EXACT AND APPROXIMATION ALGORITHMS FOR THE TACTICAL FIXED INTERVAL SCHEDULING PROBLEM

LEO G. KROON
Erasmus University, Rotterdam, The Netherlands

MARC SALOMON
Tilburg University, Tilburg, The Netherlands

LUK N. VAN WASSENHOVE
INSEAD, Boulevard de Constance, France

(Received December 1993; revision received March 1995; accepted March 1996)

The Tactical Fixed Interval Scheduling Problem (TFISP) is the problem of determining the *minimum number* of parallel nonidentical machines, such that a feasible schedule exists for a given set of jobs. In TFISP, each job must belong to a specific job class and must be carried out in a prespecified time interval. The problem is complicated by the restrictions that (1) each machine can handle only one job at a time, (2) each machine can handle only jobs from a subset of the job classes, and (3) 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 concludes with a computational study.

The Tactical Fixed Interval Scheduling Problem (TFISP) is the problem of determining the minimum number of parallel nonidentical machines, such that a feasible nonpreemptive 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 that equals the length of the time interval between the job's start and finish time. Each machine is allowed to process jobs only from a prespecified subset of job classes and can process, at most, only one job at a time.

In what follows, we describe two practical situations in which variants of this problem occur.

In a strategic expansion study for Schiphol Amsterdam Airport, one of the objectives was to obtain insight into the future required gate capacity at the terminal for different scenarios relative to flight intensities.[1]

At the airport, passengers are transferred between the platform where the aircraft arrive and the terminal either by gate, or if no gate is available, by bus. Since most passengers prefer a transfer by gate over a transfer by bus one of the important service objectives of the airport is to handle as many passengers as possible by gate. Aside from this service aspect, a number of other side-constraints must be taken into account with respect to the gates. For example, if an aircraft is assigned to a gate upon arrival at the airport, the aircraft occupies the gate during its complete ground time (i.e., the time between arrival and departure). During its ground time, no other aircraft can use the gate. Another important side constraint is that for technical reasons, each gate is suitable to handle only a limited set of different aircraft types.

In this study, the management of the airport wanted to obtain insight into the influence of higher flight intensities on the required gate capacity. To do so, different scenarios were developed with respect to the timetables. For each flight intensity, a number of timetables were generated. TFISP was used to calculate the number of required gates for each scenario. In TFISP, each incoming aircraft was modeled as a job with a fixed arrival and departure time, and each gate was modeled as a machine that could handle one job at a time. The technical constraint that gates could handle only aircraft from a predetermined set of aircraft types was also taken into account. TFISP then determined the required number of gates, such that, in principle, all incoming aircraft could be assigned to a gate. Of course, in day-to-day planning, buses are still needed to handle unforeseen circumstances, such as delays of aircraft.

The above study was carried out by the authors in cooperation with ORTEC Consultants, Gouda, The Netherlands. It resulted in the implementation of a Decision Support System (DSS) to support the strategic planning department at the airport.

TFISP is also used as the core model in a DSS for tactical capacity planning of aircraft maintenance personnel for KLM Royal Dutch Airlines. The problem context ere is as follows. Aircraft that arrive at Schiphol Amsterdam Airport usually require a number of short maintenance inspections. If the carrier has a maintenance contract with KLM, then these inspections are carried out by engineers from KLM's maintenance department. The processing times, as well as the order in which the inspections have to be carried out, are specified by the maintenance norms. As a

Subject classifications: Programming: integer; production/scheduling: approximations, heuristics; analysis of algorithms: computational complexity.
Area of review: MANUFACTURING.

consequence, the timetables of the airline companies and the maintenance norms determine the fixed intervals in which the maintenance inspections have to be carried out in order to avoid aircraft delays. The capacity planning problem is further complicated by the rule that, for safety reasons, each engineer is licensed to carry out inspections on two different aircraft types only.

The major problem the management of the maintenance department is faced with is to determine the most efficient 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 under different scenarios with respect to flight intensities and composition of the fleet. A detailed description of this problem, and the DSS that has been developed to analyse it, is found in Kroon (1990), Dijkstra et al. (1991), and Dijkstra et al. (1994).

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 1. In Section 2 we discuss the literature on TFISP. The computational complexity of TFISP and its *preemptive* variant is analyzed in Section 3. Lower and upper bounding procedures are described in the Sections 4 and 5, respectively. In Section 6 the results of a computational study are presented. Finally, Section 7 summarizes the results and conclusions of this paper.

1. 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 nonidentical machines required to carry out a number of jobs over a fixed planning horizon. Each job has a fixed start and finish time. Preemption of a job is not allowed.

We introduce the following notation to formally define TFISP: suppose there are J jobs to be carried out over a planning horizon $[0, T]$. The start and finish time of job j are represented by s_j and f_j, and 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. The set $\mathcal{A}_c$ is defined as the set of job classes that can be carried out by machines in machine class c ($c = 1, \ldots, C$). 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:

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

If α 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_α, 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 required 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 total number of machines. Now, TFISP can be formulated as the following integer program:

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

subject to:

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

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

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

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

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 nonpreemptive 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. The nodeset 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. A particular job j with $j \in \mathcal{J}_c$ is represented in G_c by an arc from

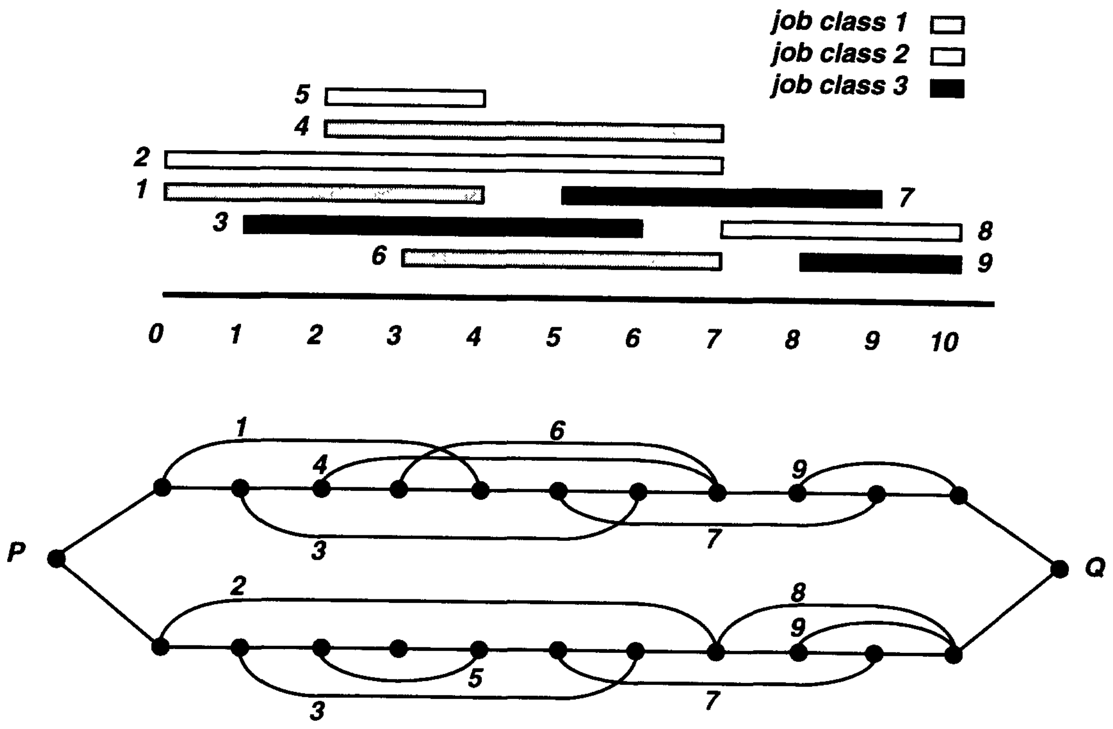

Figure 1. An instance of TFISP and the corresponding graph G. Here $A = 3$, $C = 2$, and a machine in machine class c can handle jobs in the job classes c and 3.

the node corresponding to s_j to the node corresponding to f_j. This arc has upper capacity one. 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, we refer to the graph compression procedure of Kroon et al. (1992).

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.

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.

2. 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 nonpreemptive schedule exists for all jobs. This problem was studied by Dantzig and Fulkerson (1954) and by Gertsbakh and Stern (1978) in the context of fleet planning. It was also studied by Hashimoto and 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 and 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. 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 $C = 2$, in which a machine in machine class c is allowed to carry out jobs in job

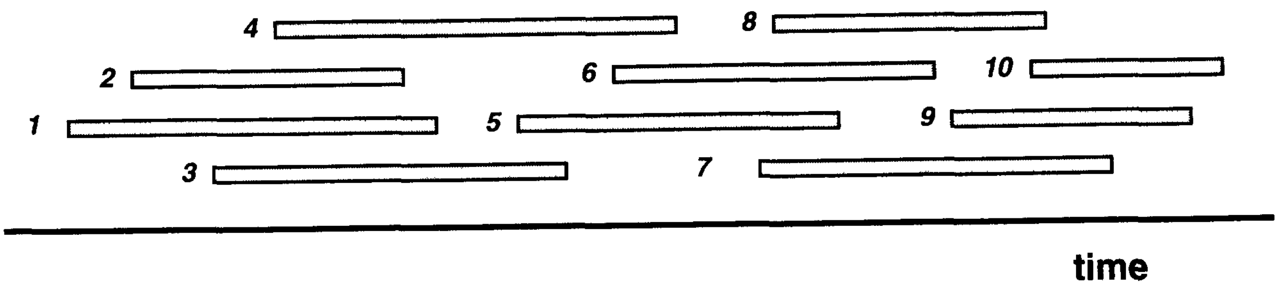

Figure 2. An instance of FJSP with $J = 10$ and $L = 4$.

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 and 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

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

STEP (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 $Y_1 + Y_2 = \lceil Z_{\text{LP}} \rceil$ to the original LP relaxation.

Kolen and Kroon (1992) prove that the nonpreemptive variant of TFISP is NP-hard in the strong sense if the number of machine classes C satisfies $C > 2$ (except for a few special cases). Dondeti and Emmons (1993) consider the preemptive variant of TFISP. They *conjecture* that this problem is NP-hard. In this paper we show that for a fixed number of machine classes, this problem can be solved in polynomial time. However, if the number of machine classes is *not* fixed, 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 consists of a given set of parallel nonidentical machines, 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 and 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 Lagrangian relaxation algorithm for a variant of OFISP occurring in the context of classroom scheduling. Carter and Tovey (1992) present an analysis of the computational complexity of several variants of this classroom scheduling problem. Finally, Kroon et al. (1992) present an approximation algorithm for solving OFISP. Their algorithm is based on Lagrangian relaxation and decomposition, and on the application of a straightforward dual-ascent heuristic.

3. 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 of spliting 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 was conjectured by Dondeti and Emmons (1993). We provide a complete overview of the complexity of the preemptive and nonpreemptive variant of TFISP. We start with the following auxiliary lemma concerning the preemptive variant of TFISP.

Lemma 1. *Each preemptive schedule R can be transformed into a preemptive schedule R', in which all preemptions occur only at time instants $t \in \mathcal{T}$. 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 (σ, 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 τ be defined as $\tau = \min(\{s_j | j = 1, \ldots, J; s_j > t\} \cup \{T\})$.

Now, the schedule during the interval (t, τ) is modified in the following way. If machine m was idle during the time interval (σ, t), then machine m is also idle during the interval (t, r). If machine m was carrying out job j_m during the time interval (σ, t), then let e_m be defined by $e_m = \min\{f_{jm}, \tau\}$. 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, τ). By this transformation we obtain a new preemptive schedule, where the preemption at time instant t no longer

Table I
Computational Complexity of TFISP.

	C Variable	C Fixed	
		$C \leq 2$	$C > 2$
Preemption Allowed	NP-Hard (Lemma 3)	Polynomially Solvable (Section 2)	Polynomially Solvable (Lemma 4)
Preemption *Not* Allowed	NP-Hard (Lemma 3)	Polynomially Solvable (Section 2)	NP-hard (Kolen and Kroon, 1992)

occurs. 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 1. 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 $\mathcal{W}$ that 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 equals $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{Z}$. 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 nonoverlapping. □

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

Proof. First, note that if the number of machine classes is fixed, then the maximum number of job classes is fixed 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 2.

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 set of constraints exists:

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

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

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 previously noted by Dondeti and 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 problems (6) and (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:

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

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 different 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 fixed, 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 2. If $C > 2$, then TFISP is NP-hard (Kolen and Kroon, 1992). Table I summarizes the complexity results obtained for TFISP.

4. 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 nonpreemptive way. This relaxation is called Nonpreemptive 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 $\Sigma_c Y_c$ subject to (6) and (7), and Y_c integer for all c) using a standard branch-and-bound algorithm.
- Relaxation of the integrality constraints (4) and (5). The remaining LP relaxation can be solved in polynomial time. The corresponding lower bound is denoted by Z_{LP}.
- Lagrangian relaxation of the constraints (3), ensuring that each job is carried out exactly once. The resulting Lagrangian Relaxation (LR) is written as:

LR:

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

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

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

where v_j is the (unconstrained) Lagrangian 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 $Z_{\text{LR}}(v)$ 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, $Z_{\text{LR}}(v)$ 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 $Z_{\text{LR}}(v)$ to Z_{IP} is now obtained as $\Sigma_{c=1}^{C} (Z_c^* + Y_c) + \Sigma_{j=1}^{J} v_j$. This lower bound can be obtained in polynomial time, since (*i*) the minimum cost-flow problems on the subgraphs $G_c(v)$ can be solved by a strongly polynomial algorithm (see Ahuja et al. 1993), (*ii*) the amount of flow Y_c is bounded by L_c, and (*iii*) $L_c \leq J$.

For *updating* the Lagrangian 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*) Lagrangian multipliers have been updated J times, or (*ii*) the difference between the Lagrangian lower bound and the "*greedy*" upper bound (as discussed in Section 5) has become less than 1. The latter implies that the optimal solution has been obtained. The best Lagrangian lower bound that we found upon execution of the procedure is denoted by Z_{LR}. Due to convergence problems, we often found $Z_{\text{LR}} < \max_v Z_{\text{LR}}(v)$ in our computational study (Section 6).

Remark. In the initial implementation of the LR procedure we have tried several *alternative* stopping criteria and updating mechanisms for the Lagrangian multipliers, including dual ascent. However, the effects of these alternative stopping criteria and updating mechanisms on the quality of the solutions and on the convergence speed were only marginal.

Lemma 5. *No other than the following dominance relations exist between the introduced lower bounds*:

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

$$Z_{\text{CR}} \leq \max_v Z_{\text{LB}}(v).$$

Proof.

- Problem CR is obtained by replacing $\mathscr{A}_c$ 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 that is totally unimodular. As a consequence, $Z_{\text{CR}} \leq Z_{\text{LP}}$.
- Obviously, the maximum job overlap does not change by allowing preemption of the jobs. Furthermore, Z_{NR} is greater than or equal to the maximum job overlap. This implies $Z_{\text{CR}} \leq Z_{\text{NR}}$.

- The relation $\max_v Z_{LR}(v) = Z_{LP}$ follows from the fact that in LR the side constraints (3), which disturb the network flow structure of TFISP, are dualized into the objective (1′). Hence, upon dualization of (3), LR decomposes into C subproblems which all have a network flow structure. As a consequence, each of these subproblems has the integrality property, which states that its LP relaxation has only integral extremal solutions. Thus, the relation $\max_v Z_{LR}(v) = Z_{LP}$ is a direct consequence of Geoffrion (1974).
- The relation $Z_{CR} \leq \max_v Z_{LR}(v)$ follows from the relations $\max_v Z_{LR}(v) = Z_{LP}$ and $Z_{CR} \leq Z_{LP}$.
- The absence of a dominance relation between Z_{LP} and Z_{NR} can be seen from the following instances involving three job classes and three machine classes. Let $\mathcal{A}_c = \{1, 2, 3\}\backslash\{c\}$ for $c = 1, 2, 3$; if the three jobs $(s_j, f_j, a_j) = (0, 1, 1)$, $(1, 3, 2)$, and $(3, 4, 3)$ must 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 $\max_v Z_{LR}(v)$ 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}. □

5. UPPER BOUNDS

Upper bounds to Z_{IP} are obtained by constructing a feasible solution to TFISP. 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 α be a set of job classes. We say that α 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, $\cup_{n=1}^{N} \alpha_n = \{1, \ldots, A\}$. A feasible subset α_n in this cover is said to be *nonredundant* if the collection $\{\alpha_1, \ldots, \alpha_N\}\backslash\{\alpha_n\}$ 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 $\Sigma_{n=1}^{N} L_{\alpha_n}$ is an upper bound to Z_{IP}. If all feasible subsets in the cover are nonredundant, 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_{α_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 $\Sigma_{n=1}^{N} L_{\alpha_n}$ machines. The fact that the bound is tight if all feasible subsets in $\{\alpha_1, \ldots, \alpha_N\}$ are nonredundant can be seen as follows. Obviously, each subset α_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 $a(n)$. We now consider an instance of TFISP with the following structure: the time horizon has been split up into N subintervals, and the nth subinterval contains only jobs in job class $a(n)$. For this instance of TFISP we have $Z_{IP} = \Sigma_{n=1}^{N} L_{\alpha_n}$. □

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:

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

subject to:

$$\sum_{\{s \mid a \in \alpha_s\}} x_s = 1 \quad a = 1, \ldots, A,$$

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

If the numbers A and C are fixed, then Z_{CC} can be obtained by complete enumeration in an amount of time that is independent of the number of jobs. If A and C are *not* fixed, then calculating Z_{CC} involves the solution of an uncapacitated facility location problem, which can be accomplished by the DUALOC procedure of Erlenkotter (1978).

A special covering of all job classes is obtained by assuming that all feasible subsets are singletons. In that case, one obtains the upper bound $\Sigma_{a=1}^{A} L_a$. 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. The heuristic uses the Lagrangian multipliers v obtained from the Lagrangian relaxation procedure. More formally, a *single pass* of the heuristic is described as follows:

Greedy Heuristic:

```
$\mathcal{J}$ := {all jobs};
$Y_v$ := 0 for $c = 1, \ldots, C$;
Repeat.
  Search for the locally best machine class $c^*$;
  $Y_{c^*} := Y_{c^*} + 1$.
  $\mathcal{J} := \mathcal{J}\backslash${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 to carry 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 (Lagrangian) 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 $\bar{G}_c(v)$. Here the graph $\bar{G}_c(v)$ is obtained from the graph $G_c(v)$ (as defined in the Lagrangian relaxation procedure) by deleting all job arcs corresponding to jobs $j \notin \mathcal{J}$. The resulting solution for the graph $\bar{G}_c(v)$ is called $\bar{Z}_c(v)$. Now, the locally best machine class is defined by $c^* := \text{argmin}\ \{\bar{Z}_c(v) | c = 1, \ldots, C\}$.

The single-pass upper bound $Z_{\text{GR}}(v)$ equals $\Sigma_c Y_c$. Note that the computational effort required to compute $Z_{\text{GR}}(v)$ is $\mathcal{O}(J\ \Sigma_c |J_c|^2)$. The single-pass upper bound $Z_{\text{GR}}(v)$ is computed for different values of the Lagrangian multipliers v after every five iterations of the Lagrangian lower bounding procedure. The resulting multipass, greedy upper bound Z_{GR} equals the minimum over all single-pass upper bounds; i.e., $Z_{\text{GR}} = \min_v Z_{\text{GR}}(v)$, where the minimum is taken over all Lagrangian multipliers that are considered upon execution of the algorithm.

6. COMPUTATIONAL EXPERIMENTS

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.

6.1. Implementation of the Procedures

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

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

6.2. Design of Test Problems

We have generated four sets of problem instances. For all instances, the planning horizon T has been set to 1,000. 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, a normal distribution function with mean μ_a, and standard deviation σ_a. Here, μ_a is the expected peak time of jobs in job class a, and σ_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 $s_j = m_j - D/2$ and the finish time $f_j = m_j + D/2$. The specific parameter settings for p_a, μ_a, and σ_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 $C = \binom{A}{2}$. In this case, we have restricted ourselves to $A = 4$ and $A = 5$. Note that $A = 6$ implies $C = 15$. This large number of machine classes could not be handled by our software.
- *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 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.

Table II
Summary of Set Characteristics

	Machine Classes	
Job Overlap	All	Limited
Uniform	Set I (180 Instances)	Set III (270 Instances)
Peaks	Set II (180 Instances)	Set IV (270 Instances)

Within each set we have generated 10 problem instances for each considered (A, J, D) combination. In Table II we summarize the characteristics of each set.

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

Symbol	Definition
#	Number of instances per set.
$\overline{\Delta}_{(\mathrm{LB})}$	Average quality of Lower Bound (LB). It is computed by taking the average of $(Z_{\mathrm{IP}} - Z_{(\mathrm{LB})})/Z_{\mathrm{IP}}$ 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.
$\overline{E}_{\mathrm{LP}}$	The number of times that Z_{LP} equals Z_{IP} over all instances in a set.
$\overline{I}_{\mathrm{LP}}$	The number of times that the LP relaxation yields an integer solution over all instances in a set.
$\overline{\Delta}_{(\mathrm{UB})}$	The average quality of the Upper Bound (UB). It is computed by taking the average of $(Z_{\mathrm{UB}} - Z_{\mathrm{IP}}/Z_{\mathrm{IP}})$ 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 III(b). The figures shown 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.

From Tables III(a) and III(b), 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 Sets I and II. Furthermore, note that solving the linear program can be done relatively fast (on average within nine 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, Lagrangian relaxation is not very successful for solving TFISP. 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 III(a) that, with respect to the quality of the solutions, the LR-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

Table III
(a) Summary of Computational Results with Respect to Quality

		Lower Bounds						Upper Bounds	
	#	$\overline{\Delta}_{\mathrm{CR}}$	$\overline{\Delta}_{\mathrm{NR}}$	$\overline{\Delta}_{\mathrm{LP}}$	$\overline{E}_{\mathrm{LP}}$	$\overline{I}_{\mathrm{LP}}$	$\overline{\Delta}_{\mathrm{LR}}$	$\overline{\Delta}_{\mathrm{CC}}$	$\overline{\Delta}_{\mathrm{GR}}$
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 III
(b) Summary of Computational Results with Respect to CPU Times

		Lower Bounds				Upper Bounds		Optimal
	#	$\overline{\text{CPU}_{\text{CR}}}$	$\overline{\text{CPU}_{\text{NR}}}$	$\overline{\text{CPU}_{\text{LP}}}$	$\overline{\text{CPU}_{\text{LR}}}$	$\overline{\text{CPU}_{\text{CC}}}$	$\overline{\text{CPU}_{\text{GR}}}$	$\overline{\text{CPU}_{\text{IP}}}$
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

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 depends heavily 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 become dependent of the Lagrangian multipliers in the Lagrangian objective (1″).

- It is somewhat counterintuitive that 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 the GR-bound is larger than the CPU time required to compute the CC-bound.
- From a comparison of the results for the instances in Sets I and II, and Sets III and IV, it is concluded rather unexpectedly 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-and-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-and-bound technique (CPU times may go up to 10 minutes on average). The latter is *also* true when the LP solution is fractional and $Z_{\text{IP}} = Z_{\text{LP}}$.

7. 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 fixed; 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. 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 characteristics that closely resemble those of the problems that we met in the practical settings previously reported) could be solved relatively fast by some of the procedures described in this paper. Linear programming, in combination with a branch-and-bound procedure, turned out to be effective for medium-sized problems.

APPENDIX I. DEFINITION OF 3DM

Instance of 3DM:

- Three 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	μ_a	σ_a	p_a	μ_a	σ_a	p_a	μ_a	σ_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

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.
P	The maximum job duration.
#	Number of instances.

Table A1
Quality of Lower and Upper Bounding Procedures (Set I)

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

Table A2
CPU Times (in seconds) for 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	10	0.010	0.521	0.293	1.553	0.013	0.296	0.575
4	100	200	10	0.010	0.567	0.331	1.650	0.012	0.744	1.241
4	100	300	10	0.011	0.585	0.314	1.990	0.011	0.424	0.816
4	200	100	10	0.013	2.158	0.884	8.689	0.013	0.671	4.216
4	200	200	10	0.011	2.088	1.078	11.244	0.012	1.186	12.693
4	200	300	10	0.013	1.968	1.398	14.351	0.012	2.382	16.171
4	300	100	10	0.011	4.161	1.969	28.259	0.011	2.276	20.722
4	300	200	10	0.012	4.246	2.705	38.566	0.012	4.156	59.050
4	300	300	10	0.012	4.262	3.522	50.284	0.012	5.719	52.158
5	100	100	10	0.012	1.012	0.513	2.046	0.012	0.683	1.499
5	100	200	10	0.010	1.218	0.700	2.283	0.012	1.129	5.579
5	100	300	10	0.011	1.436	0.629	2.534	0.013	1.011	5.103
5	200	100	10	0.012	5.727	2.126	10.358	0.013	2.957	41.019
5	200	200	10	0.012	5.108	2.240	13.019	0.012	2.597	74.585
5	200	300	10	0.013	4.803	2.420	15.560	0.013	2.793	79.543
5	300	100	10	0.011	10.945	4.325	30.405	0.011	5.921	200.648
5	300	200	10	0.013	11.540	4.363	42.398	0.014	3.687	300.193
5	300	300	10	0.014	11.856	5.567	50.576	0.014	9.382	566.045

Table A3
Quality of Lower and Upper Bounding Procedures (Set III)

				Lower Bounds						Upper Bounds	
A	J	D	#	Δ_{CR}	Δ_{NR}	Δ_{LP}	E_{LP}	I_{LP}	Δ_{LR}	Δ_{CC}	Δ_{GR}
4	100	100	10	0.077	0.000	0.023	4	3	0.051	0.198	0.008
4	100	200	10	0.038	0.000	0.005	8	4	0.028	0.236	0.010
4	100	300	10	0.023	0.000	0.004	8	5	0.018	0.238	0.008
4	200	100	10	0.056	0.000	0.009	5	3	0.051	0.220	0.004
4	200	200	10	0.039	0.000	0.003	8	2	0.034	0.263	0.008
4	200	300	10	0.004	0.000	0.002	9	4	0.020	0.272	0.008
4	300	100	10	0.053	0.000	0.006	7	3	0.052	0.300	0.022
4	300	200	10	0.031	0.000	0.001	9	4	0.026	0.256	0.009
4	300	300	10	0.015	0.000	0.001	8	2	0.019	0.250	0.008
5	100	100	10	0.133	0.006	0.014	6	2	0.041	0.368	0.000
5	100	200	10	0.055	0.000	0.010	5	2	0.046	0.379	0.000
5	100	300	10	0.046	0.011	0.012	5	2	0.033	0.418	0.000
5	200	100	10	0.149	0.000	0.017	2	1	0.063	0.388	0.000
5	200	200	10	0.061	0.000	0.005	6	3	0.035	0.409	0.005
5	200	300	10	0.011	0.000	0.001	9	2	0.031	0.310	0.009
5	300	100	10	0.136	0.000	0.008	4	2	0.063	0.371	0.003
5	300	200	10	0.074	0.002	0.005	6	0	0.042	0.303	0.002
5	300	300	10	0.023	0.003	0.004	6	1	0.025	0.201	0.003

$\Delta_{(LB)}$ Average quality of lower bound (LB). It is computed by taking the average of $(Z_{IP} - Z_{(LB)})/Z_{IP}$ 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 $(Z_{(UB)} - Z_{IP})/Z_{IP}$ 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.

Table A4
CPU Times (in second) for 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	10	0.012	0.583	0.341	1.455	0.013	0.357	1.010
4	100	200	10	0.014	0.613	0.375	1.687	0.012	0.841	1.063
4	100	300	10	0.011	0.671	0.401	1.988	0.013	1.223	1.216
4	200	100	10	0.010	2.115	1.147	9.172	0.013	1.330	5.212
4	200	200	10	0.013	2.087	1.422	12.092	0.010	3.050	10.245
4	200	300	10	0.012	2.413	1.703	14.197	0.010	3.655	23.417
4	300	100	10	0.012	4.803	2.730	29.323	0.011	5.816	14.512
4	300	200	10	0.013	5.037	3.735	42.721	0.013	8.890	31.273
4	300	300	10	0.014	5.470	4.331	52.829	0.015	14.803	69.193
5	100	100	10	0.012	1.282	0.625	1.871	0.013	0.719	1.843
5	100	200	10	0.011	1.607	0.702	2.183	0.011	0.993	5.068
5	100	300	10	0.011	1.528	0.743	2.322	0.014	1.527	4.570
5	200	100	10	0.012	5.899	2.301	10.431	0.012	2.371	25.063
5	200	200	10	0.012	5.690	2.511	12.552	0.016	2.197	49.218
5	200	300	10	0.014	5.133	2.914	14.227	0.013	6.406	58.288
5	300	100	10	0.011	12.601	5.225	32.379	0.014	4.817	106.940
5	300	200	10	0.012	12.396	7.301	40.896	0.011	12.136	235.694
5	300	300	10	0.011	12.661	8.618	49.011	0.010	20.494	405.905

Table A5
Quality of the Lower and Upper Bounding Procedures (Set III)

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

Table A6
CPU Times (in seconds) for 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	10	0.012	0.259	0.111	0.347	0.012	0.128	0.145
4	100	200	10	0.013	0.275	0.137	0.619	0.010	0.326	0.196
4	100	300	10	0.012	0.273	0.144	0.845	0.012	0.290	0.189
4	200	100	10	0.012	0.906	0.317	3.290	0.010	0.336	0.507
4	200	200	10	0.011	0.964	0.416	8.762	0.010	0.686	0.707
4	200	300	10	0.013	0.892	0.562	9.428	0.010	1.065	0.835
4	300	100	10	0.011	2.070	0.722	21.964	0.010	1.133	1.237
4	300	200	10	0.011	2.000	1.220	27.650	0.012	2.332	1.723
4	300	300	10	0.013	1.812	1.524	45.644	0.012	3.185	2.206
5	100	100	10	0.013	0.327	0.123	0.329	0.010	0.158	0.174
5	100	200	10	0.012	0.407	0.137	0.747	0.011	0.302	0.212
5	100	300	10	0.014	0.424	0.150	1.365	0.011	0.392	0.220
5	200	100	10	0.012	1.456	0.344	4.352	0.010	0.634	0.656
5	200	200	10	0.012	1.612	0.480	7.649	0.011	0.964	0.790
5	200	300	10	0.011	1.402	0.628	9.913	0.011	2.398	0.937
5	300	100	10	0.012	3.093	0.757	20.601	0.011	1.191	1.433
5	300	200	10	0.013	3.342	1.210	34.666	0.012	2.569	2.455
5	300	300	10	0.014	3.102	1.493	50.976	0.010	7.961	2.438
6	100	100	10	0.015	0.471	0.131	0.633	0.012	0.291	0.201
6	100	200	10	0.011	0.566	0.148	0.750	0.010	0.400	0.219
6	100	300	10	0.011	0.655	0.168	1.463	0.010	0.433	0.260
6	200	100	10	0.011	2.079	0.402	5.206	0.010	0.759	0.697
6	200	200	10	0.010	2.227	0.558	7.369	0.011	1.961	1.111
6	200	300	10	0.013	2.376	0.630	11.234	0.012	2.421	1.202
6	300	100	10	0.012	5.246	0.859	23.762	0.012	1.881	1.843
6	300	200	10	0.015	4.995	1.222	37.239	0.011	4.764	3.023
6	300	300	10	0.014	4.798	1.577	48.853	0.012	12.060	3.560

Table A7
Quality of Lower and Upper Bounding Procedures (Set IV)

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

Table A8
CPU Times (in seconds) for 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	10	0.014	0.201	0.117	0.201	0.012	0.130	0.163
4	100	200	10	0.012	0.222	0.141	0.829	0.011	0.208	0.205
4	100	300	10	0.013	0.234	0.162	0.904	0.011	0.339	0.225
4	200	100	10	0.012	0.728	0.440	4.802	0.010	0.408	0.609
4	200	200	10	0.012	0.750	0.582	7.638	0.011	0.619	0.773
4	200	300	10	0.012	0.758	0.631	9.064	0.011	1.025	0.804
4	300	100	10	0.010	1.502	0.881	26.323	0.010	0.630	1.429
4	300	200	10	0.014	1.634	1.394	33.657	0.011	1.627	1.958
4	300	300	10	0.016	1.540	1.735	46.209	0.010	1.898	2.021
5	100	100	10	0.010	0.263	0.128	0.485	0.010	0.226	0.188
5	100	200	10	0.012	0.320	0.153	0.967	0.012	0.467	0.234
5	100	300	10	0.013	0.282	0.178	0.833	0.011	0.531	0.243
5	200	100	10	0.012	1.039	0.462	7.138	0.010	0.976	0.691
5	200	200	10	0.013	1.055	0.560	6.897	0.010	1.492	0.798
5	200	300	10	0.012	1.058	0.688	11.505	0.010	2.956	1.002
5	300	100	10	0.010	2.262	1.086	26.783	0.010	3.216	1.728
5	300	200	10	0.011	2.344	1.508	39.572	0.011	2.495	2.382
5	300	300	10	0.014	2.272	1.741	50.359	0.013	3.766	2.586
6	100	100	10	0.013	0.392	0.146	0.950	0.010	0.305	0.216
6	100	200	10	0.011	0.443	0.151	0.756	0.011	0.556	0.227
6	100	300	10	0.012	0.484	0.177	1.046	0.012	0.775	0.275
6	200	100	10	0.013	1.665	0.460	5.937	0.011	0.946	0.752
6	200	200	10	0.011	1.623	0.649	8.736	0.011	3.851	0.913
6	200	300	10	0.014	1.819	0.696	11.558	0.011	4.585	0.991
6	300	100	10	0.014	3.734	1.056	25.702	0.010	2.812	1.932
6	300	200	10	0.013	3.815	1.612	42.594	0.010	2.641	2.549
6	300	300	10	0.013	3.893	1.858	54.338	0.011	6.951	2.948

ENDNOTES

1. Opposite to the situation that is common in the United States, the *airport* is owner of the gates, and is responsible for the assignment of gate capacity to the incoming aircraft.

2. 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.

ACKNOWLEDGMENTS

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

REFERENCES

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin. 1993. Network Flows: Theory, Algorithms, and Applications. Prentice Hall, Englewood Cliffs, NJ.

Arkin, E. M. and E. L. Silverberg. 1987. Scheduling Jobs With Fixed Start and Finish Times. *Discrete Appl. Math.* **18,** 1-8.

Carter, M. W. 1989. A Lagrangian Relaxation Approach to the Classroom Assignment Problem. *INFOR* **27,** 230-246.

Carter, M. W. and C. A. Tovey. 1992. When is the Classroom Assignment Problem Hard? *Opns. Res.* **40,** S28-S39.

Dantzig, G. L. and D. R. Fulkerson. 1954. Minimizing the Number of Tankers to Meet a Fixed Schedule. *Naval Res. Logist.* **1,** 217-222.

Dijkstra, M. C., L. G. Kroon, J. A. E. E. van Nunen, and M. Salomon. 1991. A DSS for Capacity Planning of Aircraft Maintenance Personnel. *International J. of Production Res.* **23,** 69-78.

Dijkstra, M. C., L. G. Kroon, M. Salomon, J. A. E. E. van Nunen, and L. N. Van Wassenhove. 1994. Planning the Size and Organization of KLM's Maintenance Personnel. *Interfaces* **24,** 47-58.

Dilworth, R. P. 1950. A Decomposition Principle for Partially Ordered Sets. *Ann. Math.* **51,** 161-166.

Dondeti, V. R. and H. Emmons. 1992. Interval Scheduling With Processors of Two Types. *Opns. Res.* **40,** S76-S85.

Dondeti, V. R. and H. Emmons. 1993. Algorithms for Preemptive Scheduling of Different Classes of Processors to Do Jobs With Fixed Times. *Eur. J. Opnl. Res.* **70,** 316-326.

Erlenkotter, D. 1978. A Dual-Based Procedure for Uncapacitated Facility Location. *Opns. Res.* **26,** 992-1000.

Fischetti, M., S. Martello, and P. Toth. 1987. The Fixed Job Schedule Problem With Spread Time Constraints. *Opns. Res.* **6,** 849-858.

Fischetti, M., S. Martello, and P. Toth. 1989. The Fixed Job Schedule Problem With Working Time Constraints. *Opns. Res.* **3,** 395-403.

Fischetti, M., S. Martello and P. Toth. 1992. Approximation Algorithms for Fixed Job Schedule Problems. *Opns. Res.* **40,** S96-S108.

Fisher, M. L. 1981. The Lagrangian Relaxation Method for Solving Integer Programming Problems. *Mgmt. Sci.* **27,** 1-18.

Geoffrion, A. M. 1974. Lagrangian Relaxation and Its Uses in Integer Programming. *Math. Prog. Study*, **2,** 82-114.

Gertsbakh, I. and H. I. Stern. 1978. Minimal Resources for Fixed and Variable Job Schedules. *Opns. Res.* **18,** 68-85.

Gupta, U. L., D. T. Lee, and J. Y.-T Leung. 1979. An Optimal Solution to the Channel Assignment Problem. *IEEE Trans. Comput.* **C-28,** 807-810.

Hashimoto, A. and J. Stevens. 1971. Wire Routing by Optimizing Channel Assignments Within Large Apertures. In *Proceedings of the 8th Design Automation Workshop*, 155-169.

IBM. 1991. Optimization Subroutine Library. Release 2: Guide and References. Third Edition.

Kolen, A. W. J. and L. G. Kroon. 1991. On the Computational Complexity of (Maximum) Class Scheduling. *Eur. J. Opnl. Res.* **54,** 23-38.

Kolen, A. W. J. and L. G. Kroon. 1992. License Class Design: Complexity and Algorithms. *Eur. J. Opnl. Res.* **63,** 432-444.

Kroon, L. G. 1990. Job Scheduling and Capacity Planning in Aircraft Maintenance. Ph.D. Thesis, Rotterdam School of Management, Erasmus University.