Interval Scheduling on Identical Machines

Authors:

Khalid I. Bouzina. Senior Analyst, Risk Management Solutions Inc., 149 Commonwealth Drive, Menlo Park, California 94025.

Hamilton Emmons. Professor and Chairman, Department of Operations Research, Case Western Reserve University, Cleveland, Ohio 44106-7235.

Date: January, 1995.
First Revision Date: September 20, 1995

Abstract: Interval Scheduling problems (IS) address the situation where jobs with fixed start and fixed end times are to be processed on parallel identical machines. The optimization criteria of interest are the maximization of the number of jobs completed and, in case weights are associated with jobs, the subset of jobs with maximal total weight. We present polynomial solutions to several IS problems and study computational complexity issues in the situation where bounds are imposed on the total operating time of the machines. With this constraint, we show that tractability is achieved again when job preemption is allowed.

Key Words: Sequencing/Scheduling, Interval Scheduling, Parallel Processing, Combinatorial Optimization, Computational Complexity.

Journal of Global Optimization 9: 379–393, 1996.

Interval Scheduling on Identical Machines

Interval Scheduling (IS) is a class of scheduling problems where independent tasks, each with a fixed start time and fixed end time, are to be processed in a parallel machine environment. In this study, it is further assumed that all machines are identical. Therefore, with no constraint on machine availability, it is easy to see that the maximal job overlap over time equals the minimal number of machines required to complete all jobs. However, an interesting combinatorial issue rises when, given the total number of machines available, not all jobs can be processed. In this situation, two problems require attention. First, the finding of the maximal number of completed jobs (Maximal IS). Then, in case a weight representing the task priority or value is associated with each job, the finding of a subset of jobs with maximal total weight (Maximal Weight IS). The aim of this paper is to address such problems under different scenarios.

IS problems have received much attention during the last decade or so and may be encountered under different names in the literature, such as "Fixed Job Scheduling" (Arkin & Silverberg 1987, Fishetti et al. 1989 and Dondeti & Emmons 1992), "The Classroom Assignment Problem" (Carter and Tovey 1992), "The Bus Driver Scheduling Problem" (Martello and Toth 1986), "Class Scheduling" (Kolen and Kroon 1991). However, we prefer to refer to them as IS because of the analogy to an even broader class of problems addressed in Graph Theory, namely, Interval Graph problems (Golumbic 1980 and Kolen et al. 1986).

In this paper, we first present an algorithm solving Maximal IS. Next, a Minimal Cost Flow formulation is presented for Maximal Weight IS and is extended to solve Maximal IS with minimal total processing time. Thereafter, we introduce a constraint on the working time of the machines, namely that no machine operates for more than T units of time. This problem is referred to as TIS. We provide an algorithm

solving Maximal Preemptive TIS (that is, when job preemption is allowed) and show that Maximal Weight Preemptive TIS is NP-Hard. Integer programming formulations are also presented.

1. An algorithm for Maximal IS

Given n jobs with fixed start times s_j and fixed end times e_j, j=1,...,n, we seek the largest subset, S, of jobs that can be processed on m parallel identical machines. The following algorithm finds a solution to Maximal IS in O(n max(log n, m)) time.

Algorithm 1:

1.1 Set S to the empty set.

1.2. Sort jobs in chronological order of arrival.

1.3. Consider jobs sequentially. At each start time add the arriving job to S. If there is no machine to accommodate it, remove from S the job with latest ending time.

It is easy to see that Algorithm 1 produces a feasible schedule with the desired complexity. To establish optimality, we first state the following lemma without proof.

Lemma 1: Given a set of jobs to be scheduled, suppose one job is shortened (its start time is unchanged, but its end time is earlier), with all else unchanged. Then the maximal number of jobs that can be scheduled is either unchanged or increased.

Without loss of generality, we can assume all jobs have distinct start times. Thus, in case i >1 jobs have the same arrival time s, assign new start times s, s+ε, s+2ε,..., s + (i-1)ε for infinitesimal ε > 0, in any order. We now present the proposition on which the algorithm depends.

Proposition 1: Let t_0 = earliest time at which more than m jobs require processing, J = {jobs requiring processing at t_0} and $e_k = \max_{j \in J} e_j$.

Then there exists an optimal schedule which does not include k.

Proof: Clearly, one of the (m+1) jobs in J cannot be done. The choice depends only on the future ($t > t_0$), since all jobs started earlier must have been successfully completed or be in J, and the amount of processing already undergone at time t_0 by jobs in J is irrelevant for resolving the present and future conflicts. Thus, redefine the start times of all $j \in J$ to be t_0. Now, suppose we remove some job i rather than job k. By switching them (remove k rather than i), we are replacing a longer job with a shorter, or in effect shortening a job. By Lemma 1, this can only increase the number of jobs we can eventually schedule. ❐

2. Minimal Cost Flow Formulation of Maximal Weight IS

Given an instance of IS, assume that a weight w_j is associated with each job j, j=1,...,n and let us now address the problem of finding a subset of jobs S such that S is feasible and the sum of the weights of the jobs in S is maximum. In Arkin and Silverberg (1987) a solution to this problem based on a minimal cost flow formulation is given. Their proposed algorithm requires a preliminary construction of the interval graph corresponding to the IS instance and the identification of all maximal cliques in such graph (see Golumbic 1980). We will present an algorithm for the problem that is also based on the solving of a minimal cost flow problem. However, its related network construction is obtained directly from the IS instance and requires exactly n+1 nodes and 2n arcs.

Algorithm 2:

2.1 Index jobs in chronological order of arrival.

2.2 Create nodes $s = v_1, v_2,...,v_n$ *for each job and dummy node* $t = v_{n+1}$. *Connect* v_j *to* v_{j+1} *with arc of cost zero and capacity m, j=1,...,n.*

2.3 Create arc (v_j, v_k) *where k is the first job not overlapping with job j, j=1,...n. If no such job k exists, create arc* (v_j, t). *Each of these arc has cost* $-w_j$ *and capacity 1.*

2.4 Require a flow of m from s to t and solve the minimal cost flow problem.

Proposition 2: The optimal solution to the Minimal Cost Flow problem is also an optimal solution to Maximal Weight IS.

Proof: Let V be the subset of jobs corresponding to arcs with cost $-w_j$ and a flow of one unit on them. First, let us show that V is feasible for Maximal Weight IS. Note that having a flow of one unit on such arcs is equivalent to assigning a machine to the corresponding job. Also, since the node to which the flow is surrendered corresponds to the first job that does not overlap with the currently processed job, at no time do we have a machine assigned to more than one job. Furthermore, since only a total of m units is allowed through the network, at most m machines at all times will be assigned to jobs. Hence, V is feasible for Maximal Weight IS. Now, an optimal solution to Maximal Weight IS is equivalent to an optimal solution to the minimal cost flow problem when we note that job j is processed if and only if the corresponding arc (v_j, v_k) has a non-zero flow on it. Therefore, the absolute value of the minimal cost solution is the maximal total value of the jobs processed for Maximal Weight IS. ❐

Example: Consider the jobs in Figure 1.a, to be scheduled on 2 machines.

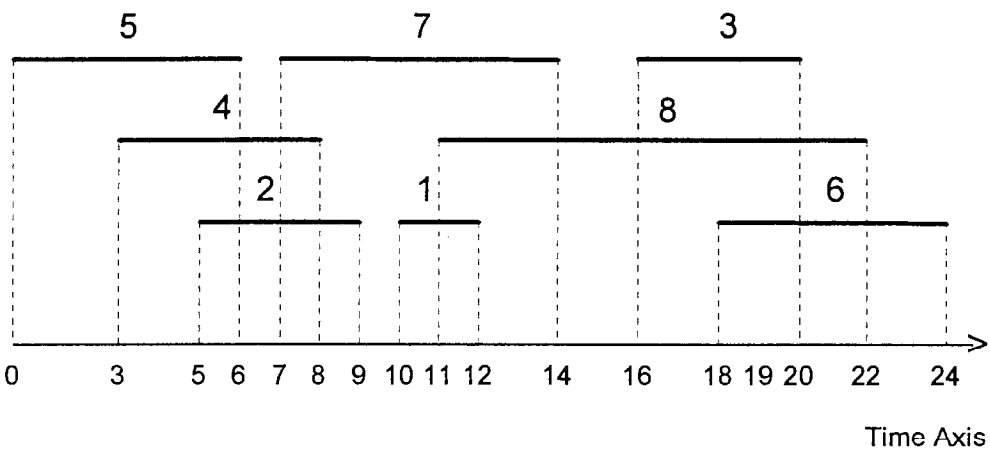

Figure 1.a: Instance of IS Problem.

For Maximal IS, Algorithm 1 shows that 6 jobs can be processed. It selects job 2 and 8 for omission, though other pairs of jobs are possible. For Maximal Weight IS, the network structure for Algorithm 2 is given in Figure 1.b, where each arc is labeled with its cost, $-w_j$, and capacity, 1, except for the arcs generated in Step 2 (those on the straight line from s to t) whose costs and capacities are 0 and 2, respectively. If, for example, we charge by the hour and wish to find the subset of jobs that will keep our 2 machines busiest, then with $w_j = p_j$, $j = 1,...,8$, the minimal cost flow is -5 and utilizes the arcs corresponding to the 5 jobs $\{4,5,6,7,8\}$.

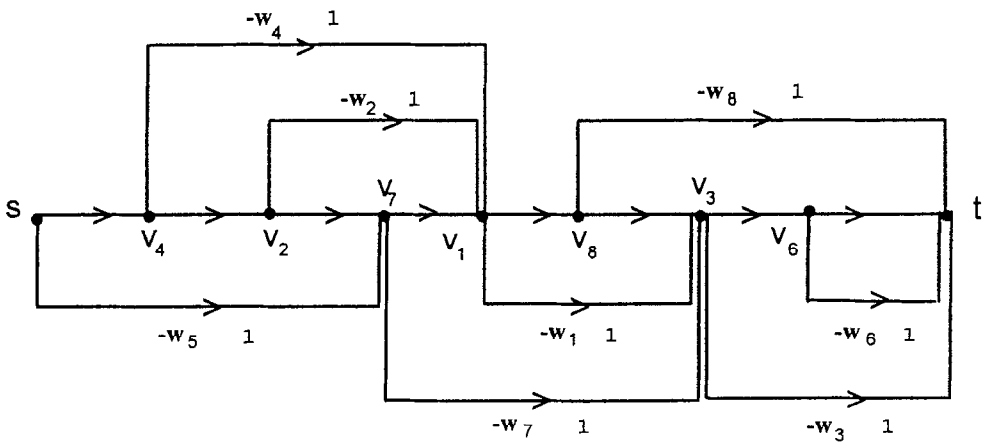

Figure 1.b: Network Representation

Let us now reexamine the Maximal IS problem and address the dual criterion of determining a subset S of maximal cardinality with the property that the sum of the weights of the jobs in S is maximal.

Corollary 1: Let M is a positive real number such that $M \geq \sum_{j=1}^{n} w_j$, and redefine w_j as $w_j + M$ for each job j. Then Algorithm 2 gives an optimal solution to Maximal IS with Maximal Total Weight.

Proof: Since M is so large, maximization of total weights will first maximize the number of jobs scheduled, since each contributes M to the objective. Second, it will maximize the weights added. □

The following special case of Corollary 1, where jobs weights are not initially given, will be needed later in this paper.

Corollary 2: For each job j with processing time $p_j = e_j - s_j$, define a weight $w_j = M - p_j$, where M is a positive real number such that $M \geq \sum_{j=1}^{n} p_j$. Then Algorithm 2 gives an optimal solution to Maximal IS with Minimal Total Processing Time

3. Model and Complexity of Maximal TIS

Let us add the constraint that no machine may work, in total, for more than a given operating time T and consider an instance of TIS. Notice that the decision version of TIS, that is, the problem of determining whether there is a feasible schedule for all jobs is NP-Complete (Fischetti et al.1989). The authors proved completeness through transformation from the Bin Packing problem (Garey and Johnson 1979) by showing that solving an instance of TIS where no two jobs overlap will solve the instance of the Bin Packing problem as well. Consequently, Maximal TIS is an NP-Hard problem. Maximal TIS and Maximal Weight TIS can be stated as a 0-1 Integer programming problem as follows. Let $\{t_1, t_2, \ldots\}$ be the sorted sequence of the s_j's and e_j's in chronological order with duplicates removed and let P_k be the set of jobs available in interval $[t_k, t_{k+1})$, $k = 1, 2, \ldots$ Finally, create binary variables x_{ij} taking the value one if and only if machine i processes job j for i=1,...,m and j=1,...,n.

$$\text{Maximize} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} w_j x_{ij}$$

Subject to :

$$\sum_{i=1}^{m} x_{ij} \leq 1 \qquad j = 1,...,n \qquad (1)$$

$$\sum_{j \in P_k} x_{ij} \leq 1 \qquad i = 1,...,m;\ k = 1, 2,... \qquad (2)$$

$$\sum_{j=1}^{n} p_j x_{ij} \leq T \qquad i = 1,...,m \qquad (3)$$

$$x_{ij} \in \{0,1\} \qquad i = 1,...,m;\ j = 1,...,n \qquad (4)$$

Constraints (1) require that each job j be processed on only one machine. Constraints (2) state that each machine cannot process more than one job at a time. Constraints (3) prohibit any machine from having a total operating time exceeding T.

4. An algorithm for Maximal Preemptive TIS

Given an instance of TIS, let us introduce the relaxation that jobs may be split and assigned to two or more machines. Then, this preemptive version of TIS becomes tractable when we seek a feasible set of jobs S with maximal cardinality. The following algorithm for Maximal Preemptive TIS provides only the set S of jobs of maximal cardinality which satisfies $\sum_{i \in S} p_i \leq mT$ and has no more than m jobs in process at any time. The actual (preemptive) schedule of the jobs is constructed by keeping the machines equally loaded over time. The start and end times of jobs may be used as increments in this procedure.

Algorithm 3:

3.1 Index jobs in increasing order of their processing time $p_j = e_j - s_j$, $j = 1, ...,n$. *Break ties arbitrarily.*

3.2 Initialize: $S = \varnothing$, $j = 0$.

3.3 Repeat

$j = j + 1$.

If $m+1$ *jobs in* $S \cup \{j\}$ *are conflicting at some time point,*

apply ***Algorithm 1*** *to the jobs in* $\{1,...,j\}$.

Let S' *be the resulting schedule.*

If $|S'| = |S| + 1$,

apply ***Corollary 2 of Algorithm 2*** *to the jobs in* $\{1,...,j\}$.

Let S^* *be the resulting schedule.*

If $\sum_{i \in S^*} p_i \leq mT$, *make* $S = S^*$.

Endif

Endif

Else make $S = S \cup \{j\}$.

Endif

Until $j = n$ ***or*** $p_{j+1} + \sum_{i \in S} p_i > mT$.

Feasibility is insured in step 3.3 of the algorithm . Complexity is dictated by step 3.3, when Algorithm 2 is used, in which case a minimal cost flow is sought for in a network having j+1 nodes and 2j arcs, j = 1,...,n. Optimality will be shown in Proposition 3, but first, two related lemmas are stated (the proof of Lemma 2 is trivial and is omitted).

Lemma 2: Consider an instance of TIS where at no time are there more than m overlapping jobs, and assume the jobs ordered in increasing order of their processing times. An optimal schedule for Maximal Preemptive TIS for such an instance includes the first k jobs for which $k = \min\{j: \sum_{i=1}^{j+1} p_i > mT\}$.

Lemma 3: Consider an instance of TIS where the jobs are ordered in increasing order of their processing times and assume that, at each iteration of Algorithm 3, a feasible set S with maximal cardinality and minimal sum of the processing

times is maintained. If at some iteration j, the total processing time of the jobs in $S \cup \{j\}$ exceeds mT, then the current schedule S is optimal.

Proof: Given the ordered sequence of jobs, consider all jobs in $J = \{1,...,j-1\}$. We have that, for any feasible set S' in J, $|S'| \leq |S|$ and if $|S'| = |S|$ then $\sum_{i \in S'} p_i \geq \sum_{i \in S} p_i$. Also, from Lemma 2, it follows that replacing any job in S by a job not in J, will only result in a schedule of bigger sum of the processing times. Now, consider adding a job j' such that $p_{j'} \geq p_j$ to the list S of scheduled jobs. Whether j' conflicts or not with any of the jobs in S, we still have $\sum_{i \in S' \cup \{j\}} p_i > mT$ for a feasible set S' in J that can accommodate j'. Hence, if S is as described in Lemma 3, it is optimal for Maximal Preemptive TIS. □

Proposition 3: The schedule S produced by Algorithm 3 is optimal for Maximal Preemptive TIS.

Proof: It follows from Lemma 3 that, at this stage, we need only show that at each iteration the schedule S produced by Algorithm 3 has maximal cardinality (property 1) and the sum of the processing times of the jobs in S is minimal (property 2). Assume the result true up to the (j-1)th iteration and consider adding the jth job to S from the ordered list. Three cases need to be examined. If the addition of job j does not create m+1 conflicting jobs at any time point then, naturally properties 1 and 2 are maintained for the new schedule $S \cup \{j\}$. Now, if a conflict does occur, but the addition of job j does not produce a schedule with higher cardinality, then it follows from Proposition 1 and from the fact that the processing time of job j is greater than or equal to any of the jobs scanned so far, that the current schedule S should be kept and consequently, no property is violated. Finally, suppose that the adjunction of job j creates a conflict but there is at least a set S' in $\{1,...,j\}$ such that $|S'| = |S|+1$. Then the schedule S* provided, when using Algorithm 2 for the jobs in $\{1,...,j\}$ satisfies properties 1 and 2 as shown in Corollary 1. Therefore, if the sum of the processing times of the jobs in S*

does not exceed mT, S* is optimal up to iteration j. If $\sum_{i \in S^*} p_i > mT$, S* cannot be accepted and neither will be any other schedule in {1,...,j} with the same cardinality as S* for the obvious reason that the sum of the processing times of the jobs in S* is already minimal.

Hence, the current set S remains optimal for the jth iteration. However, this does not constitute a stopping criterion for Algorithm 3. Indeed, the set S* has been created in respect to the adjunction of job j. For any other job k, k>j, it is possible for the algorithm to produce (at a later iteration) a set S* such that $\sum_{i \in S^* \cup \{k\}} p_i \leq mT$.

Considering that all outcomes have been examined, the subset of jobs S constructed by Algorithm 3 is optimal for Maximal Preemptive TIS. ❐

Example: Consider the IS instance depicted in Figure 1.a, with m =2 and T = 15 hours. Applying Algorithm 3, the jobs are already indexed in increasing order of their processing times, and the first four jobs may be added to S without conflict (more than 2 simultaneous jobs) or overload (over 30 hours total). The following audit shows the algorithm's performance for the remaining jobs.

S	j	$\Sigma_{S \cup \{j\}} > mT$?	Conflict?	S'	$\lvert S' \rvert > \lvert S \rvert$?	S*	$\Sigma_{S^*} \leq mT$?
1,2,3,4	5	No	Yes	1,3,4,5	No	—	—
1,2,3,4	6	No	No	—	—	—	—
1,2,3,4,6	7	No	Yes	1,3,4,5,6,7	Yes	1,2,3,5,6,7	Yes
1,2,3,5,6,7	8	Yes	—	—	—	—	—

We end with the optimal subset S = {1,2,3,5,6,7}. Note how job 5, which is rejected at first (because a smaller job can be chosen instead), is later brought back (because it has less overlap with other jobs); an unusual feature in a single-pass optimization algorithm. The actual schedule of the jobs in S, if we keep the machines equally loaded, can be constructed as follows. Let $\{t_1, t_2, ...\}$ be the sorted sequence of the start and end times of the jobs in S in chronological order. Consider each interval

$[t_i, t_{i+1})$, i = 1, 2, ... Generally, if there are m machines and k ($k \leq m$) jobs are in process during $[t_i, t_{i+1})$, partition the interval into m equal parts, and assign each machine in rotation to one part of each job. Here, with m = 2, if two jobs are to be processed in a given time interval, assign a machine to each job. If only one job is present, assign each machine to half of the interval. The preemptive schedule of the jobs in S on the two machines is depicted in Figure 2, with machines identified by roman numerals. Note that, although this procedure is needed to guarantee a feasible schedule in a single pass, schedules with far fewer preemptions are usually available with a little trial-and-error.

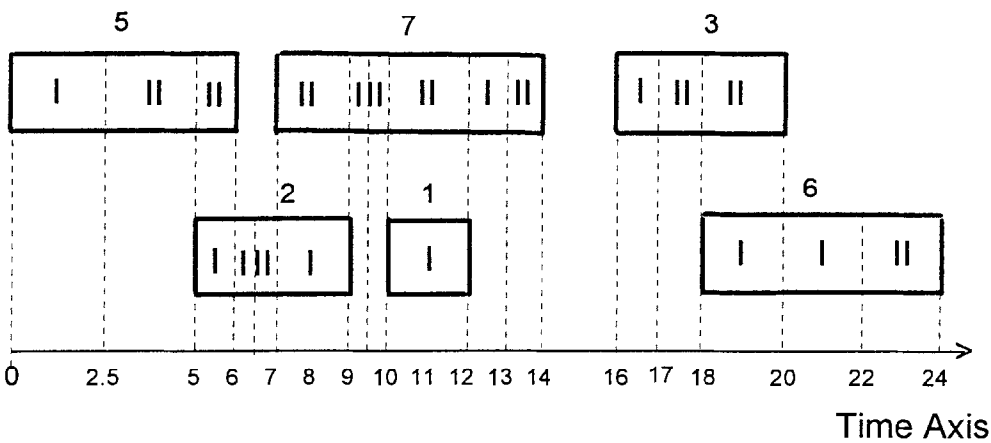

Figure 2: Preemptive Job Schedule for TIS.

5. Complexity of Maximal Weight Preemptive TIS

Consider the problem addressed in Section 4, but assume now that a weight w_j is associated with each job j, j=1,...,n and the goal is to provide a subset of jobs with maximal total weight. We now show that this problem is intractable.

Proposition 4: Maximal Weight Preemptive TIS is NP-Hard

Proof: Without loss of generality, assume that all the numerical data are positive real integers. Also, for proof purposes, replace the problem addressed in Maximal Weight Preemptive TIS by the question of whether there exists a feasible subset of jobs S such that $\sum_{i \in S} w_i \geq w$, where w is a positive integer. To show completeness, we reduce the Knapsack problem to Maximal Weight Preemptive TIS. In an instance of Knapsack (Garey and Johnson 1979), we are given a set $\{1,...,p\}$ and with each one of its elements j is associated two positive integers z_j and v_j reflecting respectively the size and the value of j, j=1,...,p. Also, let Z and V be two positive integers. We are asked to find a subset $U \subseteq \{1,...,p\}$ such that $\sum_{i \in U} z_i \leq Z$ and $\sum_{i \in U} v_i \geq V$. Given such an instance of Knapsack, we construct the following instance for Maximal Weight Preemptive TIS.

$n = p;$

$s_j = 0,\ e_j = z_j,\ w_j = v_j,\ j=1,...,n;$

$m = p;$

$T = Z/p;$

$w = V.$

Claim: There exist a subset U in $\{1,...,p\}$ such that $\sum_{i \in U} z_i \leq Z$ and $\sum_{i \in U} v_i \geq V$ if and only if there exist a subset of jobs S such that all jobs in S can be scheduled preemptively on m machines, no machine working more than T units of time and such that $\sum_{i \in S} w_i \geq w$ and $\sum_{i \in S} p_i \leq mT$.

Proof: Suppose that a subset U that solves the knapsack problem exists and let S = U. It is easy to see that both inequalities $\sum_{i \in S} p_i \leq mT$ and $\sum_{i \in S} w_i \geq w$ are satisfied. Now, a preemptive assignment of the jobs in S to machines satisfying the conditions that at no time are there more than m jobs in process and no machine operates more than T units of time is the following. Consider the units time intervals [0,1), [1,2), etc. Starting

from 0 and moving one unit at a time, consider all jobs in S and assign all m = n machines to each unit interval one by one in rotation. With n machines being allocated and all of them being used an equal amount of time, the schedule S is feasible for Maximal Weight Preemptive TIS. Reciprocally, it is trivial that if a solution exists for Maximal Weight Preemptive TIS, the same solution is optimal for the Knapsack problem. □

6. Concluding Remarks

Our contribution to the field of IS problems on identical machines through this paper has been twofold. We first added to the work of Arkin and Silverberg (1987) and provided a "greedy" type of algorithm solving Maximal IS together with a more straightforward solution to Maximal Weight IS. Second, we introduced a new dimension to the study done by Fishetti et al. (1989) and analyzed the maximization criterion for IS problems with machine working time constraints. We developed an algorithm solving Maximal Preemptive TIS and proved that Maximal Weight Preemptive TIS is NP-Hard. Until now, we focused only on polynomially solvable cases. Our next task is to design and analyze approximate algorithms for the intractable problems described in this paper. Heuristic approaches based on the preemptive cases are currently under investigation. Our study of IS problems is not limited to the situation where machines are identical. In forthcoming publications, constraints such as hierarchy (jobs can be processed only on machines of a certain type or bigger) and machine intervals of availability (or personnel shifts) will be added to an IS instance.

References

Arkin, A.M. and Silverberg E.L. (1987), Scheduling Jobs with Fixed Start and End Times, Discrete Applied Mathematics, 18, 1-8.

Carter, M.W. and Tovey C.A. (1992), When is the Classroom Assignment Problem Hard? Operations Research, 40, Supp. No 1, 28-39.

Dondeti, V.R. and Emmons H. (1992). Fixed job Scheduling with Two types of Processors, Operations Research, 40, Supp. No 1, 76-85.

Fischetti, M., Martello S. and Toth P. (1989), The Fixed Job Schedule Problem with Working Time Constraints, Operations Research, 3, 395-403.

Garey, M.R. and Johnson D.S. (1979), Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco.

Golumbic, M.C. (1980), Algorithmic Graph Theory and Perfect Graphs, Academic Press

Kolen, A.J.W. and Kroon L.G. (1991), On the Computational Complexity of (Maximum) Class Scheduling, European Journal Of Operations Research, 54, 23-38.

Kolen, A.J.W., Lenstra J.K. and Papadimitriou C.H. (1986), Interval Scheduling Problems, Manuscript, Centre for Mathematics and Computer Science, C.W.I., Kruislaan 413, 1098 SJ Amsterdam.

Martello S. and Toth P. (1986), A Heuristic Approach to the Bus Driver Scheduling Problem, European Journal Of Operations Research, 24, 106-117.