European Journal of Operational Research 82 (1995) 190–205

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Theory and Methodology

Exact and approximation algorithms for the operational fixed interval scheduling problem

Leo G. Kroon [a], Marc Salomon [a,*] and Luk N. Van Wassenhove [b]

[a] *Erasmus University (R.S.M.), P.O. Box 1738, NL-3000 DR Rotterdam, The Netherlands*
[b] *INSEAD, Boulevard de Constance, F-77305 Fontainebleau, France*

Received April 1993; revised August 1993

Abstract

The Operational Fixed Interval Scheduling Problem (OFISP) is characterized as the problem of scheduling a number of jobs, each with a fixed starting time, a fixed finishing time, a priority index, and a job class. The objective is to find an assignment of jobs to machines with maximal total priority. The problem is complicated by the restrictions that: (i) each machine can handle only one job at a time, (ii) each machine can handle only jobs from a prespecified subset of all possible job classes, and (iii) preemption is not allowed. It follows from the above that OFISP has both the character of a job scheduling problem and the character of an assignment problem. In this paper we discuss the occurrence of the problem in practice, and we present newly developed exact and approximation algorithms for solving OFISP. Finally, some computational results are shown.

Keywords: Job scheduling; Integer programming; Lagrangean relaxation; Heuristics

1. Introduction

The authors were first confronted with the Operational Fixed Interval Scheduling Problem (OFISP) during the development of a decision support system for the maintenance department of the major dutch airline company KLM at Schiphol Airport (Dijkstra et al., 1990). Planes arriving at the airport may require a number of maintenance jobs. The processing times as well as the order in which these jobs have to be carried out are specified by strict maintenance norms. As a consequence, the maintenance norms and the time-table determine the fixed intervals in which the jobs have to be carried out in order not to delay the departure of the airplanes on their next flights. The problem is further complicated by the safety rule that each of the available engineers is licensed to carry out jobs on at most two different aircraft types.

One of the problems to be solved by the decision support system is to develop maintenance schedules, such that in principle all jobs are carried out. However, jobs with low priority that cannot be carried out within their required interval might be postponed until the next stop of the airplane at an airport.

* Corresponding author.

0377-2217/95/$09.50 © 1995 Elsevier Science B.V. All rights reserved
SSDI 0377-2217(93)E0335-U

Later on, the authors became aware of, or involved in several other projects in which OFISP plays an important role. These projects are briefly discussed below:

- *The assignment of airplanes to gates* (Hagdorn-van der Meijden and Kroon, 1990, and de Wit, 1991). This problem also occurs at Schiphol Airport, where airplanes of different types have to be assigned to gates during fixed intervals. However, each gate can only handle a limited set of aircraft types, due to technical restrictions. The problem here is to find an assignment of airplanes to gates where the number of unassigned airplanes – whose passengers have to be transported to the terminal by bus – is minimized.
- *The scheduling of operating rooms in a hospital.* In most hospitals a limited number of operating rooms is available. Some of these operating rooms may be general purpose, but others may be suitable for only a subset of the various types of operations. In general the time slot for an operation is fixed some time ahead. Now the problem to be solved in this context is to find a feasible schedule for as many as possible of the planned operations, taking into account the restricted suitability of the operating rooms.
- *The assignment of holiday bungalows to vacationists* (Kolen et al., 1987). Usually holiday bungalows are booked a long time in advance for a period of one or more weeks. The holiday bungalows may differ in several aspects, like size, location, accommodation, quality, and price. Each season the booking office is faced with the problem of finding an assignment of holiday bungalows to vacationists, such that there is a matching between the desires of the vacationists with respect to e.g. comfort, and the available accommodation.

Also the classroom assignment problem (CAP) considered by Carter (1989) is closely related to OFISP. These examples illustrate that OFISP is an interesting problem from a practical point of view. However, the number of algorithms that is available for solving OFISP is quite limited. The research reported here is conducted in an attempt to fill this gap. This paper is organized as follows. In Section 2 we consider the special case of OFISP, where all machines are identical and where all jobs belong to the same job class. Section 3 proceeds with the general case, in which several machine classes and job classes exist. We formulate OFISP as an integer linear program, and present an approximation algorithm based on Lagrangean relaxation and decomposition. In Section 4 we compare the computational results of our heuristic with the results obtained from the integer linear program and with the results obtained from the linear programming relaxation thereof. We make some final remarks in Section 5. In this paper we follow the literature on job scheduling. Therefore we address the maintenance engineers (gates, operating rooms, holiday bungalows, classroom) as 'machines', and the inspections (airplanes, operations, holiday periods, classes) as 'jobs'.

2. Identical machines and one job class

OFISP is a generalization of the Fixed Job Scheduling Problem (FSP) and the Maximum Fixed Job Scheduling Problem (Max.FSP). In these problems all jobs have a fixed starting time and a fixed finishing time and belong to the same job class. Furthermore, the machines are identical. We will discuss the problems FSP and Max.FSP first. Thereafter we will consider the case with several machine classes and several job classes.

Suppose there are J jobs to be carried out in the time-interval $[0, T]$, where each job j is represented by the triple (s_j, f_j, p_j). Here s_j and f_j are the fixed starting and finishing time of job j, respectively, and p_j represents the priority of job j. For carrying out these jobs M identical machines are available. FSP is the feasibility problem of determining whether there exists a feasible non-preemptive schedule for all jobs. A necessary and sufficient condition for the existence of a feasible schedule for all jobs is given by the following lemma.

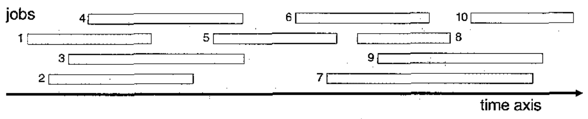

Fig. 1. Instance of FSP with $J = 10$ and $L = 4$.

Lemma 1. *A feasible non-preemptive schedule for all jobs exists if and only if the maximum job overlap is less than or equal to the number of available machines.*

Here the job overlap at time t, denoted by L_t, and the maximum job overlap, denoted by L, are defined as follows.

$$L_t = |\{j \mid s_j \le t < f_j\}|, \qquad L = \max\{L_t \mid 0 \le t \le T\}.$$

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 O(J log(J)) algorithm for determining the maximum job overlap of the jobs is described by Hashimoto and Stevens (1971) and Gupta, Lee and Leung (1979). Note that a feasible *preemptive* schedule also exists if and only if the maximum job overlap is less than or equal to the number of available machines. So in this case nothing can be gained by allowing preemption.

Fig. 1 gives an example of an instance of FSP. In this figure the bars indicate the jobs to be carried out. In this example the maximum job overlap L equals 4. Hence the minimum number of machines required for carrying out all jobs equals 4 as well.

If the maximum job overlap exceeds the number of available machines, then Max.FSP becomes interesting. Max.FSP is the problem of finding a subset of jobs with maximum total value that can be processed by the available machines. Max.FSP is considered by Arkin and Silverberg (1987), Kolen et al. (1987), and Kolen and Kroon (1991). They show that Max.FSP can be solved in polynomial time by a minimum cost flow algorithm. Arkin and Silverberg and Kolen et al. first construct a clique-graph. Thereafter, this graph is used as the underlying graph of a minimum cost flow problem with M units of flow.

The construction of the underlying directed graph G that we use in this paper is more direct than those constructions, and can be described as follows. The set $\{t_r \mid r = 0, 1, \dots, R\}$ is used to represent all starting and finishing times of the jobs in chronological order. That is, $\{t_r \mid r = 0, 1, \dots, R\} = \{s_j, f_j \mid j = 1, \dots, J\}$, and $t_{r-1} < t_r$ for $r = 1, \dots, R$. The set of nodes of the graph G is in one-to-one correspondence with the set $\{t_r \mid r = 0, 1, \dots, R\}$. A particular job j is represented in G by an arc from the node corresponding to s_j to the node corresponding to f_j. This arc has an upper capacity of one on the amount of flow that can be transported, and associated costs of $-p_j$ per unit of flow transported. Furthermore, for $r = 1, \dots, R$, there is an arc from t_{r-1} to t_r with zero costs and unlimited capacity. Obviously, a feasible schedule for a subset of jobs of maximum total value corresponds to a minimum cost flow of M units of flow from t_0 to t_R in the graph G. Fig. 2 shows the graph G corresponding to the set of jobs represented in Fig. 1. A job is carried out if and only if in the solution to the minimum cost flow problem one unit of flow passes through the corresponding arc. The minimum cost flow problem on

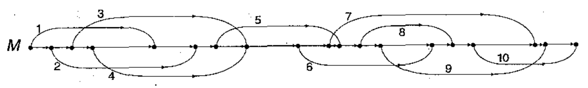

Fig. 2. The graph G corresponding to the set of jobs represented in Fig. 1.

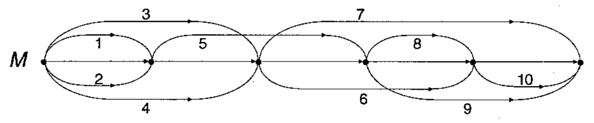

Fig. 3. The reduced graph with $R = 5$ corresponding to the graph of Fig. 2.

the graph G can be solved e.g. by the strongly polynomial time algorithm of Orlin (1988). In order to speed up the algorithm and to save storage space the following graph compression procedure can be applied to G:

Graph compression procedure:

Step 1. Search for a pair of nodes (t_{r-1}, t_r) in G such that node t_{r-1} does not have any *outgoing* job arcs **or** node t_r does not have any *incoming* job arcs. *If* such a pair (t_{r-1}, t_r) does not exist, *then* STOP *else Goto* Step 2.

Step 2. Replace the pair (t_{r-1}, t_r) by one single node and update the incoming and the outgoing arcs accordingly. *Repeat* Step 1.

Lemma 2. *There is a one-to-one correspondence between the feasible flows of M units of flow in G, and the feasible flows of M units of flow in the graph obtained after applying the graph compression procedure to G.*

Proof. Two consecutive nodes (t_{r-1}, t_r) are replaced by one single node in the following cases: *Case 1.* Node t_{r-1} does not have any *outgoing* job arcs. If a job finishing at t_{r-1} is carried out by one of the machines, then this machine will be idle during the interval (t_{r-1}, t_r). Therefore it may be assumed as well that the finishing time of such a job equals t_r. Thus node t_{r-1} is superfluous. *Case 2.* Node t_r does not have any *incoming* job arcs. If a job starting at t_r is carried out by one of the machines, then this machine must have been idle during the interval (t_{r-1}, t_r). Therefore it may be assumed as well that the starting time of such a job equals t_{r-1}. Thus node t_r is superfluous. □

Application of the Graph Compression Procedure to the graph of Fig. 2 yields the reduced graph of Fig. 3.

3. Several machine and job classes

Here we assume that there are C different machine classes, and A different job classes, where each machine class is allowed to handle jobs from a limited number of job classes. Each job j belongs to a certain job class a_j. For $c = 1,\dots,C$, the integer M_c represents the predetermined number of machines in machine class c. Furthermore, $\mathscr{A}_c$ is the set of job classes that can be carried out by machines in machine class c. For $j = 1,\dots,J$, the set $\mathscr{C}_j$ consists of all machine classes that can be used for carrying out job j. Mathematically, OFISP can be formulated as:

OFISP:

$$Z_{\text{OFISP}} = \max \sum_{j=1}^{J} \sum_{c \in \mathscr{C}_j} p_j x_{j,c} \tag{1}$$

subject to

$$\sum_{\{j \mid a_j \in \mathscr{A}_c \wedge s_j \leq t_r < f_j\}} x_{j,c} \leq M_c, \quad c = 1,\ldots,C;\ r = 0,\ldots,R, \tag{2}$$

$$\sum_{c \in \mathscr{C}_j} x_{j,c} \leq 1, \qquad j = 1,\ldots,J, \tag{3}$$

$$x_{j,c} \in \{0, 1\}, \qquad j = 1,\ldots,J;\ c \in \mathscr{C}_j, \tag{4}$$

where $x_{j,c}$ is a binary decision variable, indicating whether job j is assigned to a machine in machine class c ($j = 1,\ldots,J$, and $c \in \mathscr{C}_j$). The objective function (1) states that we look for a feasible schedule for a subset of jobs with maximum total value.

Lemma 3. *Each solution satisfying constraints* (2)–(4) *can be interpreted as a feasible non-preemptive schedule.*

Proof. Constraints (3) and (4) guarantee that each job is assigned to at most one machine class. Furthermore, constraints (2) ensure that at any point in time the total number of jobs assigned to machine class c does not exceed the number of machines available in machine class c. Thus Lemma 1 assures that there exists a feasible non-preemptive schedule for all jobs that are assigned to machine class c. The latter holds for each machine class c. □

Remark 1. The restrictions corresponding to values of r for which t_r is not a starting time of any job j with $a_j \in \mathscr{A}_c$ are in fact redundant, and can be eliminated from the model formulation.

Remark 2. Another problem closely related to OFISP is the Tactical Fixed Interval Scheduling Problem (TFISP), where the objective is to carry out all jobs against minimum total machine costs. The complexity of several variants of this problem is studied extensively by Kolen and Kroon (1992). The case of TFISP with identical machines and one job class is considered by Hashimoto and Stevens (1971), Gertsbakh and Stern (1978), and Gupta, Lee and Leung (1979). This problem is equivalent to FSP and can be solved in $O(J \log(J))$ time. Dondeti and Emmons (1992) study a generalization of this problem, with 3 job classes and 2 machine classes. The machines in machine class c ($c = 1, 2$) are allowed to carry out jobs in the job classes c and 3. It is shown that this variant of TFISP can be solved in polynomial time by repeatedly solving a Max Flow problem. Another algorithm for solving this variant of TFISP, based on Linear Programming and a Max Flow algorithm, is presented by Kolen and Kroon (1992). Fischetti, Martello and Toth (1987, 1989, 1992) describe variants of TFISP with side constraints either on the total workload per machine or on the spread time per machine (i.e. the difference between the finishing time of the last assigned job and the starting time of the first assigned job). It is shown that these variants of TFISP, which are related to the bus driver scheduling problem, are NP-hard. Kroon, Salomon and Van Wassenhove (1993) present an approximation algorithm for solving the general variant of TFISP with several machine classes and several job classes. Their algorithm is based on Lagrangean relaxation and decomposition. It is very similar to the algorithm for solving OFISP in the present paper.

Kolen and Kroon (1991) show that OFISP is NP-hard when $C > 1$, except for some trivial cases. As a consequence, solving OFISP to optimality when $C > 1$ requires the use of (potentially very) time-consuming algorithms. An example of such an algorithm is given by Arkin and Silverberg (1987). Their algorithm is based on dynamic programming. Unfortunately, since the number of nodes of the corresponding network is $O(J^M)$ and the number of arcs is $O(J^{M+1})$, the practical applicability of this approach is

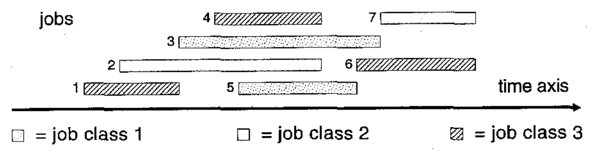

Fig. 4. Instance of OFISP with $A = 3$ and $C = 2$. The jobs in job class 1 can be carried out by the machines in machine class 1, the jobs in job class 2 can be carried out by the machines in machine class 2, and the jobs in job class 3 can be carried out by all machines.

small. Since OFISP must be solved routinely in practice, we concentrate on fast procedures that yield satisfactory (and not necessarily optimal) solutions. The number of available approximation algorithms for solving OFISP is still quite limited. One example is provided by Carter (1989), who presents an approximation algorithm based on Lagrangean relaxation for the classroom assignment problem, which is a problem closely related to OFISP. In the Lagrangean subproblems the restriction that the jobs must be processed in a non-preemptive way is relaxed. Also a heuristic for constructing feasible solutions is provided.

The algorithm described in the present paper exploits the observation that OFISP can be modelled as a minimum cost flow problem if all machines are identical. In our algorithm the restriction that each job must be processed at most once is relaxed. If the number of machine classes is greater than 1, then we construct for each machine class c a corresponding graph G_c, representing all jobs j for which $a_j \in \mathscr{A}_c$. Although the problem obtained in this way is still related to the minimum cost flow problem, it is complicated by the set of restrictions (3), which state that each job may be processed at most once. As a consequence, the graphs G_c are coupled by a set of constraints. These constraints must ensure that the total amount of flow that passes through the arcs corresponding to job j ($j = 1,\ldots, J$) is at most one. An instance of OFISP is shown in Fig. 4, and the corresponding graphs G_c and their coupling constraints are shown in Fig. 5. The dual-cost heuristic we propose can be summarized as follows:

Dual-cost heuristic:

```
Repeat
    Apply upper bounding procedure;
    Apply lower bounding procedure;
    Update dual-cost multipliers
Until Stop Criterion is fulfilled
```

The upper (lower) bounding procedure is described in Section 3.1 (Section 3.2). To update the dual-cost multipliers (introduced below), we use the standard subgradient optimization procedure as

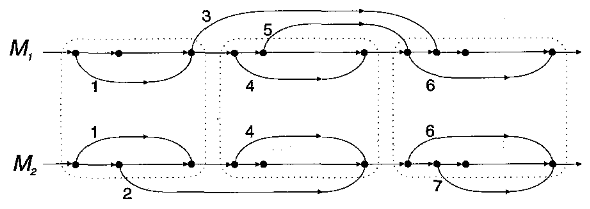

Fig. 5. The graphs G_1 and G_2 corresponding to the instance of OFISP shown in Fig. 4. The coupling constraints are indicated by dotted boxes. Note that, for ease of representation, we did not apply the Graph Compression Procedure here.

described by Fisher (1981) as well as a dual-descent procedure. The dual-cost multiplier updating procedure is described in Section 3.3. Finally, the stop criterion is based upon (i) the gap between upper and lower bound, (ii) the computation time, and (iii) the number of iterations.

3.1. Upper bounding procedure

Upper bounds to Z_{OFISP} are obtained by Lagrangean relaxation of the linking constraints (3), using non-negative multipliers $u=(u_1,\dots,u_J)$. The resulting Lagrangean problem LR(OFISP) is formulated as follows.

LR(OFISP):

$$Z_{\text{LR(OFISP)}}(u)=\max\sum_{j=1}^{J}\sum_{c\in\mathscr{C}_j}(p_j-u_j)x_{j,c}+\sum_{j=1}^{J}u_j \tag{1'}$$

subject to (2) and (4)

LR(OFISP) decomposes into C minimum cost flow problems with M_c units of flow on the graphs G_c, where the transportation costs per unit of flow are now equal to u_j-p_j for job j. It follows that $Z_{\text{LR(OFISP)}}(u)=-\Sigma_{c=1}^{C}Z(G_c)+\Sigma_{j=1}^{J}u_j$ where $Z(G_c)$ is the solution to the minimum cost flow problem on the graph G_c. Since it is a well known result (Fisher, 1981) that $\min_{u\geq 0}Z_{\text{LR(OFISP)}}(u)\geq Z_{\text{OFISP}}$, it follows that $Z_{\text{LR(OFISP)}}(u)$ yields an upper bound to Z_{OFISP} for all $u\geq 0$.

Furthermore, an alternative upper bound – not exploited in our heuristic, but used in Section 4 to compare the computational results with – is obtained by solving the linear programming relaxation of OFISP (or LP(OFISP) for short). Note that this bound equals $\min_{u\geq 0}Z_{\text{LR(OFISP)}}(u)$, since the Lagrangean problem satisfies the integrality property (Geoffrion, 1974). Finally, a third upper bound is obtained by relaxation of the restriction that each job j can only be carried out by a machine in machine class c when $j\in\mathscr{J}_c$. Upon relaxation of this set restriction a single class problem with $\Sigma_{c=1}^{C}M_c$ machines remains. This '*set relaxed*' problem, denoted by SR(OFISP), can be solved by the procedure described in Section 2.

3.2. Lower bounding procedure

The lower bounding procedure – which generates feasible solutions to OFISP – can be described as follows: at each iteration we start out with a tentative schedule with some jobs that have been assigned to a machine, while others are still unassigned. As long as there are idle machines, we search in a greedy fashion for an idle machine with the highest potential profit in terms of the unassigned jobs that can be processed by that machine. The latter is accomplished by repetitively solving a shortest path problem on the graphs G_c, from which all arcs corresponding to already assigned jobs have been eliminated. More formally, the lower bounding procedure is stated as follows:

Lower bounding procedure:

```
$M := \Sigma_{c=1}^{C} M_c$;
$\mathscr{J} :=$ {all jobs};
Repeat
    Search for a 'locally best' machine class $c^*$ with $M_{c^*} > 0$;
    $M_{c^*} := M_{c^*} - 1$;
    $M := M - 1$:
    $\mathscr{J} := \mathscr{J} \setminus$ {all jobs that can be carried out by one machine of $c^*$};
Until $M = 0$ or $\mathscr{J} = \emptyset$;
```

As already stated, the greedy search for a '*locally best*' machine class c^* is done by solving for each $c = 1,\dots,C$, a shortest path problem on the graph G_c, where all job arcs corresponding to jobs not in $\mathscr{J}$ have been deleted. In this problem the length of each job arc corresponding to job j equals $u_j - p_j$. Let the value of the resulting solution be equal to P_c. Then the locally best machine class c^* is taken as argmin $\{P_c \mid c = 1,\dots,C\}$. As can be seen easily, this procedure results in a feasible solution to OFISP, since it is ensured that (i) each job is processed at most once, and (ii) each machine processes at most one job at the same time. Consequently, the total value corresponding to the obtained schedule yields a lower bound to Z_{OFISP}. During the course of the heuristic the lower bounding procedure is executed every LB_{freq} iterations.

3.3. Dual-cost adaptation procedure

Our heuristic subsequently iterates between the upper bounding procedure and the lower bounding procedure, updating the dual-cost multipliers u each round, until some prespecified stop-criterion is satisfied. To update the dual-cost multipliers we apply the well-known subgradient optimization procedure (Fisher, 1981) in our first heuristic (*H1*):

$$u_j := \max\left(0,\ u_j + \lambda\left(1 - \sum_{c\in\mathscr{C}_j} x_{j,c}\right)\right)$$

where λ is a positive scalar step size, determined as:

$$\lambda := \frac{\mu\left(Z_{UB(H1)} - Z_{LB(H1)}\right)}{\sum_{j=1}^{J}\left(1 - \sum_{c\in\mathscr{C}_j} x_{j,c}\right)^2}$$

Here $Z_{UB(H1)}(Z_{LB(H1)})$ is the upper (lower) bound value obtained by heuristic *H1*. The dualcost multipliers are initialized at $u_j = 0$ for $j = 1,\dots,J$. The scalar μ has an initial value μ_0 which is halved whenever the upper bound has failed to decrease during $H1_{half}$ iterations.

As an alternative to *H1* we have also developed a second heuristic (*H2*), in which the subgradient optimization procedure is combined with a dual-descent procedure. In *H2* the dual-descent procedure starts (with the multipliers obtained by the upper bounding procedure) when the upper bound has failed to decrease during $H2_{decrease}$ subgradient iterations. It modifies the dual-costs of a job assigned more than once in such a way that this job will be assigned to at most one machine class in the next iteration of the upper bounding procedure. This implies a non-negative improvement of the upper bound in the next iteration of the upper bounding procedure. The number of dual-descent iterations is set equal to $H2_{iter}$. Thereafter the subgradient procedure is called again. More formally, the dual-descent procedure is described as follows:

Dual-descent procedure:

```
Initialization. Solve a minimum cost flow problem on $G_c$, $c = 1,\dots,C$;
                Let the corresponding objective value be $Z(G_c)$, $c = 1,\dots,C$;
Step 1.         Search for a job $j^*$ which is assigned to more than one machine class; If no such job
                exists then STOP else pick the first one and goto Step 2;
Step 2.         Remove all arcs corresponding to job $j^*$ from the graphs $G_c$, $c \in \mathscr{C}_{j^*}$, and denote the
                remaining graphs by $H_c$;
Step 3.         Solve a minimum cost flow problem on $H_c$, $c \in \mathscr{C}_{j^*}$, and let the corresponding objective
                value be $Z(H_c)$, $c \in \mathscr{C}_{j^*}$;
```

Step 4. Define $\Delta_1 := \max_{c \in \mathscr{C}_{j^*}} \{Z(H_c) - Z(G_c)\}$ and let this maximum be obtained for $c = c^*$;
Define $\Delta_2 := \max_{c \in \mathscr{C}_{j^*} \setminus \{c^*\}} \{Z(H_c) - Z(G_c)\}$;
Update $u_{j^*} := u_{j^*} + \frac{1}{2}(\Delta_1 + \Delta_2)$;

Remark 3. Note that, for the reader's convenience, we have included the initialization step in our description of the dual-descent procedure. However, the minimum cost flow problem has already been solved in the upper bounding procedure. Note further that in the dual-descent procedure both Δ_1 and Δ_2 are non-negative.

Lemma 4. *If $\Delta_2 < \Delta_1$, then job j^* is assigned to exactly one machine class in the next iteration of the upper bounding procedure. The improvement of the upper bound in the next iteration equals $\Sigma_{c \in \mathscr{C}_{j^*} \setminus \{c^*\}}(Z(H_c) - Z(G_c))$.*

Proof. Note that $\Delta_2 < \frac{1}{2}(\Delta_1 + \Delta_2) < \Delta_1$. If job j^* is assigned to machine class c^* in the next iteration, then $Z(G_{c^*})$ increases by $\frac{1}{2}(\Delta_1 + \Delta_2)$. If job j^* is *not* assigned to machine class c^* in the next iteration, then $Z(G_{c^*})$ increases by Δ_1. As we want to minimize $Z(G_{c^*})$, job j^* is assigned to machine class c^* in the next iteration. Now let c be different from c^*. If job j^* is assigned to machine class c in the next iteration, then $Z(G_c)$ increases by $\frac{1}{2}(\Delta_1 + \Delta_2)$. If job j^* is *not* assigned to machine class c in the next iteration, then $Z(G_c)$ increases by $Z(H_c) - Z(G_c)$, which is less than or equal to Δ_2. Therefore job j^* is *not* assigned to machine class c in the next iteration. The objective function of LR(OFISP) equals $\Sigma_{j=1}^{J} \Sigma_{c \in \mathscr{C}_j} (p_j - u_j) x_{j,c} + \Sigma_{j=1}^{J} u_j$. This implies that the objective function changes by

$$\left[\sum_{c \in \mathscr{C}_{j^*} \setminus \{c^*\}} (Z(H_c) - Z(G_c)) - \frac{\Delta_1 + \Delta_2}{2} \right] + \frac{\Delta_1 + \Delta_2}{2},$$

if u_{j^*} increases by $\frac{1}{2}(\Delta_1 + \Delta_2)$. □

Note that the improvement of the upper bound in the next iteration also equals $\Sigma_{c \in \mathscr{C}_{j^*} \setminus \{c^*\}}(Z(H_c) - Z(G_c))$ if $\Delta_1 = \Delta_2$. However, in this case job j^* may be assigned to more than one machine class in the next iteration.

4. Computational results

Heuristics *H1* and *H2* have been implemented with Borland's Turbo Pascal 5.0 on an Olivetti M380 with 80386 processor and 80387 mathematical co-processor [1]. Two different sets of problem instances are considered. The first set of instances contains a number of *randomly* generated instances, whereas the second set of instances comes from the *real-life* situation of the maintenance department at Schiphol Airport.

4.1. Randomly generated problem instances (Set I)

The first set of instances that we created to test our heuristics was generated randomly. In order to obtain information on the robustness of our heuristics, a number of problem parameters have been

[1] As the developed DSS had to run on a personal computer, we have obtained our computational results, as far as possible, on a personal computer.

varied:
- the number of jobs J,
- the number of job classes A,
- the a priori utilization rate ρ.

Here the *a priori utilization rate* ρ of the system is an indicator of the expected workload per capacity unit of the workforce. More formally, the utilization rate is defined as follows:

$$\rho = \frac{\text{expected total workload } (\textit{in time-units})}{\text{total workforce } (\textit{in time-units})} = \frac{J \times \frac{1}{2} D}{T \times M}$$

where D indicates the maximum job duration, T represents the length of the planning horizon, and M denotes the total number of machines. We consider instances with *low* utilization rate ($\rho = 0.8$), *medium* utilization rate ($\rho = 1.0$), and *high* utilization rate ($\rho = 1.2$). With respect to the machine classes it is assumed that each machine can process jobs from two different job classes, which results in the relation $C = \binom{A}{2}$. Remember that this reflects the situation at the maintenance department of KLM at Schiphol Airport, where each engineer is allowed to carry out jobs on at most two different aircraft types. As we consider instances with $A = 3$, $A = 4$, and $A = 5$, this results in instances with $C = 3$, $C = 6$, and $C = 10$. Furthermore, based upon the situation at Schiphol Airport we set the total number of machines to $M = 18$ or $M = 20$. The machines were equally divided over the different machine classes [2]. The parameter D was set in such a way that the required values for the utilization rate ρ were obtained.

The procedure for generating the jobs is as follows. We consider a planning horizon of $T = 1000$ time-units and instances with $J = 100$, $J = 200$ and $J = 300$. For each job j the class a_j is chosen randomly from the set $\{1, \ldots, A\}$ and the processing time d_j is generated randomly from the $U(0, D)$-distribution. The starting time s_j is generated randomly from the $U(0, T - d_j)$-distribution, and the finishing time f_j is set equal to $s_j + d_j$. The priorities p_j of the jobs are determined in such a way that the *total amount of work* that is carried out is maximized, which is achieved by putting $p_j = d_j$. For each (J, A, ρ)-combination obtained in this way we have generated 10 instances, which yields a total of 270 instances [3]. Table 1 shows some other parameter settings and stop-criteria used for *H1* and *H2* [4].

Table 2 shows the *average relative quality* $\overline{\Delta_H^R}$ and the *average absolute quality* $\overline{\Delta_H^A}$ for heuristic H. Here, and in the remainder of this paper, a performance measure with a bar is the *average* performance, computed over 10 instances per cell. The above quality measures are defined as:

$$\overline{\Delta_H^R} = \overline{\frac{Z_{UB(H)} - Z_{LB(H)}}{Z_{UB(H)}}} \quad \text{and} \quad \overline{\Delta_H^A} = \overline{\frac{Z_{\text{OFISP}} - Z_{LB(H)}}{Z_{\text{OFISP}}}}$$

OFISP and LP(OFISP) were, as far as possible, solved by LINDO (Schrage, 1987) on the personal computer [5]. For instances that were too large to be handled by LINDO on the personal computer we used OSL (IBM, 1991) on an IBM RS/6000. So, all instances were solved to optimality, either by LINDO or by OSL. Results in Tables 2 and 3 obtained by OSL on the IBM RS/6000 are placed in brackets.

[2] For $C = 3$ we set $M_c = 6$, for $C = 6$ we set $M_c = 3$, and for $C = 10$ we set $M_c = 2$ ($c = 1, \ldots, C$).

[3] The problem generator is available from the authors on request.

[4] The parameter settings are determined based on a small preliminary study.

[5] As an alternative to LINDO we experimented with NETSIDE (Kennington and Wishman, 1988) to solve LP(OFISP). NETSIDE is a specialized code for solving network problems with a number of linear side constraints. However, for our instances the computation times of NETSIDE were even higher than the ones required by LINDO. The latter may be caused by the fact that, although OFISP has clearly a network structure, the number of side constraints (3) is too high.

Table 1
Parameter settings and stop-criteria for heuristics

parameter setting	numerical value
subgradient procedure	$\mu_0 = 1.0$ when $\rho = 0.8$ $\mu_0 = 1.5$ when $\rho = 1.0$ $\mu_0 = 2.0$ when $\rho = 1.2$ $H1_{\text{half}} = 6$
dual-descent procedure	$H2_{\text{decrease}} = 6$ $H2_{\text{iter}} = 12$
lower bounding frequency	$LB_{\text{freq}} = 5$
stop criterion	**numerical value**
optimal solution	$\dfrac{Z_{UB(H)} - Z_{LB(H)}}{Z_{UB(H)}} \le 0.005$
maximum no. of iterations	J (number of jobs)
maximum CPU-time	$2J$ seconds

Table 3 shows the average quality ($\overline{\Delta^A_{LP}}$) and the average CPU-time ($\overline{LP_t}$, in seconds) required to solve LP(OFISP) [6]. In addition, the number of times the LP-relaxation yields a fractional solution is denoted by LP_f, while the number of times the bound obtained by solving SR(OFISP) is better than the bound obtained by the subgradient (dual-descent) procedure is found in the columns corresponding to d_{H1} (d_{H2}). From the computational results of Tables 2 and 3 it can be concluded that the heuristics *H1* and *H2* perform almost equally well with respect to absolute and relative deviation from optimality. The fact that *H2* does not perform significantly better than *H1* may be caused by the way we implemented *H2*. In the current implementation of *H2* we do not use sensitivity analysis to obtain the difference between $Z(H_c)$ and $Z(G_c)$. Hence in an alternative implementation of *H2* the time spent in the dual-descent procedure may possibly be reduced in favour of the number of iterations, which may lead to a better

Table 2
Quality of the heuristics

		$\rho = 0.8$				$\rho = 1.0$				$\rho = 1.2$			
A	J	$\overline{\Delta^R_{H1}}$	$\overline{\Delta^A_{H1}}$	$\overline{\Delta^R_{H2}}$	$\overline{\Delta^A_{H2}}$	$\overline{\Delta^R_{H1}}$	$\overline{\Delta^A_{H1}}$	$\overline{\Delta^R_{H1}}$	$\overline{\Delta^A_{H2}}$	$\overline{\Delta^R_{H1}}$	$\overline{\Delta^A_{H1}}$	$\overline{\Delta^R_{H2}}$	$\overline{\Delta^A_{H2}}$
3	100	0.02	0.01	0.02	0.01	0.01	0.01	0.01	0.01	0.02	0.00	0.02	0.00
	200	0.03	0.01	0.03	0.01	0.01	0.01	0.02	0.01	0.01	0.01	0.01	0.01
	300	0.04	0.02	0.04	0.02	0.02	0.02	0.03	0.02	0.01	0.01	0.01	0.01
4	100	0.02	0.02	0.03	0.02	0.01	0.01	0.01	0.01	0.02	0.00	0.02	0.00
	200	0.05	0.03	0.05	0.03	0.02	0.02	0.03	0.02	0.01	0.01	0.02	0.01
	300	0.06	(0.02)	0.06	(0.02)	0.04	(0.02)	0.04	(0.02)	0.02	(0.02)	0.02	(0.02)
5	100	0.03	0.02	0.03	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
	200	0.05	(0.03)	0.06	(0.03)	0.03	(0.02)	0.03	(0.02)	0.02	(0.01)	0.02	(0.01)
	300	0.06	(0.03)	0.06	(0.03)	0.03	(0.02)	0.03	(0.02)	0.02	(0.02)	0.02	(0.02)

[6] In case the problem was solved by OSL no CPU-times are reported in the table.

Table 3
Quality of upper bounding procedures

		$\rho = 0.8$					$\rho = 1.0$					$\rho = 1.2$				
A	J	$\overline{\Delta^A_{LP}}$	$\overline{LP_t}$	LP_f	d_{H1}	d_{H2}	$\overline{\Delta^A_{LP}}$	$\overline{LP_t}$	LP_f	d_{H1}	d_{H2}	$\overline{\Delta^A_{LP}}$	$\overline{LP_t}$	LP_f	d_{H1}	d_{H2}
3	100	< 0.01	40	2	9	10	< 0.01	57	1	4	5	< 0.01	52	2	7	7
	200	< 0.01	130	3	10	10	< 0.01	416	4	9	10	< 0.01	492	6	4	8
	300	< 0.01	181	3	10	10	< 0.01	1012	7	9	10	< 0.01	1583	7	1	4
4	100	< 0.01	199	6	8	9	< 0.01	211	4	0	0	< 0.01	140	5	3	4
	200	< 0.01	1080	8	10	10	< 0.01	1955	10	3	7	< 0.01	2193	10	0	0
	300	(< 0.01)	–	(7)	10	10	(< 0.01)	–	(10)	7	9	(< 0.01)	–	(10)	0	1
5	100	< 0.01	566	5	6	7	< 0.01	428	5	1	2	< 0.01	288	4	1	2
	200	(< 0.01)	–	(10)	10	10	(< 0.01)	–	(10)	1	2	(< 0.01)	–	(10)	0	0
	300	(< 0.01)	–	(10)	10	10	(< 0.01)	–	(10)	1	2	(< 0.01)	–	(10)	0	1

upper bound. The average relative difference (after $2J$ seconds of computation time [7]) between lower- and upper bound ranges (on average) from 1% for instances with high utilization rate and $A = 3$ to 6% for instances with low utilization rate and $A = 5$. The small absolute differences (from 0 to 3% on average) between the heuristic lower bound and the optimal solution indicates that the lower bounding routine is rather effective in finding good solutions to OFISP. In general, both the lower bounding procedure and the upper bounding procedures perform best for instances with high utilization rate. Furthermore, for instances with low utilization rate the alternative upper bound obtained from SR(OFISP) is often better than the subgradient and dual-descent upper bound. However, for instances with high utilization rate this bound is often outperformed by these upper bounds.

Table 3 shows further that the upper bound obtained from LP(OFISP) is remarkably tight. For some instances the solution to LP(OFISP) turns out to be all integer, whereas for others the value obtained from LP(OFISP) equals Z_{OFISP}, although the corresponding solution is fractional. Unfortunately, the CPU-time required to compute this upper bound increases strongly in the number of jobs and in the number of machine classes. Furthermore, the number of instances for which an optimal integer solution is obtained by solving LP(OFISP) is high for the set of instances with low utilization rate and a small number of machine classes and jobs, but decreases fast when the utilization rate, the number of machine classes, or the number of jobs increases. A fractional solution to LP(OFISP) for an instance with a small number of machine classes generally contains only a small number of fractional variables. On the other

Table 4
Overview of workload and number of jobs

	Mon		Tue		Wed		Thu		Fri		Sat		Sun		Total	
	hrs.	jobs	hrs.	jobs	hrs.	jobs	hrs.	jobs	hrs.	jobs	hrs.	jobs	hrs.	jobs	hrs.	jobs
A310	31.2	37	33.1	42	36.8	47	29.8	41	48.1	58	35.8	48	41.4	55	251.2	328
B737	68.1	120	69.3	122	68.1	115	64.4	115	71.2	123	61.3	103	67.2	116	469.6	814
B747	83.7	48	99.8	67	96.5	64	102.7	63	69.3	50	122.5	84	80.9	56	655.4	432
DC10	19.4	15	31.6	21	16.0	12	22.6	15	13.9	12	30.9	21	18.7	14	153.1	110
Total	202.4	220	233.8	252	217.4	238	219.5	234	197.5	243	250.5	256	208.2	241	1529.3	1684

[7] *H1* (*H2*) stopped for 22 (17) out of 270 instances before $2J$ seconds of CPU-time because of the optimality criterion, and for 4 (2) out of 270 instances because of having reached the maximum number of J iterations.

Table 5
License combinations in each scenario

license combination	scenario No. 1	No. 2	No. 3	No. 4	No. 5	No. 6	No. 7
A310/B737	3	4	5	4	4	3	3
A310/B747	3	4	5	4	4	3	3
A310/DC10	3	4	5	2	6	1	5
B737/B747	3	4	5	6	2	5	1
B737/DC10	3	4	5	4	4	3	3
B747/DC10	3	4	5	4	4	3	3
Total teamsize	18	24	30	24	24	18	18

hand, a fractional solution for an instance with a high number of machine classes can be *almost completely fractional*. In such cases rounding down the fractional solution does not lead to a useful integer solution. One is then committed to an enumeration scheme in order to obtain a good integer solution. It appears from our experiments that -notwithstanding the small gap between Z_{OFISP} and $Z_{\text{LP(OFISP)}}$-finding an optimal integer solution may be an enormous task, not only for LINDO on the personal computer, but also for OSL on the IBM RS/6000. For example, solving the largest instances of our experiments to optimality takes a CPU-time which varies from 10 minutes to 8 hours on the RS/6000. The latter makes an enumeration approach interesting from a theoretical point of view [8], but in practice, where OFISP has to be solved on a personal computer quickly and routinely in a dynamic environment, this approach has only a limited value.

4.2. Real-life instances (Set II)

Apart from the randomly generated instances of OFISP, we also analyze a number of instances of OFISP that come from a study for the maintenance department of KLM at Schiphol Airport. In these

Table 6
Day-by-day results for scenario No. 4

	Quality of heuristics				Upper bounding procedures			
	Δ^R_{H1}	Δ^A_{H1}	Δ^R_{H2}	Δ^A_{H2}	Δ^A_{LP}	LP_f	d_{H1}	d_{H2}
Mon	0.07	0.06	0.08	0.06	<0.01	n	n	n
Tue	0.07	0.06	0.08	0.06	<0.01	y	n	n
Wed	0.06	0.04	0.06	0.04	<0.01	n	y	y
Thu	0.06	0.05	0.06	0.05	<0.01	n	n	n
Fri	0.08	0.06	0.09	0.06	<0.01	n	n	n
Sat	0.08	0.07	0.05	0.04	<0.01	n	n	n
Sun	0.03	0.02	0.03	0.02	<0.01	n	n	n

[8] Research is in progress (Kroon 1990) to develop polyhedral approaches for solving OFISP. This research is based on a slightly different formulation of OFISP as a Node Packing problem. In this formulation (aggregated) decision variables $x_{j,c}$ are replaced by (disaggregated) decision variables $x'_{j,m}$, reflecting the 0–1 decision on processing job j at machine m (instead of processing job j on a machine from class c). This approach has the advantage that the corresponding coefficient matrix is a clique matrix and that the *lifted odd-hole inequalities* of Padberg (1974) can sometimes be used to turn a fractional solution into an integer one. Unfortunately, the size of this alternative formulation grows very fast in the number of machines, making this formulation not useful on a personal computer for the problem dimensions studied here. For the aggregated formulation of OFISP given in Section 3 we did not yet succeed in finding useful valid inequalities. Furthermore, we are currently looking for specialized branch-and-bound procedures for solving OFISP.

Table 7
Weekly results for all scenarios

	Quality of heuristics				Upper bounding procedures			
	Δ^R_{H1}	Δ^A_{H1}	Δ^R_{H2}	Δ^A_{H2}	Δ^A_{LP}	LP_f	d_{H1}	d_{H2}
Scen. No. 1	0.06	0.05	0.06	0.05	<0.01	y	n	n
Scen. No. 2	0.07	0.05	0.08	0.05	<0.01	y	y	y
Scen. No. 3	0.08	0.03	0.08	0.03	<0.01	y	y	y
Scen. No. 4	0.06	0.03	0.06	0.03	<0.01	y	y	y
Scen. No. 5	0.07	0.06	0.07	0.05	<0.01	y	n	n
Scen. No. 6	0.05	0.03	0.05	0.03	<0.01	y	y	y
Scen. No. 7	0.08	0.07	0.05	0.04	<0.01	y	n	n

instances of OFISP four different aircraft types are present: *A310, B737, B747*, and *DC10*. For each of these aircraft types the workload per day (in hours) and the number of jobs per day are represented in Table 4. For example, on Monday for the *A310* the total workload is 31.2 hours, generated by 37 jobs. Each engineer has a license for two aircraft types. KLM's management is interested in the best size and composition of her workforce. In order to obtain this information, seven different scenarios have been generated, as shown in Table 5. Each scenario represents one composition of the workforce. For example, in scenario No. 1 all license combinations are obtained by three engineers, and the total teamsize is 18 people. For each scenario we obtain day-by-day and weekly results. The results for our heuristics have been obtained on a personal computer, whereas the solutions to OFISP and LP(OFISP) were obtained by OSL on an IBM RS/6000. All instances were solved to proven optimality. In Table 6 some of the day-by-day results are shown for scenario No. 4. In the Tables 6 and 7 LP_f shows whether or not LP(OFISP) has a fractional solution, and d_{H1} and d_{H2} indicate if the upper bound obtained from SR(OFISP) was better than the upper bound obtained by the heuristics *H1* and *H2*. In the corresponding columns 'y' indicates 'yes', and 'n' indicates 'no'. The weekly results have been obtained for all scenarios by aggregating the day-by-day results. These results are shown in Table 7. The results obtained for these real-life instances are a little worse (1–2%) than the results obtained for the randomly generated ones. However, the absolute deviation from optimality is still quite acceptable. Furthermore, for a given scenario the upper bounding procedures perform best on days with a relatively low utilization rate (workload). The weekly results, which all correspond to the same workload, show the same trend. That is, the upper bounds are strong if the total number of engineers in the scenario is relatively low.

As before, the upper bound obtained from LP(OFISP) is excellent. In many cases this upper bound equals Z_{OFISP}, although the corresponding solution is fractional. In general, solving these real-life instances to optimality requires less computational effort than solving the (equally sized) randomly generated instances. For most of the real-life instances OSL finds an optimal integer solution after a search through a limited number of nodes of the branch-and-bound tree, whereas for some of the randomly generated instances OSL has to search through well over five thousand nodes. Still, on a personal computer the enumeration approach has only a limited value, even for these real-life instances.

5. Final remarks

In this paper we consider the Operational Fixed Interval Scheduling Problem (OFISP) and its appearance in practice. We suggest an exact algorithm for the single machine class variant. This algorithm is a simplification of the algorithms of Arkin and Silverberg (1987) and Kolen et al. (1987). Furthermore, we formulate the multiple machine class variant as an integer program, and we present two dual-cost heuristics for solving this general problem. Finally, we compare the performance of our heuristics with the performance of solving LP(OFISP) followed by a standard enumeration scheme. Although LP(OFISP) yields excellent upper bounds, this approach has as a serious draw-back associated with it that for larger sized instances it requires too much time and memory, both on a personal computer and on a workstation. This draw-back together with the observation that our heuristics yield -within an acceptable amount of CPU-time on a personal computer- feasible solutions that are on average only 0–7% from optimality makes our heuristics better suitable for use in practice than the enumeration approach. Nevertheless, future research has to focus on polyhedral methods and/or tailor made enumeration schemes which hopefully improve the performance of the enumeration approach.

Acknowledgements

The authors want to express their gratitude to professors Antoon Kolen and Jo van Nunen for their stimulating discussions. Furthermore, the authors want to thank professor Jef Kennington for making NETSIDE available, and IBM (Netherlands) for making OSL available. Finally the referees are acknowledged for useful suggestions which lead to improvements of the paper.

References

Arkin, E.M., and Silverberg, E.L. (1987), "Scheduling jobs with fixed starting and finishing times", *Discrete Applied Mathematics* 18, 1–8.

Carter, M.W., (1989), "A lagrangean relaxation approach to the classroom assignment problem", *INFOR* 27, 230–246.

Dondeti, V.R., and Emmons, H. (1992), "Interval scheduling with processors of two types", *Operations Research* 40, S76–S85.

Dijkstra, M.C., Kroon, L.G., van Nunen, J.A.E.E., and Salomon, M. (1991), "A DSS for capacity planning of aircraft maintenance personnel", *International Journal of Production Economics* 23, 69–78.

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

Fischetti, M., Martello, S., and Toth, P. (1987), "The Fixed Job Schedule Problem with spread time constraints", *Operations Research* 6, 849–858.

Fischetti, M., Martello, S., and Toth, P. (1989), "The Fixed Job Schedule Problem with working time constraints", *Operations Research* 3, 395–403.

Fischetti, M., Martello, S., and Toth, P. (1992), "Approximation algorithms for Fixed Job Schedule Problems", *Operations Research* 40, S96–S108.

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

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

Gertsbakh, I., and Stern, H.I. (1978), "Minimal resources for fixed and variable job schedules", *Operations Research* 26, 68–85.

Gupta, U.L., Lee, D.T., and Leung, J.Y.-T. (1979), "An optimal solution to the channel assignment problem", *IEEE Trans. Comp.* 28, 807–810.

Hagdorn-van der Meijden, E., and Kroon, L.G. (1989), "Pitfalls in obtaining solutions for an aircraft to gate assignment problem", Man. Rep. Series 59, Rotterdam School of Management, Erasmus University, The Netherlands.

Hashimoto, A., and Stevens, J. (1971), "Wire routing by optimizing channel assignments within large apertures", in: *Proceedings of the 8th Design Automation Workshop*, 155–169.

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

Kennington, J.L., and Whisman, A. (1988), "NETSIDE user guide", Technical Report 86-OR-01 (revised), Southern Methodist University, Dallas.

Kolen, A.W.J. and Kroon, L.G. (1991), "On the computational complexity of (maximum) class scheduling", *European Journal of Operational Research* 54, 23–38.

Kolen, A.W.J., and Kroon, L.G. (1992), "License class design: complexity and algorithms", *European Journal of Operational Research* 63, 432–444.

Kolen, A.W.J., Lenstra, J.K., and Papadimitriou, C.H. (1987), "Interval scheduling problems", unpublished manuscript.

Kroon, L.G. (1990), "Job scheduling and capacity planning in aircraft maintenance", PhD Thesis, Rotterdam School of Management, Erasmus University, The Netherlands.

Kroon, L.G., Salomon, M., and Van Wassenhove, L.N. (1993), "Exact and approximation algorithms for the tactical fixed interval scheduling problem", Working Paper, Rotterdam School of Management, Erasmus University, The Netherlands.

Orlin, J.B. (1988), "A faster strongly polynomial minimum cost flow algorithm", in: *Proceedings of the 20th ACM symposium on the theory of computing*.

Padberg, M.W. (1973), "On the facial structure of set packing polyhedra", Mathematical Programming 5, 199–215.

Schrage, L., (1987), *User Manual for Linear, Integer and Quadratic Programming with LINDO*, The Scientific Press, Redwood City, 1987.

Wit, J. de, (1991), "On the assignment of aircraft to gates", Technical Report, ORTEC Consultants, Gouda, The Netherlands, 1991. (Paper presented at the EURO XI Conference, Aachen (Germany), July 16–19, 1991).