Pergamon

S0305-0548(96)00016-0

Computers Ops Res. Vol. 23, No. 11, pp. 1059–1067, 1996

0305-0548/96 $15.00+0.00

LOW-COMPLEXITY ALGORITHMS FOR SEQUENCING JOBS WITH A FIXED NUMBER OF JOB-CLASSES

Jack A. A.van der Veen[1]†‡, Shuzhong Zhang[2]§

[1]Nijenrode University, The Netherlands Business School, Straatweg 25, 3621 BG Breukelen, The Netherlands

[2]Econometric Institute, Erasmus University Rotterdam, P.O. Box 1738, 3000 DR Rotterdam, The Netherlands

(Received November 1995; in revised form February 1996)

Scope and Purpose—One of the key problems in manufacturing operations is to determine the assignment of jobs to machines and the sequence of the jobs on each machine. In this paper we consider an environment in which the machines are identical, change-over times in between the processing of two consecutive jobs are job-dependent and the objective is to maximize the utilization of the machines which is equivalent to minimizing the makespan. Unfortunately, such problems fall into the strongly *NP*-hard category. However, it is the purpose of this paper to show that the computational complexity drastically improves if the jobs are divided into a small number of classes where each class consists of "similar" jobs. Such situations can be found in many applications where the jobs in a group are identical, such as for example in the Aircraft Sequencing Problem (where an optimal sequence has to be determined for the landing of airplanes which are divided into several classes according to their size) or if the jobs in a group require a similar state of the machine, like e.g. color for a painting machine or tool-loading in flexible manufacturing systems. In such cases the change-over times are actually determined by the class of the jobs. For a fixed number of job classes, we give fast algorithms for the one-machine problem and the problem with multiple machines and identical processing times.

Abstract—In this paper we consider the problem of scheduling n jobs such that makespan is minimized. It is assumed that the jobs can be divided into K job-classes and that the change-over time between two consecutive jobs depends on the job-classes to which the two jobs belong. In this setting, we discuss the one machine scheduling problem with arbitrary processing times and the parallel machines scheduling problem with identical processing times. In both cases it is assumed that the number of job-classes K is fixed. By using an appropriate integer programming formulation with a fixed number of variables and constraints, it is shown that these two problems are solvable in polynomial time. For the one machine scheduling case it is shown that the complexity of our algorithm is linear in the number of jobs n. Moreover, if the problem is encoded according to the high multiplicity model of Hochbaum and Shamir, the time complexity of the algorithm is shown to be a polynomial in log n. In the parallel machine scheduling case, it is shown that if the number of machines is fixed the same results hold. Copyright © 1996 Elsevier Science Ltd

1. INTRODUCTION

In this paper we consider scheduling problems with the following characteristics. There are m parallel machines and n jobs (denoted by $J_1,..., J_n$) are to be processed on these machines. All jobs are available at time zero and preemption is not allowed. Furthermore, there is a job-dependent change-over time between any pair of jobs, and the objective is to find a distribution of the jobs over the machines and a sequence of the jobs on each machine such that makespan is minimized.

The largest part of this paper is devoted to the single machine case (i.e. $m=1$). Clearly, in this case the problem of dividing the jobs over the machines vanishes. To keep the presentation clear, we will from now on assume that there is only a single machine. In Section 5 we come back to the case $m\geq 2$.

Applications of the above described single machine sequencing problems appear if the machine must be in some *beginning state* B_i (e.g. temperature, tool-loading or paint-color) in order to process job J_i. In some applications the state of the machine after job J_i has been processed, the *ending state* E_i, differs from the beginning state B_i, e.g. if the state of the machine is given by the temperature (see Gilmore and

† To whom all correspondence should be addressed (e-mail:vanderveen@nijenrode.nl).

‡ Jack A. A. van der Veen is Assistant Professor in the area of Production and Logistics at Nijenrode University, the Netherlands Business School. He obtained a Ph.D. in Economics from the University of Groningen, the Netherlands. His research interests include Solvable cases of combinatorial optimization problems, Sensitivity analysis and Scheduling. He has published in several journals including the *European Journal of Operational Research*, the *Journal of the Operational Research Society* and *Discrete Applied Mathematics*.

§ Shuzhong Zhang is an Assistant Professor of Operations Research at the Econometric Institute at the Erasmus University of Rotterdam. He received a B.Sc. degree in Mathematics from Fudan University Shanghai, P.R. China, and a Ph.D. degree in Operations Research from the Erasmus University Rotterdam. His papers have been published in *Mathematical Programming*, *Operations Research*, *Journal of Optimization Theory and Applications*, etc. Dr Zhang's current research interests include Analysis of algorithms, Special cases of combinatorial optimization and Interior point methods for mathematical programming.

Gomory [9]). Let $d[i,j]$ denote the change-over time needed to transform the machine from the ending state E_i for job J_i to the beginning state B_j for job J_j ($i,j=1,\ldots, n$) and p_i the processing time of job J_i ($i=1,\ldots, n$). A dummy job J_0 (with $p_0=0$) can be used to model the fact that the machine is in a beginning state B_0 before the jobs are being processed and should be in an ending state E_0 after all jobs have been processed. In mathematical terms, by introducing a dummy job J_0, a *sequence* of jobs is transformed to a *cyclic permutation* of 0, 1,..., n where J_0 marks the beginning (and end) of the sequence. For a given cyclic permutation τ the completion time of the last job in the sequence (i.e. the *makespan*) is given by the sum of all processing times and change-over times:

$$\sum_{i=0}^{n}\left(p_{\tau(i)}+d[i,\tau(i)]\right)=\sum_{i=0}^{n}p_i+\sum_{i=0}^{n}d[i,\tau(i)].$$

Clearly, the sum of the processing times is a constant, hence the problem of finding a sequence that minimizes the makespan can be formulated as

$$\min\left\{\sum_{i=0}^{n}d[i,\tau(i)]:\tau \text{ is a cyclic permutation of } 0,1\ldots,n\right\},$$

i.e. as an asymmetric *Traveling Salesman Problem* (TSP) with distance matrix $D=(d[i,j])$. In TSP-terms, a cyclic permutation is exactly the same as a *tour*. For notational simplicity, we will assume, without loss of generality, that it is our objective to find a shortest tour with respect to an $(n \times n)$-matrix $D=(d[i,j])$, i.e. we do not consider a separate dummy job J_0. So we want to solve the TSP

$$\min\left\{d(\tau)=\sum_{i=1}^{n}d[i,\tau(i)]:\tau \text{ is a cyclic permutation of } 1,\ldots,n\right\}.$$

We will consider a *special case* of this problem in the following sense. On top of the previous assumptions, it will be assumed that the jobs are divided into K *groups* $N_1,\ldots, N_K$ and that the change-over time between two jobs is determined by the job groups to which they belong. We will denote the number of jobs in group k by n_k (i.e. $n_k=|N_k|$, $k=1,\ldots, K$). In this paper the terms "group" and "class" both refer to the same and will be used interchangeably. Let $C=(c[p,q])$ be a $(K\times K)$-matrix, where $c[p,q]$ denotes the change-over time if a job from group N_q is scheduled directly after a job from group N_p ($p,q=1,\ldots, K$). Then, the distance matrix $D=(d[i,j])$ of the corresponding TSP is given by

$$d[i,j]=c[p,q] \text{ if } J_i\in N_p \text{ and } J_j\in N_q$$

for all $i,j\in\{1,\ldots, n\}$ and all $p,q\in\{1,\ldots K\}$. We will refer to the TSP restricted to the class of matrices satisfying this property as the *K-group TSP*.

Throughout this paper we will use the following example for illustration purposes.

Example: Assume that there are nine jobs $J_1,\ldots, J_9$ divided into three job classes (i.e. $n=9$ and $K=3$), and that the first job-group consists of jobs J_1, J_2 and J_3 (so $N_1=\{J_1,J_2,J_3\}$, $N_2=\{J_4,J_5\}$ and $N_3=\{J_6,J_7,J_8,J_9\}$). Furthermore, assume that the matrix $C=(c[p,q])$ is given by

$$C=\begin{bmatrix}12 & 36 & 25\\ 42 & 19 & 30\\ 56 & 29 & 44\end{bmatrix}.$$

It follows that the 9×9 distance matrix $D=(d[i,j])$ of the 3-group TSP is given by

$$D=\begin{bmatrix}12 & 12 & 12 & 36 & 36 & 25 & 25 & 25 & 25\\ 12 & 12 & 12 & 36 & 36 & 25 & 25 & 25 & 25\\ 12 & 12 & 12 & 36 & 36 & 25 & 25 & 25 & 25\\ 42 & 42 & 42 & 19 & 19 & 30 & 30 & 30 & 30\\ 42 & 42 & 42 & 19 & 19 & 30 & 30 & 30 & 30\\ 56 & 56 & 56 & 29 & 29 & 44 & 44 & 44 & 44\\ 56 & 56 & 56 & 29 & 29 & 44 & 44 & 44 & 44\\ 56 & 56 & 56 & 29 & 29 & 44 & 44 & 44 & 44\\ 56 & 56 & 56 & 29 & 29 & 44 & 44 & 44 & 44\end{bmatrix}.\ \square$$

It is easy to see that for variable K (i.e. K is part of the input) the K-group TSP is NP-hard. This follows immediately from the observation that the ordinary asymmetric TSP can be formulated as an n-group TSP. It is the purpose of this paper to show that for K *fixed*, the K-group TSP is solvable with low time complexity in n or in $\log n$.

In order to derive the complexity of our algorithms let us make clear how the input of the K-group TSP can be encoded. A straightforward way of encoding the input is to encode every single job by its processing time and by the number of the group it belongs to, and to encode the change-over times by a $(K \times K)$-matrix. Clearly, this "standard" encoding is of size $O(n)$. However, as mentioned above, the processing times of the jobs are of no relevance, hence we can encode an instance of the K-group TSP by the $(K \times K)$-matrix of change-over times together with the K numbers $n_1, \ldots n_K$, which yields an encoding of size $O(\log n)$. Following Hochbaum and Shamir [11] we will refer to this encoding as the *high multiplicity* (HM) encoding. In this paper it is shown that if the standard encoding is used, the K-TSP is solvable in $O(n)$ time. Moreover, it is shown that our algorithm can be adapted to run in $O(p(\log n))$ time with $p(\cdot)$ a certain polynomial, if the high multiplicity encoding of the input and an appropriate way of representing the output (i.e. an optimal tour) is used.

A number of authors have discussed algorithms for the K-group TSP. In 1980, Psaraftis [13] presented a dynamic programming approach to solve the K-group TSP. Using the standard encoding of an instance, the computational complexity of Psaraftis' algorithm is

$$O(K^2 \Pi_{k=1}^{K} (1+n_k)^K).$$

Later, Bianco *et al.* ([3] and [4]) introduced a branch and bound method for solving the same problem, using a Lagrange dual for estimating the lower bound of the optimal value. In particular, jobs are assumed to subject to arbitrary release times in [3] and [4]. In 1984, Cosmadakis and Papadimitriou [6] gave an algorithm for the K-group TSP (which they referred to as the *many-visits TSP*) based on solving an exponential number of transportation problems. Under the assumption that K is fixed and the high-multiplicity encoding is used, it was shown that their algorithm runs in $O(p(\log n))$ time, i.e. it has similar time complexity as the algorithm presented in this paper. However, the approach proposed in this paper has several advantages. It is flexible in implementation and easy in proving the correctness. Moreover, our method is more general. As we will show in Section 5, the same technique is applicable to the case where there are $m \geq 2$ parallel machines and all processing times are identical.

The remainder of this paper is organized as follows. In Section 2 we will discuss some applications of the K-group TSP. In Section 3 we will give an $O(n)$ algorithm for the K-group TSP for fixed K using the standard encoding. The high multiplicity model is discussed in Section 4. Finally, in Section 5, we discuss the generalization of our algorithm for the parallel machine case.

2. SOME APPLICATIONS

Although we formulated the K-group TSP in terms of minimizing the summation of change-over *times*, it is clear that the same model can be used if the problem is to minimize the summation of change-over *costs*.

Applications of the K-group TSP occur if the jobs can be divided into K groups of *identical* jobs. One important motivation for studying the K-group TSP is the *Aircraft Sequencing Problem* (ASP), see Bianco *et al.* [3] and Psaraftis [13]. Assume there are n airplanes waiting for permission to land on a single runway. According to the number of passenger seats, the airplanes can be divided into $K=3$ classes: small, medium and large. Safety regulations require a certain time-gap between the landing of two airplanes. This time-gap depends on the size of the two airplanes. Now assume that there are n_1 small airplanes, n_2 medium sized airplanes and n_3 large airplanes and that the flight controller wants to determine the sequence in which the $n=n_1+n_2+n_3$ airplanes will land according to the safety regulations, and the total amount of time is minimized. Clearly, this ASP can be modeled as a 3-group TSP. Some heuristic methods for solving the ASP are discussed in [10].

In another application of the K-group TSP, the machine needs to be in one of K "states" in order to be able to process a job. Jobs are classified with respect to their required state of the machine. Examples of such "states" are:

- *Color* for a painting-machine (cf. Conway *et al.* [5]). Here a change-over time is needed for cleaning the machine and inserting a new paint-color. The change-over times might differ for different colors, e.g. changing over from "black" to "white" might require considerable more time than the other way

round. Obviously, in this case a job-group consists of all jobs that need to be painted with the same color, and K denotes the number of different colors.

- *Tool-loading* for a machine in a flexible manufacturing system (see e.g. Tang and Denardo [15]). In this case, each job requires a certain set of tools to be loaded in the limited capacity tool magazine of the machine. Tools can be switched between the magazine of the machine and a (central) tool storage area. The change-over time between two jobs is determined by the time it takes to replace the set of required tools. Here a job group consists of all jobs that need the same set of tools and K denotes the number of possible tool-loads.

A real-life application including a two-dimensional machine-state was recently given in Al-Haboubi and Selim [1]. In that paper a problem in the weaving industry is discussed. A job represents an order for a batch of cloth pieces with a given specification (width, length and type). There is one weaving machine that has to be set-up with respect to the width and the type (i.e. type of fiber used and thickness) of the job but that can produce any required length. Assuming that there are W possible widths and T possible types, the jobs can be classified into $K=W \cdot T$ classes, where each job class consists of all jobs with a given width and a given type. Clearly, the problem of sequencing the jobs such that the total set-up time is minimized can again be formulated as a K-group TSP.

3. A LINEAR TIME ALGORITHM

In this section we will give a linear time algorithm for the K-group TSP based on the standard encoding of the input. Our approach is the following. First we will give a lower bound for the K-group TSP by means of an integer programming problem (IPP). Thereafter it will be shown that the solution of (IPP) can be transformed in linear time to a cyclic permutation with the same length as the value of the solution of (IPP). Since (IPP) has an input length of $O(\log n)$, it is solvable, using Lenstra's algorithm, in time complexity polynomial in $O(\log n)$, which means that this approach leads to an $O(n)$ algorithm.

For a given cyclic permutation, let x_{pq} denote the number of times a job from group N_q is scheduled directly after a job from group N_p $(p,q=1,\ldots, K)$. Consider the following integer programming problem (IPP):

$$\text{minimize} \sum_{q=1}^{K} \sum_{p=1}^{K} c[p,q]x_{pq}$$

$$\text{s.t.} \begin{cases} \sum_{p=1}^{K} x_{pq}=n_q & \text{for} \quad q=1,\ldots,K & (1) \\ \sum_{q=1}^{K} x_{pq}=n_p & \text{for} \quad p=1,\ldots,K & (2) \\ \sum_{p\in S} \sum_{q\notin S} x_{pq}\geq 1 & \text{for all} \quad S\subset\{1,\ldots,K\},\ S\neq\varnothing & (3) \\ x_{pq}\geq 0 \text{ and integer} & \text{for} \quad p,q=1,\ldots,K. & (4) \end{cases}$$

Let $X=(x_{pq})$ be a $(K \times K)$-matrix that satisfies all constraints (1)–(4) in (IPP). According to the *degree-constraints* (1)–(2), the pth row-sum and the pth column-sum of X both have to be equal to n_p. This reflects the fact that since there are n_p jobs in group N_p, there have to be n_p jobs that are scheduled directly after (and before, respectively) jobs in this group. Note that from the degree constraints it follows that

$$\sum_{p\in S} \sum_{q\notin S} x_{pq} = \sum_{p\notin S} \sum_{q\in S} x_{pq} \text{ for all } S\subset\{1,\ldots,K\}.$$

This corresponds to the observation that if the job-groups are divided into two nonempty parts (S and $\{1,\ldots, K\}\backslash S$), a job from one part has to precede a job from the other part as many times as the other way round.

The third type of constraints in (IPP), the *connectivity constraints* (3), assure that the jobs are scheduled such that for any partition of the job-groups, there is always a job in one part preceded by a job in the other part. Clearly, the connectivity constraints in (IPP) correspond to the subtour-elimination constraints for the integer programming formulation of the ordinary TSP (which can be traced back to Dantzig *et al.* [7]).

Intuitively it is clear that if X corresponds to an *optimal* tour it must satisfy all four conditions in (IPP). This intuition is formally supported by the following lemma.

Lemma 1: Let $X^*=(x^*_{pq})$ be an optimal solution of (IPP). Furthermore, let τ^* be an optimal solution of the K-group TSP with the same C and $N_1,\ldots, N_K$. Then,

$$\sum_{p=1}^{K}\sum_{q=1}^{K} c[p,q]x^*_{pq}\leq d(\tau^*).$$

Proof: From an optimal tour τ^* we will construct a $(K\times K)$-matrix $Y=(y_{pq})$ such that

$$\sum_{p=1}^{K}\sum_{q=1}^{K} c[p,q]y_{pq}=d(\tau^*)$$

and such that Y satisfies conditions (1)–(4) in (IPP). The assertion then follows immediately. Define

$$E_{pq}(\tau^*):=\{(i,\tau^*(i)):J_i\in N_p \text{ and } J_{\tau^*(i)}\in N_q\} \text{ and } y_{pq}:=|E_{pq}(\tau^*)|$$

for $p,q = 1,\ldots, K$. Note that the sets $E_{pq}(\tau^*)$ are mutually disjunct (i.e. $E_{pq}(\tau^*)\cap E_{p'q'}(\tau^*)=\varnothing$ *if* $(p,q)\neq(p',q')$) and that $d[i,\tau^*(i)] = c[p,q]$ for all $(i,\tau^*(i))\in E_{pq}(\tau^*)$. Furthermore, note that y_{pq} is nonnegative and integer for all p,q, hence satisfies condition (4) in (IPP). Since for all q

$$\sum_{p=1}^{K} y_{pq}=\sum_{p=1}^{K}|E_{pq}(\tau^*)|=\left|\left\{(i,\tau^*(i)):J_i\in(N_1\cup\ldots\cup N_K) \text{ and } J_{\tau^*(i)}\in N_q\right\}\right|=n_q,$$

condition (1) in (IPP) is satisfied by $Y=(y_{pq})$. Similarly, it can be shown that Y also satisfies condition (2) in (IPP). Finally, assume that Y does not satisfy condition (3) in (IPP), i.e. that there is an $S\subset\{1,\ldots, K\}$ such that

$$\sum_{p\in S}\sum_{q\notin S} y_{pq}=0.$$

Since conditions (1) and (2) of (IPP) are satisfied, we have

$$\sum_{p\notin S}\sum_{q\in S} Y_{pq}=\sum_{p\in S}\sum_{q\notin S} y_{pq}=0.$$

By defining $S':=\cup_{p\in S}N_p$, it is easy to see that these equalities imply that for all $i\in S'$ it holds that $\tau^*(i)\in S'$ and for all $i\notin S'$ that $\tau^*(i)\notin S'$, which contradicts with the fact that τ^* is a tour. □

An important observation is that (IPP) can be solved with time complexity polynomial in log n. This follows from the fact that (IPP) is an integer program with K^2 variables and 2^K+K^2+2K constraints. The encoding length of the inequalities is a constant, and the length of each equality constraint is $O(\log n)$. It can be shown that in this case the problem is solvable with the number of operations polynomial in log n, using e.g. Lenstra's algorithm (cf. Lenstra [12] and Schrijver [14]). For practical purposes, K being small, one can apply a branch and bound method to solve (IPP) efficiently (cf. Balas and Toth [2]).

It will now be shown that, given an optimal solution X^* of (IPP), a tour τ can be constructed such that

$$d(\tau)=\sum_{p=1}^{K}\sum_{q=1}^{K} c[p,q]x^*_{pq}.$$

Note that, by Lemma 1, a tour τ that satisfies this equality is an *optimal* tour. In order to find a tour for which equality holds, consider the following procedure.

Procedure Construct_Tour:

Step 1. Construct a weighted directed multi-graph $G=(V,A,w)$ with vertex- set $V = \{1,\ldots, K\}$ as follows. There are x^*_{pq} arcs from vertex p to vertex q $(p,q=1,\ldots, K)$. Note that self-loops are allowed. The weight $w(p,q)$ of an arc from vertex p to vertex q is given by $c[p,q]$ (for all p and q).

Step 2. Construct an Eulerian cycle in G.

Step 3. Construct a tour from the Eulerian cycle by simply replacing the vertices k in V by the jobs in the corresponding job group N_k $(k=1,\ldots, K)$. □

It is easy to see that the number of arcs in the graph G constructed in Step 1 is equal to n. Moreover, from the degree constraints (1)–(2) in (IPP) it follows that for all vertices $k\in V$ we have

$$\text{in-degree}(k)=\text{out-degree}(k)=n_k.$$

Furthermore, by the connectivity constraints (3) in (IPP), G is connected. From these two properties it follows that G is *Eulerian* (i.e. contains a Eulerian cycle), which makes Step 2 feasible. Since the length (defined by the sum of the weights of the arcs) of all Eulerian cycles in G is equal to the value of the optimal solution of (IPP), the length of the tour constructed in Step 3 is equal to the value of the optimal solution of (IPP).

Example (continued): The integer program (IPP) is solved by

$$X^* = \begin{bmatrix} 2 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 2 & 2 \end{bmatrix}.$$

The graph G is shown in Fig. 1. Note that (1,1,1,3,3,3,2,3,2) forms a Eulerian cycle in G. The tour constructed from this cycle is τ = (1,2,3,6,7,8,4,9,5). The length of the tour is 267 which is both equal to the value of the optimal solution of (IPP) and the weight of the Eulerian cycle. □

So, by first solving (IPP) and then using Procedure Construct_Tour we have determined an optimal solution for the K-group TSP. By evaluating the time requirement for this algorithm we obtain the following theorem.

Theorem 1: For fixed K, the K-group TSP is solvable in $O(n)$ time.

Proof: In order to find τ we first had to solve (IPP), which took time polynomial in log n. Since Procedure Construct_Tour takes linear time, an optimal solution can be determined in $O(n)$ time. □

4. THE HIGH MULTIPLICITY MODEL

In Hochbaum and Shamir [11] the notion of *high multiplicity* for scheduling problems is introduced. In their model, jobs are divided into relatively small number of groups. Each group contains identical jobs. Clearly, our problem can be modelled as a high multiplicity one-machine scheduling problem. Interestingly, the complexity results contained in Section 3 can be further sharpened and interpreted as a polynomial procedure under the high multiplicity model. To be more specific, in this section it will be shown that if K is fixed and if the input and output of the problem are recorded in a compact way, then it is even possible to solve the problem in time that is polynomial in log n.

Recall that the HM-encoding of the input of the K-group TSP is of size $O(\log n)$. Clearly, due to Lenstra's result [12], the complexity required to solve (IPP) is polynomial in the HM-encoding length, because in this case

$$O\left(\sum_{k=1}^{K} \log n_k\right) = O(\log n).$$

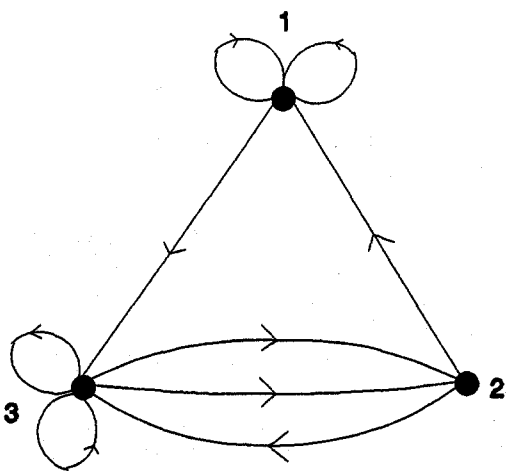

Fig. 1. The graph G in the example.

So, using the HM-encoding, an optimal solution X^* of (IPP) can be computed in $O(p(\log n))$ time.

This brings us to the question of how the output should be represented. Clearly, in case of the HM-encoding of the input, the output can not be represented by the exact optimal tour, because then the output size is exponential in the input size. So, we have to represent the output by a compact description of the Eulerian cycle in G.

Having an optimal solution X^* of (IPP), we should not construct the graph G explicitly because the number of arcs in G is equal to n, i.e. exponential in the HM-encoded input. A more compact way is to construct G' as a complete graph on K vertices (including self-loops) with x^*_{ij} as capacity on the arc (i,j). Using a labeling procedure we can find a cycle in G'. Now, circulate a flow along the cycle until the minimum capacity is saturated. Delete the saturated arc(s) and repeat the procedure until there is no arc left. Clearly, there can be at most K^2 iterations, each resulting in a cycle, and a Eulerian cycle in G can be easily constructed based on the cycles generated by this procedure. The capacity being achieved at each iteration is counted as the degree of multiplicity of the cycle. This procedure is polynomial according to the HM-model. Note that the output of the algorithm is given as a series of cycles of job-groups, where each cycle is labeled with its degree of multiplicity. We will call this the *HM-output*. It is easily seen that a sequence of jobs can directly be constructed from the HM-output. Hence, we conclude that the K-group TSP discussed in this paper is indeed solvable in time polynomial in log n when K is fixed. In other words, we have proven the following theorem.

Theorem 2: Suppose that K is a fixed integer. If an instance of the K-group TSP is described by the HM-input, then the HM-output of the K-group TSP can be determined in $O(p(\log n))$ time.

5. PARALLEL MACHINES

In this section we discuss the sequencing problem where there are $m \geq 2$ parallel identical machines $M_1,..., M_m$ instead of a single machine. As before, the jobs are divided into K job-groups $N_1,..., N_K$, and the change-over time for scheduling a job from N_q directly after a job from class N_p is given by $c[p,q]$ on any machine $(p,q \in \{1,..., K\})$. It is assumed that both m and K are fixed. The objective is to minimize the makespan, i.e. to minimize the largest workload on the m machines.

If the jobs have arbitrary processing times, the problem is NP-hard. This follows immediately from the observation that the special case with $m=2$, $K=1$ and zero change-over times is the *Partition Problem* which is well-known to be NP-hard (see e.g. Garey and Johnson [8]). However, as will be shown below, the same technique used in Section 3 for the single machine case can be used to derive similar results for the parallel machine case if all the n jobs have *identical processing times*. Without losing generality, we assume that the processing times are *unit*.

The output of the parallel machine problem consists of the sequences of jobs processed on each machine. We introduce a group of dummy jobs N_0 with $n_0=m$ where it is assumed that one dummy job is placed on each machine. The dummy job marks the beginning and end of the sequence on each machine. The dummy jobs are assumed to have zero processing times. Furthermore, the corresponding change-over times are given by $c[i,0]=c[0,j]=0$ for $i,j=1,..., K$.

As the single machine problem was reformulated as a TSP, the parallel machine problem can be seen as a (one-depot, uncapacitated) *Vehicle Routing Problem*. The jobs correspond to the customers and the machines to the trucks. The output describes which truck visits which customer (which job is processed on which machines) and in what sequence the customers are visited by each truck (the sequence of jobs on the machines). The central depot, where all trucks start and return after their tour, is represented by the group of dummy jobs N_0.

In order to solve the parallel machine problem with unit processing times, let x^j_{pq} denote the number of times a job from class N_q is scheduled directly after a job from class N_p on machine M_j $(p,q = 0,1,..., K; j = 1,..., m)$. Furthermore, let n^j_i be the number of jobs belonging to group N_i that are processed on machine M_j $(i=1,..., K; j=1,..., m)$. Note that, by definition $x^j_{00}=0$ and $n^j_0=1$ for all j. Because of the unit processing times of the "regular" jobs, we can formulate the workload on machine M_j by

$$\sum_{i=1}^{K} n^j_i + \sum_{p=1}^{K} \sum_{q=1}^{K} c[p,q]x^j_{pq}.$$

Consider the following mixed integer programming formulation (MIP).

minimize t

$$\text{s.t.}\begin{cases} t \geq \sum_{i=1}^{K} n_i^j + \sum_{p=1}^{K}\sum_{q=1}^{K} c[p,q]x_{pq}^j \text{ for } j=1,\ldots,m & (1)\\ \sum_{j=1}^{m} n_i^j = n_i \text{ for } i=1,\ldots,K & (2)\\ n_0^j = 1 \text{ for } j=1,\ldots,m & (3)\\ \sum_{p=0}^{K} x_{pq}^j = n_q^j \text{ for } j=1,\ldots,m;\ q=0,1,\ldots,K & (4)\\ \sum_{q=0}^{K} x_{pq}^j = n_p^j \text{ for } j=1,\ldots,m;\ p=0,1,\ldots,K & (5)\\ n\left(\sum_{p\in S}\sum_{q\in S} x_{pq}^j\right) \leq (n-1)\sum_{i\in S} n_i^j \text{ for all } j=1,\ldots,m;\ S\subseteq\{1,\ldots,K\} & (6)\\ x_{pq}^j \geq 0 \text{ and integer, } j=1,\ldots,m;\ p,q=0,1,\ldots,K & (7)\\ n_i^j \geq 0 \text{ and integer, } j=1,\ldots,m;\ i=1,\ldots,K. & (8) \end{cases}$$

Clearly, constraints (1) together with the objective function make sure that makespan is minimized. Constraints (2)–(5) are a generalization of the degree constraints in (IPP), and are used to assure that the jobs are correctly divided over the machines. Constraints (6) are generalized from the connectivity constraints in (IPP). Note that it is not necessary that jobs from each job-group are present on a given machine. Therefore, two cases are to be distinguished.

1. $\Sigma_{i\in S} n_i^j = 0$, i.e. no jobs belonging to a group in S are scheduled on machine M_j. In this case the constraint reads

$$\sum_{p\in S}\sum_{q\in S} x_{pq}^j = 0.$$

2. $\Sigma_{i\in S} n_i^j > 0$, i.e. there is a job belonging to a group in S that is scheduled on machine M_j. Note that, since S does not contain 0, we have $\Sigma_{i\notin S} n_i^j > 0$. So, in this case we have the constraint

$$\sum_{p\in S}\sum_{q\in S} x_{pq}^j < \sum_{i\in S} n_i^j$$

or equivalently

$$\sum_{p\in S}\sum_{q\notin S} x_{pq}^j \geq 1.$$

Note also that under constrain (6) it is not possible that there is a subtour in $\{1,\ldots, K\}\backslash S$ because, clearly, this set is also a subset of $\{1,\ldots, K\}$, i.e. that subtour is prohibited by constraint (6) for that particular set.

Finding an optimal solution of (MIP), again using Lenstra's algorithm, can be done in time complexity polynomial in log n when both m and K are fixed as constants. Analogous to the one machine case, we can construct an optimal schedule in the compact way as we described in the previous section, based on the optimal solution of the above mixed integer program in time complexity polynomial in m, K and log n.

Example (continued): Assume that there are $m=2$ parallel machines. The optimal solution of the integer programming problem is

$$X^1 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \\ 1 & 0 & 1 & 0 \end{bmatrix} \text{ and } X^2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix},$$

so that $n_1^1=0, n_2^1=2, n_3^1=2, n_1^2=3, n_2^2=0$, *and* $n_3^2=2$, hence four jobs are scheduled on M_1 and five on M_2 The corresponding tours are $(D_1,4,6,5,7)$ on machine M_1 with length 89 and $(D_2,1,2,3,8,9)$ on machine M_2 with length 93 (where D_j is the dummy job on machine M_j, j =1,2). The makespan is therefore given by $t=\max\{89+4, 93+5\}=98$. □

REFERENCES

1. M.H. Al-Haboubi and S.Z. Selim, A sequencing problem in the weaving industry, *Eur. J. Opl Res.*, **66**, 65–71 (1993).
2. E. Balas, and P. Toth. Branch and bound methods. In *The Traveling Salesman Problem* (Edited by E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys) John Wiley & Sons, New York (1985).
3. L. Bianco, R. Rinaldi and A. Sassano: A combinatorial optimization approach to aircraft sequencing problem. In Flow Control of Congested Networks (Edited by A. R. Odoni *et al.*) NATO-ASI Series **38**, 324–339, (1987).
4. L. Bianco, S. Ricciardelli, G. Rinaldi and A. Sassano, Scheduling tasks with sequence-dependent processing times, *Naval Res. Logist.*, **35**, 177–184 (1988).
5. R. W. Conway, W. L. Maxwell, and L. W. Miller. *Theory of Scheduling*. Addison-Wesley, Reading MA (1967).
6. S.S. Cosmadakis and C.H. Papadimitriou, The traveling salesman problem with many visits to a few cities, *SIAM J. Computg*, **13**, 99–108 (1984).
7. G.B. Dantzig, D.R. Fulkerson and S.M. Johnson, Solution of a large-scale traveling salesman problem, *Ops Res.*, **2**, 393–410 (1954).
8. M. R. Garey, and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company (1979).
9. P.C. Gilmore and R.E. Gomory, Sequencing a one state-variable machine: a solvable case of the traveling salesman problem, *Ops Res.*, **12**, 655–679 (1964).
10. A. ten Have. A new approach to aircraft sequencing. Master's Thesis, Department of Econometrics, University of Groningen, The Netherlands (1992).
11. D.S. Hochbaum and R. Shamir, Strongly polynomial algorithms for the high multiplicity scheduling problem, *Ops Res.*, **39**, 648–653 (1991).
12. H.W. Lenstra, Integer programming with a fixed number of variables, *Math. Ops Res.*, **8**, 538–548 (1983).
13. H.N. Psaraftis, A dynamic programming approach for sequencing groups of identical jobs, *Ops Res.*, **28**, 1347–1359 (1980).
14. A. Schrijver *Theory of Linear and Integer Programming*. John Wiley, New York (1986).
15. C.S. Tang and E.V. Denardo, Models arising from a flexible manufacturing machine, part I: minimization of the number of tool switches, *Ops Res.*, **36**, 767–7771 (1988).