European Journal of Operational Research 79 (1994) 417–430
North-Holland

Theory and Methodology

An analysis of shift class design problems

Antoon W.J. Kolen
University of Limburg, P.O. Box 616, 6200 MD Maastricht, Netherlands

Leo G. Kroon
Erasmus University, P.O. Box 1738, 3000 DR Rotterdam, Netherlands

Received October 1992; revised January 1993

Abstract: In this paper we consider a generalization of the Fixed Job Schedule Problem (FJSP) which appears in the aircraft maintenance process at an airport. A number of jobs must be carried out where each job requires processing from a fixed start time to a fixed finish time. These jobs must be carried out by a number of machines which are available in specific shifts only. The jobs must be carried out in a non-preemptive way, although at the end of a shift preemption of a job is allowed sometimes. The problem is to choose the number of machines in each of the shifts in such a way that all jobs can be carried out and that the total costs of the machines or the total number of machines are minimum. In this paper we present an analysis of the computational complexity of these problems. We also analyse the worst case behaviour of the preemptive variant versus the non-preemptive variant.

Keywords: Computational complexity; Fixed Job Schedule Problem; Preemptive and non-preemptive scheduling

1. Introduction

Between the time of arrival and the time of departure of an aircraft at the main airport in the Netherlands the aircraft must be inspected before being allowed to take off again. Such an inspection is a job with a fixed start time and a fixed finish time. The start time and the finish time of an inspection may coincide with the time of arrival and the time of departure of the aircraft, but this is not necessary: a list of maintenance norms is available which is used to determine the start time and the finish time of each inspection. The inspections must be carried out by a number of ground engineers. These ground engineers are available at the airport in specific shifts only. In principle the inspections must be carried out in a non-preemptive way. This implies that an inspection can be carried out by a ground engineer only if the interval between the start and finish time of the inspection is a subinterval of the shift of the engineer. However, at the end of a shift preemption of a job is allowed sometimes.

In the practical situation at the airport the begin and end times of the shifts are fixed. Thus an important capacity planning problem to be solved is to determine the number of engineers in

Correspondence to: Dr. L.G. Kroon, Erasmus University, P.O. Box 1738, 3000 DR Rotterdam, Netherlands.

SSDI 0377-2217(93)E0121-D

each of the shifts in such a way that all inspections can be carried out and that the total costs of the engineers or the total number of engineers are minimum.

In the practical situation at the airport still other constraints must be satisfied. For example, an engineer is allowed to carry out a specific inspection only if he has a license for the corresponding aircraft type. The problems introduced by the licenses of the engineers have been studied by Kolen and Kroon [13,14]. In this paper we neglect these problems by assuming that all engineers have licenses for all aircraft types.

This paper is organized as follows. In Section 2 we give a formal definition of the capacity planning problems to be studied. In Section 3 the results of Kolen and Kroon [15] are summarized, as they are used several times in the Sections 4 and 5 of this paper. In Section 4 we present an analysis of the problem of finding the minimum total number of engineers required for carrying out all jobs. Both the non-preemptive and the preemptive variant are considered. The problem of minimising the total costs of the engineers such that all jobs can be carried out is studied in Section 5. We also describe the extent to which the presented classification of the computational complexity of this problem is complete. We finish with some concluding remarks in Section 6.

In order to follow the literature on job scheduling, in the remainder of this paper the inspections are addressed as *jobs* and the engineers are addressed as *machines*.

2. Problem definition

Suppose that J jobs must be carried out where job j requires processing from a fixed start time s_j to a fixed finish time f_j. In principle the jobs must be processed in a non-preemptive way. The jobs must be carried out by a number of parallel machines which are available in specific shifts only. The number of shifts is denoted by Z and shift z has a fixed begin time b_z and a fixed end time e_z. In this paper it is assumed that the shifts have been sorted according to their begin time. Thus $b_{z-1} \leq b_z$ for $z = 2, \ldots, Z$. With each machine in shift z we associate costs k_z.

The notation $t_0, \ldots, t_P$ is used to represent all relevant instants of time (i.e. the start and finish times of the jobs and the begin and end times of the shifts) in chronological order. Thus

$$\{t_p \mid p = 0, \ldots, P\} = \{s_j, f_j \mid j = 1, \ldots, J\} \cup \{b_z, e_z \mid z = 1, \ldots, Z\}$$

and $t_{p-1} < t_p$ for $p = 1, \ldots, P$. In a similar way we use the notation $u_0, \ldots, u_Q$ to represent the begin and end times of the shifts in chronological order. That is,

$$\{u_q \mid q = 0, \ldots, Q\} = \{b_z, e_z \mid z = 1, \ldots, Z\}$$

and $u_{q-1} < u_q$ for $q = 1, \ldots, Q$.

Shift x is said to have a *unique begin time* if $b_y \neq b_x$ for all $y \neq x$. A *unique end time* of a shift is defined similarly. Finally, shift x is said to be *dominated* by shift y if shift x is a subinterval of shift y. That is, $b_y \leq b_x < e_x \leq e_y$. Note that, if $b_x = b_y$, then at least one of the shifts x or y is dominated. A similar remark holds if $e_x = e_y$.

Preemption of a job (s_j, f_j) means that the job is split into (at least) two parts (s_j, p_j) and (p_j, f_j) which are carried out by different machines. If preemption of a job is *not* allowed, then it must be assumed that each job is contained in at least one of the intervals (b_z, e_z). Otherwise a feasible non-preemptive schedule can not exist, whatever the number of machines in each shift. If preemption of a job is allowed, then it must be assumed that each job is contained in the interval (b_1, e_Z).

Suppose that a set S containing Z shifts (b_z, e_z) has been given. Then the problem Shift Class Design with respect to S (abbreviated to SCD(S)) is the following capacity planning problem.

Instance of SCD(S):
- J jobs (s_j, f_j) to be carried out;
- Z numbers k_z representing the costs per machine in each of the Z shifts.

Question:
- How to choose the numbers of machines in the Z shifts such that a feasible non-preemptive schedule exists for all jobs and such that the total costs of the machines are minimum?

Note that the set of shifts S belongs to the type of the problem SCD(S) and that it does not

belong to the instances of the problem. Therefore a whole class of problems has been defined which is indexed by the sets of shifts S.

A special case of SCD(S) is obtained by assuming that the costs per machine are shift-independent. This problem is called Shift Class Design with Uniform Costs (abbreviated to SCDUC(S)). Related problems are the problems SCD and SCDUC, which are conceptually similar to the problems SCD(S) and SCDUC(S). However, in SCD and SCDUC the set of shifts S belongs to the instances of the problem instead of to the type of the problem.

These problems are generalizations of the well-known Fixed Job Schedule Problem (FJSP). In this problem all jobs have a fixed start and finish time and all machines are continuously available. FJSP is the capacity planning problem of determining the minimum number of machines such that all jobs can be carried out in a non-preemptive way. Thus FJSP is equivalent to the problem SCDUC(S) if the set of shifts S contains only one shift. The optimal solution to an instance of FJSP is described in the following lemma.

Lemma 1. *In* FJSP *the minimum number of machines required for carrying out all jobs in a non-preemptive way equals the maximum job overlap of the jobs.*

Here the job overlap at time instant t, denoted by D_t, and the maximum job overlap, denoted by D, are defined by

$$D_t = \left|\{j \mid s_j \leqslant t < f_j\}\right|, \quad D = \max_t D_t.$$

The shift overlap at time instant t and the maximum shift overlap are defined similarly.

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 equals the size of a maximum antichain (Dilworth [3]).

An $O(J \log J)$ algorithm for determining the maximum job overlap of the jobs is described by Hashimoto and Stevens [12] and by Gupta, Lee and Leung [11]. Note that the minimum number of machines required for carrying out all jobs in a *preemptive* way also equals the maximum job overlap of the jobs. So in this case nothing can be gained by allowing preemption of the jobs.

Dondeti and Emmons [4] study a generalization of FJSP 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 a minimum cost configuration of machines allowing a feasible schedule for all jobs can be found in polynomial time by repeatedly solving a network flow problem.

Fischetti, Martello and Toth [5–7] 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. Furthermore, lower and upper bounds are described together with their corresponding worst case behaviour.

Other problems closely related to the mentioned problems are studied by Carter and Tovey [2] and by Arkin and Silverberg [1]. Carter and Tovey [2] study the complexity of the classroom assignment problem, which is also a Fixed Job Schedule Problem. Here the number of available machines is given. This is also assumed by Arkin and Silverberg [1]. For each job j a subset of machines W_j is given and it is assumed that job j can be carried out by the machines in W_j only. The objective is to find a feasible schedule for all jobs. It is shown that this problem is NP-complete. Further, a dynamic programming formulation is presented that can be used for finding in $O(J^{M+1})$ time a subset of jobs of maximum total value for which a feasible schedule exists. Here M represents the number of machines. Hence if M is fixed, then this optimization problem can be solved in polynomial time.

3. Preliminary remarks

The computational complexity of the problems SCD(S) and SCDUC(S) depends on the size and the structure of the set of shifts S. In the Sections 4 and 5 a classification of the computational complexity of a large subset of these problems is presented. To a great extent this classification is based on the results of Kolen and Kroon [15]. Therefore these results are summarized in this section.

Kolen and Kroon [15] analyse the computational complexity of the problem Shift Class Scheduling with respect to S (abbreviated to SCS(S)). This problem is a feasibility problem asking whether or not a feasible schedule exists for a given set of jobs if the number of machines in each of the shifts is known. For a given set of shifts S this problem is described more formally as follows.

Instance of SCS(S):
- J jobs (s_j, f_j) to be carried out;
- Z integers M_z for $z = 1, \ldots, Z$ representing the number of machines in shift z and satisfying $M_z \leqslant J$.

Question:
- Does there exist a feasible non-preemptive schedule for all jobs?

Note that the set of shifts S again belongs to the type of the problem SCS(S) and that it does not belong to the instances of the problem. The results of Kolen and Kroon [15] on the computational complexity of the problem SCS(S) are expressed in terms of the undirected graph $G(S)$ which is defined as follows.
- Each node of $G(S)$ corresponds to one of the shifts in S.
- Two nodes of $G(S)$ are connected $\Leftrightarrow$ the corresponding shifts are overlapping, and the corresponding shifts have different begin times and different end times.

Examples of sets of shifts S and corresponding graphs $G(S)$ are given in Figure 1.

In this paper it is always assumed that the graph $G(S)$ is connected, because otherwise the corresponding capacity planning or job scheduling problem can be decomposed into a number of subproblems. Now the main results of Kolen and Kroon [15] on the computational complexity of the problem SCS(S) are summarized as follows.

Theorem 2. *If the graph $G(S)$ is bipartite, then* SCS(S) *can be solved in polynomial time. If the graph $G(S)$ contains a triangle as a node-induced subgraph, then* SCS(S) *is NP-complete.*

4. The problems SCDUC and SCDUC(S)

In the problems SCDUC and SCDUC(S) the costs of the engineers are shift-independent. As a consequence, if shift x is dominated by shift y and I is an instance of SCDUC or SCDUC(S), then an optimal solution for I exists where the number of machines in shift x is zero. Indeed, if a feasible solution exists containing Y_x machines in shift x and Y_y machines in shift y, then another feasible solution exists containing 0 machines in shift x and $Y_x + Y_y$ machines in shift y.

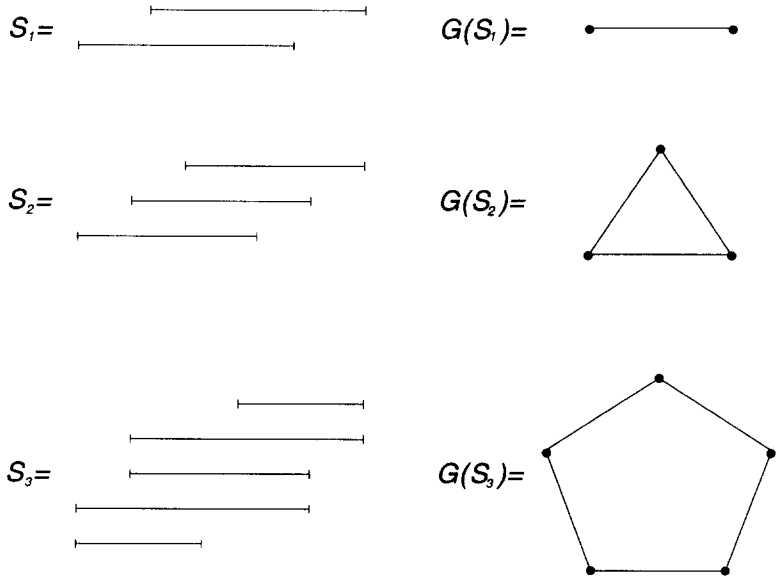

Figure 1. Examples of sets of shifts S and corresponding graphs $G(S)$

As a consequence, it may be assumed that all shifts are non-dominated. Note that this together with the assumed connectivity of $G(S)$ implies $b_z < b_{z+1} < e_z < e_{z+1}$ for $z = 1, \ldots, Z-1$.

4.1. Preemptive scheduling

In this subsection it is shown that the preemptive variant of SCDUC and SCDUC(S) can be solved in polynomial time. Furthermore, we analyse the worst case behaviour of the preemptive variant versus the non-preemptive variant.

Lemma 3. *A feasible preemptive schedule can be transformed into a feasible preemptive schedule where each preemption coincides with the end time of a shift.*

Proof. Suppose that at least one preemption does not coincide with the end time of a shift. That is, (p_{i-1}, p_i) and (p_i, p_{i+1}) are parts of one job carried out by different machines m_1 and m_2, and p_i does not coincide with the end time of a shift. Let e_1 and e_2 denote the end times of the shifts of the machines m_1 and m_2 respectively. Then the subschedules on the machines m_1 and m_2 between p_i and $\min\{e_1, e_2\}$ can be swapped. An example of such a swap is given in Figure 2. A consequence of the swap is that the preemption on p_i no longer exists. Note that a new preemption may have been introduced. However, this new preemption coincides with the end time of a shift, which is allowed. Thus the number of preemptions not coinciding with the end time of a shift has been reduced by one. By applying similar swaps as often as required, one finally arrives at a feasible schedule where all preemptions coincide with the end time of a shift. □

Next we prove that the preemptive variant of SCDUC and SCDUC(S) can be solved in polynomial time by the following greedy algorithm (A.1). Recall that D_t denotes the job overlap at time instant t.

Algorithm (A.1).
$\tilde{D}_t := D_t$ for $b_1 \leqslant t \leqslant e_Z$
For $z = 1, \ldots, Z$ do
  $Y_z := \max\{\tilde{D}_t \mid b_z \leqslant t < b_{z+1}\}$
  $\tilde{D}_t := \max\{0, \tilde{D}_t - Y_z\}$ for $b_z \leqslant t < e_z$
End do

In this description b_{Z+1} should be interpreted as e_Z. Note that each iteration requires $O(J \log J)$ time. Thus the running time of algorithm (A.1) is $O(ZJ \log J)$, which is $O(J \log J)$ if the number of shifts is fixed. Next we prove that the algorithm produces an optimal solution.

Lemma 4. *Algorithm* (A.1) *produces an optimal solution for the preemptive variant of* SCDUC *and* SCDUC(S).

Proof. First we will prove that a non-preemptive schedule exists if the numbers of machines Y_z have been determined by algorithm (A.1).

As was pointed out earlier, the non-dominance of the shifts together with the connectivity of $G(S)$ implies $b_z < b_{z+1} < e_z < e_{z+1}$ for $z =$

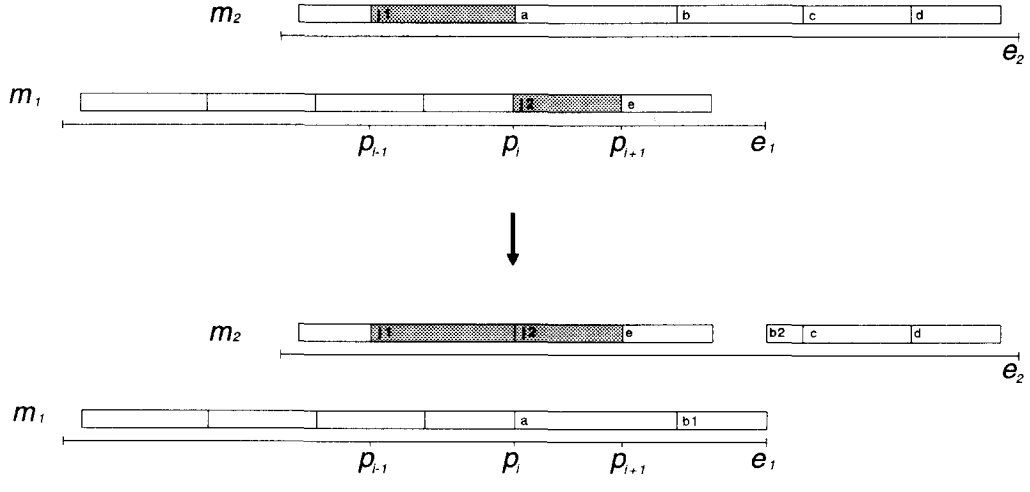

Figure 2. Example of a swap as described in the proof of Lemma 3

$1, \ldots, Z-1$. Thus the Y_z machines that start at b_z work at least until b_{z+1}. As a consequence, for all time instants t we have the inequality

$$\sum_{\{z \mid b_z \leqslant t < e_z\}} Y_z \geqslant D_t. \tag{1}$$

Now the jobs can be scheduled one by one. If job j is about to be scheduled, then inequality (1) implies that during the whole interval (s_j, f_j) at least one machine is free. It may happen that this is not one single machine. Nevertheless, the job can be scheduled, because preemption is allowed.

Next we will show that the obtained solution is optimal. Indeed, the inequalities $b_z < b_{z+1} < e_z < e_{z+1}$ for $z = 1, \ldots, Z-1$ imply that a machine in shift z, which is completely idle during the interval (b_z, b_{z+1}), can be replaced by a machine in shift $z+1$. Thus algorithm (A.1), which guarantees that a machine in shift z is *not* completely idle during the interval (b_z, b_{z+1}) and which finally assigns a minimum number of machines to shift Z, produces an optimal solution. □

Let the optimal solution to the non-preemptive variant be denoted by Y and let the optimal solution to the preemptive variant be denoted by Y_{pre}. The following lemma presents a general result on the relation of Y and Y_{pre}.

Lemma 5. $Y < 2Y_{\text{pre}}$ *for the problem* SCDUC.

Proof. Suppose one has a preemptive schedule with Y_{pre} machines. As was shown in Lemma 3, this schedule can be transformed into a preemptive schedule where each preemption coincides with the end time of a shift. As a consequence, the number of preempted jobs is less than or equal to Y_{pre}, because now each machine contains at most one first part of a preempted job. By choosing one appropriate machine for each preempted job one obtains a non-preemptive schedule with at most $2Y_{\text{pre}}$ machines, which implies $Y \leqslant 2Y_{\text{pre}}$. The strict inequality is caused by the fact that a job cannot be preempted at e_Z. □

The result of Lemma 5 is the best possible result, as is shown by the set of instances represented in Figure 3. In these instances $4n+3$ machines are required for a feasible non-preemptive schedule. However, a feasible preemptive schedule exists already if 1 machine is available in each of the odd-numbered shifts giving a total of $2n+2$ machines.

Lemma 5 is a very general result which holds for all sets of shifts. For each specific set of shifts this result can be improved. An example of this is given in the following lemma concerning the set of shifts S_2 which was shown in Figure 1.

Lemma 6. $Y \leqslant \frac{5}{3}Y_{\text{pre}}$ *for the problem* SCDUC(S_2).

Proof. Suppose one has a preemptive schedule with Y_z machines in shift z ($z = 1, 2, 3$). The schedule can be transformed into a preemptive schedule where each preemption coincides with the end of shift 1 or with the end of shift 2. Let y_{12} denote the number of preempted jobs at the end of shift 1 that are continued on a machine in shift 2, and let y_{13} and y_{23} be defined similarly. Then it follows that there exists a feasible preemptive schedule with Y_1 machines in shift 1, $Y_2 + y_{12}$ machines in shift 2, and $Y_3 + y_{13} + y_{23}$ machines in shift 3. Thus

$$Y \leqslant Y_1 + (Y_2 + y_{12}) + (Y_3 + y_{13} + y_{23}). \tag{2}$$

Note that $y_{12} + y_{13} \leqslant Y_1$. Furthermore, $y_{23} \leqslant \min\{Y_2, Y_3\}$. Thus we obtain the inequalities

$$Y \leqslant 2Y_1 + 2Y_2 + Y_3, \tag{3}$$

$$Y \leqslant 2Y_1 + Y_2 + 2Y_3. \tag{4}$$

By reversing the time-axis the schedule can be transformed into a preemptive schedule where

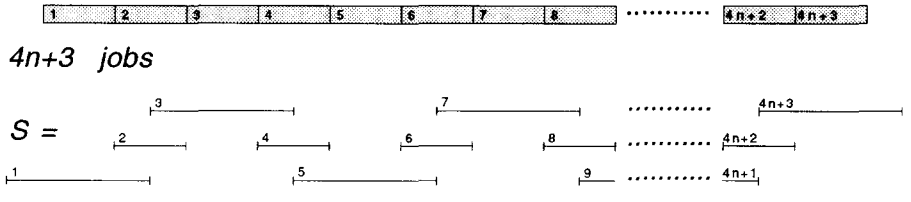

Figure 3. A set of instances of SCDUC

each preemption coincides with the begin time of shift 2 or with the begin time of shift 3. In a similar way as above this gives the inequality

$$Y \leqslant Y_1 + 2Y_2 + 2Y_3. \tag{5}$$

By adding the inequalities (3), (4) and (5) one finds $3Y \leqslant 5(Y_1 + Y_2 + Y_3)$, which is the desired result. □

The result of Lemma 6 is the best possible result, as is shown by the instance represented in Figure 4. In this instance 5 machines are required for a feasible non-preemptive schedule. However, a feasible preemptive schedule exists already if 1 machine is available in each shift giving a total of 3 machines.

Although the results of the Lemmas 5 and 6 show that in the worst case the gap between the solutions of the non-preemptive and the preemptive variant can be substantial, an experimental study has revealed that in general this gap is very small. More details on this statement are provided in a forthcoming paper.

4.2. Non-preemptive scheduling

The computational complexity of the problem SCDUC(S) depends on the size and the structure of the set of shifts S. A complete classification of this computational complexity is expressed by Theorem 7. Note that the fact that there exist sets of shifts S such that SCDUC(S) is NP-hard implies that SCDUC is NP-hard.

Theorem 7. *If the maximum shift overlap of the non-dominated shifts is less than or equal to* 2, *then* SCUDC(S) *can be solved in polynomial time. If the maximum shift overlap of the non-dominated shifts is greater than 2, then* SCDUC(S) *is NP-hard.*

Proof. As was pointed out earlier, it may be assumed that all shifts are non-dominated. If the maximum shift overlap is less than or equal to 2, then the graph $G(S)$ does not contain any circuit, and, therefore, it is bipartite. Thus Theorem 2 implies that SCS(S) can be solved in polynomial time. Let I be an instance of SCDUC(S) containing J jobs (s_j, f_j), and let the integers Y_z represent the numbers of machines that must be available in each of the Z shifts in an optimal solution for I. Then it may be assumed that the numbers Y_z satisfy the relation

$$\sum_{z=1}^{Z} Y_z \leqslant J. \tag{6}$$

Indeed, if more than J machines were available, then at least one machine would be completely inactive. As SCS(S) can be solved in polynomial time, an integer K exists such that for each set $\{Y_z \mid z = 1, \ldots, Z\}$ satisfying (6) the existence of a feasible schedule can be verified in $O(J^K)$ time. By checking each set $\{Y_z \mid z = 1, \ldots, Z\}$ satisfying (6) one can find a solution with a minimum number of machines that permits a feasible schedule for all jobs. Obviously, the number of sets $\{Y_z \mid z = 1, \ldots, Z\}$ satisfying (6) is $O(J^Z)$. Therefore SCDUC(S) can be solved in $O(J^{K+Z})$ time where both K and Z are fixed.

Next, suppose there is a subset of 3 non-dominated overlapping shifts. As all shifts in S are non-dominated, each shift has a unique begin time and a unique end time, which implies that $G(S)$ contains a triangle. Thus SCS(S) is NP-complete, according to Theorem 2. Now the NP-hardness of SCDUC(S) is established by a reduction from SCS(S).

Let I_1 be an instance of SCS(S) containing J jobs to be carried out and Z integers M_z for $z = 1, \ldots, Z$ representing the number of machines

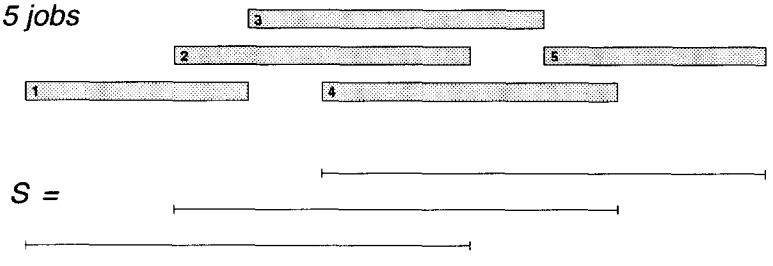

Figure 4. An instance of SCDUC(S_2)

in shift z and satisfying $M_z \leqslant J$. It may be assumed that in I_1 at any time instant the job overlap is less than or equal to the number of available machines, as otherwise I_1 is clearly a no-instance. In particular, as $M_z \leqslant J$ for $z = 1,\ldots,Z$, one can add $O(J^2)$ *short* dummy jobs to the set of jobs without changing the feasibility of I_1, in order to have the job overlap at any time instant *exactly* equal to the number of available machines. [1]

Now an instance I_2 of SCDUC(S) is created containing the same jobs as I_1, and the following statement is proved: I_1 is a yes-instance if and only if all jobs in I_2 can be carried out by $\sum_{z=1}^{Z} M_z$ machines.

If I_1 is a yes-instance, then for $z = 1,\ldots,Z$ one can choose M_z machines in shift z, in order to obtain a feasible schedule for all jobs of I_2. Then the total number of machines in I_2 equals $\sum_{z=1}^{Z} M_z$.

Conversely, suppose all jobs in I_2 can be carried out by $\sum_{z=1}^{Z} M_z$ machines. Let Y_z denote the number of machines that must be available in shift z. For $q = 1,\ldots,Q$ the job overlap in the interval (u_{q-1}, u_q) equals $\sum_{\{z \mid b_z < u_q \leqslant e_z\}} M_z$. Hence the following Q inequalities, each one corresponding to an interval (u_{q-1}, u_q), must be satisfied:

$$\sum_{\{z \mid b_z < u_q \leqslant e_z\}} Y_z \geqslant \sum_{\{z \mid b_z < u_q \leqslant e_z\}} M_z \quad \text{for } q = 1,\ldots,Q. \tag{7}$$

It is proved in Lemma 8 that the only solution to the inequalities (7) with $\sum_{z=1}^{Z} M_z$ machines is the solution $Y_z = M_z$ for $z = 1,\ldots,Z$. But then it is evident that I_1 is a yes-instance. This completes the proof of Theorem 7. □

Lemma 8. *If all shifts in S are non-dominated, then the only solution to the inequalities* (7) *and* $\sum_{z=1}^{Z} Y_z = \sum_{z=1}^{Z} M_z$ *is the solution* $Y_z = M_z$ *for* $z = 1,\ldots,Z$.

Proof. If S contains only 1 shift, then the statement is obvious. Suppose that the statement has been proved for Z shifts and let S contain $Z+1$ shifts now. Next, a partition $W_1, W_2,\ldots,W_n$ of the set S is constructed by algorithm (A.2).

Algorithm (A.2).

$k := 1$
$w_k := b_1$
Repeat
  $W_k := \{z \mid b_z \leqslant w_k < e_z\}$
  $S := S - W_k$
  $w_{k+1} := \max\{e_z \mid z \in W_k\}$
  $k := k + 1$
Until $S = \emptyset$
$n := k - 1$

By construction, the sets $W_1, W_2,\ldots,W_n$ are pairwise disjoint. Furthermore, the non-dominance of the shifts together with the connectivity of $G(S)$ implies $b_z < b_{z+1} < e_z < e_{z+1}$ for $z = 1,\ldots,Z-1$. It follows that the above procedure is finite and, as a consequence, $W_1, W_2,\ldots,W_n$ is a partition of S. The inequalities in (7) corresponding to the sets W_k are the following:

$$\sum_{z \in W_k} Y_z \geqslant \sum_{z \in W_k} M_z \quad \text{for } k = 1,\ldots,n. \tag{8}$$

By combining the inequalities corresponding to the sets W_k for $k = 2,\ldots,n$ and by noting that $W_1 = \{1\}$, one obtains

$$\sum_{z \neq 1} Y_z \geqslant \sum_{z \neq 1} M_z \quad \text{and} \quad Y_1 \geqslant M_1. \tag{9}$$

As the total number of machines equals $\sum_{z=1}^{Z+1} M_z$, one finds that $Y_1 = M_1$. But then the induction hypothesis shows that $Y_z = M_z$ for $z = 1,\ldots,Z+1$. This completes the proof of Lemma 8. □

Figure 5 illustrates the construction used in the proof of Lemma 8. Here $n = 3$ and the sets W_1, W_2 and W_3 are equal to {1}, {2,3} and {4} respectively. The inequalities in (7) corresponding to these sets are $Y_1 \geqslant M_1$, $Y_2 + Y_3 \geqslant M_2 + M_3$ and $Y_4 \geqslant M_4$.

The problems SCDUC and SCDUC(S) are integer-valued minimisation problems and for any

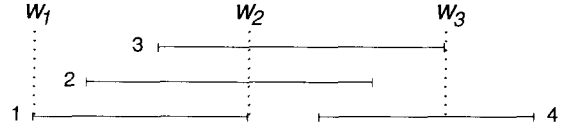

Figure 5. Example of the construction of Lemma 8

[1] Indeed, if the job overlap in the interval (t_{p-1}, t_p) is Δ less than the number of available machines in that time interval, then Δ jobs of the form (t_{p-1}, t_p) can be added to the set of jobs. In this way the feasibility of I_1 is not changed. As the number of intervals (t_{p-1}, t_p) is $O(J+Z) = O(J)$ and the number of added jobs per interval is $O(JZ) = O(J)$, the total number of added jobs is $O(J^2)$.

instance of these problems the minimum number of required machines is less than or equal to the total number of jobs. Furthermore, if these problems are NP-hard, then they are NP-hard in the strong sense. This follows from the fact that SCS and SCS(S) are NP-complete in the strong sense (Kroon and Kolen [15]). These remarks imply (Garey and Johnson [8]) that, assuming P $\neq$ NP, a fully polynomial approximation scheme for SCDUC or for an NP-hard variant of SCDUC(S) does not exist.

5. The problems SCD and SCD(S)

5.1. Preemptive scheduling

In this subsection we show that the preemptive variant of SCD can be solved in polynomial time. Note that this implies that for any set of shifts S the preemptive variant of SCD(S) can be solved in polynomial time as well. It should be noted further that the result of Lemma 3 stating that a feasible preemptive schedule can be transformed into a feasible preemptive schedule where each preemption coincides with the end time of a shift also holds for the problems SCD and SCD(S).

Lemma 9. *The preemptive variant of* SCD *can be solved in polynomial time.*

Proof. Let I be an instance of SCD containing J jobs (s_j, f_j) and Z tuples (b_z, e_z) representing the intervals of the Z shifts. We will show that an optimal solution can be obtained by finding a minimum cost circulation flow in a directed network N to be defined.

The nodeset of N is the set $\{t_p \mid p = 0, \ldots, P\}$. For each job j there is an arc from node s_j to node f_j with lower and upper capacity 1 and with cost coefficient 0. Furthermore, for each shift z there is an arc from node e_z to node b_z without capacity constraints and with cost coefficient k_z. Finally, for $p = 1, \ldots, P$ there is a dummy arc from node t_{p-1} to node t_p without capacity constraints and with cost coefficient 0. An example of the construction is given in Figure 6.

It is not difficult to see that a circulation flow in the described network can be interpreted as a feasible preemptive schedules for all jobs where each preemption coincides with the end time of a shift (Kroon [16]). As a minimum cost circulation flow in the directed network N can be found in polynomial time, the preemptive variant of SCD can be solved in polynomial time. □

5.2. Non-preemptive scheduling

In this subsection we present a classification of the computational complexity of a subset of problems SCD(S). Obviously, if the maximum shift

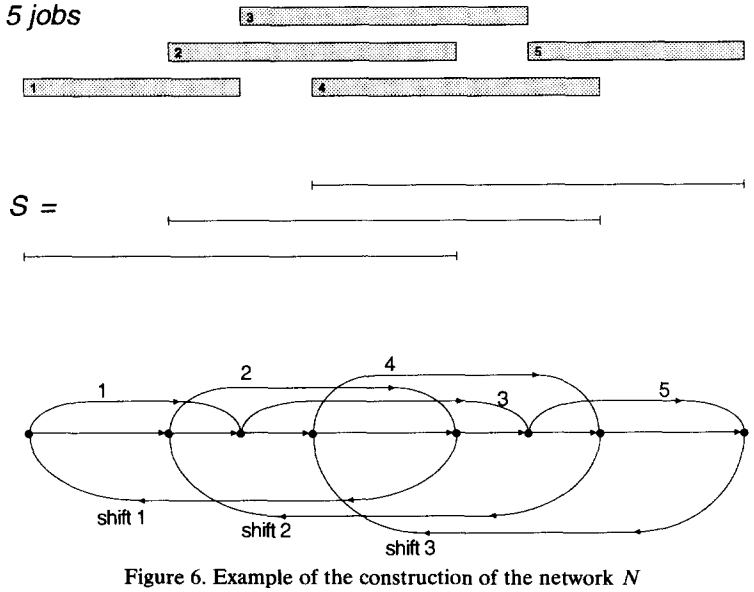

Figure 6. Example of the construction of the network N

overlap of the non-dominated shifts in the set of shifts S is greater than 2, then the graph $G(S)$ contains a triangle. In this case SCDUC(S) is NP-hard, according to Theorem 7. Hence a straightforward reduction from SCDUC(S) can be used to show that SCD(S) is NP-hard as well.

However, in the classification of the computational complexity of the problems SCD(S) we also have to consider sets of shifts S containing *dominated* shifts, as low costs for the machines in a dominated shift can make this shift still attractive. In these cases a reduction from SCDUC(S) is inappropriate, as SCDUC(S) is not necessarily NP-hard then. The results for SCD(S) are expressed in Theorem 10. These results are complete for the sets of shifts S where each shift has a unique begin and end time.

Theorem 10. *If* SCS(S) *can be solved in polynomial time, then* SCD(S) *can be solved in polynomial time. If the graph $G(S)$ contains a triangle as a node-induced subgraph, then* SCD(S) *is NP-hard.*

Proof. The first part of Theorem 10 can be proved by a similar enumeration argument as in the first part of Theorem 7. The only difference with the proof of the first part of Theorem 7 is that in comparing the solutions $\{Y_z \mid z = 1, \ldots, Z\}$ one has to consider the costs of these solutions instead of the number of machines in these solutions.

The second part of Theorem 10 is proved by a reduction from SCS(T) where T denotes the set of 3 shifts corresponding to the triangle in $G(S)$. According to Theorem 2, SCS(T) is NP-complete. Let I_1 be an instance of SCS(T) containing J jobs to be carried out and 3 integers M_z for $z = 1, 2, 3$ representing the number of machines in shift z and satisfying $M_z \leqslant J$. It may be assumed that in I_1 at any time instant the job overlap is less than or equal to the number of available machines, as otherwise a feasible schedule can not exist. In the same way as in the proof of Theorem 7, one can add $O(J^2)$ *short* dummy jobs to the set of jobs without changing the feasibility of I_1, in order to have the job overlap at any time instant *exactly* equal to the number of available machines.

Next, an instance I_2 of SCD(S) is created as follows. In I_2 the same jobs must be carried out as in I_1 and for $z = 1, 2, 3$ the costs of each machine in shift z are equal to $e_z - b_z$. Next the following statement is proved: I_1 is a yes-instance if and only if the minimum total costs for hiring machines in I_2 are equal to $\sum_{z=1}^{3} M_z(e_z - b_z)$.

If I_1 is a yes-instance, then in I_2 one can choose M_z machines in shift z for $z = 1, 2, 3$. This solution has total costs $\sum_{z=1}^{3} M_z(e_z - b_z)$ and is clearly optimal, because the total workload equals $\sum_{z=1}^{3} M_z(e_z - b_z)$ and the machines are paid per time unit.

Conversely, if the minimum total costs for hiring machines in I_2 are equal to $\sum_{z=1}^{3} M_z(e_z - b_z)$, then all machines are uninterruptedly busy, because the machines are paid per time unit and the total workload equals $\sum_{z=1}^{3} M_z(e_z - b_z)$. Let Y_z denote the number of machines in shift z in the optimal solution for I_2. For $q = 1, \ldots, Q$ the job overlap in the interval (u_{q-1}, u_q) equals $\sum_{\{z \mid b_z < u_q \leqslant e_z\}} M_z$. Hence the following Q equations, each one corresponding to an interval (u_{q-1}, u_q), must be satisfied. Note that in this situation with $Z = 3$ where all shifts have different begin and end times we actually have $Q = 5$.

$$\sum_{\{z \mid b_z < u_q \leqslant e_z\}} Y_z = \sum_{\{z \mid b_z < u_q \leqslant e_z\}} M_z \quad \text{for } q = 1, \ldots, Q. \tag{10}$$

Using the fact that the three begin times of the shifts are different and that the three end times of the shifts are different, it is not difficult to see that the only solution satisfying these constraints is the solution $Y_z = M_z$ for $z = 1, 2, 3$. But then it is evident that I_1 is a yes-instance. As a consequence, SCD(T) is NP-hard. Now the proof of Theorem 10 is completed by noting that $T \subset S$ implies that SCD(S) is at least as difficult as SCD(T). □

If the graph $G(S)$ is bipartite, then SCS(S) can be solved in polynomial time, according to Theorem 2. If this result is combined with the first part of Theorem 10, then we find that SCD(S) can be solved in polynomial time if the graph $G(S)$ is bipartite.

According to this first part of Theorem 10, SCD(S) can be solved in polynomial time as soon as SCS(S) can be solved in polynomial time. At this moment we do not know yet if it is generally true that SCD(S) is NP-hard as soon as SCS (S) is NP-complete. Nevertheless, there is a remark-

able overlap in the complexity results for SCS(S) and SCD(S).

Theorem 10 together with the above remarks imply that the graph $G(S)$ is an important indicator of the computational complexity of the problems SCS(S) and SCD(S). Unfortunately, if $G(S)$ contains an odd hole of at least 5 nodes as a node-induced subgraph, then Theorems 2 and 10 cannot be applied. The set of shifts S_3 that was shown in Figure 1 demonstrates that $G(S)$ can be an odd hole of size 5. However, in the Lemmas 11, 12 and 13 it is proved that this is the worst thing that can happen. That is, it is shown that $G(S)$ cannot be a hole of 7 or more nodes, which also implies that $G(S)$ cannot contain a hole of 7 or more nodes as a node-induced subgraph.

In the following the graph $\tilde{G}(S)$ is the usual interval graph corresponding to the set of shifts S. Note that $\tilde{G}(S)$ is triangulated, which means that in $\tilde{G}(S)$ each cycle of 4 or more nodes contains at least one chord (Golumbic [10]).

It is obvious that the set of edges of $G(S)$ is a subset of the set of edges of $\tilde{G}(S)$. The edges of $G(S)$ are called *strong* edges, or *s-edges* for short. In the figures the s-edges are represented by curved solid lines. An edge of $\tilde{G}(S)$ that is missing in $G(S)$ corresponds to a pair of shifts with the same begin time or to a pair of shifts with the same end time. Such edges are called *b-edges* and *e-edges* respectively. In the figures the b-edges and the e-edges are represented by dashed and dotted lines. In this section all mentioned nodes and edges are nodes and edges of $\tilde{G}(S)$. In the following the result of Lemma 11 is used several times. The proof of this lemma is so obvious that we omit it.

Lemma 11. *A triangle cannot contain exactly two b-edges. A triangle cannot contain exactly two e-edges. If $\tilde{G}(S)$ does not contain a triangle of b-edges nor a triangle of e-edges, then each triangle in $\tilde{G}(S)$ contains at least one s-edge.*

Next we show that $G(S)$ cannot be a hole of more than 6 nodes. That is, Lemma 12 shows that if $G(S)$ is a hole of at least 6 nodes, then $\tilde{G}(S)$ contains a triangle of b-edges or a triangle of e-edges. Thereafter, Lemma 13 proves that if $\tilde{G}(S)$ is a hole containing a triangle of b-edges or a triangle of e-edges, then the set of shifts S contains exactly 6 shifts. By combining these results it follows that $G(S)$ cannot be a hole of more than 6 nodes. It also follows from the proof of these lemmas that there is essentially one set of shifts S such that $G(S)$ is a hole of 6 nodes.

Lemma 12. *If $G(S)$ is a hole and $Z \geqslant 6$, then $\tilde{G}(S)$ contains a triangle of b-edges or a triangle of e-edges.*

Proof. This lemma is proved by contradiction. To this end, assume $G(S)$ is a hole, $Z \geqslant 6$, and $\tilde{G}(S)$ does not contain a triangle of b-edges nor a triangle of e-edges.

As $\tilde{G}(S)$ is triangulated, nodes 1, 2 and 3 exist such that (1, 2) and (1, 3) are s-edges and (2, 3) is not an s-edge. Without loss of generality, (2, 3) is an e-edge (otherwise the time-axis is reversed) and $b_2 < b_3$ (otherwise the shifts are renumbered). Figure 7 shows a sample situation with $Z = 10$.

As $\tilde{G}(S)$ is triangulated, a node 4 different from node 1 exists such that (2, 3, 4) is a triangle.

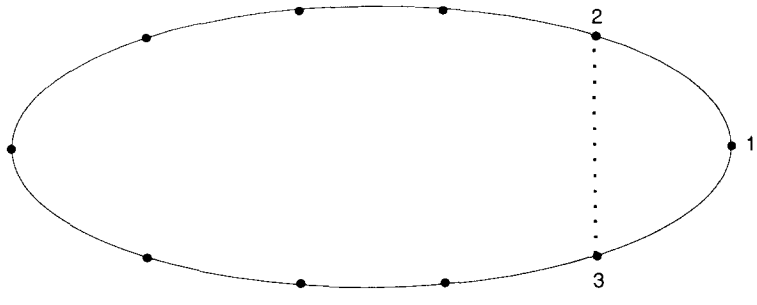

Figure 7. Sample situation with $Z = 10$

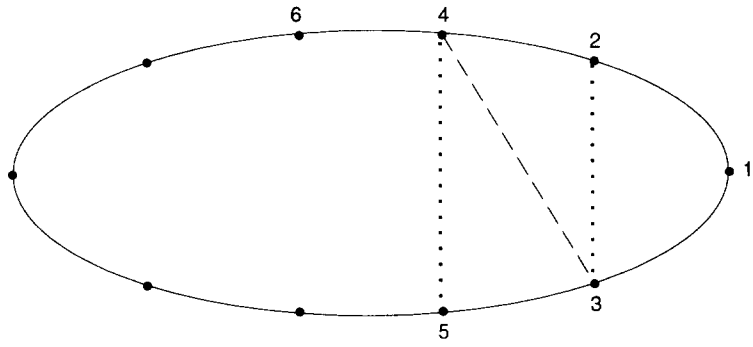

Figure 8. Sample situation with $Z = 10$ (continued)

As by assumption $\tilde{G}(S)$ does not contain a triangle of b-edges nor a triangle of e-edges, one of the edges (2, 4) or (3, 4) is an s-edge (Lemma 11). We assume (2, 4) is an s-edge (the other case is similar). Now (3, 4) is a b-edge (Lemma 11).

Analogously, it follows that a node 5 different from the nodes 1 and 2 exists such that (3, 4, 5) is a triangle and such that (3, 5) or (4, 5) is an s-edge. However, if (4, 5) is an s-edge, then (3, 5) is an e-edge (Lemma 11) and thus (2, 3, 5) is a triangle of e-edges. This contradiction implies that (3, 5) is an s-edge and that (4, 5) is an e-edge (Lemma 11). This situation is represented in Figure 8.

As shifts 3 and 5 are overlapping and shift 3 is a subinterval of shift 2, shifts 2 and 5 are overlapping as well. As $e_2 = e_3 \neq e_5$ and nodes 2 and 5 are not connected by an s-edge, nodes 2 and 5 are connected by a b-edge. Now the cases $e_3 < e_4$ and $e_3 > e_4$ are distinguished. These cases are represented in Figure 9.

First we consider the case $e_3 < e_4$. Shifts 1 and 3 are overlapping and shift 3 is a subinterval of shifts 4 and 5. Thus shifts 1, 4 and 5 are also overlapping, which implies that (1, 4, 5) is a triangle. As (1, 4) and (1, 5) are non s-edges and (4, 5) is an e-edge, (1, 4, 5) is a triangle of e-edges (Lemma 11), which contradicts our assumption.

Next we consider the case $e_3 > e_4$. Let node 6 be such that nodes 4 and 6 are connected by an s-edge. Shifts 4 and 6 are overlapping and shift 4 is a subinterval of shifts 2 and 3. Thus shifts 2, 3 and 6 are also overlapping, which implies that (2, 3, 6) is a triangle. As (2, 6) and (3, 6) are non s-edges and (2, 3) is an e-edge, (2, 3, 6) is a triangle of e-edges (Lemma 11), which again contradicts our assumption.

Thus the assumption that $\tilde{G}(S)$ does not contain a triangle of b-edges nor a triangle of e-edges always leads to a contradiction. This completes the proof of Lemma 12. □

Lemma 13. *If $G(S)$ is a hole and $\tilde{G}(S)$ contains a triangle of b-edges or a triangle of e-edges, then $Z = 6$.*

Proof. Suppose $G(S)$ is a hole and $\tilde{G}(S)$ contains a triangle of b-edges (the other case is similar). Obviously, this implies $Z \geqslant 6$. Now we will prove that $Z > 6$ cannot occur.

As $\tilde{G}(S)$ contains a triangle of b-edges, there exist shifts 1, 2 and 3 with $b_1 = b_2 = b_3$. Without loss of generality $e_1 < e_2 < e_3$. The nodes 1, 2 and 3 are not connected by s-edges. Hence two nodes 4 and 5 exist such that (1, 4) and (1, 5) are s-edges. Furthermore, a node 6 exists *between* the nodes 2 and 3 such that (2, 6) is an s-edge. Figure 10 shows a sample situation with $Z = 9$.

As shifts 1 and 4 are overlapping and shift 1 is a subinterval of shift 3, shifts 3 and 4 are overlapping as well. Furthermore, $b_4 \neq b_1 = b_3$ and nodes

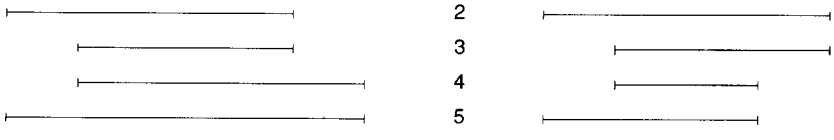

Figure 9. Cases with $e_3 < e_4$ and with $e_3 > e_4$

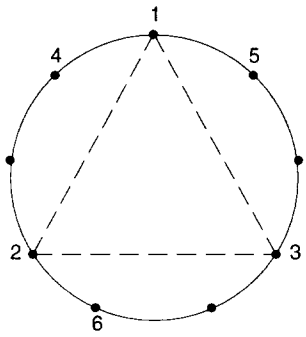

Figure 10. Sample situation with $Z = 9$

3 and 4 are not connected by an s-edge. Therefore nodes 3 and 4 are connected by an e-edge. In a similar way it follows that nodes 2 and 5 are connected by an e-edge.

If $\delta > 0$ is chosen small enough, then shifts 1 and 4 are overlapping in the interval $(e_1 - \delta, e_1)$ and the same holds for the shifts 1 and 5. It follows that the shifts 4 and 5 are overlapping in the interval $(e_1 - \delta, e_1)$. As $e_4 = e_3 > e_2 = e_5$ and nodes 4 and 5 are not connected by an s-edge, it follows that nodes 4 and 5 are connected by a b-edge. Both cases with $b_1 < b_4$ and with $b_1 > b_4$ are represented in Figure 11.

Figure 11 shows that shifts 2 and 4 are overlapping and have different begin and end times. Thus nodes 2 and 4 are directly connected by an s-edge. A similar results also holds for nodes 3 and 5.

As shifts 2 and 6 are overlapping and shift 2 is a subinterval of shift 3, shifts 3 and 6 are overlapping as well. Suppose nodes 3 and 6 are not directly connected by an s-edge. Then nodes 3 and 6 are connected by an e-edge (Lemma 11). As $e_5 = e_2 < e_3 = e_6$ and shifts 2 and 6 are overlapping, shifts 5 and 6 are overlapping and have different end times. Furthermore, nodes 5 and 6 are not connected by an s-edge. This implies that nodes 5 and 6 are connected by a b-edge. Now we have $b_6 = b_5 = b_4$ and $e_6 = e_3 = e_4$, which means that shifts 4 and 6 are identical. As this is not allowed, nodes 3 and 6 are directly connected by an s-edge. So the nodes without a number in Figure 10 do not exist, which implies $Z = 6$. □

6. Final remarks

In this paper we have analysed the problems SCD(S) and SCDUC(S) as well as their generalizations SCD and SCDUC. We have considered both the preemptive and the non-preemptive variant of these problems.

A network flow algorithm can be used for solving the preemptive variant of SCD. A polynomial algorithm for the preemptive variant of SCDUC is based on the determination of the maximum job overlap in the time intervals (b_z, b_{z+1}). We have also analysed the worst case behaviour of the preemptive variant versus the non-preemptive variant of the problems SCDUC and SCDUC(S).

Furthermore, we have derived a complete classification of the computational complexity of the problems SCDUC(S) and a classification of the computational complexity of a large subset of the problems SCD(S). The graph $G(S)$ turns out to be an important indicator of the computational complexity of the problems SCD(S).

At this moment we are involved in an experimental study to determine the average behaviour of the preemptive variant versus the non-preemptive variant of the problems SCDUC and SCDUC(S). An alternative bound that is explored at this moment is the bound obtained by solving the linear programming relaxation of a straightforward integer programming formulation. Some preliminary results at this point reveal that in practice these relaxations provide excellent bounds.

Furthermore, we are trying to extend the worst case analysis for the problems SCDUC and SCDUC(S) to the problems SCD and SCD(S). Another problem still to be solved is to complete the

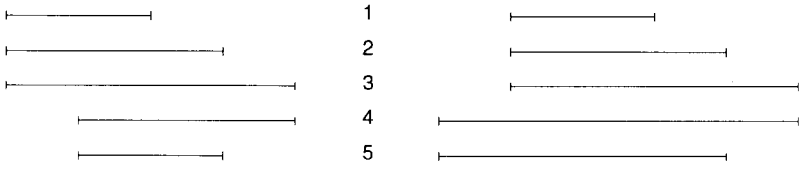

Figure 11. Cases with $b_1 < b_4$ and with $b_1 > b_4$

classification of the computational complexity of the non-preemptive variant of the problems SCD(S).

References

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

[2] Carter, M.W., and Tovey, C.A., "When is the classroom assignment problem hard?", *Operations Research* 40 (1992) S28–S39.

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

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

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

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

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

[8] Garey, M.R., and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-completeness,* Freeman, San Fransisco, CA, 1979).

[9] Gertsbakh, I., and Stern, H.I., "Minimal resources for fixed and variable job schedules", *Operations Research* 18 (1978) 68–85.

[10] Golumbic, M.C., *Algorithmic Graph theory and Perfect Graphs,* Academic Press, New York, 1980.

[11] Gupta, U.L., Lee, D.T., and Leung, J.Y.-T., "An optimal solution the channel assignment problem", IEEE *Transactions on Computers* 28 (1979) 807–810.

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

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

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

[15] Kolen, A.W.J., and Kroon, L.G., "On the computational complexity of (maximum) shift class scheduling", *European Journal of Operational Research* 64 (1993) 138–151.

[16] Kroon, L.G., "Job scheduling and capacity planning in aircraft maintenance", Ph.D. Thesis, Erasmus University Rotterdam, 1990.