An $\mathcal{O}(n \log n)$ Algorithm for the K-Template Traveling Salesman Problem

JACK A.A. VAN DER VEEN * GERHARD J. WOEGINGER †
SHUZHONG ZHANG ‡

November 23, 1995

Abstract

In this paper a one-machine scheduling model is analyzed where n different jobs are classified into K groups depending on which additional resource they require. The change-over time from one job to another consists of the removal time or of the set-up time of the two jobs. It is sequence-dependent in the sense that the change-over time is determined by whether or not the two jobs belong to the same group. The objective is to minimize the makespan. This problem can be modeled as an asymmetric Traveling Salesman Problem with a specially structured distance matrix. For this problem we give a polynomial time solution algorithm that runs in $O(n \log n)$ time.

Keywords: Single Machine Scheduling, Traveling Salesman Problem, Polynomial Time Algorithm.

*Nijenrode University, The Netherlands School of Business, Straatweg 25, 3621 BG Breukelen, The Netherlands.

†TU Graz, Institut für Mathematik B, Steyrergasse 30, A-8010 Graz, Austria.

‡Econometric Institute, Erasmus University Rotterdam, P.O. Box 1738, 3000 DR Rotterdam, The Netherlands.

1 Introduction

In this paper we consider the following problem. There are n jobs, $J_1, \ldots, J_n$, to be processed without interruption on a single machine. In order to process a job, a certain additional resource is required, e.g. a fixture or, as we will call it in this paper, a *template*. In total there are K different templates ($K \leq n$) each with a unique form. It is assumed that the machine always works with one template at a time. Moreover, it is assumed that a template can contain only one job each time and jobs can only be processed in one (job-specific) template. So, the jobs can be partitioned according to the template they require. The set of indices that belong to jobs requiring the k-th template is denoted by N_k and its cardinality by $n_k := |N_k|$, $1 \leq k \leq K$. Furthermore, there is a *change-over time* between the processing of two jobs. This change-over time is determined by the following two rules.

(R1) After a job has been processed, it has to remain in its template for a certain time period to allow some after-processing, e.g. cooling. For job J_i this time period is denoted by a_i ($i = 1, \ldots, n$). However, it is not necessary that the after-processing is done on-line, i.e. that the template remains placed on the machine. If a different template is to be used for the next job, the template, with the job still in it, can be removed from the machine in no time and the after-processing can take place off-line. It is assumed that the after-processing time is negligible compared to the processing times of the jobs on the machine. This implies that the template is free to be used whenever that is required (or, in other words, that the machine never needs to wait idle for a template becoming available). On-line after-processing is necessary only when the following job requires the same template.

(R2) If the same template is not used for the next job, then a set-up time is needed. The length of this set-up time depends on the job to be processed next. For job J_i this set-up time is denoted by b_i, $1 \leq i \leq n$.

In order to get an expression for the change-over time, suppose that job J_j is scheduled directly after job J_i, and that $i \in N_p$ and $j \in N_q$. If both jobs require the same template (i.e. $p = q$) then the change-over time is given by a_i, because the after-processing of job J_i is done on-line, so only after that time J_j can be placed in the template for processing. If J_j requires another template than J_i (i.e. $p \neq q$) then the change-over time is given by b_j because in this case the template with J_i can be taken away from the machine (the after-processing can take place off-line) and the machine has to be set up in order to process job J_j.

Our objective is to determine a sequence of the jobs (a *schedule*) such that all jobs are processed on the machine in a minimal amount of time, i.e. we want to minimize the *makespan.* (Note that in the problem described above, the makespan includes the set-up time of the first job and does not include the removal time of the last job). Clearly, this is equivalent to minimizing the summation of change-over times. It is well-known (and easy to see) that by introducing a dummy job J_0, the sequencing problem with job-dependent change-over times can be reformulated as an (asymmetric) *Traveling Salesman Problem* (TSP). The distance matrix of the corresponding TSP is given by $D = (d\,[i,j])$ where $d\,[i,j]$ denotes the change-over time if job J_j is scheduled directly after job J_i $(i, j = 0, \ldots, n)$. In general, the TSP is *NP*-hard. However, this hardness result does not necessarily carry over to our case, since in the situation described above the distance matrix has a rather special structure,

$$d[i,j] = \begin{cases} a_i & \text{if } i \text{ and } j \text{ belong to the same group} \\ b_j & \text{if } i \text{ and } j \text{ belong to different groups} \end{cases}$$

where the groups are given by $N_1, \ldots, N_K$ and a dummy group N_0 containing only job J_0. Remark that if we assume $a_0 = 0$ and $b_0 = 0$, the minimal length of the above TSP equals the minimal makespan of the K -template one machine scheduling problem. Therefore, in the remainder of this paper we will not take the dummy job into consideration but will, without loss of generality, concentrate on solving the TSP on the jobs $J_1, \ldots, J_n$.

The main objective of this paper is to show that due to the special structure of the distance matrix, the problem allows a *polynomial-time* exact solution algorithm. To be more precise: It will be shown that the TSP restricted to distance matrices satisfying the above property (which will be referred to as the *K–template TSP*) is solvable in $O(n \log n)$ time.

The scheduling problem discussed in this paper has several characteristics, including:

- One machine scheduling with sequence dependent change-over times
- Makespan minimization
- Polynomial-time solvability
- Job-groups.

Note that the first two characteristics make it possible to reformulate the problem as a TSP. In the following, we will briefly discuss results in the literature that possess at least three of the above characteristics.

An excellent survey on polynomially solvable special cases of the TSP is given in Gilmore *et al.* [8]. More recent results can be found e.g. in Van Dal [15], in Van der Veen [16] and in the recent survey paper by Burkard *et al.* [5]. One of the few polynomially solvable cases of the TSP that relate directly to the first two characteristics mentioned above, is the special case discussed in Gilmore & Gomory [7]. In that paper, a problem of sequencing jobs for heat-treatment in a furnace is discussed. The change-over time between two consecutive jobs is determined by the pre-specified required starting and ending temperature of the furnace for these jobs. In [7] it is shown that for a certain structure on the time-requirements for heating and cooling the furnace, the resulting problem is solvable in $O(n \log n)$ time.

Polynomially solvable single machine scheduling problems with job-groups are discussed in many papers of which we mention Ahn & Hyun [1], Gupta [9], Gupta *et al.* [10], Monma & Potts [12] and Potts [13].

The following two problems have all four characteristics mentioned above in common with the K–template TSP.

The first problem concerns the so called *Aircraft Sequencing Problem* (ASP), see e.g. Psaraftis [14] and Bianco *et al.* [3, 4]. Assume that 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 classes (from small to 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. Assume that for size k, $1 \leq k \leq K$, there are n_k airplanes waiting for permission to land and that the flight controller wants to determine the sequence in which all n airplanes will land such that the total time (landing time plus safety time for all airplanes) is minimized. Let $c[p,q]$ denote the required safety time if an airplane from class N_q is to land directly after an airplane from class N_p. Clearly, the ASP can be reformulated as a TSP on the n airplanes where the distance matrix $D = (d[i,j])$ is given by

$$d[i,j] = c[p,q]$$

if $i \in N_p$ and $j \in N_q$ for all $i,j \in \{1,\ldots,n\}$ and all $p,q \in \{1,\ldots,K\}$. Psaraftis [14] proposed a dynamic programming approach for the ASP, whereas Bianco *et al.* ([3, 4]) suggested a branch-and-bound method for this problem. In Van der Veen & Zhang [17] (see also Cosmadakis & Papadimitriou [6]) it is shown that for any *fixed* value K (that is not part of the input), the ASP is solvable in $O(n)$ time.

The second problem occurs in *printed circuit board* (PCB) design. Assume that n parts are to be inserted by a robot arm at specified points on a PCB. Each of the parts belongs to one of K part-types. All parts of a given type are stored in a bin. The K bins are located along one side of the PCB. Furthermore, it is assumed that the robot arm can only move sequentially in either horizontal or vertical direction, so that the robot arm travel time from one location to another is proportional to the rectilinear distance between the two locations. The *Robot Tour Problem* (RTP) is to find the optimal sequence of insertions of the parts in the PCB, given the locations of the n insertion points and the locations of the K bins. For more details and extensions see Hamacher [11].

For fixed locations of the bins, the RTP can be reformulated as a TSP on the n insertion points. Obviously, the distance from one insertion point to another equals the distance from the first insertion point to the bin containing the part to be inserted into the second insertion point plus the distance from this bin to the second insertion point. Note that the second part of this distance is fixed, in the sense that independent of the sequence this distance has to be travelled. However, the first part (the distance from first insertion point to bin for the second insertion point) is sequence-dependent. Since all bin-locations are located at the same side of the PCB and since distances are measured rectilinearly, the only part of this distance that is sequence dependent is the distance from the projection of the insertion point on the side of the PCB where the bins are located to the next bin. Let N_k be the set of all parts of type k ($k = 1,\ldots,K$). If a part from N_k is to be inserted after a given part i, the sequence dependent "distance" $d[i,j]$ to travel from i to $j \in N_k$ has the same value for all $j \in N_k$. It follows that the distance matrix $D = (d[i,j])$ of the TSP modelling the RTP is given by

$$d[i,j] = a^k[i] \quad \text{if} \quad j \in N_k$$

for all $i,j \in \{1,\ldots,n\}$, where $a^1, a^2, \ldots, a^K$ are given n-dimensional vectors. A polynomial time algorithm for solving this RTP has been given in Ball & Magazine [2].

The rest of this paper is organized into sections as follows. Section 2 gives some definitions and states an illustrating example for the K–template TSP. Sections 3, 4 and 5 provide the combinatorial and algorithmical tools for the polynomial time solution algorithm. The solution algorithm itself is given in Section 6 together with an application to a more general special case of the TSP. Finally, the paper is concluded in Section 7.

2 The K-Template TSP

In the following sections, we will derive an $O(n \log n)$ time solution algorithm for the K–template TSP. An instance of this problem is fully described by the numbers n and K with $1 \le K \le n$, by two vectors $a = [a_1, \dots, a_n]$ and $b = [b_1, \dots, b_n]$, and by the partition $N_1, \dots, N_K$ of the jobs. For notational simplicity, we assume without losing generality that all elements in the vectors a and b are pairwise distinct (i.e. $a_i \neq a_j$ and $b_i \neq b_j$ for $i \neq j$, and $a_i \neq b_j$ for $1 \le i, j \le n$). Notice that vectors a and b are not necessarily integral. Since the proposed algorithm is strongly polynomial, by perturbation the above conditions can always be satisfied without changing the complexity of the algorithm. Furthermore, we assume that the jobs are ordered according to the template they require, i.e. $N_k = \{m_k + 1, \dots, m_k + n_k\}$ for $k = 1, \dots, K$, where $m_1 := 0$ and $m_k := \sum_{t=1}^{k-1} n_t$ for $k = 2, \dots, K$. Since we are going to solve the scheduling problem via a traveling salesman formulation, we will use the terms *job* and *city* interchangeably.

Throughout the paper, we will use the following example to illustrate the introduced concepts and algorithms.

Example. Let there be $n = 9$ cities and $K = 3$ templates, where $N_1 = \{1, 2, 3\}$; $N_2 = \{4, 5\}$ and $N_3 = \{6, 7, 8, 9\}$. Furthermore, let

$$a = [34 \;\; 67 \;\; 83 \;\; 55 \;\; 68 \;\; 94 \;\; 56 \;\; 65 \;\; 88]$$

and

$$b = [73 \;\; 23 \;\; 54 \;\; 77 \;\; 81 \;\; 39 \;\; 50 \;\; 45 \;\; 29].$$

It follows that the distance matrix is given by

$$D = \begin{bmatrix} 34 & 34 & 34 & 77 & 81 & 39 & 50 & 45 & 29 \\ 67 & 67 & 67 & 77 & 81 & 39 & 50 & 45 & 29 \\ 83 & 83 & 83 & 77 & 81 & 39 & 50 & 45 & 29 \\ 73 & 23 & 54 & 55 & 55 & 39 & 50 & 45 & 29 \\ 73 & 23 & 54 & 68 & 68 & 39 & 50 & 45 & 29 \\ 73 & 23 & 54 & 77 & 81 & 94 & 94 & 94 & 94 \\ 73 & 23 & 54 & 77 & 81 & 56 & 56 & 56 & 56 \\ 73 & 23 & 54 & 77 & 81 & 65 & 65 & 65 & 65 \\ 73 & 23 & 54 & 77 & 81 & 88 & 88 & 88 & 88 \end{bmatrix}. \qquad \square$$

3 The Combinatorial Structure of Tour-Sets

Our first step towards a solution of the K–template TSP is triggered by the following observation. Given a tour, it is trivial to find the corresponding set of entries in the distance matrix that are "used" in this tour. Such a set of entries corresponding to a tour consists of exactly n entries in the two vectors a and b (since each element in a and b can only be used once). However, the converse of this statement in general does not hold. There are sets of exactly n entries in the two vectors a and b for which no tour exists that exactly uses this set of entries.

Example (continued). The tour that visits the cities $(1, 3, 5, 7, 9, 8, 6, 4, 2)$ in this order uses the nine distinct entries $\{a_1, a_2, a_7, a_8, a_9, b_2, b_4, b_5, b_7\}$. On the other hand it is easy to verify that there is no tour that uses e.g. the nine entries $\{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9\}$. $\square$

This observation rules out simple solution strategies for the K –template TSP, like selecting the smallest n values from the two vectors a and b and constructing a tour with these values. Moreover, they lead to the problem of finding necessary and sufficient conditions under which a given subset of n values from the vectors a and b corresponds to a tour. Our first goal is to find a solution for this problem.

We start with some definitions. Let

$$\begin{aligned} A_k &:= \{a_{m_k+1}, \ldots, a_{m_k+n_k}\} \\ B_k &:= \{b_{m_k+1}, \ldots, b_{m_k+n_k}\} \\ V_k &:= A_k \cup B_k \end{aligned}$$

for $k = 1, \ldots, K$ and $V := V_1 \cup \ldots \cup V_K = \{a_1, \ldots, a_n; b_1, \ldots, b_n\}$. Also, σ_k and ρ_k are defined as permutations of $m_k + 1, \ldots, m_k + n_k$ such that

$$\begin{aligned} a_{\sigma_k(m_k+1)} &\leq \ldots \leq a_{\sigma_k(m_k+n_k)} \quad \text{and} \\ b_{\rho_k(m_k+1)} &\leq \ldots \leq b_{\rho_k(m_k+n_k)} \end{aligned}$$

for $k = 1, \ldots, K$. Furthermore, for a subset S of V, define

$$\begin{aligned} x_k(S) &:= |S \cap A_k| \\ y_k(S) &:= |S \cap B_k| \end{aligned}$$

for $k = 1, \ldots, K$.

Definition 3.1 *A set $S \subseteq V$ is called a* tour-set *if it satisfies the following four conditions for all $k \in \{1, \ldots, K\}$:*

(T1) $x_k(S) + y_k(S) = n_k$;

(T2) $y_k(S) \leq \sum_{t \neq k} y_t(S)$;

(T3) $y_k(S) \geq 1$;

(T4) *Either $y_k(S) = n_k$ or there is an $i \in N_k$ such that both $a_i \in S$ and $b_i \in S$.* □

Observe that Condition (T4) in fact implies Condition (T3). However, Conditions (T1), (T2) and (T3) together are concerned only with the cardinalities of the intersection sets $|S \cap A_k|$ and $|S \cap B_k|$, whereas Condition (T4) reveals further structure of the intersections. In later analysis this feature will be used in the construction of related tour-sets and value functions.

The *length of a tour-set $S \subseteq V$*, denoted by $d(S)$, is defined as the sum of the values of its elements, i.e.

$$d(S) := \sum_{s \in S} s.$$

A tour τ on the cities $\{1, \ldots, n\}$ is denoted by the order in which the cities are visited, i.e. $\tau = (i_1, i_2, \ldots, i_n)$. The length of a tour τ is given by

$$d(\tau) := d[i_n, i_1] + \sum_{t=1}^{n-1} d[i_t, i_{t+1}].$$