementing minimum cost flow procedure:

**Incrementing minimum cost flow procedure:**

*For* $c := 1, \ldots, C$ *do*
{ $f := 0$ ; $Z_c^* := 0$ ; Optimal := false;
  *Repeat*
    $f := f + 1$;
    Find a minimum cost flow of $f$ units of flow from $n_{c,1}$ to $n_{c,p_c}$ on the graph $G_c(v)$;

Call the resulting solution $Z_c$;
*If* $Z_c + 1 < Z_c^*$, *then* $Z_c^* := Z_c$ *else* Optimal := true;
*Until* Optimal;
$Y_c := f - 1$; }

The lower bound $Z_{LR}(v)$ to $Z_{IP}$ is now obtained as $\sum_{c=1}^{C}(Z_c^* + Y_c) + \sum_{j=1}^{J} v_j$. The minimum cost flow problems on the subgraphs $G_c(v)$ can be solved by the strongly polynomial algorithm of Orlin (1988).
For *updating* the Lagrangean multipliers we use the subgradient optimization procedure described by Fisher (1981). For the actual implementation of our procedure, we have adopted the following stopping criteria: the procedure is stopped when either *(i)* Lagrangean multipliers have been updated $J$ times, or *(ii)* the difference between the Lagrangean lower bound and the *'greedy'* upper bound that will be introduced in Section 6 has become less than 1. The latter implies that the optimal solution has been obtained. The best Lagrangean lower bound that we found upon execution of the procedure is denoted by $Z_{LR}$. Note that, due to convergence problems, we often found $Z_{LR} < \max_v Z_{LR}(v)$.

**Remark**: At the initial design of the LR procedure we have tried several alternative stopping criteria and updating mechanisms for the Lagrangean multipliers, including dual-ascent. However, the effects of these alternative stopping criteria and updating mechanism on the quality of the solutions and on CPU-times were only marginal.

**Lemma 5** *The following dominance relations exist between the above lower bounds:*

$$Z_{CR} \leq Z_{LP}, \quad Z_{CR} \leq Z_{NR}, \quad \max_v Z_{LR}(v) = Z_{LP}, \quad Z_{CR} \leq \max_v Z_{LR}(v).$$

*These dominance relations are the only ones within the given set of lower bounds.*

**Proof.**

- Problem CR is obtained by replacing $\mathcal{A}_c$ by $\{1, \ldots, A\}$. By doing so the problem reduces to FJSP, which can be solved to optimality by Linear Programming. Indeed, the coefficient matrix of any instance of FJSP is an interval matrix which is totally unimodular. As a consequence, $Z_{CR} \leq Z_{LP}$.

- Both the job overlap at the instant $t$ and the maximum job overlap do not change by allowing preemption of the jobs. Furthermore, $Z_{NR}$ is obviously greater than or equal to the maximum job overlap. This implies $Z_{CR} \leq Z_{NR}$.

- The relation $\max_v Z_{LR}(v) = Z_{LP}$ follows from the fact that LR decomposes into $C$ subproblems, which are all network flow problems. As a consequence, each of these subproblems has the integrality property. Hence the dominance relation follows from Geoffrion (1974).

- The relation $Z_{CR} \leq \max_v Z_{LR}(v)$ follows from the relations $\max_v Z_{LR}(v) = Z_{LP}$ and $Z_{CR} \leq Z_{LP}$ which were described above.

- The absence of a dominance relation between $Z_{LP}$ and $Z_{NR}$ can be seen from the following instances involving 3 job classes and 3 machine classes, where $\mathcal{A}_c = \{1,2,3\} \setminus \{c\}$ for $c = 1,2,3$: if the 3 jobs $(s_j, f_j, a_j) = (0,1,1)$, $(1,3,2)$ and $(3,4,3)$ have to be carried out, then $Z_{LP} = \frac{3}{2}$ and $Z_{NR} = 2$. However, if the jobs $(0,2,3)$ and $(2,4,1)$ are added, then $Z_{LP} = \frac{5}{2}$ and $Z_{NR} = 2$.

  The absence of a dominance relation between $Z_{LR}$ and $Z_{NR}$ follows from $Z_{LP} = \max_v Z_{LR}(v)$ and the absence of a dominance relation between $Z_{LP}$ and $Z_{NR}$. □

# 6 Upper bounds

Upper bounds to $Z_{IP}$ are obtained by constructing a feasible solution for all jobs. We consider the following upper bounds:

- An upper bound obtained by covering all job classes by an *appropriate* set of machine classes. This bound, denoted by Class Covering (CC), is described below.

- An upper bound obtained by applying a greedy heuristic to obtain a feasible solution for all jobs. This greedy bound is denoted by GR.

First we will explain the CC-bound. Let $\alpha$ be a set of job classes. We say that $\alpha$ is a *feasible subset of job classes* if $\alpha \subset \mathcal{A}_c$ for some machine class $c$. Next, let $\{\alpha_1, \ldots, \alpha_N\}$ be a collection of feasible subsets *'covering'* all job classes. Thus, $\cup_n \alpha_n = \{1, \ldots, A\}$. A feasible subset $\alpha_n$ in this cover is said to be *non-redundant* if the collection $\{\alpha_1, \ldots, \alpha_N\} \setminus \{\alpha_n\}$ does not cover all job classes.

**Lemma 6** *If $\{\alpha_1, \ldots, \alpha_N\}$ is a collection of feasible subsets covering all job classes, then $Z_{CC} = \sum_{n=1}^{N} L_{\alpha_n}$ is an upper bound to $Z_{IP}$. If all feasible subsets in the cover are non-redundant, then this upper bound is tight.*

**Proof.** If $\alpha_n \subset \mathcal{A}_c$ is a feasible subset, then all jobs $j$ with $a_j \in \alpha_n$ can be carried out by $L_{\alpha_n}$ machines in machine class $c$. Hence if $\{\alpha_1, \ldots, \alpha_N\}$ is a collection of feasible subsets covering all job classes, then all jobs can be carried out by $\sum_{n=1}^{N} L_{\alpha_n}$ machines.
The fact that the bound is sharp if all feasible subsets in $\{\alpha_1, \ldots, \alpha_N\}$ are non-redundant can be seen as follows. Obviously, each subset $\alpha_n$ contains at least one job class that is not covered by the other subsets. The first job class with this property is denoted by $a(n)$. Now we consider an instance of TFISP with the following structure: the time horizon has been split up into $N$ subintervals, and the $n^{th}$ subinterval contains only jobs in job class $a(n)$. For this instance of TFISP we have $Z_{IP} = \sum_{n=1}^{N} L_{\alpha_n}$. □

In general, finding a cover $\{\alpha_1, \ldots, \alpha_N\}$ of all job classes is not difficult. However, if one is interested in a tight upper bound, then the cover of the job classes should be *minimal* in

some sense. In order to obtain a minimal cover, let $\{\alpha_1, \ldots, \alpha_S\}$ be an exhaustive list of all feasible subsets. Note that $S = \mathcal{O}(C2^A)$. The covering problem to be solved involves the binary decision variables $x_s$ $(s = 1, \ldots, S)$. These variables indicate whether feasible subset $s$ should be taken into the cover of the job classes.

$$\min Z_{CC} = \sum_{s=1}^{S} L_{\alpha_s} x_s$$

subject to

$$\sum_{\{s|a \in \alpha_s\}} x_s = 1 \qquad a = 1, \ldots, A$$

$$x_s \in \{0, 1\} \qquad s = 1, \ldots, S$$

If the numbers $A$ and $C$ are predetermined, then a minimum for $Z_{CC}$ can be obtained by complete enumeration in a fixed amount of time. However, if $A$ and $C$ are not predetermined, then calculating a minimum for $Z_{CC}$ involves the solution of an uncapacitated facility location problem, which can be accomplished by the DUALOC procedure of Erlenkotter (1978).
A special case of $Z_{CC}$ is obtained by assuming that all feasible subsets are singletons. In that case one obtains the upper bound $\sum_{a=1}^{A} L_a$. This upper bound can be determined in $\mathcal{O}(J \log J)$ time.

In the remaining part of this section we describe a *greedy heuristic* that we have implemented to obtain a feasible solution for an instance of TFISP, and hence an upper bound $Z_{GR}$ to $Z_{IP}$. This heuristic is based on repetitively increasing the number of machines until finally all jobs can be carried out by the minimum cost machine configuration. It uses the Lagrangean multipliers $v$ obtained from the Lagrangean relaxation procedure. More formally, a single pass of the heuristic is described as follows:

**Greedy heuristic:**

$\mathcal{J} :=$ {all jobs};
$Y_c := 0$ *for* $c = 1, \ldots, C$;
*Repeat*
  Search for the *'locally best'* machine class $c^*$;
  $Y_{c^*} := Y_{c^*} + 1$;
  $\mathcal{J} := \mathcal{J} \setminus$ {all jobs that can be carried out by one additional machine of $c^*$};
*Until* $\mathcal{J} = \emptyset$;

Note that in the description of the upper bounding procedure the method for finding the locally best machine class $c^*$ is still unspecified. The method that we have implemented for obtaining $c^*$ is as follows. Suppose that, after a number of iterations, we are faced with a machine configuration and $\mathcal{J} \neq \emptyset$. The latter means that the capacity of the machine configuration is still not large enough for carrying out all jobs. Then we tentatively increase for each machine class the number of machines by one, and obtain $c^*$ as the machine class for which such an increase is *'most profitable'*. Here the obtained profit is

defined as the total value of the additional jobs that can be carried out. It is easy to see that $c^*$ can be obtained by solving $C$ shortest path problems on the graphs $\overline{G}_c(v)$. Here the graph $\overline{G}_c(v)$ is obtained from the graph $G_c(v)$ (as defined in the Lagrangean relaxation procedure) by deleting all job arcs corresponding to jobs $j \notin \mathcal{J}$. The resulting solution for the graph $\overline{G}_c(v)$ is called $\overline{Z}_c(v)$. Now, the locally best machine class is defined by $c^* :=$ argmin $\{\overline{Z}_c(v)|c = 1, \ldots, C\}$.
The single pass upper bound $Z_{GR}(v)$ equals $\sum_c Y_c$. Note that the computational effort required to compute $Z_{GR}(v)$ is $\mathcal{O}(J \sum_c |J_c|^2)$. The single pass upper bound $Z_{GR}(v)$ is computed for different values of the Lagrangean multipliers $v$ after every 5 iterations of the Lagrangean lower bounding procedure. The resulting multi pass greedy upper bound $Z_{GR}$ equals the minimum over all single pass upper bounds, i.e. $Z_{GR} = \min_v Z_{GR}(v)$, where the minimum is taken over all applied Lagrangean multipliers.

# 7 Computational experiments

In order to test the quality of the lower and upper bounding procedures, we have implemented the procedures on an IBM RS/6000 model 370 with 128 Mb internal memory and a clock-frequency of 62.5 MHz under the AIX operating system.

*Implementation of the procedures*

The details on implementation of the aforementioned procedures are as follows:

- The procedures to compute the lower bounds of class-relaxation ($Z_{CR}$), Lagrangean relaxation ($Z_{LR}$), and the greedy upper bound ($Z_{GR}$) have been implemented in the programming language PASCAL,
- The procedures to determine the value of the non-preemptive relaxation ($Z_{NR}$), the value of the linear programming relaxation $Z_{LP}$, and the value of the optimal solution to TFISP ($Z_{IP}$) have been implemented in the programming language FORTRAN using the optimization library OSL (IBM, 1991),
- The class covering upper bounding procedure has been implemented partly in PASCAL (to generate the subsets $\alpha$), and partly in FORTRAN (using Erlenkotter's DUALOC-code to solve the location problem),

*Design of test problems*

We have generated four sets of problem instances. For all instances the planning horizon $T$ has been fixed at 1000 time units. Within each set, the following parameters have been varied:

- the number of job classes $A$. We consider instances with $A = 4$, $A = 5$, or $A = 6$,
- the number of jobs $J$. We consider instances with $J = 100$, $J = 200$, or $J = 300$,

- the maximum job duration $D$. We consider instances with $D = 100$, $D = 200$, or $D = 300$.

Besides this, the sets differ in *two* characteristics, i.e. *(i)* the job overlap over time, and *(ii)* the available machine classes. The details for *(i)* and *(ii)* are as follows:

**Job overlap over time:**

- *Uniformly distributed*: For each job $j$ the job class $a_j$ is chosen randomly from the set $\{1, \ldots, A\}$, and the processing time $d_j$ is chosen randomly from the uniform $U(0, D)$ distribution. The start time $s_j$ is chosen randomly from the uniform $U(0, T - d_j)$ distribution, and the finishing time $f_j$ is set equal to $s_j + d_j$.

- *Peak distribution*: With each job class $a$ we associate a probability $p_a$, and a normal distribution function with mean $\mu_a$ and standard deviation $\sigma_a$. Here $\mu_a$ is the expected peak time of jobs in job class $a$, and $\sigma_a$ is a measure for the spread of the peak. For each job $j$ the job class $a_j$ is randomly chosen from the set $\{1, \ldots, A\}$, where $p_a$ is the probability that the job class is $a$. The job's processing time $d_j$ is generated randomly from the $U(0, D)$ distribution, and its midpoint $m_j$ is drawn from the normal $N(\mu_{a_j}, \sigma_{a_j})$ distribution. The start time of the job $s_j = m_j - \frac{D}{2}$ and the finish time $f_j = m_j + \frac{D}{2}$.[1] The specific parameter settings for $p_a$, $\mu_a$, and $\sigma_a$ are found in Appendix II.

**Available machine classes:**

- *All combinations*: In this case each machine can handle jobs from *two* job classes, and the number of machine classes is chosen such that each possible combination of two job classes is included. This results in the relation $C = \begin{pmatrix} A \\ 2 \end{pmatrix}$,

- *Limited number of combinations*: Here, each machine can also handle jobs from two job classes, but *not all* combinations of two job classes occur. For $A = 4$, $A = 5$ and $A = 6$ we have set $C = 3$, $C = 4$, and $C = 5$ respectively. In all cases the sets $\mathcal{A}_c$ have been constructed such that $\mathcal{A}_c = \{c, c+1\}$ for $c = 1, \ldots, A-1$. For example, if $A = 4$, then $C = 3$, and $\mathcal{A}_1 = \{1, 2\}$, $\mathcal{A}_2 = \{2, 3\}$, and $\mathcal{A}_3 = \{3, 4\}$.

**Remark**: We have also experimented with problem instances where each machine could handle jobs from *three* different job classes. However, since there was not much difference in quality and CPU-times between the results obtained for instances with two job classes per machine and the results obtained for instances with three job classes per machine, we have not included (a discussion on) the latter instances.

[1] Obviously, it may happen that part of a job falls outside the planning horizon. If this occurs, the part of the job that falls outside the planning horizon is not taken into account. If a complete job falls outside the planning horizon, then a new job is generated.

In Table 2 we summarize the characteristics of each set.

| | | **machine classes** | |
|---|---|---|---|
| | | *all* | *limited* |
| **job overlap** | *uniform* | Set I (180 instances) | Set III (270 instances) |
| | *peaks* | Set II (180 instances) | Set IV (270 instances) |

Tables A1–A8 in Appendix III give a detailed presentation of the computational results for Sets I–IV. Here, we summarize the results: Table 3a summarizes the average quality of the solutions aggregated over all instances in each set. Table 3b summarizes the CPU-times over all instances in each set. In Tables 3a and 3b the following notation is used:

| *symbol* | *definition* |
|---|---|
| $\overline{\Delta}_{(LB)}$ | average quality of lower bound (LB). It is computed by taking the average of $\frac{Z_{IP}-Z_{(LB)}}{Z_{IP}}$ over all instances in a set. The lower bounds that are considered are the CR-bound, the NR-bound, the LP-bound, and the LR-bound, |
| $\overline{E}_{LP}$ | the number of times that $Z_{LP}$ equals $Z_{IP}$ over all instances in a set, |
| $\overline{I}_{LP}$ | the number of times that the LP-relaxation yields an integer solution over all instances in a set, |
| $\overline{\Delta}_{(UB)}$ | the average quality of the upper bound (UB). It is computed by taking the average of $\frac{Z_{(UB)}-Z_{IP}}{Z_{IP}}$ over all instances in a set. The upper bounds that are considered are the CC-bound, and the GR-bound, |
| $\overline{CPU}_{(.)}$ | the average CPU-time (in seconds) of procedure (.) over all instances in a set. |

**Remark**: One should be careful when comparing the results listed in Table 3b. The figures listed in the table are averages over instances of *different* dimensions. So, a comparison of the results between different sets is not appropriate, and may lead to wrong conclusions. The table is used to compare the CPU-times of different bounding procedures *within* a *single* set only.

| | | *lower bounds* | | | | | | *upper bounds* | |
|---|---|---|---|---|---|---|---|---|---|
| | # | $\overline{\Delta_{CR}}$ | $\overline{\Delta_{NR}}$ | $\overline{\Delta_{LP}}$ | $\overline{E_{LP}}$ | $\overline{I_{LP}}$ | $\overline{\Delta_{LR}}$ | $\overline{\Delta_{CC}}$ | $\overline{\Delta_{GR}}$ |
| Set I | 180 | 0.000 | 0.000 | 0.000 | 9.889 | 2.389 | 0.059 | 0.158 | 0.001 |
| Set II | 180 | 0.057 | 0.001 | 0.007 | 6.389 | 2.500 | 0.038 | 0.299 | 0.006 |
| Set III | 270 | 0.064 | 0.029 | 0.000 | 9.852 | 9.556 | 0.005 | 0.105 | 0.017 |
| Set IV | 270 | 0.266 | 0.006 | 0.000 | 9.741 | 9.519 | 0.004 | 0.158 | 0.014 |

**Table 3a**: Summary of computational results with respect to quality.

| | | *lower bounds* | | | | *upper bounds* | | *optimal* |
|---|---|---|---|---|---|---|---|---|
| | # | $CPU_{CR}$ | $CPU_{NR}$ | $CPU_{LP}$ | $CPU_{LR}$ | $CPU_{CC}$ | $CPU_{GR}$ | $CPU_{IP}$ |
| Set I | 180 | 0.012 | 4.122 | 1.965 | 18.098 | 0.012 | 2.667 | 80.103 |
| Set II | 180 | 0.012 | 4.588 | 2.618 | 18.408 | 0.012 | 5.090 | 58.318 |
| Set III | 270 | 0.012 | 1.779 | 0.599 | 14.284 | 0.011 | 1.890 | 1.081 |
| Set IV | 270 | 0.012 | 1.209 | 0.654 | 13.765 | 0.011 | 1.467 | 0.965 |

**Table 3b.** Summary of computational results with respect to CPU-times.

From Tables 3a and 3b, and from the Tables A1–A8 the following conclusions can be drawn:

**Lower bounding procedures:**

- The value of the LP-bound turns out to be surprisingly good: in many cases the bound equals the value of the optimal solution. The LP-bound outperforms all other bounds, except for the instances of Set II, where the NR-bound is somewhat better. However, although the LP-bound is very tight for all sets, the corresponding solution is often fractional, especially for Set I and Set II. Furthermore, solving the linear program can be done relatively fast (on average within 9 seconds for the largest instances listed in Tables A2 and A4),

- The NR-bound behaves fairly well. Gaps are small (within 3% on average), and CPU-times are acceptable (on average within 13 seconds for the largest instances listed in Tables A2 and A4). Comparing the NR-bound with the LP-bound, we conclude that the quality of the LP-bound is very often better than the quality of the NR-bound, whereas CPU-times of the LP-bound are less. As for the LP-bound, the quality of the NR-bound is not very sensitive to the characteristics of the instances in each set,

- The quality of the LR-bound varies among the sets. For Sets III and IV the results are better than for Sets I and II. For the latter sets, gaps may go up to 10%. Furthermore, CPU-times are large compared to the CPU-times for all other bounds. Summarizing, we conclude that in comparison with the other bounding procedures Lagrangean relaxation is not a very successful method here. A similar conclusion was drawn by Kroon et al. (1992) in the context of OFISP,

- The quality of the CR-bound varies significantly among the sets, but the CPU-times required to calculate the CR-bound are always very small. In Set I there is almost *no gap* between the CR-bound and the optimal solution, whereas in Set IV the gap goes up to 27% on average,

- As we have shown (Lemma 5), no dominance relations exist between the NR-bound and the LP-bound, nor between the NR-bound and the LR-bound. However, we conclude from Table 3a that, with respect to the quality of the solutions, the LP-bound is frequently better than the NR-bound, and the NR-bound is frequently better than the LR-bound,

- With respect to the influence of the available machine classes, we conclude that the quality of the CR-bound and the NR-bound is better for instances of Sets I and II than for instances of Sets III and IV. This may be caused by the fact that for determining the CR-bound and the NR-bound *less* of the problem structure is relaxed if the number of machine classes is *large*. The quality of the LP-bound and the LR-bound becomes *better* when the number of machine classes *decreases*. This is caused by the fact that the dimension of the corresponding model formulations heavily depends on the number of machine classes: the larger the number of machine classes, the larger the number of decision variables $x_{j,c}$ and $Y_c$ in the LP formulation, and the larger the number of decision variables in the LR formulation that are dependent of the Lagrangean multipliers in the Lagrangean objective ($1''$).

- Somewhat counter intuitive, there is no clear direction in the influence of peaks in the job overlap on the quality of the lower bounds. With peaks, the quality of the CR-bound decreases, the quality of the LR-bound increases, and for the NR and LP-bounds no clear conclusion can be drawn on the (direction of) quality changes.

**Upper bounding procedures:**

- The GR-bound clearly outperforms the CC-bound with respect to the quality of the solutions. However, the CPU-time required to compute GR is larger than the CPU-time required to compute CC,

- Somewhat counter intuitive it is concluded from a comparison between the results for the instances in Sets I and II, and Sets III and IV, that, when the number of machine classes *decreases*, the quality of the GR-bound also *decreases*. Furthermore, the occurrence of peaks in the job overlap has only a *marginal* effect on the quality of the GR-bound,

- The quality of the CC-bound becomes *better* with *fewer* machine classes, and becomes *worse* with the occurrence of peaks.

**Optimal solutions:**

- Obtaining optimal integer solutions using the standard Branch & Bound procedure implemented in OSL works relatively well for medium sized problem instances. This is due to the fact that the LP-bound is very tight. However, if the LP solution is *fractional*, it may be a very time consuming task for larger sized instances to find an optimal integer solution using a Branch & Bound technique (CPU-times may go up to 10 minutes on average). The latter is *also* true when the LP solution is fractional and $Z_{IP} = Z_{LP}$.

# 8 Summary and conclusions

In this paper we consider the Tactical Fixed Interval Scheduling Problem. We formulate the problem as an integer program, and as a network flow problem with side constraints.

We show that some special cases of TFISP can be solved in polynomial time, and we settle the status of the computational complexity of the preemptive variant of TFISP: the problem is solvable in polynomial time if the number of machine classes is predetermined, otherwise the problem is NP-hard.
Furthermore, we develop a number of lower bounding procedures for TFISP, and we derive all dominance relations between these bounds. Also, we describe two upper bounding procedures.
Finally, in a computational study we analyse all upper and lower bounding procedures, both with respect to quality of the solutions, and required CPU-times.

We conclude that, although TFISP is categorized as NP-hard, the instances of the problems that we have considered in this study (which had dimensions and other characteristics that closely resemble the dimensions of the problems that we met in a practical setting reported in Dijkstra et al. (1991)), could be solved relatively fast by some of the procedures described in this paper. Especially linear programming in combination with a Branch & Bound procedure turned out to be rather effective for medium sized problems.
Our future research on TFISP includes the search for polyhedral methods to reduce the number of fractional variables in the solution to the linear programming relaxation, and the development and implementation of special purpose Branch & Bound procedures.

**Acknowledgments** We want to thank professor Antoon Kolen from Limburg University, the Netherlands, for several helpful suggestions.

# 9 References

E.M. ARKIN, AND E.L. SILVERBERG, 'Scheduling jobs with fixed start and finish times,' *Discrete Applied Mathematics*, 18 (1987), 1–8.

M.W. CARTER, 'A Lagrangean relaxation approach to the classroom assignment problem,' *INFOR*, 27 (1989), 230–246.

M.W. CARTER, AND C.A. TOVEY, 'When is the class room assignment problem hard?' *Operations Research*, 40 (1992) S28–S39.

G.L. DANTZIG, AND D.R. FULKERSON, 'Minimizing the number of tankers to meet a fixed schedule,' *Naval Research Logistics Quarterly*, 1 (1954), 217–222.

M.C. DIJKSTRA, L.G. KROON, J.A.E.E. VAN NUNEN, AND M. SALOMON, 'A DSS for capacity planning of aircraft maintenance personnel,' *International Journal of Production Economics*, 23 (1991), 69–78.

R.P. DILWORTH, 'A decomposition principle for partially ordered sets,' *Annals of Mathematics*, 51 (1950) 161–166.

V.R. DONDETI, AND H. EMMONS, 'Interval scheduling with processors of two types,' *Operations Research*, 40 (1992) S76–S85.

V.R. DONDETI, AND H. EMMONS, 'Algorithms for preemptive scheduling of different classes of processors to do jobs with fixed times,' *European Journal of Operational Research*, 70 (1993) 316–326.

D. ERLENKOTTER, 'A dual-based procedure for uncapacitated facility location,' *Operations Research*, 26 (1978) 992–1000.

M. FISCHETTI, S. MARTELLO, AND P. TOTH, 'The Fixed Job Schedule Problem with spread time constraints,' *Operations Research*, 6 (1987) 849–858.

M. FISCHETTI, S. MARTELLO, AND P. TOTH, 'The Fixed Job Schedule Problem with working time constraints,' *Operations Research*, 3 (1989) 395–403.

M. FISCHETTI, S. MARTELLO AND P. TOTH, 'Approximation algorithms for Fixed Job Schedule Problems,' *Operations Research*, 40 (1992) S96-S108.

M.L. FISHER, 'The Lagrangean relaxation method for solving integer programming problems,' *Management Science*, 27 (1981), 1–18.

A.M. GEOFFRION, 'Lagrangean relaxation and its uses in integer programming,' *Mathematical Programming Study*, 2 (1974), 82–114.

I. GERTSBAKH, AND H.I. STERN, 'Minimal resources for fixed and variable job schedules,' *Operations Research*, 18 (1978), 68–85.

U.L. GUPTA, D.T. LEE, AND J.Y.-T LEUNG, 'An optimal solution to the channel assignment problem,' *IEEE Trans. Comp*, C-28 (1979), 807–810.

E. HAGDORN, AND L.G. KROON, 'Pitfalls in obtaining solutions for an aircraft to gate assignment problem,' Man. Rep. Series 59, Rotterdam School of Management, Erasmus University, 1990.

A. HASHIMOTO, AND J. STEVENS, 'Wire routing by optimizing channel assignments within large apertures,' *In:* Proceedings of the 8th Design Automation Workshop, (1971), 155–169.

IBM, 'Optimization Subroutine Library, Release 2: Guide and References,' Third edition, July 1991.

A.W.J. KOLEN, AND L.G. KROON, 'On the computational complexity of (maximum) class scheduling,' *European Journal of Operational Research*, 54 (1991), 23–38.

A.W.J. KOLEN, AND L.G. KROON, 'Licence Class Design: complexity and algorithms,' *European Journal of Operational Research*, 63 (1992), 432–444.

A.W.J. KOLEN, AND L.G. KROON, 'On the computational complexity of (maximum) shift class scheduling,' *European Journal of Operational Research*, 64 (1993), 138–151.

L.G. KROON, 'Job Scheduling and Capacity Planning in Aircraft Maintenance,' PhD Thesis, Rotterdam School of Management, Erasmus University, 1990.

L.G. KROON, M. SALOMON, AND L.N. VAN WASSENHOVE, 'Exact and approximation algorithms for the Operational Fixed Interval Scheduling Problem,' Man. Rep. Series 103, Rotterdam School of Management, Erasmus University, 1992. *(Forthcoming in European Journal of Operational Research).*

J.B. ORLIN, 'A faster strongly polynomial minimum cost flow algorithm,' *In:* proceedings of the 20th ACM symposium on the theory of computing, 1988.

## Appendix I. Definition of 3DM

**Instance of 3DM:**

- 3 disjoint sets $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$, each one containing $A$ elements.
- A set $\mathcal{W}$ which is a subset of $\mathcal{X} \times \mathcal{Y} \times \mathcal{Z}$.

**Question:**

- Does there exist a subset $\mathcal{W}'$ of $\mathcal{W}$ with $A$ elements such that no two elements of $\mathcal{W}'$ agree in any coordinate?

## Appendix II. Parameter settings for Sets II and IV

| | Case $A = 4$ | | | Case $A = 5$ | | | Case $A = 6$ | | |
|---|---|---|---|---|---|---|---|---|---|
| $a$ | $p_a$ | $\mu_a$ | $\sigma_a$ | $p_a$ | $\mu_a$ | $\sigma_a$ | $p_a$ | $\mu_a$ | $\sigma_a$ |
| 1 | 0.20 | 250 | 100 | 0.15 | 200 | 100 | 0.15 | 200 | 100 |
| 2 | 0.20 | 500 | 100 | 0.15 | 350 | 100 | 0.20 | 350 | 200 |
| 3 | 0.40 | 500 | 300 | 0.40 | 500 | 400 | 0.15 | 500 | 100 |
| 4 | 0.20 | 750 | 100 | 0.15 | 650 | 100 | 0.15 | 500 | 100 |
| 5 | | | | 0.15 | 800 | 100 | 0.20 | 650 | 200 |
| 6 | | | | | | | 0.15 | 800 | 100 |

## Appendix III. Computational results.

The Tables A1–A8 give a detailed presentation of the computational results for Sets I–IV. In the tables we use the following notation:

| *symbol* | *definition* |
|---|---|
| $A$ | the number of job classes, |
| $J$ | the number of jobs, |
| $D$ | the maximum job duration, |
| $\Delta_{(LB)}$ | average quality of lower bound (LB). It is computed by taking the average of $\frac{Z_{IP}-Z_{(LB)}}{Z_{IP}}$ over 10 instances. The lower bounds that are considered are the CR-bound, the NR-bound, the LP-bound, and the LR-bound, |
| $E_{LP}$ | the number of times that $Z_{LP}$ equals $Z_{IP}$ over 10 instances, |
| $I_{LP}$ | the number of times that the LP-relaxation yields an integer solution over 10 instances, |
| $\Delta_{(UB)}$ | the average quality of the upper bound (UB). It is computed by taking the average of $\frac{Z_{(UB)}-Z_{IP}}{Z_{IP}}$ over 10 instances. The upper bounds that are considered are the CC-bound, and the GR-bound, |
| $CPU_{(.)}$ | the average CPU-time (in seconds) of procedure (.) over 10 instances. |

Tables A1 and A2 present the results for Set I.

| | | | lower bounds | | | | | | upper bounds | |
|---|---|---|---|---|---|---|---|---|---|---|
| $A$ | $J$ | $D$ | $\Delta_{CR}$ | $\Delta_{NR}$ | $\Delta_{LP}$ | $E_{LP}$ | $I_{LP}$ | $\Delta_{LR}$ | $\Delta_{CC}$ | $\Delta_{GR}$ |
| 4 | 100 | 100 | 0.000 | 0.000 | 0.000 | 10 | 5 | 0.078 | 0.169 | 0.000 |
| 4 | 100 | 200 | 0.000 | 0.000 | 0.000 | 10 | 6 | 0.032 | 0.123 | 0.000 |
| 4 | 100 | 300 | 0.000 | 0.000 | 0.000 | 10 | 6 | 0.042 | 0.082 | 0.000 |
| 4 | 200 | 100 | 0.000 | 0.000 | 0.000 | 10 | 5 | 0.073 | 0.120 | 0.000 |
| 4 | 200 | 200 | 0.003 | 0.003 | 0.002 | 9 | 3 | 0.053 | 0.064 | 0.000 |
| 4 | 200 | 300 | 0.000 | 0.000 | 0.000 | 10 | 2 | 0.033 | 0.047 | 0.000 |
| 4 | 300 | 100 | 0.000 | 0.000 | 0.000 | 10 | 4 | 0.087 | 0.102 | 0.000 |
| 4 | 300 | 200 | 0.000 | 0.000 | 0.000 | 10 | 2 | 0.044 | 0.081 | 0.000 |
| 4 | 300 | 300 | 0.000 | 0.000 | 0.000 | 10 | 3 | 0.027 | 0.051 | 0.000 |
| 5 | 100 | 100 | 0.000 | 0.000 | 0.000 | 10 | 1 | 0.093 | 0.255 | 0.000 |
| 5 | 100 | 200 | 0.000 | 0.000 | 0.000 | 10 | 0 | 0.053 | 0.220 | 0.006 |
| 5 | 100 | 300 | 0.000 | 0.000 | 0.000 | 10 | 3 | 0.054 | 0.133 | 0.004 |
| 5 | 200 | 100 | 0.000 | 0.000 | 0.000 | 10 | 2 | 0.090 | 0.445 | 0.000 |
| 5 | 200 | 200 | 0.003 | 0.003 | 0.001 | 9 | 0 | 0.049 | 0.138 | 0.000 |
| 5 | 200 | 300 | 0.000 | 0.000 | 0.000 | 10 | 0 | 0.041 | 0.225 | 0.000 |
| 5 | 300 | 100 | 0.000 | 0.000 | 0.000 | 10 | 0 | 0.098 | 0.336 | 0.000 |
| 5 | 300 | 200 | 0.000 | 0.000 | 0.000 | 10 | 0 | 0.063 | 0.162 | 0.000 |
| 5 | 300 | 300 | 0.000 | 0.000 | 0.000 | 10 | 1 | 0.045 | 0.093 | 0.000 |

**Table A1.** Quality of lower and upper bounding procedures (Set I).

| | | | lower bounds | | | | upper bounds | | optimal |
|---|---|---|---|---|---|---|---|---|---|
| $A$ | $J$ | $D$ | $CPU_{CR}$ | $CPU_{NR}$ | $CPU_{LP}$ | $CPU_{LR}$ | $CPU_{CC}$ | $CPU_{GR}$ | $CPU_{IP}$ |
| 4 | 100 | 100 | 0.010 | 0.521 | 0.293 | 1.553 | 0.013 | 0.296 | 0.575 |
| 4 | 100 | 200 | 0.010 | 0.567 | 0.331 | 1.650 | 0.012 | 0.744 | 1.241 |
| 4 | 100 | 300 | 0.011 | 0.585 | 0.314 | 1.990 | 0.011 | 0.424 | 0.816 |
| 4 | 200 | 100 | 0.013 | 2.158 | 0.884 | 8.689 | 0.013 | 0.671 | 4.216 |
| 4 | 200 | 200 | 0.011 | 2.088 | 1.078 | 11.244 | 0.012 | 1.186 | 12.693 |
| 4 | 200 | 300 | 0.013 | 1.968 | 1.398 | 14.351 | 0.012 | 2.382 | 16.171 |
| 4 | 300 | 100 | 0.011 | 4.161 | 1.969 | 28.259 | 0.011 | 2.276 | 20.722 |
| 4 | 300 | 200 | 0.012 | 4.246 | 2.705 | 38.566 | 0.012 | 4.156 | 59.050 |
| 4 | 300 | 300 | 0.012 | 4.262 | 3.522 | 50.284 | 0.012 | 5.719 | 52.158 |
| 5 | 100 | 100 | 0.012 | 1.012 | 0.513 | 2.046 | 0.012 | 0.683 | 1.499 |
| 5 | 100 | 200 | 0.010 | 1.218 | 0.700 | 2.283 | 0.012 | 1.129 | 5.579 |
| 5 | 100 | 300 | 0.011 | 1.436 | 0.629 | 2.534 | 0.013 | 1.011 | 5.103 |
| 5 | 200 | 100 | 0.012 | 5.727 | 2.126 | 10.358 | 0.013 | 2.957 | 41.019 |
| 5 | 200 | 200 | 0.012 | 5.108 | 2.240 | 13.019 | 0.012 | 2.597 | 74.585 |
| 5 | 200 | 300 | 0.013 | 4.803 | 2.420 | 15.560 | 0.013 | 2.793 | 79.543 |
| 5 | 300 | 100 | 0.011 | 10.945 | 4.325 | 30.405 | 0.011 | 5.921 | 200.648 |
| 5 | 300 | 200 | 0.013 | 11.540 | 4.363 | 42.398 | 0.014 | 3.687 | 300.193 |
| 5 | 300 | 300 | 0.014 | 11.856 | 5.567 | 50.576 | 0.014 | 9.382 | 566.045 |

**Table A2.** CPU-times (in seconds) for lower and upper bounding procedures (Set I)

Tables A3 and A4 present the results for Set II.

| | | | lower bounds | | | | | | upper bounds | |
|---|---|---|---|---|---|---|---|---|---|---|
| $A$ | $J$ | $D$ | $\Delta_{CR}$ | $\Delta_{NR}$ | $\Delta_{LP}$ | $E_{LP}$ | $I_{LP}$ | $\Delta_{LR}$ | $\Delta_{CC}$ | $\Delta_{GR}$ |
| 4 | 100 | 100 | 0.077 | 0.000 | 0.023 | 4 | 3 | 0.051 | 0.198 | 0.008 |
| 4 | 100 | 200 | 0.038 | 0.000 | 0.005 | 8 | 4 | 0.028 | 0.236 | 0.010 |
| 4 | 100 | 300 | 0.023 | 0.000 | 0.004 | 8 | 5 | 0.018 | 0.238 | 0.008 |
| 4 | 200 | 100 | 0.056 | 0.000 | 0.009 | 5 | 3 | 0.051 | 0.220 | 0.004 |
| 4 | 200 | 200 | 0.039 | 0.000 | 0.003 | 8 | 2 | 0.034 | 0.263 | 0.008 |
| 4 | 200 | 300 | 0.004 | 0.000 | 0.002 | 9 | 4 | 0.020 | 0.272 | 0.008 |
| 4 | 300 | 100 | 0.053 | 0.000 | 0.006 | 7 | 3 | 0.052 | 0.300 | 0.022 |
| 4 | 300 | 200 | 0.031 | 0.000 | 0.001 | 9 | 4 | 0.026 | 0.256 | 0.009 |
| 4 | 300 | 300 | 0.015 | 0.000 | 0.001 | 8 | 2 | 0.019 | 0.250 | 0.008 |
| 5 | 100 | 100 | 0.133 | 0.006 | 0.014 | 6 | 2 | 0.041 | 0.368 | 0.000 |
| 5 | 100 | 200 | 0.055 | 0.000 | 0.010 | 5 | 2 | 0.046 | 0.379 | 0.000 |
| 5 | 100 | 300 | 0.046 | 0.011 | 0.012 | 5 | 2 | 0.033 | 0.418 | 0.000 |
| 5 | 200 | 100 | 0.149 | 0.000 | 0.017 | 2 | 1 | 0.063 | 0.388 | 0.000 |
| 5 | 200 | 200 | 0.061 | 0.000 | 0.005 | 6 | 3 | 0.035 | 0.409 | 0.005 |
| 5 | 200 | 300 | 0.011 | 0.000 | 0.001 | 9 | 2 | 0.031 | 0.310 | 0.009 |
| 5 | 300 | 100 | 0.136 | 0.000 | 0.008 | 4 | 2 | 0.063 | 0.371 | 0.003 |
| 5 | 300 | 200 | 0.074 | 0.002 | 0.005 | 6 | 0 | 0.042 | 0.303 | 0.002 |
| 5 | 300 | 300 | 0.023 | 0.003 | 0.004 | 6 | 1 | 0.025 | 0.201 | 0.003 |

**Table A3.** Quality of lower and upper bounding procedures (Set II).

| | | | lower bounds | | | | upper bounds | | optimal |
|---|---|---|---|---|---|---|---|---|---|
| $A$ | $J$ | $D$ | $CPU_{CR}$ | $CPU_{NR}$ | $CPU_{LP}$ | $CPU_{LR}$ | $CPU_{CC}$ | $CPU_{GR}$ | $CPU_{IP}$ |
| 4 | 100 | 100 | 0.012 | 0.583 | 0.341 | 1.455 | 0.013 | 0.357 | 1.010 |
| 4 | 100 | 200 | 0.014 | 0.613 | 0.375 | 1.687 | 0.012 | 0.841 | 1.063 |
| 4 | 100 | 300 | 0.011 | 0.671 | 0.401 | 1.988 | 0.013 | 1.223 | 1.216 |
| 4 | 200 | 100 | 0.010 | 2.115 | 1.147 | 9.172 | 0.013 | 1.330 | 5.212 |
| 4 | 200 | 200 | 0.013 | 2.087 | 1.422 | 12.092 | 0.010 | 3.050 | 10.245 |
| 4 | 200 | 300 | 0.012 | 2.413 | 1.703 | 14.197 | 0.010 | 3.655 | 23.417 |
| 4 | 300 | 100 | 0.012 | 4.803 | 2.730 | 29.323 | 0.011 | 5.816 | 14.512 |
| 4 | 300 | 200 | 0.013 | 5.037 | 3.735 | 42.721 | 0.013 | 8.890 | 31.273 |
| 4 | 300 | 300 | 0.014 | 5.470 | 4.331 | 52.829 | 0.015 | 14.803 | 69.193 |
| 5 | 100 | 100 | 0.012 | 1.282 | 0.625 | 1.871 | 0.013 | 0.719 | 1.843 |
| 5 | 100 | 200 | 0.011 | 1.607 | 0.702 | 2.183 | 0.011 | 0.993 | 5.068 |
| 5 | 100 | 300 | 0.011 | 1.528 | 0.743 | 2.322 | 0.014 | 1.527 | 4.570 |
| 5 | 200 | 100 | 0.012 | 5.899 | 2.301 | 10.431 | 0.012 | 2.371 | 25.063 |
| 5 | 200 | 200 | 0.012 | 5.690 | 2.511 | 12.552 | 0.016 | 2.197 | 49.218 |
| 5 | 200 | 300 | 0.014 | 5.133 | 2.914 | 14.227 | 0.013 | 6.406 | 58.288 |
| 5 | 300 | 100 | 0.011 | 12.601 | 5.225 | 32.379 | 0.014 | 4.817 | 106.940 |
| 5 | 300 | 200 | 0.012 | 12.396 | 7.301 | 40.896 | 0.011 | 12.136 | 235.694 |
| 5 | 300 | 300 | 0.011 | 12.661 | 8.618 | 49.011 | 0.010 | 20.494 | 405.905 |

**Table A4.** CPU-times (in seconds) for lower and upper bounding procedures (Set II)

Tables A5 and A6 present the results for Set III.

| | | | lower bounds | | | | | | upper bounds | |
|---|---|---|---|---|---|---|---|---|---|---|
| $A$ | $J$ | $D$ | $\Delta_{CR}$ | $\Delta_{NR}$ | $\Delta_{LP}$ | $E_{LP}$ | $I_{LP}$ | $\Delta_{LR}$ | $\Delta_{CC}$ | $\Delta_{GR}$ |
| 4 | 100 | 100 | 0.057 | 0.016 | 0.000 | 10 | 10 | 0.000 | 0.131 | 0.016 |
| 4 | 100 | 200 | 0.074 | 0.049 | 0.003 | 9 | 9 | 0.000 | 0.082 | 0.010 |
| 4 | 100 | 300 | 0.041 | 0.041 | 0.000 | 10 | 10 | 0.000 | 0.068 | 0.012 |
| 4 | 200 | 100 | 0.063 | 0.034 | 0.003 | 9 | 9 | 0.000 | 0.084 | 0.015 |
| 4 | 200 | 200 | 0.009 | 0.009 | 0.000 | 10 | 10 | 0.000 | 0.089 | 0.027 |
| 4 | 200 | 300 | 0.022 | 0.022 | 0.000 | 10 | 10 | 0.000 | 0.054 | 0.016 |
| 4 | 300 | 100 | 0.035 | 0.025 | 0.000 | 10 | 10 | 0.017 | 0.082 | 0.018 |
| 4 | 300 | 200 | 0.030 | 0.019 | 0.000 | 10 | 10 | 0.002 | 0.077 | 0.015 |
| 4 | 300 | 300 | 0.023 | 0.023 | 0.000 | 10 | 10 | 0.000 | 0.052 | 0.019 |
| 5 | 100 | 100 | 0.127 | 0.027 | 0.000 | 10 | 10 | 0.000 | 0.103 | 0.007 |
| 5 | 100 | 200 | 0.084 | 0.033 | 0.000 | 10 | 10 | 0.000 | 0.107 | 0.015 |
| 5 | 100 | 300 | 0.043 | 0.025 | 0.000 | 10 | 10 | 0.000 | 0.091 | 0.027 |
| 5 | 200 | 100 | 0.123 | 0.014 | 0.000 | 10 | 10 | 0.000 | 0.159 | 0.014 |
| 5 | 200 | 200 | 0.045 | 0.036 | 0.000 | 10 | 8 | 0.006 | 0.107 | 0.008 |
| 5 | 200 | 300 | 0.030 | 0.016 | 0.000 | 10 | 10 | 0.000 | 0.098 | 0.019 |
| 5 | 300 | 100 | 0.073 | 0.021 | 0.000 | 10 | 10 | 0.020 | 0.139 | 0.018 |
| 5 | 300 | 200 | 0.028 | 0.024 | 0.000 | 10 | 10 | 0.008 | 0.081 | 0.020 |
| 5 | 300 | 300 | 0.016 | 0.016 | 0.000 | 10 | 9 | 0.006 | 0.075 | 0.024 |
| 6 | 100 | 100 | 0.156 | 0.030 | 0.000 | 10 | 10 | 0.000 | 0.148 | 0.016 |
| 6 | 100 | 200 | 0.095 | 0.028 | 0.000 | 10 | 10 | 0.000 | 0.121 | 0.014 |
| 6 | 100 | 300 | 0.086 | 0.050 | 0.000 | 10 | 10 | 0.000 | 0.104 | 0.022 |
| 6 | 200 | 100 | 0.142 | 0.034 | 0.000 | 10 | 10 | 0.013 | 0.154 | 0.020 |
| 6 | 200 | 200 | 0.085 | 0.040 | 0.001 | 9 | 8 | 0.009 | 0.132 | 0.009 |
| 6 | 200 | 300 | 0.058 | 0.046 | 0.000 | 10 | 10 | 0.000 | 0.112 | 0.022 |
| 6 | 300 | 100 | 0.092 | 0.037 | 0.002 | 9 | 9 | 0.031 | 0.175 | 0.014 |
| 6 | 300 | 200 | 0.059 | 0.033 | 0.000 | 10 | 8 | 0.014 | 0.102 | 0.015 |
| 6 | 300 | 300 | 0.035 | 0.035 | 0.000 | 10 | 8 | 0.004 | 0.107 | 0.027 |

**Table A5.** Quality of the lower and upper bounding procedures (Set III)

| | | | lower bounds | | | | upper bounds | | optimal |
|---|---|---|---|---|---|---|---|---|---|
| $A$ | $J$ | $D$ | $CPU_{CR}$ | $CPU_{NR}$ | $CPU_{LP}$ | $CPU_{LR}$ | $CPU_{CC}$ | $CPU_{GR}$ | $CPU_{IP}$ |
| 4 | 100 | 100 | 0.012 | 0.259 | 0.111 | 0.347 | 0.012 | 0.128 | 0.145 |
| 4 | 100 | 200 | 0.013 | 0.275 | 0.137 | 0.619 | 0.010 | 0.326 | 0.196 |
| 4 | 100 | 300 | 0.012 | 0.273 | 0.144 | 0.845 | 0.012 | 0.290 | 0.189 |
| 4 | 200 | 100 | 0.012 | 0.906 | 0.317 | 3.290 | 0.010 | 0.336 | 0.507 |
| 4 | 200 | 200 | 0.011 | 0.964 | 0.416 | 8.762 | 0.010 | 0.686 | 0.707 |
| 4 | 200 | 300 | 0.013 | 0.892 | 0.562 | 9.428 | 0.010 | 1.065 | 0.835 |
| 4 | 300 | 100 | 0.011 | 2.070 | 0.722 | 21.964 | 0.010 | 1.133 | 1.237 |
| 4 | 300 | 200 | 0.011 | 2.000 | 1.220 | 27.650 | 0.012 | 2.332 | 1.723 |
| 4 | 300 | 300 | 0.013 | 1.812 | 1.524 | 45.644 | 0.012 | 3.185 | 2.206 |
| 5 | 100 | 100 | 0.013 | 0.327 | 0.123 | 0.329 | 0.010 | 0.158 | 0.174 |
| 5 | 100 | 200 | 0.012 | 0.407 | 0.137 | 0.747 | 0.011 | 0.302 | 0.212 |
| 5 | 100 | 300 | 0.014 | 0.424 | 0.150 | 1.365 | 0.011 | 0.392 | 0.220 |
| 5 | 200 | 100 | 0.012 | 1.456 | 0.344 | 4.352 | 0.010 | 0.634 | 0.656 |
| 5 | 200 | 200 | 0.012 | 1.612 | 0.480 | 7.649 | 0.011 | 0.964 | 0.790 |
| 5 | 200 | 300 | 0.011 | 1.402 | 0.628 | 9.913 | 0.011 | 2.398 | 0.937 |
| 5 | 300 | 100 | 0.012 | 3.093 | 0.757 | 20.601 | 0.011 | 1.191 | 1.433 |
| 5 | 300 | 200 | 0.013 | 3.342 | 1.210 | 34.666 | 0.012 | 2.569 | 2.455 |
| 5 | 300 | 300 | 0.014 | 3.102 | 1.493 | 50.976 | 0.010 | 7.961 | 2.438 |
| 6 | 100 | 100 | 0.015 | 0.471 | 0.131 | 0.633 | 0.012 | 0.291 | 0.201 |
| 6 | 100 | 200 | 0.011 | 0.566 | 0.148 | 0.750 | 0.010 | 0.400 | 0.219 |
| 6 | 100 | 300 | 0.011 | 0.655 | 0.168 | 1.463 | 0.010 | 0.433 | 0.260 |
| 6 | 200 | 100 | 0.011 | 2.079 | 0.402 | 5.206 | 0.010 | 0.759 | 0.697 |
| 6 | 200 | 200 | 0.010 | 2.227 | 0.558 | 7.369 | 0.011 | 1.961 | 1.111 |
| 6 | 200 | 300 | 0.013 | 2.376 | 0.630 | 11.234 | 0.012 | 2.421 | 1.202 |
| 6 | 300 | 100 | 0.012 | 5.246 | 0.859 | 23.762 | 0.012 | 1.881 | 1.843 |
| 6 | 300 | 200 | 0.015 | 4.995 | 1.222 | 37.239 | 0.011 | 4.764 | 3.023 |
| 6 | 300 | 300 | 0.014 | 4.798 | 1.577 | 48.853 | 0.012 | 12.060 | 3.560 |

**Table A6.** CPU-times (in seconds) for lower and upper bounding procedures (Set III).

Tables A7 and A8 present the results for Set IV.

| | | | lower bounds | | | | | | upper bounds | |
|---|---|---|---|---|---|---|---|---|---|---|
| $A$ | $J$ | $D$ | $\Delta_{CR}$ | $\Delta_{NR}$ | $\Delta_{LP}$ | $E_{LP}$ | $I_{LP}$ | $\Delta_{LR}$ | $\Delta_{CC}$ | $\Delta_{GR}$ |
| 4 | 100 | 100 | 0.238 | 0.000 | 0.000 | 10 | 10 | 0.000 | 0.055 | 0.000 |
| 4 | 100 | 200 | 0.248 | 0.004 | 0.002 | 9 | 9 | 0.000 | 0.077 | 0.015 |
| 4 | 100 | 300 | 0.182 | 0.006 | 0.002 | 9 | 9 | 0.003 | 0.086 | 0.009 |
| 4 | 200 | 100 | 0.209 | 0.000 | 0.000 | 10 | 10 | 0.000 | 0.111 | 0.022 |
| 4 | 200 | 200 | 0.217 | 0.000 | 0.000 | 10 | 10 | 0.000 | 0.087 | 0.015 |
| 4 | 200 | 300 | 0.171 | 0.000 | 0.000 | 10 | 9 | 0.000 | 0.091 | 0.010 |
| 4 | 300 | 100 | 0.255 | 0.000 | 0.000 | 10 | 10 | 0.019 | 0.074 | 0.015 |
| 4 | 300 | 200 | 0.183 | 0.000 | 0.000 | 10 | 9 | 0.005 | 0.096 | 0.011 |
| 4 | 300 | 300 | 0.192 | 0.000 | 0.000 | 10 | 10 | 0.001 | 0.080 | 0.014 |
| 5 | 100 | 100 | 0.353 | 0.010 | 0.000 | 10 | 10 | 0.005 | 0.114 | 0.004 |
| 5 | 100 | 200 | 0.269 | 0.014 | 0.000 | 10 | 10 | 0.000 | 0.176 | 0.014 |
| 5 | 100 | 300 | 0.270 | 0.014 | 0.000 | 10 | 10 | 0.000 | 0.145 | 0.003 |
| 5 | 200 | 100 | 0.331 | 0.006 | 0.002 | 9 | 9 | 0.013 | 0.190 | 0.026 |
| 5 | 200 | 200 | 0.287 | 0.004 | 0.000 | 10 | 9 | 0.000 | 0.169 | 0.009 |
| 5 | 200 | 300 | 0.265 | 0.006 | 0.000 | 10 | 10 | 0.000 | 0.183 | 0.013 |
| 5 | 300 | 100 | 0.322 | 0.009 | 0.000 | 10 | 9 | 0.021 | 0.191 | 0.020 |
| 5 | 300 | 200 | 0.290 | 0.007 | 0.000 | 10 | 10 | 0.009 | 0.222 | 0.019 |
| 5 | 300 | 300 | 0.284 | 0.005 | 0.000 | 10 | 8 | 0.006 | 0.169 | 0.008 |
| 6 | 100 | 100 | 0.399 | 0.016 | 0.003 | 9 | 9 | 0.000 | 0.121 | 0.017 |
| 6 | 100 | 200 | 0.371 | 0.004 | 0.000 | 10 | 10 | 0.000 | 0.196 | 0.011 |
| 6 | 100 | 300 | 0.207 | 0.021 | 0.003 | 8 | 8 | 0.000 | 0.225 | 0.005 |
| 6 | 200 | 100 | 0.318 | 0.000 | 0.000 | 10 | 10 | 0.000 | 0.231 | 0.026 |
| 6 | 200 | 200 | 0.297 | 0.004 | 0.000 | 10 | 10 | 0.000 | 0.252 | 0.015 |
| 6 | 200 | 300 | 0.246 | 0.019 | 0.000 | 10 | 10 | 0.000 | 0.214 | 0.011 |
| 6 | 300 | 100 | 0.320 | 0.000 | 0.000 | 10 | 10 | 0.014 | 0.213 | 0.018 |
| 6 | 300 | 200 | 0.243 | 0.006 | 0.001 | 9 | 9 | 0.004 | 0.244 | 0.026 |
| 6 | 300 | 300 | 0.217 | 0.008 | 0.000 | 10 | 10 | 0.000 | 0.247 | 0.011 |

**Table A7.** Quality of lower and upper bounding procedures (Set IV).

| | | | lower bounds | | | | upper bounds | | optimal |
|---|---|---|---|---|---|---|---|---|---|
| $A$ | $J$ | $D$ | $CPU_{CR}$ | $CPU_{NR}$ | $CPU_{LP}$ | $CPU_{LR}$ | $CPU_{CC}$ | $CPU_{GR}$ | $CPU_{IP}$ |
| 4 | 100 | 100 | 0.014 | 0.201 | 0.117 | 0.201 | 0.012 | 0.130 | 0.163 |
| 4 | 100 | 200 | 0.012 | 0.222 | 0.141 | 0.829 | 0.011 | 0.208 | 0.205 |
| 4 | 100 | 300 | 0.013 | 0.234 | 0.162 | 0.904 | 0.011 | 0.339 | 0.225 |
| 4 | 200 | 100 | 0.012 | 0.728 | 0.440 | 4.802 | 0.010 | 0.408 | 0.609 |
| 4 | 200 | 200 | 0.012 | 0.750 | 0.582 | 7.638 | 0.011 | 0.619 | 0.773 |
| 4 | 200 | 300 | 0.012 | 0.758 | 0.631 | 9.064 | 0.011 | 1.025 | 0.804 |
| 4 | 300 | 100 | 0.010 | 1.502 | 0.881 | 26.323 | 0.010 | 0.630 | 1.429 |
| 4 | 300 | 200 | 0.014 | 1.634 | 1.394 | 33.657 | 0.011 | 1.627 | 1.958 |
| 4 | 300 | 300 | 0.016 | 1.540 | 1.735 | 46.209 | 0.010 | 1.898 | 2.021 |
| 5 | 100 | 100 | 0.010 | 0.263 | 0.128 | 0.485 | 0.010 | 0.226 | 0.188 |
| 5 | 100 | 200 | 0.012 | 0.320 | 0.153 | 0.967 | 0.012 | 0.467 | 0.234 |
| 5 | 100 | 300 | 0.013 | 0.282 | 0.178 | 0.833 | 0.011 | 0.531 | 0.243 |
| 5 | 200 | 100 | 0.012 | 1.039 | 0.462 | 7.138 | 0.010 | 0.976 | 0.691 |
| 5 | 200 | 200 | 0.013 | 1.055 | 0.560 | 6.897 | 0.010 | 1.492 | 0.798 |
| 5 | 200 | 300 | 0.012 | 1.058 | 0.688 | 11.505 | 0.010 | 2.956 | 1.002 |
| 5 | 300 | 100 | 0.010 | 2.262 | 1.086 | 26.783 | 0.010 | 3.216 | 1.728 |
| 5 | 300 | 200 | 0.011 | 2.344 | 1.508 | 39.572 | 0.011 | 2.495 | 2.382 |
| 5 | 300 | 300 | 0.014 | 2.272 | 1.741 | 50.359 | 0.013 | 3.766 | 2.586 |
| 6 | 100 | 100 | 0.013 | 0.392 | 0.146 | 0.950 | 0.010 | 0.305 | 0.216 |
| 6 | 100 | 200 | 0.011 | 0.443 | 0.151 | 0.756 | 0.011 | 0.556 | 0.227 |
| 6 | 100 | 300 | 0.012 | 0.484 | 0.177 | 1.046 | 0.012 | 0.775 | 0.275 |
| 6 | 200 | 100 | 0.013 | 1.665 | 0.460 | 5.937 | 0.011 | 0.946 | 0.752 |
| 6 | 200 | 200 | 0.011 | 1.623 | 0.649 | 8.736 | 0.011 | 3.851 | 0.913 |
| 6 | 200 | 300 | 0.014 | 1.819 | 0.696 | 11.558 | 0.011 | 4.585 | 0.991 |
| 6 | 300 | 100 | 0.014 | 3.734 | 1.056 | 25.702 | 0.010 | 2.812 | 1.932 |
| 6 | 300 | 200 | 0.013 | 3.815 | 1.612 | 42.594 | 0.010 | 2.641 | 2.549 |
| 6 | 300 | 300 | 0.013 | 3.893 | 1.858 | 54.338 | 0.011 | 6.951 | 2.948 |

**Table A8.** CPU-times (in seconds) for lower and upper bounding procedures (Set IV).

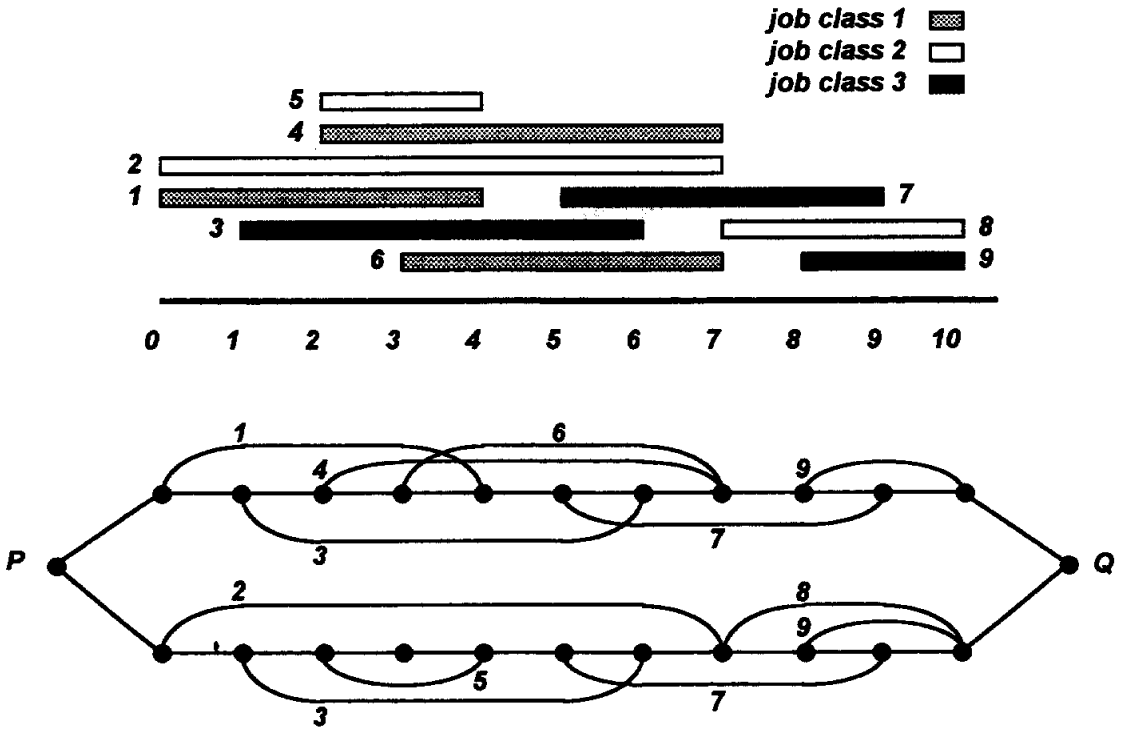
job class 1
job class 2
job class 3
5
4
2
1
7
3
8
6
9
0 1 2 3 4 5 6 7 8 9 10
1
6
4
9
3
7
P
Q
2
8
9
5
3
7

Figure 1

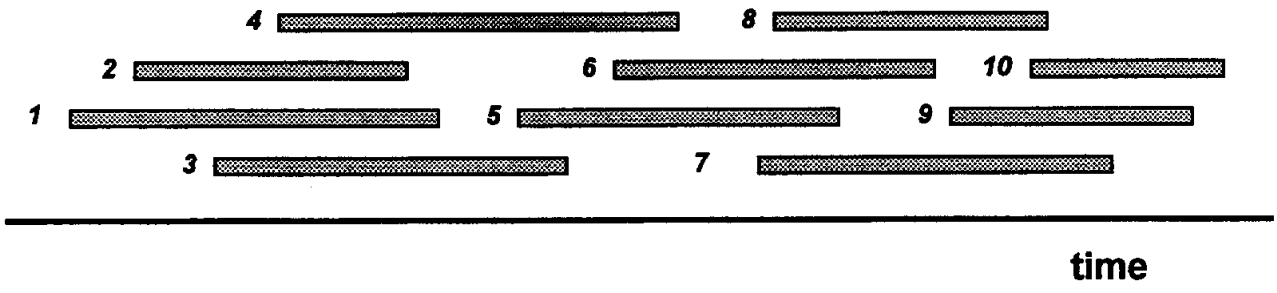


Figure 2