Journal of Combinatorial Optimization, 9, 381–399, 2005

Robotic-Cell Scheduling: Special Polynomially Solvable Cases of the Traveling Salesman Problem on Permuted Monge Matrices

VLADIMIR G. DEĬNEKO v.deineko@warwick.ac.uk
Warwick Business School, The University of Warwick, Coventry CV4 7AL, UK

GEORGE STEINER steiner@mcmaster.ca
ZHIHUI XUE xzhm@yahoo.com
DeGroote School of Business, McMaster University, Hamilton, Ontario L8S 4M4, Canada

Received January 28, 2004; Revised March 8, 2005; Accepted April 8, 2005

Abstract. In this paper, we introduce the $1-K$ robotic-cell scheduling problem, whose solution can be reduced to solving a TSP on specially structured permuted Monge matrices, we call *b*-decomposable matrices. We also review a number of other scheduling problems which all reduce to solving TSP-s on permuted Monge matrices. We present the important insight that the TSP on *b*-decomposable matrices can be solved in polynomial time by a special adaptation of the well-known subtour-patching technique. We discuss efficient implementations of this algorithm on newly defined subclasses of permuted Monge matrices.

Keywords: robotic-cell scheduling, traveling salesman problem, permuted Monge matrix, polynomial-time algorithm

1. Introduction

Robots and cellular manufacturing are widely used in modern production systems. In cellular manufacturing, a group of similar parts is to be produced together in a specialized and automated manufacturing cell. Typically, the manufacturing cell consists of a small number of versatile machines that can perform a variety of tasks. A *robotic cell* is a cluster of machines arranged within the reach of a robot, which is used to load, unload, and move parts between machines. There are a number of issues to be considered for a robotic cell, e.g., cell design, robot movement, part processing sequence, to name just a few. In this paper, we study scheduling problems in certain types of robotic cells. As most frequently there is only one robot in a cell in the normal setting, it is often the bottleneck of systems. Therefore, unlike traditional scheduling models in which we are looking for the processing sequence of parts, *both* the robot activities and the part processing sequence should be optimized in order to improve the productivity of these automated systems.

High-volume manufacturing environments are often controlled by *cyclic production*. This means that instead of sequentially processing a large batch of each part, a smaller set of parts is loaded into the system and processed repetitively. For example, consider the need

to process 2,000 units of part A, 4,000 units of part B, and 3,000 units of part C in a day. A part mix ratio or *minimal part set* (MPS) is calculated as {2A, 4B, 3C}—namely, two of A, four of B, and three of C—which is the smallest set of parts in proportion to the day's production. Then the MPS is fed into the system and produced 1,000 times to fulfill the production target. In this context, we develop the class of *cyclic schedules* that perform each required operation on an MPS exactly once. When such a schedule is formed, it will be identically repeated at regular intervals. To measure the performance of such a repetitive manufacturing system, where there is no need to track each individual order, one often-used objective is to minimize the *cycle time* of an MPS—the time between completions of successive MPS-s, which is equivalent to maximizing the throughput rate over the long run. Due to its simplicity in management and control, cyclic production is suitable for producing large quantities of different parts which have small setup cost.

The *traveling salesman problem* (TSP) is one of the most widely studied problems in combinatorial optimization. Simply, the problem may be stated as follows: Given a collection of "cities," find the shortest tour that visits all of them exactly once and returns to the starting city. The TSP belongs to the class of difficult optimization problems, as it is strongly $\mathcal{NP}$-hard. Nevertheless, many special cases of it can be solved in polynomial time when the distance matrix satisfies certain properties. For a comprehensive review of the extensive results on this subject, the interested reader is referred to the earlier survey by Gilmore et al. (1985), the papers by Burkard et al. (1998) and Kabadi (2002).

The TSP plays an important role in applications like production scheduling. A large number of scheduling problems can be formulated as special cases of the TSP, and many well-solvable cases of the TSP originate from scheduling problems. For example, we know that a special case of the TSP with distance matrix $C = (c_{ij}) = \max\{f_i, e_j\}$ can be solved by the Gilmore-Gomory algorithm (Gilmore and Gomory, 1964), which was originally designed to solve the problem of sequencing jobs on a one-state-variable machine. This also can be used to find the minimum makespan in the two-machine no-wait flow-shop scheduling problem in $O(n \log n)$ time (Reddi and Ramamoorthy, 1972). Thus investigating special cases of the TSP that can be solved by polynomial algorithms is of great practical significance.

An $n \times n$ matrix $C = (c_{ij})$ is a *Monge* (*distribution*) *matrix*, if it fulfills the so-called Monge property:

$$c_{ij} + c_{i'j'} \leq c_{ij'} + c_{i'j} \quad \text{for all } 1 \leq i < i' \leq n \text{ and } 1 \leq j < j' \leq n.$$

Furthermore, $C = (c_{ij})$ is a *permuted Monge* (*distribution*) *matrix*, if there exists a permutation ϕ such that $C^{\phi} = (c_{i\phi(j)})$ is a Monge matrix. These matrices can be recognized and their permutation ϕ can be found in $O(n^2)$ time (Burkard et al., 1998). The Monge property has received considerable attention in combinatorial optimization (Burkard et al., 1996), as its particular structure often leads to easier solutions for problems. For instance, for the linear assignment problem with a Monge cost matrix, the *identity permutation* $\phi(i) = i$ for $i = 1, 2, \ldots, n$ is an optimal solution. Based on this fact, if ϕ is the permutation for which C^{ϕ} is a Monge matrix, then ϕ is an optimal assignment for C. See e.g. Gilmore et al. (1985) for details.

There are many previously studied robotic-cell scheduling problems where determining the minimum cycle time is based on solving special cases of the TSP on certain permuted Monge matrices. For a two-machine bufferless robotic cell which uses only a fixed robot move cycle, Sethi et al. (1992) show that the minimum cycle time can be determined by solving an auxiliary TSP using the Gilmore-Gomory algorithm (Gilmore and Gomory, 1964). Hall et al. (1997) extend this result to a general two-machine bufferless robotic cell with n parts by showing that, through the repeated use of the Gilmore-Gomory algorithm, one can find the optimal cycle time and part sequence in $O(n^4)$ time, even with multiple robot move cycles and parts. Aneja and Kamoun (1999) show that this last problem can be solved by finding a minimum tour in an n-city TSP with distance matrix $C = (c_{ij}) = \min\{b_i + a_j, \max\{\mu, b_i, a_j\}\}$, where μ, b_i, a_j are given non-negative numbers. They also claim to reduce the complexity of the solution to $O(n \log n)$ by solving a complicated set of auxiliary problems, again using the Gilmore-Gomory algorithm repeatedly. For recent surveys on cyclic scheduling in robotic cells with various configurations, we refer the reader to Crama et al. (2000), Middendorf and Timkovsky (2002), Xue (2004), and Steiner and Xue (2005).

In this paper, we introduce the $1 - K$ robotic-cell scheduling problem. It can be shown (Steiner and Xue, 2002; Xue, 2004) that this problem also reduces to solving an n-city TSP with a specially structured permuted Monge distance matrix which can be defined as follows: $\tilde{C} = (\tilde{c}_{ij}) = \min\{\lambda + \eta + Kb_i + a_j, \max\{\lambda + (K-1)b_i + \max\{\eta + \rho, b_i\}, a_j\}\}$, where all parameters are given non-negative numbers. We present an $O(n \log n)$ algorithm to solve the TSP with distance matrix $\tilde{C}$. Notice that when $K = 1$ and $\lambda = \eta = 0$, then this last distance matrix reduces to the matrix of Aneja and Kamoun (1999). As it often happens in cases when a general approach to tackle a problem is found, our approach not only solves the problem for a much larger class of instances, but it is also easier to understand and implement. The Aneja-Kamoun algorithm for the special case mentioned above contains a very large number of branches with calls to the Gilmore-Gomory algorithm using different auxiliary distance matrices. The algorithm is extremely difficult to understand and validate. Our approach uses an *important insight*, by showing that both matrices share the newly defined crucial property of 'b-decomposability' that we introduce later. Based on this insight, we define a hierarchy of subclasses within the class of permuted Monge matrices, and show that the TSP is polynomially solvable on each of these subclasses. As a result, we present a new $O(n \log n)$ algorithm for solving the TSP on the smallest one of these subclasses. We also describe an efficient implementation of our algorithm for the largest subclass, which has an $O(n^2)$ time complexity.

The remainder of the paper is organized as follows. In Section 2, we describe in more detail the $1 - K$ robotic-cell scheduling problem, which triggered our investigations of special classes of the TSP. In Section 3, we introduce several preliminary definitions and review the theory of subtour patching. In Section 4, we present the main structural insight and the algorithm, which enable us to solve the TSP with a 'b-decomposable' distance matrix. We also demonstrate our new algorithm on an example in the paper. The detailed calculations for this example are contained in the Appendix. Section 5 contains our final remarks.

2. Scheduling a $1 - K$ robotic cell

We introduce a new scheduling problem, which is a generalization of the classical robotic-cell scheduling model. In a flexible manufacturing cell, there are two machines, M_1 and M_2, and a robot, all without buffer. The parts are available at the input station (*In*) at time zero. Each part i ($i = 1, 2, \ldots, n$) consists of $K \geq 1$ identical components to be processed *together* first on machine M_1 for a_i time, then processed on machine M_2 *item-by-item*, each component requiring b_i processing time. As each part involves one processing step at the first machine and K processing steps at the second machine, we call this type of processing $1 - K$ *processing*.

The robot first picks up each part at *In*, then moves it to the first machine, M_1, and loads it on that machine for processing; after the processing is completed, the robot unloads the part and moves it to the second machine, M_2, for processing its individual components. Since the components need the robot to perform the loading and unloading tasks at M_2 before or after each of the K processing steps, these K steps cannot be combined into one single step. After all of its components are processed on M_2, the robot moves the finished part to an output station (*Out*) and drops it there. During the entire process, both the operations of the robot and the processing of parts (components) on machines are nonpreemptive. The K components of a part are handled as a whole except at machine M_2. *In*, M_1, M_2, and *Out* are located on the arc of a circle or on a straight line with the robot at the center. Figure 1 depicts the movement of a part (or component) in such a two-machine robotic cell. The input and output stations have unlimited storage capacity. Since there is no buffer space on machines, any part (component) being produced must be either on one of the machines or on the robot. Neither a machine nor the robot can handle more than one part at a time.

This model has a variety of applications. In particular, our study was inspired by a real-life application in an automated pharmaceutical laboratory in which a large number of samples need to go through a given chemical process. Each sample is a set of test tubes. Sample i ($1 \leq i \leq n$) is composed of K test tubes. After a sample is picked up from an input station, it is moved to an instrument (M_1)—a versatile and flexible (multifunctional) machine—for a chemical process. When the process finishes, the sample is moved to the reader (M_2) to measure the data for each test tube (one by one). When all measurements are taken, the sample is dropped at an output station. To realize automation and increase efficiency, a robot performs all moving, loading and unloading tasks during the entire process.

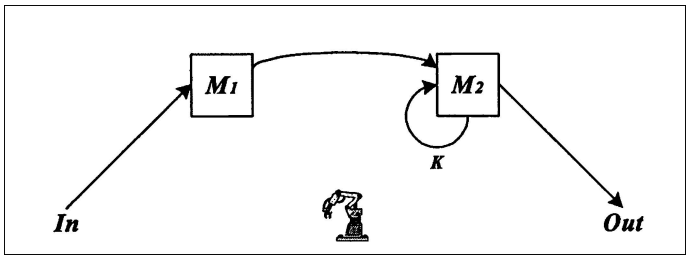

Figure 1. $1 - K$ processing in a two-machine robotic cell.

This model is also encountered in semiconductor test facilities consisting of burn-in ovens and testers in sequence, where the oven is viewed as a batch processing machine and testers are modeled as a unit-capacity machine. A batch processing machine is one where a number of components (jobs) are processed together as a batch. Examples of batch processing machines in semiconductor manufacturing are etchers in wafer fab and burn-in ovens in final test. Here the processing of each batch consists of two stages. The first stage is undertaken on the batch processing machine common to all components in a batch. All of these components start and finish processing at the same time. The second stage is undertaken on the unit-capacity machine for one component at a time. Components are regarded as single indivisible entities. Thus, once the processing of a component on a machine has started, this operation must be completed before this component can be processed on any other machine.

We have examined the problem of determining the part processing sequence in the MPS and the corresponding robot move activities, which jointly minimize the steady-state cycle time required for repeated production of the MPS in the aforementioned $1-K$ robotic cell. We have established that an optimal solution can be found by solving an n-city TSP with distance matrix $\tilde{C} = (\tilde{c}_{ij}) = \min\{\lambda + \eta + Kb_i + a_j, \max\{\lambda + (K-1)b_i + \max\{\eta + \rho, b_i\}, a_j\}\}$, where $\lambda = 2(K-1)\varepsilon$, $\eta = 2\varepsilon + 2\delta$, $\rho = 2\delta$, ε is the time needed by the robot to pick up, load or unload a part (or component) and δ is the travel time of the robot between adjacent locations in the cell. The technical details of this reduction can be found in Steiner and Xue (2002) and Xue (2004). Using extensive case analysis, it is also shown there that $\tilde{C}$ is a permuted Monge matrix. This application has served as our motivation to study the TSP on permuted Monge matrices. In the next section, we briefly review some relevant preliminary definitions and results for this problem.

3. Preliminary definitions and results

3.1. Permutations

For any n-city TSP with a given $n \times n$ distance matrix $C = (c_{ij})$, we denote the set of cities by $\{1, 2, \ldots, n\}$. A *permutation* ϕ on these n elements, which may be written as

$$\phi = \begin{pmatrix} 1 & 2 & \cdots & n \\ \phi(1) & \phi(2) & \cdots & \phi(n) \end{pmatrix},$$

corresponds to an assignment $i \to \phi(i)$ for $i = 1, 2, \ldots, n$ with the associated cost $c(\phi) = \sum_{i=1}^{n} C_{i\phi(i)}$. An *optimal assignment* is one that minimizes the cost. An assignment can also be expressed as a directed graph with arcs $(i, \phi(i))$ for $i = 1, 2, \ldots, n$, representing the fact that $\phi(i)$ is visited immediately after city i. If the associated graph is connected, then ϕ forms a (TSP) tour, and ϕ is said to be a *cyclic permutation*. Otherwise, ϕ consists of several subtours.

We can modify an assignment ϕ by multiplying it with a permutation ψ, which produces a new assignment ϕ' defined as $\phi'(i) = \phi \circ \psi(i) = \phi(\psi(i))$ for $i = 1, 2, \ldots, n$. A *transposition* $\langle i, j \rangle$ is a permutation that interchanges i and j. An *adjacent transposition*

is of the form $\langle i, i+1\rangle$. Performing $\langle i, j\rangle$ on the permutation ϕ yields the permutation $\phi' = \phi \circ \langle i, j\rangle$ with $\phi'(i) = \phi(j)$, $\phi'(j) = \phi(i)$ and $\phi'(k) = \phi(k)$ for $k \neq i, j$. Recall that the product of two transpositions is a non-commutative operation if they have a common index, otherwise it is commutative. For a given permutation ϕ, the *cost of performing transposition* $\langle i, j\rangle$ on ϕ is defined by $c_\phi(i, j) = c(\phi \circ \langle i, j\rangle) - c(\phi) = c_{i\phi(j)} + c_{j\phi(i)} - c_{i\phi(i)} - c_{j\phi(j)}$.

For the TSP with a Monge distance matrix, there exists an optimal tour which is pyramidal. If we number the cities by $1, 2, \ldots, n$, then a tour is called *pyramidal*, if starting from the initial city 1, they are first visited in increasing order of their index, and then the remaining cities are visited in decreasing order. For example, the six-city tours (1, 3, 5, 6, 4, 2) i.e., $1 \to 3 \to 5 \to 6 \to 4 \to 2 \to 1$, and (1, 4, 5, 6, 3, 2) are pyramidal, but the tour (1, 5, 3, 6, 4, 2) is not. For any distance matrix $C = (c_{ij})$, a shortest pyramidal tour can be found by an efficient dynamic programming scheme in $O(n^2)$ time. See e.g. Gilmore et al. (1985) for details. Further improvement in the time complexity can be achieved if the distance matrix C is a Monge matrix. By exploiting the combinatorial structure of Monge matrices, Park (1991) showed that the calculation can be speeded up so that the TSP on Monge matrices is solvable in *O(n)* time.

3.2. Review of the theory of subtour patching

A matrix $C = (c_{ij})$ of the form $c_{ij} = a_i b_j$ with real numbers a_i and b_j, $1 \leq i, j \leq n$, is called a *product matrix*. Sarvanov (1980) proved that the TSP on product matrices is *NP*-hard. Since product matrices are contained in the more general class of permuted Monge matrices, it follows immediately that the TSP with a permuted Monge matrix is also *NP*-hard. Nevertheless, as we shall describe in detail later, the TSP is solvable in polynomial time on certain subclasses of permuted Monge matrices, if they satisfy some additional crucial properties. As the solution technique is based on the theory of *subtour patching*, we briefly review its important points that will be used in this paper; the interested reader is referred to Gilmore et al. (1985) and Burkard et al. (1998) for a more comprehensive coverage.

In general, the strategy of subtour patching works as follows: First solve an assignment problem for the given distance matrix. If the optimal assignment ϕ is a tour, it is clearly optimal for the underlying TSP, as the cost of ϕ is a lower bound on the length of an optimal tour. Otherwise, the assignment, ϕ consists of $r \geq 2$ subtours $\phi_1, \phi_2, \ldots, \phi_r$. In this case, patch the subtours together by a series of transpositions so as to yield an optimal solution for the TSP. More precisely, if i and j are in different subtours of ϕ, then performing the transposition $\langle i, j\rangle$ on ϕ will patch these two subtours into a single tour. A series of transpositions, in any order, can be expressed as a single permutation by forming their product. For an optimal assignment ϕ, a permutation ψ is called a *patching permutation* if $\phi \circ \psi$ is a cyclic permutation (tour). Thus the problem is, "Given an optimal assignment ϕ, find an *optimal patching permutation* ψ^* such that $\phi \circ \psi^*$ is an optimal tour."

In order to determine an optimal patching permutation, it is often useful to examine the so-called *patching graph*. With respect to a given optimal assignment ϕ, a patching graph $G_\phi = (V, E)$ may be constructed as follows: The vertices are the subtours ϕ_i of ϕ, $1 \leq i \leq r$. Every edge $e \in E$ corresponds to an adjacent transposition $\langle i, i+1\rangle$, that is, if city i is in subtour ϕ_j and city $i+1$ is in subtour ϕ_k, $j \neq k$, then the two

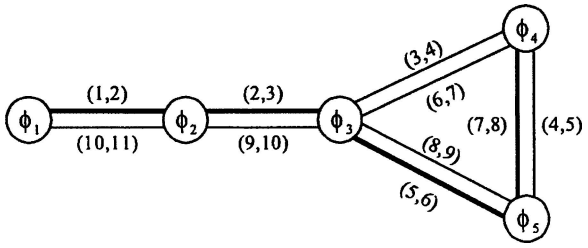

Figure 2. A patching graph $G_\phi = (V, E)$.

corresponding vertices in G_ϕ are connected by an edge labeled $(i, i+1)$. For the sake of simplicity, we will refer to edge $(i, i+1)$ as $\overline{i}$. Since the same pair of vertices may be connected by multiple edges, this construction will yield a connected *multigraph* with r vertices and at most $n-1$ edges. As an example, suppose we have an 11-city TSP with the optimal assignment $\phi = \phi_1\phi_2\phi_3\phi_4\phi_5$ = (1,11)(2,10) (3,6,9)(4,7)(5,8). The corresponding patching graph $G_\phi = (V, E)$ is shown in figure 2, where $V = \{\phi_i : 1 \le i \le 5\}$ and $E = \{(i, i+1) : 1 \le i \le 10\}$.

A *spanning tree* T of a graph G is a tree connecting all its vertices. A permutation obtained by multiplying a set of adjacent transpositions which correspond to a spanning tree in G_ϕ is called a *tree permutation*. Gilmore and Gomory (1964) have shown that every tree permutation is a patching permutation. For example, the edges $\overline{1}, \overline{2}, \overline{5}$ and $\overline{7}$ form a spanning tree in G_ϕ of the above example (see the bold lines of figure 2). Thus the product of the transpositions $\langle 1, 2\rangle$, $\langle 2, 3\rangle$, $\langle 5, 6\rangle$ and $\langle 7, 8\rangle$, in *any* order, forms a tree permutation, which patches together the subtours of ϕ into a tour. With each edge $\overline{i}$ in G_ϕ, we associate a non-negative *weight* w_i that represents the cost of performing the corresponding transposition $\langle i, i+1\rangle$ on ϕ, i.e., $w_i = c(\phi \circ \langle i, i+1\rangle) - c(\phi) = c_\phi(i, i+1)$. The *weight* $w(T)$ of a *tree* T is defined as the sum of the weights of the edges in T. A *minimum-weight spanning tree*, or *MST* for short, is a spanning tree of G whose weight is minimum.

Recall that if a permutation (assignment) ψ consists of t subtours $\psi_1, \psi_2, \ldots, \psi_t$, then

$$c(\phi \circ \psi) - c(\phi) = \sum_{i=1}^{t} (c(\phi \circ \psi_i) - c(\phi)). \tag{1}$$

(See Theorem 14 in Gilmore et al., 1985). Corresponding to a given spanning tree T, we define ψ_T as a patching permutation that minimizes (1). To solve a special case of the TSP, Gilmore and Gomory (1964) developed a subtour-patching strategy that uses only adjacent transpositions with minimum total cost, which can be found by determining an MST for the patching graph. Burdyuk and Trofimov (1976) and Gilmore et al. (1985) proved that this basic patching strategy of using only adjacent transpositions is extendible also to permuted Monge matrices. Their result is essentially contained in the following theorem.

Theorem 1 (Burdyuk and Trofimov 1976, Gilmore et al. 1985)**.** *Let* $C^\phi = (c_{i\phi(j)})$ *be a Monge matrix. For any cyclic permutation* π*, there exists a spanning tree* $T =$

$\{\overline{i_1}, \overline{i_2}, \cdots, \overline{i_{r-1}}\}$ of the patching graph G_ϕ and a sequence σ for performing the transpositions of T such that the permutation $\phi_T = \phi \circ \langle i_{\sigma(1)}, i_{\sigma(1)} + 1\rangle \circ \langle i_{\sigma(2)}, i_{\sigma(2)} + 1\rangle \circ \cdots \circ \langle i_{\sigma(r-1)}, i_{\sigma(r-1)} + 1\rangle$ is a cyclic permutation with $c(\phi_T) \le c(\pi)$.

Theorem 1 is important as it allows us to restrict our search for an optimal patching permutation only to those permutations which can be formed of adjacent transpositions of G_ϕ, but it *does not* say anything about how to find the spanning tree T corresponding to these transpositions and the sequence σ in which they have to be multiplied. In the following we take a closer look at these problems. A set of edges in G_ϕ is said to be *dense* if it is of the form $\{\bar{i}, \overline{i+1}, \ldots, \bar{j}\}$ with $j \ge i$. Let T be a spanning tree for the patching graph G_ϕ. We partition the set of edges (i.e., the transpositions) of T into t ($1 \le t \le r-1$) dense, pairwise disjoint subsets $\mathcal{I}(i_1, j_1) = \{\bar{i}_1, \overline{i_1+1}, \ldots, \overline{j_1}\}, \mathcal{I}(i_2, j_2) = \{\bar{i}_2, \overline{i_2+1}, \ldots, \overline{j_2}\}, \ldots, \mathcal{I}(i_t, j_t) = \{\bar{i}_t, \overline{i_t+1}, \ldots, \overline{j_t}\}$, which will be called *branches* hereafter, such that $T = \mathcal{I}(i_1, j_1) \cup \mathcal{I}(i_2, j_2) \cup \cdots \cup \mathcal{I}(i_t, j_t)$ and $j_k + 1 < i_{k+1}$ for $k = 1, 2, \ldots, t-1$. Refer to the example illustrated in figure 2. Suppose again that $T = \{\overline{1}, \overline{2}, \overline{5}, \overline{7}\}$. Then T is composed of three branches with $\mathcal{I}(1, 2) = \{\overline{1}, \overline{2}\}, \mathcal{I}(5, 5) = \{\overline{5}\}$, and $\mathcal{I}(7, 7) = \{\overline{7}\}$. Since $j_k + 1 < i_{k+1}$ for any two branches $\mathcal{I}(i_k, j_k)$ and $\mathcal{I}(i_{k+1}, j_{k+1})$, $k = 1, 2, \ldots, t-1$, the permutations corresponding to transpositions of different branches have no common element and thus are commutative and can be performed independent of each other in any order. Furthermore, since the patching costs of these permutations are additive by (1), the best patching permutation corresponding to T can be identified by finding the minimum cost permutation for each branch and taking the product of these. It is well known that, as branches are dense, performing transpositions of a branch in any order will yield a pyramidal subtour. In summary, the best patching permutation ψ_T can be derived by constructing a shortest pyramidal subtour for each branch $\mathcal{I}(i, j)$ of T. Since finding the shortest pyramidal tour on a given set of vertices is solvable in polynomial time, the remaining hard part of the problem is how to find the best spanning tree.

Let us now define the *b-weight w^b_{ij} of a branch $\mathcal{I}(i, j)$* by

$$w^b_{ij} = c(\phi \circ \psi^*_i) - c(\phi),$$

where ψ^*_i is a shortest pyramidal subtour corresponding to the branch $\mathcal{I}(i, j)$. Also, we define the *weight* w_{ij} of a *branch* $\mathcal{I}(i, j)$ as the total weight of the edges in the branch, i.e., $w_{ij} = \sum_{k=i}^{j} w_k$. Note that from these definitions, we have $w^b_{ii} = w_{ii} = w_i = c_\phi(i, i+1)$ for a branch $\mathcal{I}(i, i)$ that contains only edge $\bar{i}$. Further, we define the *b-weight $w^b(T)$ of a spanning tree T* as the total b-weight of its branches, i.e., $w^b(T) = \sum_{k=1}^{t} w^b_{i_k j_k}$. It is easy to verify from the definitions of ψ_T and $w^b(T)$ that $w^b(T) = c(\phi \circ \psi_T) - c(\phi)$. As $c(\phi)$ is constant, ψ_T is an optimal patching permutation *if and only if* the corresponding spanning tree T has minimum b-weight. In other words, the best spanning tree is actually a *minimum-b-weight spanning tree* of G_ϕ. Thus, the TSP on permuted Monge matrices is essentially reduced to the problem of finding a spanning tree of G_ϕ with minimum b-weight.

If the distance matrix C is a permuted Monge matrix, then it has been established by Burkard et al. (1998) that for any branch $\mathcal{I}(i, j)$, we have $w^b_{ij} \ge w^b_{ik} + w^b_{k+1,j}$ for $1 \le i \le n-1$ and $i \le k < j \le n-1$, and $w^b_{ij} \ge \sum_{k=i}^{j} w_k (= w_{ij})$ for $1 \le i \le j \le n$,

i.e., the b-weight is *super-additive*. This further implies that for a spanning tree T its b-weight is never lower than its weight, i.e., $w^b(T) \geq w(T)$. We call a permuted Monge matrix *b-weight-additive* if the b-weight of any branch is additive, i.e., $w^b_{ij} = \sum_{k=i}^{j} w_k$ for $1 \leq i \leq j \leq n$. It is important to note that these matrices can be recognized in $O(n^2)$ time combining Park's (1991) method for computing the b-weights with the algorithm for recognizing permuted Monge matrices. From the definition of an MST and Theorem 1, we know that the cost of the optimal assignment ϕ plus the weight of the MST $\hat{T}$ for G_ϕ, i.e., $c(\phi) + w(\hat{T})$, is a lower bound on the length of an optimal tour. Consequently, if the subtours of ϕ can be patched with cost $w(\hat{T})$, the resulting tour is clearly optimal, and we are done. Since for b-weight-additive matrices, the b-weight of a spanning tree is the same as its weight, the MST is the best patching tree in this case. It can be shown that the b-weight-additivity property is satisfied by Gilmore–Gomory matrices (Burkard et al., 1998). Hence, Gilmore–Gomory matrices form a subclass of the b-weight-additive matrices.

4. Polynomially solvable classes

4.1. b-decomposable matrices

Consider an $n \times n$ permuted Monge matrix $C = (c_{ij})$ such that $D = (d_{ij})$ with $d_{ij} = c_{i\phi(j)}$ is a Monge matrix. We call C *b-decomposable* if D can be partitioned by an index $i_0 (1 \leq i_0 \leq n)$ into two b-weight-additive sub-matrices $D' = (d'_{ij})$ for $i, j \leq i_0$ and $D'' = (d''_{ij})$ for $i, j > i_0$. Note that if $i_0 = n$, then D'' is empty and D itself is a b-weight-additive matrix. We show in this section the important *insight* that the class of TSP with a b-decomposable distance matrix is solvable in polynomial time.

As we have already noted, this special case of the TSP originated from robotic-cell scheduling problems. It was shown in Steiner and Xue (2002) and Xue (2004) that the distance matrix $\tilde{C}$ is b-decomposable. To avoid overloading the paper with unnecessary technical details, we show this fact only for the special subclass $C = (c_{ij}) = \min\{b_i + a_j, \max\{\mu, b_i, a_j\}\}$ of $\tilde{C}$. Without loss of generality, we assume that the cities have been renumbered so that $b_1 \leq \cdots \leq b_n$. Let ϕ be an assignment (ordering) for which $a_{\phi(i)} \leq a_{\phi(i+1)}$. It can be shown that ϕ is an assignment for which $C^\phi = (c_{i\phi(j)})$ is a Monge matrix (Aneja and Kamoun, 1999). Observe that $b_i + a_{\phi(i)} \leq b_{i+1} + a_{\phi(i+1)}$ for all $i = 1, 2, \ldots, n-1$. If there exists an index i_0, $i_0 < n$, such that $b_{i_0} + a_{\phi(i_0)} \leq \mu < b_{i_0+1} + a_{\phi(i_0+1)}$, then the Monge matrix $D = C^\phi = (c_{i\phi(j)})$ can be split into two submatrices $D' = (d'_{ij})$ and $D'' = (d''_{ij})$, where D' is a *sum matrix*, i.e., $d'_{ij} = b_i + a_{\phi(j)}$ for $i, j \leq i_0$, and $d''_{ij} = \max\{\mu, b_i, a_{\phi(j)}\}$ for $i, j > i_0$ is a Gilmore–Gomory matrix. Thus C is b-decomposable indeed.

The Aneja-Kamoun algorithm is specifically designed for solving the TSP with the above distance matrix C and it cannot be adapted for solving the $1-K$ robotic-cell scheduling problem with distance matrix $\tilde{C}$, which was introduced in Section 2. However, both scheduling problems lead to matrices belonging to the *subclass* of b-decomposable matrices in which D' is a sum matrix. Of course, the class of b-decomposable matrices is larger, as a matrix in it can have *any* b-weight-additive submatrix for its two parts. Figure 3 depicts the hierarchy of these newly defined matrix classes.

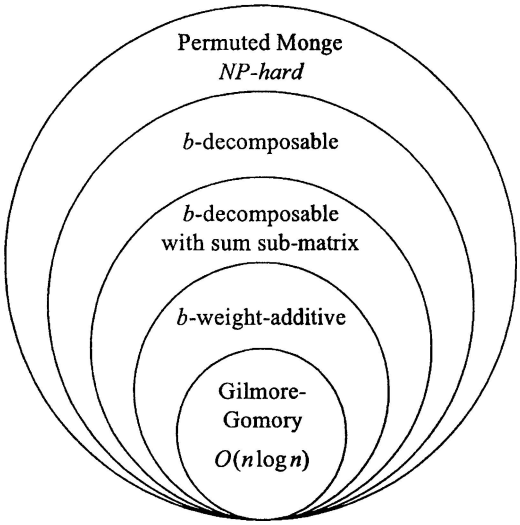

Figure 3. Hierarchical classes of permuted Monge matrices.

Let us return now to b-decomposable TSP-s in general. Since ϕ is the permutation for which $C^{\phi} = (c_{i\phi(j)})$ is a Monge matrix, ϕ is an optimal assignment for C, and ϕ is also optimal for the TSP in case it is a tour. As a result, hereafter, we concentrate on the case when ϕ consists of $r \geq 2$ subtours. Let $G_{\phi} = (V, E)$ be the patching graph relative to ϕ. We will use the subtour-patching technique by reformulating our TSP as a minimum-b-weight spanning tree problem.

For a given spanning tree T of G_{ϕ}, we begin with a characterization of the branches of T. Let $\mathcal{I}(i, j)$ be a branch of T. If $\mathcal{I}(i, j)$ does not contain the edge $\overline{i_0}$, then the weights of the whole edge set of $\mathcal{I}(i, j)$ are either in D' or in D'', and thus $w^b_{ij} = w_{ij}$, because both D' and D'' are b-weight-additive. Otherwise, it is known that $w^b_{ij} \geq w_{ij}$. As we will demonstrate later, the branch containing the edge $\overline{i_0}$, if it exists, is of primary interest among all branches of T, as it is the *only* branch which may not be b-weight-additive. From here on, we will refer to this branch as a *b-branch*.

Let us examine now specifically the MST $\hat{T}$ of G_{ϕ}. Note that during the construction of $\hat{T}$ by Kruskal's (1956) algorithm (see e.g. Ahuja et al., 1993), if there is an edge available with the same weight as edge $\overline{i_0}$, we consider that edge first. This manner of construction will ensure that $\hat{T}$ does not have a b-branch unless it is necessary. If $\hat{T}$ does not contain a b-branch, then the b-weight of $\hat{T}$ is the same as the weight of the tree, and the TSP is solved by using this MST as a patching tree. If $\hat{T}$ contains a b-branch, then it must be of the form $\mathcal{I}(h_0, j_0) = \{\overline{h_0}, \ldots, \overline{i_0 - 1}, \overline{i_0}, \overline{i_0 + 1}, \ldots, \overline{j_0}\}$ for some $h_0 \leq i_0$ and $j_0 \geq i_0$. Then the b-weight of $\hat{T}$ is calculated as the b-weight $w^b_{h_0, j_0}$ of the branch $\mathcal{I}(h_0, j_0)$ plus the weights of all edges in $\hat{T} \backslash \mathcal{I}(h_0, j_0)$. If $\mathcal{I}(h_0, j_0) = \{\overline{i_0}\}$ then $w^b_{h_0, j_0} = w_{h_0, j_0} = w_{i_0}$, implying that $w^b(\hat{T}) = w(\hat{T})$, which means $\hat{T}$ is a minimum-b-weight spanning tree. Thus we assume for the remainder of the discussion that $|\mathcal{I}(h_0, j_0)| > 1$. Clearly, $w^b_{h_0, j_0}$ can be

Table 1. The example instance.

Cities	1	2	3	4	5	6	7	8	9	10
a_i	100	120	10	32	130	90	30	110	20	39
b_i	5	15	25	36	45	83	95	106	117	125
$a_{\phi(i)}$	10	20	30	32	39	90	100	110	120	130
$\phi(i)$	3	9	7	4	10	6	1	8	2	5

obtained by using an algorithm for finding a shortest pyramidal tour on the set of cities $\{h_0, \cdots, i_0 - 1, i_0, i_0 + 1, \cdots, j_0, j_0 + 1\}$ with the corresponding distances extracted from the matrix $D = (d_{ij}) = (c_{i\phi(j)})$. Since D is a Monge matrix, this can be done in linear time by using Park's (1991) recursions. Note that in Park's recursions, by computing the length of a shortest pyramidal tour on the set of cities $\{1, 2, \ldots, n\}$, we can obtain at the same time the lengths of the shortest pyramidal tours for all sets of cities $\{1, 2, \ldots, j\}$ for $j = 2, \ldots, n$. Therefore, these recursions allow us to calculate in linear time all b-weights $w^b_{h_0,h_0+1}, w^b_{h_0,h_0+2}, \ldots, w^b_{h_0,j_0}$. Now let us compare the weight w_{h_0,j_0} of the branch $\mathcal{I}(h_0, j_0)$ with its b-weight. If $w^b_{h_0,j_0} = w_{h_0,j_0}$, then the b-weight of $\hat{T}$ is the same as the weight of the tree, and the problem is solved by using $\hat{T}$ as a patching tree. Otherwise, i.e., in the case when $w^b_{h_0,j_0} > w_{h_0,j_0}$, the MST $\hat{T}$ may not be the best spanning tree to patch the subtours of ϕ, as illustrated by the following example.

Example. Let us consider a 10-city TSP with the distance matrix $C = (c_{ij}) = \min\{b_i + a_j, \max\{\mu, b_i, a_j\}\}$, where $\mu = 80$. The b's and a's are given in Table 1. Observe that the b_i-s are in the order $b_i \le b_{i+1}$. Table 1 also shows the ordering ϕ for which $a_{\phi(i)} \le a_{\phi(i+1)}$. As discussed above, $C^\phi = (c_{i\phi(j)})$ is a Monge matrix and thus ϕ is an optimal assignment for C. Furthermore, it is easy to see that ϕ consists of six subtours $\phi = \phi_1\phi_2\phi_3\phi_4\phi_5\phi_6 = (1, 3, 7)(2, 9)(4)(5, 10)(6)(8)$, and the cost of ϕ is equal to $c(\phi) = \sum_{i=1}^n c_{i\phi(i)} = 803$. The patching graph G_ϕ, as shown in figure 4, has 6 vertices and 9 edges.

As $b_4 + a_{\phi(4)} < \mu$ and $b_5 + a_{\phi(5)} > \mu$, we have $i_0 = 4$, which means that $D' = (d'_{ij}) = b_i + a_{\phi(j)}$ for all $i, j \le 4$, and $D'' = (d''_{i,j}) = \max\{\mu, b_i, a_{\phi(j)}\}$ for all $i, j > 4$. The weights of the edges as determined by $w_i = c_{i\phi(i+1)} + c_{i+1,\phi(i)} - c_{i\phi(i)} - c_{i+1,\phi(i+1)}$ are as follows:

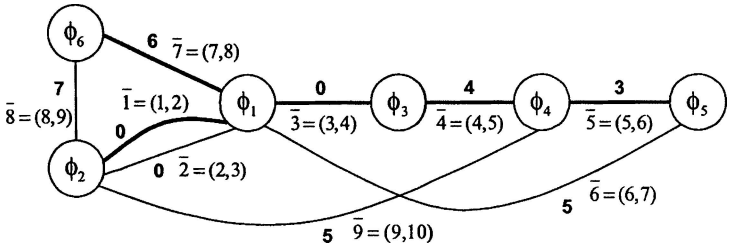

Figure 4. The patching graph G_ϕ in the example (numbers in bold are edge-weights).

$w_1 = w_2 = w_3 = 0$, $w_4 = 4$, $w_5 = 3$, $w_6 = 5$, $w_7 = 6$, $w_8 = 7$, and $w_9 = 5$. It is easy to see that $\hat{T} = \{\overline{1}, \overline{3}, \overline{4}, \overline{5}, \overline{7}\}$ is an MST in G_ϕ, and it is indicated in bold line in figure 4. The spanning tree contains three branches $\mathcal{I}(1, 1)$, $\mathcal{I}(3, 5)$ and $\mathcal{I}(7, 7)$. According to the patching scheme, we have to calculate the b-weight of the tree, which is the sum of the three branches' b-weights. The b-weight of $\mathcal{I}(1, 1)$ and $\mathcal{I}(7, 7)$ is 0 and 6, respectively. With the corresponding distances extracted from the matrix $D = (d_{ij}) = (c_{i\phi(j)})$, it is not difficult to find that (3, 5, 6, 4) is a shortest pyramidal tour on the set of cities {3, 4, 5, 6}, and the tour length is 303. (See the Appendix for the detailed calculations.) Thus, the b-weight of $\mathcal{I}(3, 5)$, which is a b-branch, can be calculated as $w^b_{3,5} = 303 - \sum_{i=3}^{6} c_{i\phi(i)} = 303 - 55 - 68 - 80 - 90 = 10$. Unfortunately, the inequality $w^b_{3,5} > w_3 + w_4 + w_5 = 7$ holds for this branch. In this case, another spanning tree may have a lower b-weight. For instance, the spanning tree $T' = \{\overline{2}, \overline{3}, \overline{5}, \overline{6}, \overline{8}\}$ has a b-weight of 15, which is lower than $w^b(\hat{T}) = w_1 + w^b_{3,5} + w_7 = 16$. (Since T' contains no edge $\overline{i_0}$, its b-weight can easily be calculated as $w^b(T') = w(T') = 15$.)

The above example demonstrates that the MST $\hat{T}$ may not have minimum b-weight if $w^b_{h_0, j_0} > w_{h_0, j_0}$. Thus we need to determine a minimum-b-weight spanning tree T^* for this case, which may contain a b-branch. Of course, if we knew which of the b-branches has to be included in the patching tree, the other edges could easily be found by any of the greedy algorithms for finding an MST, such as Kruskal's (1956) algorithm. Hence, our focus here is on the selection of the b-branch. A straightforward strategy would be to search through all possible b-branches, which would lead to an $O(n^3)$ algorithm. Before we present our more efficient search procedure, we note that we cannot restrict the choices of the b-branch only to sub-branches of $\mathcal{I}(h_0, j_0)$—the b-branch of the MST $\hat{T}$. For example, if we decide to remove edge $\overline{h_0}$ from the tree $\hat{T}$ (and from the branch $\mathcal{I}(h_0, j_0)$), a new edge has to be added to form a spanning tree in the patching graph. If the edge selected happens to be the edge $\overline{j_0 + 1}$, its addition will create a new branch $\mathcal{I}'(h_0 + 1, j_x)$ with $j_x > j_0$. For instance, in the example above, if we remove edge $\overline{3}$ from $\hat{T}$ and add edge $\overline{6}$, a new branch $\mathcal{I}'(4, 7) = \{\overline{4}, \overline{5}, \overline{6}, \overline{7}\}$ will be created.

A b-branch must start with an edge $\overline{h}$ with $1 \le h \le i_0$. We call such a branch a b_h-*branch*. Let $h_{\min}$ be the minimal possible index of the starting edge of a b-branch in the patching graph G_ϕ. The b-branches can then be chosen exclusively from among b_h-branches with $h_{\min} \le h \le i_0$. It is clear that $h_{\min}$ can be easily determined as the smallest index $h (h \ge 1)$ for which $\{\overline{h}, \overline{h+1}, \ldots, \overline{i_0}\}$ is a cycle-free path in G_ϕ. For the example above, since it will create a cycle $1 \to 2 \to 1$ when edge $\overline{1}$ is inserted, we have $h_{\min} = 2$.

We are now ready to state our search strategy: For each $h, h_{\min} \le h \le i_0$, first find a minimum-$b$-weight spanning tree T^b_h in $G_\phi \backslash \{\overline{h-1}\}$ among all trees containing a b_h-branch (we define $G_\phi \backslash \{\overline{0}\} = G_\phi$). After this, find the minimum-b-weight spanning tree T^b among the T^b_h-s.

Next we give details of a linear-time procedure to find the tree T^b_h. This means that the tree T^b can be obtained in $O(n^2)$ time. To find an MST when the underlying graph changes, we use the following lemma from Ahuja et al. (1993).

Lemma 2. *Let T be an MST for a graph G. If an edge (i, j) of T is removed from G, which results in a graph $G' = G \backslash \{(i, j)\}$ and two components of T containing sets of vertices V'*

and V'', then $T' = T \cup \{(i', j')\}\backslash\{(i, j)\}$ is an MST for G', where (i', j') is an edge with the minimum weight in the cut $[V', V'']$ of G'.

Proof: The proof follows from the "cut optimality conditions" for a minimum-weight spanning tree. That is, for every edge (i, j) of T', its weight is not larger than the weight of any edge (k, l) contained in the cut of G' formed by deleting edge (i, j) from T'. See e.g. Ahuja et al. (1993) for details. □

Given h, $h_{\min} \le h \le i_0$, we first find, if possible, a spanning tree $\hat{T}_h$ in $G'_\phi = G_\phi\backslash\{\overline{h-1}\}$ with minimum weight $w(\hat{T}_h)$ among all trees containing the edges $\overline{h}, \overline{h+1}, \ldots, \overline{i_0}$: Start with the initial working sub-tree $\{\overline{h}, \overline{h+1}, \ldots, \overline{i_0}\}$, then $\hat{T}_h$ can be determined by Kruskal's MST algorithm, which keeps adding to the tree the next smallest-weight edge from those that remain as long as it does not create a cycle. It should be noted here that the removal of edge $\overline{h-1}$ from G_ϕ may disconnect the patching graph, which would make finding a spanning tree in G'_ϕ impossible. Therefore, if $\hat{T}_h$ does not exist, then there is no T_h^b either; otherwise, let $\mathcal{I}(h, j_h) = \{\overline{h}, \overline{h+1}, \ldots, \overline{i_0}, \overline{i_0+1}, \ldots, j_h\}$ be the b-branch in $\hat{T}_h$. Then the b-weight of $\hat{T}_h$ can be calculated as the b-weight of the branch $\mathcal{I}(h, j_h)$ plus the weights of all edges in $\hat{T}_h\backslash\mathcal{I}(h, j_h)$, i.e., $w^b(\hat{T}_h) = w^b_{h,j_h} + w(\hat{T}_h) - \sum_{k=h}^{j_h} w_k$. For spanning trees of G_ϕ which contain a b_h-branch, there is a simple but very useful property as shown in the following lemma.

Lemma 3. *For any spanning tree T of G_ϕ that contains a branch $\mathcal{I}(h, j)$ such that $j \ge j_h$, its b-weight is not less than that of $\hat{T}_h$, i.e., $w^b(T) \ge w^b(\hat{T}_h)$.*

Proof: Notice that both $\hat{T}_h$ and T have the edges $\overline{h}, \overline{h+1}, \ldots, \overline{i_0}, \overline{i_0+1}, \ldots, \overline{j_h}$. Since $\hat{T}_h$ has minimum weight among all spanning trees containing a b_h-branch, the total weight of the remaining edges of T is not less than the same in $\hat{T}_h$—that is, $w(\hat{T}_h) - \sum_{k=h}^{j_h} w_k$. Moreover, as the b-weight fulfills the property $w^b_{ij} \ge w^b_{ik} + w^b_{k+1,j}$ for any $i \le k < j$, it is easy to verify that $w^b(T) \ge w^b_{h,j_h} + w(\hat{T}_h) - \sum_{k=h}^{j_h} w_k = w^b(\hat{T}_h)$. □

Lemma 3 tells us that we can restrict our search for the tree T_h^b only to $\hat{T}_h$ and trees containing sub-branches of $\mathcal{I}(h, j_h)$. We will take advantage of this observation in our search. If the b-weight of the branch $\mathcal{I}(h, j_h)$ is the same as its weight, i.e., $w^b_{h,j_h} = w_{h,j_h}$, then $T_h^b = \hat{T}_h$; otherwise, we assume without loss of generality that $j_h \ge i_0 + 1$. The search for T_h^b can be conducted by computing next, one by one, the b-weights for the MST-s containing branches $\mathcal{I}(h, j-1)$, $j = i_0+1, i_0+2, \ldots, j_h$. Notice that if we remove, for instance, edge $\overline{j}$ for some $j \in [i_0+1, j_h]$ from $\hat{T}_h$, then based on Lemma 2, the MST that contains the branch $\mathcal{I}(h, j-1) = \{\overline{h}, \overline{h+1}, \ldots, \overline{i_0}, \overline{i_0+1}, \ldots, \overline{j-1}\}$, if it exists, can be found by connecting two components of $\hat{T}_h\backslash\{\overline{j}\}$ by the edge $\overline{s(j)}$ with the minimum weight $w_{s(j)}$ in the cut between the two components of $\hat{T}_h\backslash\{\overline{j}\}$. Furthermore, it is easy to verify that the b-weight of this spanning tree is equal to $w(\hat{T}_h) - \sum_{k=h}^{j} w_k + w_{s(j)} + w^b_{h,j-1}$. As we already mentioned above, all b-weights $w^b_{h,i_0}, w^b_{h,i_0+1}, \ldots, w^b_{h,j_h}$ can be calculated in linear time. At this point, the only question left to be answered is how to find in linear time the "replacement" edges $\overline{s(j)}$ for all j with $i_0 + 1 \le j \le j_h$.

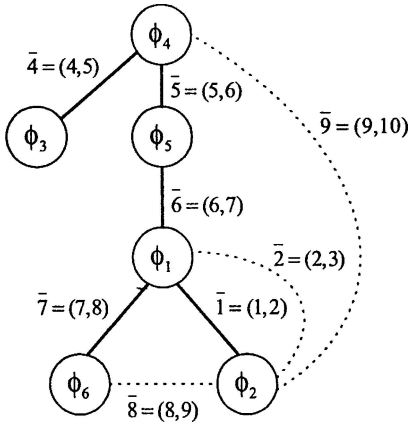

Figure 5. A rooted tree for edge replacement.

In order to identify the replacement edges, we represent the tree $\hat{T}_h$ as a rooted tree with the vertex (subtour) containing index $i_0 + 1$ as the *root*. For each vertex u in the tree $\hat{T}_h$, we assign a *pointer* $p(u)$ to be the largest index of the edges from $\mathcal{I}(h, j_h)$ on the path from the root to vertex u. For the vertices for which the path from the root does not contain any edge from $\mathcal{I}(h, j_h)$ at all, we define $p(u) = i_0$. For the root, we assign its pointer to be i_0. Now consider an edge $\overline{i}$ of $G_\phi \backslash \{\overline{h-1}\}$, which is not in the tree $\hat{T}_h$. Suppose it connects vertices v_1 and v_2 with $p(v_1) = h_1$ and $p(v_2) = h_2$, $h_1 \le h_2$. If $h_1 < h_2$, then this edge can be used as a replacement edge after removing any one of the edges $\overline{h_1+1}, \ldots, \overline{h_2}$; otherwise, i.e. when $h_1 = h_2$, it cannot be used as a replacement for any edge $\overline{j}$ with $i_0 + 1 \le j \le j_h$ at all, as it will not be in any cut between the two components of $\hat{T}_h \backslash \{\overline{j}\}$. Therefore, by looking through the sorted list of edges (e.g. in non-decreasing order of weight) not included in $\hat{T}_h$, we can identify a best replacement edge, if possible, for each edge $\overline{j}$ with $i_0 + 1 \le j \le j_h$. Thus clearly, the entire replacement search can be executed in linear time.

Let $h = 4$ in the example above (see also figure 5). It is easy to verify that an MST of $G_\phi \backslash \{\overline{h-1}\} = G_\phi \backslash \{\overline{3}\}$ that contains a b_h-branch is $\hat{T}_h = \hat{T}_4 = \{\overline{1}, \overline{4}, \overline{5}, \overline{6}, \overline{7}\}$. So $\mathcal{I}(h, j_h) = \mathcal{I}(4, 7)$. Note that edge $\overline{3}$ has been removed from the patching graph G_ϕ. The pointers p are defined as follows: $p(\phi_4) = 4$, $p(\phi_5) = 5$, $p(\phi_1) = 6$, $p(\phi_6) = 7$, $p(\phi_2) = 6$, and $p(\phi_3) = 4$. Here edges $\overline{2}$, $\overline{8}$ and $\overline{9}$ are not in the tree. For edge $\overline{2}$, as $p(\phi_1) = p(\phi_2)$, it cannot be used as replacement for any of the edges $\overline{5}$, $\overline{6}$ and $\overline{7}$. For edge $\overline{8}$, as $p(\phi_2) = 6$ and $p(\phi_6) = 7$, it could be a replacement for edge $\overline{7}$. For edge $\overline{9}$, as $p(\phi_4) = 4$ and $p(\phi_2) = 6$, it could be a replacement for edges $\overline{5}$ and $\overline{6}$. Now let us determine the minimum-b-weight spanning tree T_4^b in $G_\phi \backslash \{\overline{3}\}$ among all trees containing a b_4-branch. Using the dynamic programming scheme of Park, the b-weight of branches $\mathcal{I}(4, 4)$, $\mathcal{I}(4, 5)$, $\mathcal{I}(4, 6)$, and $\mathcal{I}(4, 7)$ can be computed as follows: $w^b_{4,4} = w_4 = 4$, $w^b_{4,5} = 10$, $w^b_{4,6} = 15$, and $w^b_{4,7} = 21$. (The detailed calculations are shown in the last paragraph of the Appendix.) Therefore, the tree $\hat{T}_4$ has a b-weight of $w^b(\hat{T}_4) = w_1 + w^b_{4,7} = 21$. It is easy to verify that the MST-s containing branches $\mathcal{I}(4, 4)$, $\mathcal{I}(4, 5)$ and $\mathcal{I}(4, 6)$ are $\{\overline{1}, \overline{4}, \overline{6}, \overline{7}, \overline{9}\}$, $\{\overline{1}, \overline{4}, \overline{5}, \overline{7}, \overline{9}\}$

and $\{\overline{1}, \overline{4}, \overline{5}, \overline{6}, \overline{8}\}$, respectively. Furthermore, the b-weight of these trees is 20, 21 and 22, respectively. Hence, we have $T_4^b = \{\overline{1}, \overline{4}, \overline{6}, \overline{7}, \overline{9}\}$ and $w^b(T_4^b) = 20$.

Similarly, we can find that $\hat{T}_3 = \{\overline{1}, \overline{3}, \overline{4}, \overline{5}, \overline{7}\}$ in $G_\phi \backslash \{\overline{2}\}$. The computation of the b-weight for branch $\mathcal{I}(3, 5)$ gives $w_{3,4}^b = 4$ and $w_{3,5}^b = 10$, see the Appendix for details. Replacing edge $\overline{5}$ with edge $\overline{6}$ in $\hat{T}_3$, we obtain the tree $\{\overline{1}, \overline{3}, \overline{4}, \overline{6}, \overline{7}\}$. Its b-weight is equal to $w_1 + w_{3,4}^b + w_6 + w_7 = 15$, which is lower than $w^b(\hat{T}_3) = w_1 + w_{3,5}^b + w_7 = 16$. This yields $T_3^b = \{\overline{1}, \overline{3}, \overline{4}, \overline{6}, \overline{7}\}$ and $w^b(T_3^b) = 15$. Again, beginning from $\hat{T}_2 = \{\overline{2}, \overline{3}, \overline{4}, \overline{5}, \overline{7}\}$ in $G_\phi \backslash \{\overline{1}\}$, we can find that $T_2^b = \{\overline{2}, \overline{3}, \overline{4}, \overline{6}, \overline{7}\}$ and $w^b(T_2^b) = 15$. Since both T_2^b and T_3^b have the same minimum b-weight among $\{T_2^b, T_3^b, T_4^b\}$, we can choose T_2^b or T_3^b for T^b.

Until now, we have assumed that the minimum-b-weight spanning tree T^* contains a b-branch. It is possible, however, that T^* does not contain a b-branch at all. For this scenario, we can simply find a spanning tree $\hat{T}_0$ with minimum weight $w(\hat{T}_0)$ in $G_\phi \backslash \{\overline{i_0}\}$. Clearly, $\hat{T}_0$ has minimum b-weight among all spanning trees that do not have a b-branch. In the case of our example, $\hat{T}_0 = \{\overline{1}, \overline{3}, \overline{5}, \overline{6}, \overline{7}\}$ with a b-weight (weight) of 14.

Finally, after T^b and $\hat{T}_0$ have been determined, we select for T^* the one with the lower b-weight. Since the entire search procedure is exhaustive in nature, T^* is obviously the optimal spanning tree for this problem.

Returning to our example, since $\hat{T}_0$ has the lowest b-weight, we have $T^* = \hat{T}_0 = \{\overline{1}, \overline{3}, \overline{5}, \overline{6}, \overline{7}\}$. Now equipped with T^*, an optimal tour τ^* can be obtained by $\tau^* = \phi \circ \langle 1, 2\rangle \circ \langle 3, 4\rangle \circ \langle 7, 8\rangle \circ \langle 6, 7\rangle \circ \langle 5, 6\rangle = (1, 3, 7)(2, 9)(4)(5, 10)(6)(8) \circ \langle 1, 2\rangle \circ \langle 3, 4\rangle \circ \langle 7, 8\rangle \circ \langle 6, 7\rangle \circ \langle 5, 6\rangle$. Note that we have followed the special order of the optimal pyramidal subtour (5, 8, 7, 6) while performing the transpositions of the branch $\mathcal{I}(5, 7)$. As a result, an optimal solution for the TSP is given by the tour $1 \to 9 \to 2 \to 3 \to 4 \to 7 \to 6 \to 10 \to 5 \to 8 \to 1$ with a length of $c(\phi) + w^b(T^*) = 803 + 14 = 817$.

The following algorithm summarizes our overall solution strategy explained above.

Algorithm *b*-Decomposable

Input: A b-decomposable permuted Monge matrix $C^\phi = (c_{i\phi(j)})$ with the permutation ϕ.
Output: An optimal TSP tour.

BEGIN
  Construct the patching graph G_ϕ;
  Define the Monge matrix $D = (d_{ij})$ as $d_{ij} = c_{i\phi(j)}$;
  Define the weights of the edges in G_ϕ by $w_i = d_{i,i+1} + d_{i+1,i} - d_{ii} - d_{i+1,i+1}$;
  Sort the edges of G_ϕ into non-decreasing order by the weights w_i;
  Find an MST $\hat{T}$ of G_ϕ trying to delay the inclusion of edge $\overline{i_0}$ in the tree as long as possible;
  IF $\hat{T}$ does not contain edge $\overline{i_0}$ THEN
    Patch all subtours of ϕ by using edge-transpositions from $\hat{T}$ in the order of an optimal pyramidal subtour for each of its branches.
  ELSE
    Find a spanning tree $\hat{T}_0$ with minimum weight in $G_\phi \backslash \{\overline{i_0}\}$;

Find a minimum-b-weight spanning tree T^b in G_ϕ among those containing a b-branch;
IF the weight of $\hat{T}_0$ is less than the b-weight of T^b THEN
Patch all subtours of ϕ by using edge-transpositions from $\hat{T}_0$ in the order of an optimal pyramidal subtour for each of its branches.
ELSE
Patch all subtours of ϕ by using edge-transpositions from T^b in the order of an optimal pyramidal subtour for each of its branches.
END

Now let us consider the running time of the algorithm. It takes $O(n \log n)$ time to sort the edges of G_ϕ. As previously described, $\hat{T}$ and $\hat{T}_0$ can be found by Kruskal's (1956) algorithm for the MST, which requires in this case only $O(n)$ time because the edges are in sorted order. Using the pointers, it takes $O(n^2)$ time to determine all T_h^b-s and T^b. The time to perform each of the remaining procedures is $O(n)$. Therefore, the running time for the entire algorithm is $O(n^2)$. Thus we have proved the following theorem.

Theorem 4. *Let $C = (c_{ij})$ be an $n \times n$ b-decomposable permuted Monge matrix with an optimal assignment ϕ. Then the TSP with distance matrix C is solvable in $O(n^2)$ time.*

4.2. A subclass with a faster solution

In the preceding section, we have studied TSP-s whose matrix can be decomposed into two b-weight-additive submatrices. Here we show that this TSP is solvable in $O(n \log n)$ time if at least one of the two submatrices is a sum matrix. In particular, we establish that the bottleneck step of finding the minimum-b-weight spanning tree T^b in Algorithm b-Decomposable can be executed in linear time. Without loss of generality, we assume that the matrix $D' = (d'_{ij})$ for $i, j \leq i_0$ is the sum matrix.

First, it is interesting to note that if the distance matrix is a sum matrix, then for a given permutation ψ, the cost of performing any transposition on ψ is always zero. Thus, when D' is a sum matrix, the edges of the patching graph G_ϕ can be classified into two classes: The first class contains the edges $\overline{i}$ for $i \leq i_0 - 1$, and their weight is zero; the second class consists of the edges $\overline{i}$ for $i \geq i_0$, which have non-negative weight.

Now consider a series of edge-transpositions $\langle i, i+1\rangle, \langle i+1, i+2\rangle, \ldots, \langle j, j+1\rangle$ in G_ϕ with $i < i_0 - 1$ and $j \geq i_0$. If we first perform, in any order, all transpositions for $i \geq i_0 - 1$, the remaining transpositions can always be performed with zero cost. This observation implies that for any b-branch $\mathcal{I}(i, j)$ in G_ϕ with $i < i_0 - 1$, its b-weight is the same as that of $\mathcal{I}(i_0 - 1, j)$. Hence, while looking for T^b, it is unnecessary to consider those spanning trees that contain a b_h-branch with $h < i_0 - 1$. In other words, to find a minimum-b-weight spanning tree T_h^b in $G_\phi \backslash \{\overline{h-1}\}$ among all trees containing a b_h-branch, we need to consider only two cases, $h = i_0 - 1$ and i_0, if $i_0 > 1$ or only one case, $h = i_0$, if $i_0 = 1$. (In the example above, the two cases are $h = 3$ and 4.) Therefore, the tree T^b can be found in only *O(n)* time.

Corollary 5. *Let $C = (c_{ij})$ be an $n \times n$ b-decomposable permuted Monge matrix in which one of the b-weight-additive components is a sum matrix. If an optimal assignment ϕ is given for C, then a minimum b-weight spanning tree and an optimal TSP tour can be found in $O(n \log n)$ time.*

Let us take a look again at the TSP studied by Aneja and Kamoun (1999), which had the permuted Monge matrix $C = (c_{ij}) = \min\{b_i + a_j, \max\{\mu, b_i, a_j\}\}$. The Corollary represents a new, simpler solution for this problem. It can be also easily seen that the distance matrix for the TSP corresponding to the $1 - K$ robotic-cell scheduling problem, $\tilde{C} = (\tilde{c}_{ij}) = \min\{\lambda + \eta + Kb_i + a_j, \max\{\lambda + (K-1)b_i + \max\{\eta + \rho, b_i\}, a_j\}\}$ also has a sum submatrix in its b-decomposition and therefore the Corollary applies to this problem too.

5. Conclusions

We have introduced the $1 - K$ robotic-cell scheduling problem, whose solution can be reduced to solving a TSP on specially structured permuted Monge matrices. We have presented the insight that the TSP on *b*-decomposable permuted Monge matrices can be solved in polynomial time by a special adaptation of the well-known subtour-patching technique. We have discussed how this new class of matrices can be recognized in polynomial time. Based on the subtour-patching technique, we formulated the TSP on this special class of matrices as a minimum-*b*-weight spanning tree problem and described an $O(n^2)$ algorithm for it. Furthermore, we considered a special case of *b*-decomposable matrices whose one component is a sum submatrix, and showed that the optimal solution can be obtained even faster for this case. As a byproduct of this, we have given a new algorithm and a simpler proof for the special TSP studied in Aneja and Kamoun (1999).

Appendix

The following are the detailed steps of calculating the lengths of the shortest pyramidal tours and the b-weights for the example in the paper.

Let $Q(i, j)$ denote the length of a shortest pyramidal path from city i to city j that visits each city in $\{1, 2, \ldots, \max\{i, j\}\}$ exactly once. Here, a path is called *pyramidal*, if it first passes through the cities in descending order of index from i to 1 and then in ascending order of index from 1 to j. By decomposing a pyramidal path into smaller parts, it is not difficult to see that

$$Q(j, j+1) = \min_{1 \le i < j} \left\{ Q(i+1, i) + c_{i,j+1} + \sum_{k=i+1}^{j-1} c_{k+1,k} \right\} \tag{2}$$

and

$$Q(j+1, j) = \min_{1 \le i < j} \left\{ Q(i, i+1) + c_{j+1,i} + \sum_{k=i+1}^{j-1} c_{k,k+1} \right\}. \tag{3}$$

Note that $\sum_{k=i+1}^{j-1} c_{k+1,k} = \sum_{k=i+1}^{j-1} c_{k,k+1} = 0$ if $i \geq j-1$. Starting from the initial conditions $Q(1,2) = c_{12}$ and $Q(2,1) = c_{21}$, this recurrence allows us to compute $Q(i,j)$ for all $1 \leq i, j \leq n$ and $i \neq j$. For example, the recurrence yields

$$
\begin{aligned}
&Q(2,3) = Q(2,1) + c_{13}, \quad Q(3,2) = Q(1,2) + c_{31},\\
&Q(3,4) = \min\{Q(2,1) + c_{14} + c_{32},\ Q(3,2) + c_{24}\},\\
&Q(4,3) = \min\{Q(1,2) + c_{41} + c_{23},\ Q(2,3) + c_{42}\},\\
&Q(4,5) = \min\{Q(2,1) + c_{15} + c_{32} + c_{43},\ Q(3,2) + c_{25} + c_{43},\ Q(4,3) + c_{35}\},\\
&Q(5,4) = \min\{Q(1,2) + c_{51} + c_{23} + c_{34},\ Q(2,3) + c_{52} + c_{34},\ Q(3,4) + c_{53}\}
\end{aligned}
$$

for $n = 5$. The length of a shortest pyramidal tour τ_n on cities $1, 2, \ldots, n$ is then given by

$$c(\tau_n) = \min\{Q(n-1,n) + c_{n,n-1},\ Q(n,n-1) + c_{n-1,n}\}.$$

See Park (1991) for details.

Now let us consider the set of cities $\{3, 4, 5, 6\}$ with the corresponding distances extracted from the matrix $D = (d_{ij}) = (c_{i\phi(j)})$. Using the above recursions, we can obtain

$$
\begin{aligned}
&Q(1,2) = 57, \quad Q(2,1) = 66,\\
&Q(2,3) = 66 + 64 = 130, \quad Q(3,2) = 57 + 75 = 132,\\
&Q(3,4) = \min\{66 + 90 + 77, 132 + 90\} = 222,\\
&Q(4,3) = \min\{57 + 83 + 75, 130 + 83\} = 213.
\end{aligned}
$$

Then we have $c(\tau_3) = \min\{130 + 77, 132 + 75\} = 207$ and $c(\tau_4) = \min\{222 + 83, 213 + 90\} = 303$. By backtracking, a shortest pyramidal tour on cities 3, 4, 5, 6 is $\tau_4 = (3, 5, 6, 4)$. The b-weight of branches $\mathcal{I}(3,4)$ and $\mathcal{I}(3,5)$ can thus be determined as $w^b_{3,4} = c(\tau_3) - \sum_{i=3}^{5} c_{i\phi(i)} = 207 - 55 - 68 - 80 = 4$ and $w^b_{3,5} = c(\tau_4) - \sum_{i=3}^{6} c_{i\phi(i)} = 303 - 55 - 68 - 80 - 90 = 10$, respectively.

Again, consider the set of cities $\{4, 5, 6, 7, 8\}$ with the corresponding distances extracted from the matrix $D = (d_{ij}) = (c_{i\phi(j)})$. The recursions are computed in a similar way:

$$
\begin{aligned}
&Q(1,2) = 75, \quad Q(2,1) = 77,\\
&Q(2,3) = 77 + 90 = 167, \quad Q(3,2) = 75 + 83 = 158,\\
&Q(3,4) = \min\{77 + 100 + 83, 158 + 100\} = 258,\\
&Q(4,3) = \min\{75 + 95 + 90, 167 + 95\} = 260,\\
&Q(4,5) = \min\{77 + 110 + 83 + 95, 158 + 110 + 95, 260 + 110\} = 363,\\
&Q(5,4) = \min\{75 + 106 + 90 + 100, 167 + 106 + 100, 258 + 106\} = 364.
\end{aligned}
$$

Then we obtain $c(\tau_3) = \min\{167 + 83, 158 + 90\} = 248$, $c(\tau_4) = \min\{258 + 95, 260 + 100\} = 353$, and $c(\tau_5) = \min\{363 + 106, 364 + 110\} = 469$. This gives us $w^b_{4,5} =$

$c(\tau_3) - \sum_{i=4}^{6} c_{i\phi(i)} = 248 - 238 = 10$, $w^b_{4,6} = c(\tau_4) - \sum_{i=4}^{7} c_{i\phi(i)} = 353 - 338 = 15$, and $w^b_{4,7} = c(\tau_5) - \sum_{i=4}^{8} c_{i\phi(i)} = 699 - 448 = 21$.

Acknowledgement

We would like to acknowledge an anonymous reviewer whose comments led to a better presentation for the paper. Partial support for this research by the Natural Sciences and Engineering Research Council of Canada under Grant No. 1798–03 is also acknowledged.

References

R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, Upper Saddle River: New Jersey, 1993.

Y.P. Aneja and H. Kamoun, "Scheduling of parts and robot activities in a two machine robotic cell," *Computers and Operations Research*, vol. 26, pp. 297–312, 1999.

V.Y. Burdyuk and V.N. Trofimov, "Generalization of the results of Gilmore and Gomory on the solution of the traveling salesman problem," *Engineering Cybernetics*, vol. 14, pp. 12–18, 1976.

R.E. Burkard, V.G. Deĭneko, R. van Dal, J.A.A. van der Veen, and G.J. Woeginger, "Well-solvable special cases of the traveling salesman problem: A survey," *SIAM Review*, vol. 40, pp. 496–546, 1998.

R.E. Burkard, B. Klinz, and R. Rudolf, "Perspectives of Monge properties in optimization," *Discrete Applied Mathematics*, vol. 70, pp. 95–161, 1996.

Y. Crama, V. Kats, J. van de Klundert, and E. Levner, "Cyclic scheduling in robotic flowshops," *Annals of Operations Research*, vol. 96, pp. 97–124, 2000.

P.C. Gilmore and R.E. Gomory, "Sequencing a one state-variable machine: A solvable case of the traveling salesman problem," *Operations Research*, vol. 12, pp. 655–679, 1964.

P.C. Gilmore, E.L. Lawler, and D.B. Shmoys, "Well-solved special cases," in E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, eds., *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, ch. 4, pp. 87–143, John Wiley & Sons, Chichester, England, 1985.

N.G. Hall, H. Kamoun, and C. Sriskandarajah, "Scheduling in robotic cells: Classification, two and three machine cells," *Operations Research*, vol. 45, pp. 421–439, 1997.

S.N. Kabadi, "Polynomially solvable cases of the TSP," in G. Gutin and A.P. Punnen, eds., *The Traveling Salesman Problem and its Variations*, ch. 11, pp. 489–583, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

J.B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, pp. 48–50, 1956.

M. Middendorf and V.G. Timkovsky, "On scheduling cycle shops: Classification, complexity and approximation," *Journal of Scheduling*, vol. 5, pp. 135–169, 2002.

J.K. Park, "A special case of the n-vertex traveling-salesman problem that can be solved in O(n) time," *Information Processing Letters*, vol. 40, pp. 247–254, 1991.

S.S. Reddi and C.V. Ramamoorthy, "On the flow-shop sequencing problem with no wait in process," *Operational Research Quarterly*, vol. 23, pp. 323–331, 1972.

V.I. Sarvanov, "On the complexity of minimizing a linear form on a set of cyclic permutations," *Soviet Mathematics-Doklady*, vol. 22, pp. 118–120, 1980.

S.P. Sethi, C. Sriskandarajah, G. Sorger, J. Blazewicz, and W. Kubiak, "Sequencing of parts and robot moves in a robotic cell," *The International Journal of Flexible Manufacturing Systems*, vol. 4, pp. 331–358, 1992.

G. Steiner and Z. Xue, "Scheduling multi-component parts in robotic cells," Working Paper, School of Business, McMaster University, Canada, 2002.

G. Steiner and Z. Xue, "Scheduling in reentrant robotic cells: Algorithms and complexity," *Journal of Scheduling*, vol. 8, pp. 25–48, 2005.

Z. Xue, *Shop Scheduling in Manufacturing Systems: Algorithms and Complexity*, Ph.D. Thesis, McMaster University, Canada, 2004.