Journal of Combinatorial Optimization 2, 333–350 (1999)

The Travelling Salesman Problem on Permuted Monge Matrices

RAINER E. BURKARD, VLADIMIR G. DEĬNEKO* AND GERHARD J. WOEGINGER
TU Graz, Institut für Mathematik B, Steyrergasse 30, A-8010 Graz, Austria

Received March 19, 1997; Revised October 29,1997

Abstract. We consider traveling salesman problems (TSPs) with a permuted Monge matrix as cost matrix where the associated patching graph has a specially simple structure: a multistar, a multitree or a planar graph. In the case of multistars, we give a complete, concise and simplified presentation of Gaikov's theory. These results are then used for designing an $O(m^3 + mn)$ algorithm in the case of multitrees, where n is the number of cities and m is the number of subtours in an optimal assignment. Moreover we show that for planar patching graphs, the problem of finding an optimal subtour patching remains NP-complete.

Keywords: travelling salesman problem, subtour patching, combinatorial optimization, computational complexity

1. Introduction

A classical problem in combinatorial optimization is the *travelling salesman problem (TSP)*: given n cities and the distances c_{ij} between them, find a shortest tour ϕ through all cities. More formally, the TSP can be stated as minimizing the function

$$F(\phi) := \sum_{i=1}^{n} c_{i\phi(i)}$$

where ϕ is a cyclic permutation of the set $\{1, 2, \ldots, n\}$. We will refer to $F(\phi)$ as *cost* of the permutation ϕ. For further results on travelling salesman problems see the excellent book of Lawler et al. (1985).

Since the TSP is NP-hard, many efforts have been made in investigating special cases which allow a polynomial solution procedure. In most cases such special cases rely on special structures of the underlying distance matrix. In our case we consider permuted Monge matrices as distance matrices. An $(n \times n)$ matrix $C = (c_{ij})$ is called a *Monge-matrix* or *distribution matrix*, if

$$c_{ik} + c_{jl} \le c_{il} + c_{jk} \quad \text{for all } 1 \le i < j \le n,\ 1 \le k < l \le n \tag{1}$$

*University of Warwick, Warwick Business School, Coventry CV4 7AL, United Kingdom.

The matrix $C = (c_{ij})$ is called a *permuted Monge-matrix* or *permuted distribution matrix*, if there is a permutation ρ of the rows and a permutation σ of the columns of C such that

$$C_{\rho\sigma} = (c_{\rho(i),\sigma(j)}) \quad \text{is a Monge matrix.}$$

Since the numbering of the cities does not play any role we can assume in the following that ρ is the identical permutation ε.

Permuted Monge matrices arise in the following context, see e.g., Burdyuk and Trofimov (1976): Let $f : \mathbb{R}^2 \to \mathbb{R}$ be a function with constant second differences, i.e.,

$$f(x + \Delta x, y + \Delta y) + f(x, y) - f(x + \Delta x, y) - f(x, y + \Delta y)$$

has constant sign for all $\Delta x, \Delta y > 0$. Moreover, let $u = (u_1, \dots, u_n)$ and $(v_1, \dots, v_n)$ be two real vectors with n components. Then the matrix $C = (c_{ij})$ defined by $c_{ij} = f(u_i, v_j)$ is a permuted Monge matrix. In particular, every *product matrix* $(u_i v_j)$ is of this form. Given an arbitrary matrix $C = (c_{ij})$, it can be tested in $O(n^2)$ time whether C is a permuted Monge matrix or not. Deineko and Filonenko (1979) designed an $O(n^2)$ algorithm for finding the permutations ρ and σ which make C Monge, if this is possible. For further results on Monge matrices and related topics see the recent survey of Burkard et al. (1996).

The TSP with a Monge matrix as a cost matrix can easily be solved via dynamic programming or via a greedy algorithm. In this case there exists an optimal tour which is *pyramidal*. Let us number the cities by $1, 2, \dots, n$. A tour is called *pyramidal*, if starting in city 1 the cities are first visited according to increasing numbers until city n is reached, and then they are visited according to decreasing numbers. For example, the tour $\langle 1, 2, 5, 7, 6, 4, 3 \rangle$ is pyramidal. Optimal pyramidal tours can be determined in $O(n^2)$ steps, see e.g., the survey article of Gilmore et al. (1985). On the other hand, Sarvanov (1980) showed that the TSP with product matrices as cost matrices is NP-hard. Since every product matrix is a permuted Monge matrix, Sarvanov's result implies that the TSP with permuted Monge matrices as cost matrices is NP-hard in general. Gilmore and Gomory (1964), however, exhibited an important special case of permuted Monge matrices which still allows a polynomial solution routine. They developed a strategy, called *subtour patching* which was later generalized by Burdyuk and Trofimov (1976).

In Section 2 we sketch the theory of subtour patching and introduce the so-called *patching graph*. The main problem will be to determine a certain spanning tree of the patching graph that has minimum weight with respect to a nonstandard weight function. This spanning tree yields an optimal solution for the underlying TSP. Section 3 gives a complete and concise description and solution of a closely related covering problem. We simplify and streamline Gaikov's (1980) original analysis which is rather involved and only available in Russian. In Section 4, we consider the case where the patching graph is a *multistar*. In Section 5, we treat the case that the patching graph is a *multitree* and design a new fast algorithm which runs by a factor of $O(n^3)$ faster than Gaikov's original algorithm. We mentioned above that the TSP is NP-hard for product matrices which implies NP-completeness of the problem with *arbitrary* patching graphs. In Section 6 it is shown that the problem is even NP-hard, if the patching graph is planar. The paper is closed with a discussion in Section 7.

2. The theory of subtour patching

2.1. *Permutations*

Let us first introduce some notations concerning permutations. A permutation ϕ is a one-to-one mapping of $\{1, 2, \dots, n\}$ onto itself. Let $i_1, i_2, \dots, i_r$ be pairwise distinct elements of $\{1, 2, \dots, n\}$. If $\phi(i_k) = i_{k+1}$ for $k = 1, 2, \dots, r-1$, $j \neq k$, and $\phi(i_r) = i_1$, then $\langle i_1, i_2, \dots, i_r \rangle$ is called a *factor* or *subtour* of the permutation ϕ. A permutation is called *cyclic* or a *tour,* if it has only one factor. A factor of the form $\langle i \rangle$ is called *trivial.* A permutation which has only one nontrivial factor of the form $\langle j, k \rangle$ is called a *transposition.* We will denote such transpositions by (j, k). A transposition is called *adjacent,* if it is of the form $(j, j+1)$.

If both ϕ and ψ are permutations on $\{1, 2, \dots, n\}$, then their composition (often written as product) is denoted by $\phi \circ \psi$ and defined by $\phi \circ \psi(i) = \phi(\psi(i))$ for $1 \leq i \leq n$. $\phi \circ \psi$ is again a permutation. Recall that the product of two transpositions is a non-commutative operation, if they have a common index. Otherwise it is commutative. The following proposition can be viewed as an illustration of this observation, see e.g., Gilmore et al. (1985).

Proposition 2.1. *The set of all permutations that can be obtained by multiplying adjacent transpositions* $(1, 2), (2, 3), \dots, (n-1, n)$ *is exactly the set of all pyramidal tours.*

2.2. *Graphs*

Now let us turn to graphs. In undirected graphs, edges between two vertices $v_1, v_2 \in V$ are denoted by $[v_1, v_2]$. In directed graphs, arcs that go from vertex v_1 to vertex v_2 are denoted by (v_1, v_2). (From the context there will be no confusion with transpositions which are also written as pairs).

In an undirected graph $G = (V, E)$, with vertex set V and edge set E an alternating sequence $v_0, e_1, v_1, e_2, \dots, e_p, v_p$ of vertices $v_i \in V$ and edges $e_i \in E$ is called a *path* if $e_i = [v_{i-1}, v_i] \in E$ for $1 \leq i \leq p$. If additionally $v_i \neq v_j$ holds whenever $i \neq j$, the sequence is a *simple path.* G is called a *multigraph,* if several edges may connect the same pair of vertices. If we replace all those edges by just a single one, we get the *underlying simple graph.* A multigraph $G = (V, E)$ is called a *multipath* (*multistar, multitree,* respectively) if the underlying simple graph is a path (star, tree, respectively). An undirected multigraph $G = (V, E)$ with edge set $E = \{e_1, e_2, \dots, e_m\}$ is a *Eulerian* multigraph if we can number the edges such that the edge sequence $e_1, e_2, \dots, e_m$ forms a path.

2.3. *The patching graph*

Let a permuted Monge matrix $C = (c_{ij})$ be the distance matrix of a TSP and let σ be a permutation such that $(c_{i\sigma(j)})$ is a Monge matrix. Then the corresponding assignment problem with cost matrix C has the permutation σ as optimal solution. If the permutation

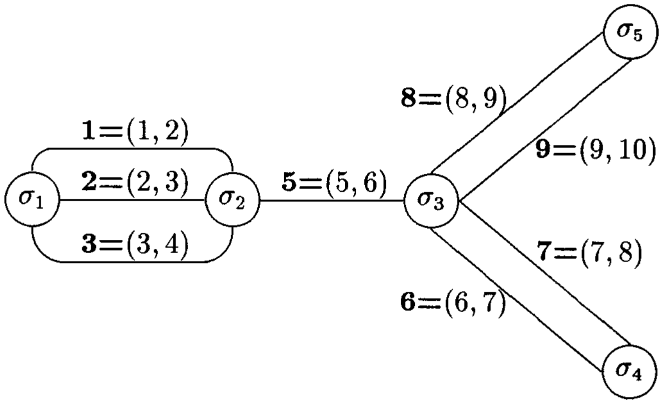


Figure 1. Graph G_σ for $\sigma = \langle 1\ 3\rangle\langle 2\ 4\ 5\rangle\langle 6\ 8\ 10\rangle\langle 7\rangle\langle 9\rangle$.

σ has only one factor, it is a tour and therefore also an optimal solution for the TSP. So we assume for the following that σ consists of $r \geq 2$ subtours σ_k, i.e. $\sigma = \sigma_1 \circ \sigma_2 \circ \cdots \circ \sigma_r$.

Now the *patching graph* $G_\sigma = (V, E)$ is constructed in the following way. Every vertex $v \in V$ corresponds to a subtour of the permutation σ. Every edge in E corresponds to an *adjacent* transposition $(i, i+1)$, namely, if index i occurs in the subtour σ_j (corresponding to vertex σ_j) and index $(i+1)$ occurs in the subtour σ_k (corresponding to vertex σ_k), then the two vertices σ_j and σ_k are connected by the edge corresponding to $(i, i+1)$. This construction yields a connected Eulerian multigraph with at most $(n-1)$ edges.

For instance, let $\sigma = \sigma_1 \circ \sigma_2 \circ \sigma_3 \circ \sigma_4 \circ \sigma_5 = \langle 1\ 3\rangle\langle 2\ 4\ 5\rangle\langle 6\ 8\ 10\rangle\langle 7\rangle\langle 9\rangle$. For this permutation σ the graph G_σ in figure 1 is a multitree with five vertices and eight edges. We denote an edge corresponding to the adjacent transposition $(i, i+1)$ by i.

2.4. *Subtour patching*

A basis for subtour patching is the following result.

Proposition 2.2 (Gilmore and Gomory, 1964). *Let* $T = \{(i_1, i_1+1), \ldots, (i_{r-1}, i_{r-1}+1)\}$ *be the edge set of a spanning tree in the graph* G_σ*. Then the permutation* $\sigma \circ (i_1, i_1+1) \circ \cdots \circ (i_{r-1}, i_{r-1}+1)$ *is a cyclic permutation.*

The composition of σ with adjacent transpositions corresponds to patching subtours which results in a tour. Let us explain this by the example given above. Choose the transpositions (1, 2), (5, 6), (6, 7) and (8, 9) as edges of the spanning tree T in G_σ. Due to the discussion prior to Proposition 2.1 there are in our case two possibilities to patch subtours in σ by using transpositions of T, thus yielding two different cyclic permutations. These are:

$$\sigma_1 = \sigma \circ (1,2) \circ (5,6) \circ (6,7) \circ (8,9) = \langle 1, 4, 5, 8, 9, 10, 6, 7, 2, 3\rangle$$

and

$$\sigma_2 = \sigma \circ (1,2) \circ (6,7) \circ (5,6) \circ (8,9) = \langle 1,4,5,7,8,9,10,6,2,3 \rangle.$$

Proposition 2.3 (Burdyuk and Trofimov (1976); Gilmore et al. (1985)). *Let $(c_{i\sigma(j)})$ be a Monge matrix. For any cyclic permutation ϕ there exist a spanning tree $T = \{(i_1, i_1+1), \ldots, (i_{r-1}, i_{r-1}+1)\}$ in the graph G_σ and a sequence τ for multiplying the transpositions of T such that the permutation $\sigma_T = \sigma \circ (i_{\tau(1)}, i_{\tau(1)}+1) \circ \cdots \circ (i_{\tau(r-1)}, i_{\tau(r-1)}+1)$ is a cyclic permutation with $F(\sigma_T) \le F(\phi)$.*

It follows from Proposition 2.3 that an optimal tour can be constructed by using a special spanning tree T^* of the graph G_σ. Unfortunately, the problem of finding a suited (“optimal”) spanning tree in G_σ is NP-hard in general (Sarvanov, 1980). However, the problem can be solved in polynomial time if the graph G_σ has a special structure: For the case when G_σ is a *multipath,* algorithms with complexity $O(n^3)$ for finding an optimal tree were proposed by Sarvanov (1980) and Deineko (1979). Recently an $O(n)$ algorithm for this case was developed by Burkard and Deineko (1995). Gaikov (1980) improved the results of Sarvanov on multipaths and considered also *multistars* and *multitrees.* In particular, he addressed the problem which spanning tree has to be chosen and in which sequence the adjacent transpositions are to be composed in order to get an optimal tour.

2.5. *The minimum spanning tree problem with branches*

According to the remarks prior to Proposition 2.1 the form of a permutation σ_T depends on the sequence τ which is used for multiplying the transpositions of the tree T. The natural question arises to find a permutation τ that minimizes the difference $F(\sigma_T) - F(\sigma)$. This can be done in the following way.

Let us divide the set of edges (transpositions) of the tree T into $t (1 \le t \le r-1)$ *dense,* pairwise disjoint subsets:

$$\begin{aligned}
I(i_1, j_1) &:= \{(i_1, i_1+1), (i_1+1, i_1+2), \ldots, (j_1, j_1+1)\},\\
I(i_2, j_2) &:= \{(i_2, i_2+1), (i_2+1, i_2+2), \ldots, (j_2, j_2+1)\},\\
&\qquad\qquad\qquad\qquad\ldots\\
I(i_t, j_t) &:= \{(i_t, i_t+1), (i_t+1, i_t+2), \ldots, (j_t, j_t+1)\},
\end{aligned}$$

such that

$$T = I(i_1, j_1) \cup I(i_2, j_2) \cup \cdots \cup I(i_t, j_t) \text{ and } j_k + 1 < i_{k+1} \quad \text{for } k = 1, 2, \ldots, t-1.$$

In the example above the edge set of the tree T is divided into three subsets with

$$I(1,1) = \{(1,2)\}, \quad I(5,6) = \{(5,6),(6,7)\}, \quad I(8,8) = \{(8,9)\}.$$

The arbitrary composition of transpositions of a dense set yields just one subtour (which depends on the sequence in which the transpositions are composed). Thus, multiplying the transpositions of T in arbitrary order yields t subtours, where each subtour corresponds to a set $I(i_k, j_k)(1 \le k \le t)$. If a permutation ϕ consists of t disjoint factors $\phi_1, \phi_2, \ldots, \phi_t$, it is well-known (see Gilmore et al. (1985), Theorem 4) that

$$F(\sigma \circ \phi) - F(\sigma) = \sum_{i=1}^{t} (F(\sigma \circ \phi_i) - F(\sigma)) \tag{2}$$

So an optimal sequence τ of multiplying the transpositions of T can be constructed by finding an optimal sequence for each dense set $I(i, j)$. According to Proposition 2.1 this can be done by using techniques for constructing optimal pyramidal tours.

Let ϕ^*_{ij} be the subtour corresponding to the set $I(i, j)$ which is constructed by multiplying transpositions of $I(i, j)$ in an optimal order. Define values w_{ij} as

$$w_{ij} := F(\sigma \circ \phi^*_{ij}) - F(\sigma). \tag{3}$$

Then it follows immediately that

$$F(\sigma_T) - F(\sigma) = \sum_{k=1}^{t} w_{i_k j_k}.$$

For a Monge matrix, all values w_{ij} for $i = 1, 2, \ldots, n$ and $j = i, \ldots, n$ can be computed by using Park's (1991) algorithm in $O(n^2)$ time. Note that the values w_{ij} do not depend on the patching graph. Moreover, for particular patching graphs only few such values will actually be needed. It can be shown (see e.g., Burkard et al., 1998) that these values w_{ij} have the following property

$$w_{ij} \ge w_{ik} + w_{k+1,j} \quad \text{for all } i \le k < j.$$

This leads to the following *minimum spanning tree problem with branches* (B-MST, for short): Let $G = (V, E)$ be a Eulerian multigraph with $E = \{1, 2, \ldots, r\}$ where the edge sequence $1, \ldots, r$ forms a Eulerian path. The edges $i, i+1, \ldots, j$ are called a *branch* $[i, j]$ with respect to some set $E' \subseteq E$, if $i - 1 \notin E'$ and $j + 1 \notin E'$. For $E' \subseteq E$, let $Br(E')$ denote the set of all branches with respect to E'.

Let w be a weight function that assigns to every edge set $\{i, i+1, \ldots, j\}$, $1 \le i \le j \le r$, a nonnegative number w_{ij} and let w fulfill the property

$$w_{ij} \ge w_{ik} + w_{k+1,j} \quad \text{for all } i \le k < j. \tag{4}$$

The *branch weight* (or *b-weight*) of $E' \subseteq E$ is defined by

$$w(E') := \sum_{[i,j] \in Br(E')} w_{ij}.$$

The *branch weight (b-weight) of a spanning tree* $T = (V, E_T)$ of G is $w(E_T)$. The problem B-MST consists in findind a spanning tree of G with minimum b-weight.

In our application every vertex of V corresponds to a subtour of σ, the permutation which makes $C = (c_{ij})$ Monge. Every edge i $(i = 1, 2, \dots, r)$ corresponds to an adjacent transposition $(i, i + 1)$. The edge i connects the vertex corresponding to the subtour containing city i with the vertex which corresponds to the subtour containing city $i + 1$. If both cities lie in the same subtour, we get a loop. Thus, G is the patching graph possibly augmented by loops. Every branch of a spanning tree of G corresponds to a maximal dense subset of transpositions. The weights w_{ij} are defined by (3). Finding a spanning tree with minimum branch weight amounts in minimizing (2) and yields an optimal solution of the underlying TSP.

3. Gaikov's covering problem

In this section we consider an auxiliary problem in directed multigraphs which will help us later to solve the minimum spanning tree problem with branches. Let $H = (N, A)$ be a directed multigraph. Some of the arcs in A are grouped into so-called *twin-pairs*: A *twin-pair* consists of two arcs a_i and a_j where $a_i = (v_s, v_t)$ and $a_j = (v_t, v_s)$ holds for some $v_s, v_t \in N$, $v_s \neq v_t$. Every arc belongs to at most one twin-pair. Arcs that are not part of any twin pair are called *single arcs*. For $B \subseteq A$, let $\mathit{Twin}(B)$ denote the set of all twin-pairs contained in B, let $\mathit{Sing}(B)$ denote the set of all non-loop single arcs in B (these arcs may have a twin-brother in A but not in B), and let $\mathit{Loop}(B)$ denote the set of all loops in B.

With every arc a in A, we associate a non-negative weight $g(a)$ and with every twin-pair $(a_i, a_j) \in \mathit{Twin}(A)$, we associate a non-negative weight $g(a_i, a_j)$. The *Gaikov weight* (or *g-weight*) of a subset $B \subseteq A$ is defined by

$$g(B) = \sum_{(a_i,a_j)\in \mathit{Twin}(B)} g(a_i, a_j) + \sum_{a_k\in \mathit{Sing}(B)} g(a_k) + \sum_{a_l\in \mathit{Loop}(B)} g(a_l).$$

We say that every arc $(v_s, v_t) \in A$ covers its target-vertex v_t. A set $B \subseteq A$ is an *exact cover* for the directed graph H, if every vertex in N is covered by exactly one arc in B. It is easy to see that H possesses an exact cover if and only if each vertex in N has in-degree at least one.

As an illustration for these definitions consider the directed multigraph H shown in figure 2 with the set of arcs $A = \{a_1, a_2, a_3, a_4, a_5\}$ and the set of twin-pairs $\mathit{Twin}(A) = \{(a_1, a_2), (a_3, a_4)\}$. The set $\{a_1, a_2, a_5\}$ is an exact cover, while the sets A, $\{a_1, a_3, a_5\}$ and $\{a_1, a_2\}$ are not.

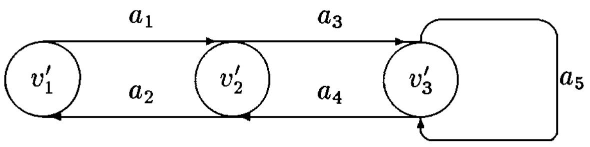


Figure 2. An illustration to Gaikov's covering problem.

Lemma 3.1. *Let $H = (N, A)$ be a directed multigraph where every vertex has in-degree at least one, and let g be a weight function on A and Twin* (A). *Then an exact cover B^* for H with minimum Gaikov weight can be computed in $O(|A|^2 \log |A|)$ time.*

Proof: The idea is to formulate the problem as a weighted matching problem in a related undirected graph $H^u = (N^u, A^u)$. To this end, let S denote a very large number (e.g., twice the sum of all arc weights and all twin-pair weights plus one). The undirected graph H^u contains all vertices in N, one additional vertex for every loop, for every single arc and for every twin-pair in A. Now let a_i be a loop (v_t, v_t) or a single arc (v_s, v_t) in A that covers v_t and let $w \in N^u$ be the vertex corresponding to a_i: Then A^u contains the undirected edge $[v_t, w]$; the weight of this edge is set to $g(a_i) - S$. Next, consider a twin pair $a_i = (v_s, v_t)$ and $a_j = (v_t, v_s)$ in A and let $w \in N^u$ denote the vertex corresponding to this pair: We introduce the edges $[v_s, v_t]$ with weight $g(a_i, a_j) - 2S$, $[v_t, w]$ with weight $g(a_i) - S$, and $[v_s, w]$ with weight $g(a_j) - S$.

It is easy to verify that H possesses an exact cover with minimum Gaikov weight W if and only if H^u contains a (not necessarily perfect) matching with minimum weight $W - |V|S$. Since the number of vertices and the number of edges in H^u both are $O(|A|)$, applying the fast weighted matching algorithm of Gabov (—) yields the claimed $O(|A|^2 \log |A|)$ algorithm (Gabov's algorithm finds a minimum weight matching in a graph with n vertices and m edges in $O(n(m + n \log n))$ time). □

In the remaining part of this section, we describe Gaikov's algorithm for multigraphs whose weight function fulfills additionally the condition

$$g(a_i) + g(a_j) \leq g(a_i, a_j) \quad \text{for all } (a_i, a_j) \in \mathit{Twin}(A). \tag{5}$$

Lemma 3.2. *Let $H = (N, A)$ be a directed multigraph and let g be a weight function on A and Twin* (A), *that fulfills condition* (5). *Then one can construct in* $O(|A|)$ *time a simple directed graph $H^s = (N, A^s)$ and a weight function g^s on A^s and Twin* (A^s), *that fulfills condition* (5) *such that the following holds*: *The exact cover B^s with minimum Gaikov weight for H^s and the exact cover B^* with minimum Gaikov weight for H both have equal weight, and B^* can easily be computed from B^s.*

Proof: The arc set A^s is constructed as follows. For every $v_t \in V$, all loops $(v_t, v_t) \in A$ with the exception of the loop ℓ with smallest weight are removed. The weight $g^s(\ell)$ of this loop equals $g(\ell)$. For every two vertices $v_s, v_t \in V$, let a_1 be the cheapest arc and a_2 be the second-cheapest arc from v_s to v_t in A, and let b_1 be the cheapest and b_2 be the second-cheapest arc going from v_t to v_s. The arc set A^s contains an arc (v_s, v_t) with weight $g(a_1)$ and an arc (v_t, v_s) with weight $g(b_1)$. In case a_1 and b_1 are a twin-pair in A, they are also a twin-pair in A^s and the corresponding weight is defined by

$$g^s(a_1, b_1) = \min\{g(a_1, b_1), g(a_1) + g(b_2), g(a_2) + g(b_1)\}.$$

This completes the description of A^s and g^s. It is easy to verify correctness of this construction: If the optimum cover B^* for H covers a vertex v_t by a loop or by an arc (v_s, v_t) where

$(v_t, v_s) \notin B^*$, then it will obviously use the cheapest such arc. Otherwise, B^* contains two arcs (v_s, v_t) and (v_t, v_s). These are either the cheapest arcs of these types or (in case the cheapest arcs form a twin-pair) the second cheapest ones. □

For $B \subseteq A$, disregarding the orientation of its arcs and omitting multiple edges naturally results in the *underlying undirected* edge set $\text{UU}(B)$ for B. An arc set B is called *connected* iff $\text{UU}(B)$ is connected. By $N_B \subseteq N$ we denote the set of vertices that are incident to $\text{UU}(B)$.

Observation 3.3. *Let B be an exact cover for $H = (N, A)$ and let C be a connected component of B. Then C contains exactly one directed cycle* (*that possibly is a loop or a twin-pair*). *If an arc (v, r) is removed from this directed cycle, the remaining arcs in $C \backslash \{(v, r)\}$ form a directed out-tree with root r.*

Next, we fix a *greedy exact cover* GREC $\subseteq A$ for H that contains for every vertex $v \in N$ a covering arc or covering loop with minimum weight. In general, GREC will not be the exact cover with minimum Gaikov weight, as it may contain twin-pairs with high g-weight.

Observation 3.4. *If the weights of $H = (N, A)$ fulfill condition* (5), *then there exists an exact cover B^* with minimum Gaikov weight for $H = (N, A)$ such that* $\text{UU}(\text{GREC}) \subseteq \text{UU}(B^*)$.

Proof: Consider an exact cover B^* with minimum Gaikov weight. Suppose that $a_1 = (v_s, v_t) \in$ GREC, $a_1 \notin B^*$ and $(v_t, v_s) \notin B^*$. Let $a_3 = (v_u, v_t) \in B^*$ be the arc that covers v_t and let a_4 denote the twin-brother of a_3 (in case it exists). By the definition of GREC, $g(a_1) \leq g(a_3)$ and by condition (5), $g(a_3) + g(a_4) \leq g(a_3, a_4)$ holds. With this it is easy to verify that $(B^* \backslash \{a_3\}) \cup \{a_1\}$ has Gaikov weight at most $g(B^*)$. Repeating this exchange procedure eventually yields a cover with the claimed property. □

Observation 3.5. *If the weights fulfill condition* (5), *then there exists a minimum weight cover B^* of the following form.*

(i) *For every connected component C of* GREC *that does not contain any twin-pair, $C \subseteq B^*$.*
(ii) *For every connected component C of* GREC *that does contain a twin-pair, there exists an arc $(v, r) \in B^*$ with $r \in N_C$ and a set $A_C \subseteq B^* \backslash \{(v, r)\}$ such that* $\text{UU}(A_C) = \text{UU}(C)$ *and such that A_C is a directed out-tree with root r.*

Proof: Consider a minimum g-weight exact cover B^* of H such that $\text{UU}(\text{GREC}) \subseteq \text{UU}(B^*)$ that exists by Observation 3.4.

Proof of (i). Since C does not contain any twin-pair, C is a minimum g-weight exact cover for N_C and $|\text{UU}(C)| = |N_C|$. According to Observation 3.3, there exists a set $A_C \subseteq B^*$ with $\text{UU}(A_C) = \text{UU}(C)$. Then $|A_C| \geq |\text{UU}(A_C)| = |\text{UU}(C)| = |N_C|$ holds. Therefore A_C covers exactly N_C and there are no other arcs in B^* that cover N_C. Hence, replacing B^* by $B^* \backslash A_C \cup C$ yields the desired property, does not introduce any new twin-pairs and consequently, does not increase the Gaikov weight.

Proof of (ii). As C contains a twin-pair, $|\text{UU}(C)| = |N_C| - 1$ and $\text{UU}(C)$ is a tree. According to Observation 3.3, there exists a set $A_C \subseteq B^*$ with $\text{UU}(A_C) = \text{UU}(C)$ and

$|A_C| = |N_C| - 1$. This set A_C covers all but one vertices in N_C and hence, there must exist an arc $(v, r) \in B^* \backslash A_C$ that covers this remaining vertex r. Clearly, in order to cover the vertices in $N_C \backslash \{r\}$, A_C must be a directed out-tree rooted at r. □

Based on the above observations, Gaikov designed the following algorithm for finding a minimum g-weight exact cover under condition (5):

(Phase 1). Compute a greedy exact cover GREC for H. Let C_0 be the union of all connected components in GREC that do not contain a twin-pair, and let $C_1, \ldots, C_k$ be an enumeration of the connected components that contain a twin-pair. For every C_i, $i \geq 1$, and for every $r \in N_{C_i}$, let $T(C_i, r)$ be the directed out-tree T with $\text{UU}(T) = \text{UU}(C_i)$ that is rooted at r and let $W(C_i, r)$ be the overall weight of the arcs in $T(C_i, r)$. In case $T(C_i, r)$ is not a subgraph of H, $W(C_i, r)$ is set to $+\infty$.

(Phase 2). Construct a new weighted directed multigraph H' with vertex set $C_0, \ldots, C_k$, arc set A' and a weight function g' defined as follows: For every arc $a = (v_s, v_t)$ in A that fulfills $v_t \notin N_{C_0}$, there is a corresponding arc a' in A' going from the component $C(v_s)$ that contains v_s, to the component $C(v_t)$ that contains v_t. If $C(v_s) \neq C(v_t)$ holds, or if $C(v_s) = C(v_t)$ and the twin-brother of arc a is not contained in $T(C(v_t), v_t)$, the weight $g'(a')$ of this new arc equals $g(a) + W(C(v_t), v_t)$. Otherwise, if $C(v_s) = C(v_t)$ and the twin-brother a^- of a is contained in $T(C(v_t), v_t)$, then arc a' receives weight $W(C(v_t), v_t) + g(a, a^-) - g(a^-)$. Moreover, there is a loop of weight zero in A' incident to C_0.

Next, we define the twin-pairs in H': Let $a_i = (v_s, v_t)$ and $a_j = (v_t, v_s)$ be a twin-pair in H such that $C(v_s) \neq C(v_t)$. Then the new corresponding arcs a_i' and a_j' form a twin-pair in H' with weight

$$g'(a_i', a_j') = g'(a_i') + g'(a_j') + g(a_i, a_j) - g(a_i) - g(a_j),$$

i.e., taking both arcs a_i' and a_j' into a cover increases the overall cost by the same amount as taking a_i and a_j into a cover.

(Phase 3). From the multigraph H', construct a simple graph H'' and a weight function g'' as described in Lemma 3.2. Recursively compute an exact cover B'' with minimum Gaikov weight for H'' with weights g''. From B'', the corresponding exact cover B^* with minimum Gaikov weight for H is constructed as follows: First, all edges in C_0 are put into B^*. For a single arc a'' in B'' that enters a component C_i with $i \geq 1$, let $a = (v_s, v_t)$ be the corresponding arc in G. B^* contains a and all arcs in $T(C_i, v_t)$. Twin pairs in B'' are handled similarly (the corresponding arcs in H are put into B^* and the concerned components are oriented in an appropriate way).

The correctness of this procedure essentially follows from Lemma 3.2 and from Observations 3.4 and 3.5, and is easy to verify. For the time complexity, observe that computing the connected components in (Phase 1) takes $O(|N| + |A|)$ time by applying depth-first-search. Computing the values $W(C_i, r)$ can trivially be done in $O(|C_i|^2)$ time per component and yields an overall time of $O(|N|^2)$ for handling all components. The running time of (Phase 2) is proportional to $|A|$. Every connected component C_i with $1 \leq i \leq k$ contains a twin-pair and hence at least two vertices. This yields that the graph H'' that is treated in the recursion in (Phase 3) has $k \leq |N|/2$ vertices. Since after applying Lemma 3.2, there remain only

$O(|N|^2)$ arcs, the overall time complexity $T(|N|, |A|)$ for treating a graph $H = (N, A)$ is

$$T(|N|, |A|) \leq T(|N|/2, |N|^2/2) + O(|A|),$$

which implies $T(|N|) = O(|N|^2 + |A|)$. Summarizing, we formulate the following theorem.

Theorem 3.6 (Gaikov, 1980). *Let $H = (N, A)$ be a directed graph where every vertex has in-degree at least one, and let g be a weight function on A and Twin(A) that fulfills condition* (5). *Then an exact cover B^* for H with minimum Gaikov weight can be computed in $O(|N|^2 + |A|)$ time.*

We illustrate some aspects of Gaikov's algorithm for the covering problem by the example presented in figure 2. Suppose that a greedy exact cover GREC contains arcs a_1, a_4 and a_5. Since there are no twin-pairs in it, it is an exact cover with minimum Gaikov weight. Suppose now that GREC contains arcs a_1, a_2 and a_3. That means that we look for an optimal cover $B^\star$ such that the underlying undirected graph UU($B^\star$) contains the edges $[v_1', v_2']$ and $[v_2', v_3']$. According to Phase 2 a new weighted directed multigraph H' contains only one vertex and five loops a_1', a_2', a_3', a_4' and a_5'. In order to compute e.g., weight $g(a_1')$ it is necessary to compute the overall weight of the arcs in the rooted tree with arcs a_2 and a_3. Since this tree contains a twin-brother of a_1, we have

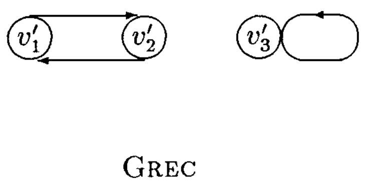
$$g(a_1') = g(a_1, a_2) + g(a_3).$$

Other weights g are computed in a similar way:

$$g(a_2') = g(a_1'), \quad g(a_3') = g(a_4') = g(a_3, a_4) + g(a_2),$$
$$g(a_5') = g(a_5) + g(a_2) + g(a_4).$$

The problem is solved after finding a loop with a minimal weight. The situation is a bit more complicated if GREC contains arcs a_1, a_2 and a_5. For this particular case a new graph H' contains two vertices, one arc and three loops (see figure 3) with the new weights

$$g(a_1') = g(a_2') = g(a_1, a_2), \quad g(a_4') = g(a_4) + g(a_2), \quad g(a_0') = 0.$$



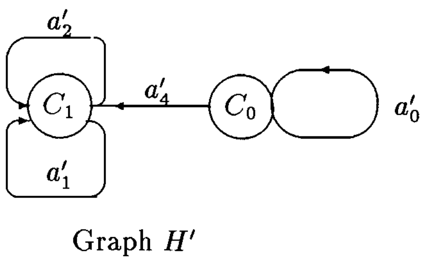


Figure 3. Illustration to the algorithm for the covering problem.

After removing one of the loops a_1' or a_2' the algorithm has to be applied to the graph H' recursively.

4. Gaikov's algorithm for multistars

In this section, we sketch Gaikov's algorithm for computing a spanning tree with minimum branch weight on *multistars*. The main idea is to translate the problem of computing a spanning tree with minimum b-weight into an equivalent problem of computing an exact cover with minimum g-weight in a related directed multigraph.

Theorem 4.1. *Let $G = (V, E)$ be a Eulerian multistar with $E = \{1, 2, \ldots, r\}$ where the edge sequence $1, \ldots, r$ forms a Eulerian path. Let w be a weight function on the edge sets $\{i, i+1, \ldots, j\}$, $1 \le i \le j \le r$, that fulfills property* (4). *Then a spanning tree for G with minimum branch weight can be computed in $O(|V|^2 + |E|)$ time.*

Proof: Let $v_0 \in V$ denote the central vertex of the multistar (the vertex that is incident to all edges), and let $v_1, \ldots, v_m$ be an enumeration of the other vertices in V. It is easy to see that for multistars, every branch of a spanning tree contains at most two edges. We call two consecutive edges $i = [v_0, v_s]$ and $i+1 = [v_0, v_t]$ in E a *critical pair,* if $v_s \neq v_t$ holds (and hence, they are potential candidates for forming a branch in some spanning tree). Since $1, \ldots, r$ forms a Eulerian path, every edge participates in at most one critical pair. Edges that are not part of any critical pair are called *non-critical edges*.

The construction of the related directed multigraph $H = (V, A)$ is done as follows. The vertex set N of H equals $\{v_1, \ldots, v_m\}$. For every non-critical edge $i = [v_0, v_t]$ in E, we introduce a loop $\ell = (v_t, v_t)$ in A that is incident to vertex v_t and has weight $g(\ell) = w_{ii}$. For every critical pair of edges $i = [v_0, v_s]$ and $i+1 = [v_0, v_t]$ in E, we introduce a twin-pair a and b of arcs in A: $a = (v_t, v_s)$ and has weight $g(a) = w_{ii}$, $b = (v_s, v_t)$ and has weight $g(b) = w_{i+1,i+1}$. The weight $g(a, b)$ is set to $w_{i,i+1}$. (A directed graph H for the multistar shown in figure 4 is depicted in figure 2. Weights g are to be defined as $g(a_i) = w_{ii}$, for $i = 1, \ldots, 5$, $g(a_1, a_2) = w_{12}$, $g(a_3, a_4) = w_{34}$.)

It is easy to see that G has a spanning tree with branch weight W if and only if H has an exact cover with Gaikov weight w: There is a one-to-one correspondence between edges

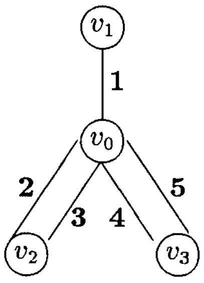


Figure 4. A multistar.

in G and arcs in H such that an edge connects some vertex v_t to v_0 in G if and only if the corresponding arc covers vertex v_t in H. Applying Theorem 3.6 completes the proof. $\square$

5. A fast algorithm for multitrees

This section deals with the problem of computing a spanning tree with minimum branch weight on *multitrees* $G = (V, E)$. A fast $O(|V|^3 + |V||E|)$ algorithm is presented that is based on dynamic programming and that uses the algorithm for multistars as a subprocedure.

Let $G = (V, E)$ be a Eulerian multigraph with $E = \{1, 2, \ldots, r\}$ where the edge sequence $1, \ldots, r$ forms a Eulerian path. Let w be a weight function on the edge sets $\{i, i+1, \ldots, j\}$, $1 \leq i \leq j \leq r$, that fulfills property (4). An edge set (a branch) $\mathcal{I} = \{i, i+1, \ldots, j\}$ is called a *maximal branch* with respect to G, if there exists a spanning tree for G that contains all edges in $\mathcal{I}$ and if this property is not fulfilled by any branch that properly contains $\mathcal{I}$. Since $1, \ldots, r$ forms a Eulerian path, every edge in E is contained in exactly one maximal branch and thus, the maximal branches constitute a partition of E. Let $\mathcal{I}_1, \ldots, \mathcal{I}_b$ be an enumeration of the maximal branches.

We root the multitree at an arbitrary vertex and derive the usual father-son relations. For $v \in V$, let $T(v)$ denote the maximal subtree rooted at v. Since there is no danger of confusion, we will use $T(v)$ also to denote its set of vertices and to denote its set of edges. For a maximal branch $\mathcal{I}_k$, consider the smallest tree $T(v)$ that contains $\mathcal{I}_k$. The corresponding vertex v is called the *anchor* of the branch $\mathcal{I}_k$. It is easily seen that $\mathcal{I}_k$ behaves as follows: $\mathcal{I}_k$ decomposes into an increasing subpath that goes up to the anchor v and into a decreasing subpath that goes down from v. The first edge in the increasing subpath and the last edge in the decreasing subpath are called the *lowermost* edges of $\mathcal{I}_k$. With this, every edge in $\mathcal{I}_k$ (with the exception of the edges that are incident to the anchor) has a unique *upper neighbor* and every edge in $\mathcal{I}_k$ (with the expection of the lowermost edges) has a unique *lower neighbor*.

For $1 \leq i, j \leq r$, we denote by $[i, j]^*$ the edge set that contains all numbers from $\min\{i, j\}$ to $\max\{i, j\}$. By $w[i, j]^*$, we denote the weight of this set.

In order to derive a fast algorithm for finding a spanning tree with minimum branch weight, we define for every vertex $v \in V$ the following arrays and a value $\textsc{Opt}_v$:

- $X_v^1[e_1, e_2]$ is a two-dimensional array where e_1 is an edge that connects v to one of its sons u, and e_2 is an edge in $T(u) \cup \{e_1\}$ that belongs to the same maximal branch $\mathcal{I}_k$ as e_1 does (in other words, e_1 and e_2 belong to the same subpath of $\mathcal{I}_k$). The entry $X_v^1[e_1, e_2]$ contains the minimum b-weight of all spanning trees for the vertices in $T(u) \cup \{v\}$ that contain all edges in $[e_1, e_2]^*$ but no other edges of $\mathcal{I}_k$.
- $X_v^2[e]$ is a one dimensional array where e is an edge that connects v to one of its sons u. $X_v^2[e]$ contains the minimum b-weight of all spanning trees for the vertices in $T(u) \cup \{v\}$ that contain the edge e.
- $X_v^3[e_1, e_2]$ is a two dimensional array where $e_1 = [v, u]$ and $e_2 = [v, u']$ are two edges in $\mathcal{I}_k$ that are both incident to v and where v is the anchor of $\mathcal{I}_k$. $X_v^3[e_1, e_2]$ contains the minimum b-weight of all spanning trees for the vertices in $T(u) \cup T(u') \cup \{v\}$ that contain the two edges e_1 and e_2.

- $Z_v^1[e]$ is a one dimensional array where $u \in V$ is a son of v. $Z_v^1[u]$ contains the minimum b-weight of all spanning trees for the vertices in $T(v) \backslash T(u)$.
- $Z_v^2[e]$ is a one dimensional array where e is an edge that connects v to one of its sons u. $Z_v^2[u]$ contains the minimum b-weight of all spanning trees for the vertices in $T(v)$ using edges in $T(v) \backslash \{e\}$.
- OPT_v contains the minimum b-weight of all spanning trees for $T(v)$.

The entries of the arrays X_v^1, X_v^2, X_v^3, Z_v^1 and Z_v^2 are only defined if the concerned vertices and edges fulfill the described relationships. The values of X_v^1, X_v^2, X_v^3, Z_v^1, Z_v^2 and OPT_v are computed in a bottom-up fashion, starting at the leaves and moving up towards the root. When we are dealing with a father, the arrays of all its sons have already been computed.

Next, we describe how to perform these computations. For a leaf v in the tree, the arrays X_v^1, X_v^2, X_v^3, Z_v^1 and Z_v^2 are not defined, and the value OPT_v clearly equals zero. Now let v be a non-leaf vertex with sons $u_1, \ldots, u_d$.

Computation of $X_v^1[e_1, e_2]$: Let $e_1 = [v, u]$ with $u \in \{u_1, \ldots, u_d\}$, and let $e_1, e_2 \in \mathcal{I}_k$. First consider the case where e_1 is a lowermost edge of $\mathcal{I}_k$. Then $e_2 = e_1$ must hold and e_1 cannot form a branch with any edge in $T(u)$. Clearly, in this case $X_v^1[e_1, e_2]$ equals $\text{OPT}_u + w(e_1)$. There remains the case where e_1 is not a lowermost edge of $\mathcal{I}_k$. In this case let $e_3 = [u, u']$ denote its lower neighbor in $\mathcal{I}_k$. If $e_2 = e_1$, then set $X_v^1[e_1, e_2] := Z_u^2[e_3] + w(e_1)$. Finally, if $e_2 \neq e_1$, then set

$$X_v^1[e_1, e_2] := X_u^1[e_3, e_2] + Z_u^1[u'] + w[e_1, e_2]^* - w[e_3, e_2]^*.$$

Computation of $X_v^2[e]$: Let $e = [v, u]$ with $u \in \{u_1, \ldots, u_d\}$. $X_v^2[e]$ equals the minimum over all defined values $X_v^1[e, e_1]$ where e_1 is an edge of $\mathcal{I}_k$ in $T(u) \cup \{e\}$.
Computation of $X_v^3[e_1, e_2]$: Let $e_1 = [v, u]$ and $e_2 = [v, u']$ be two edges in $\mathcal{I}_k$. Then $X_v^3[e_1, e_2]$ equals the minimum over all edges e_3 in $T(v)$ that belong to the same subpath of $\mathcal{I}_k$ as e_1 and over all edges e_4 in $T(v)$ that belong to the same subpath of $\mathcal{I}_k$ as e_2 of

$$\min_{e_3, e_4} X_v^1[e_1, e_3] + X_v^1[e_2, e_4] + w[e_3, e_4]^* - w[e_1, e_3]^* - w[e_2, e_4]^*.$$

This expression checks all possibilities for the first edge e_3 and the last edge e_4 of the branch $\mathcal{I}_k$ in a spanning tree. The branch weight of the two branches $[e_1, e_3]^*$ and $[e_2, e_4]^*$ has to be replaced by the weight of their union $[e_3, e_4]^*$.
Computation of OPT_v: Construct an equivalent problem on a multistar as follows: The multistar has vertex set $\{v, u_1, \ldots, u_d\}$ with central vertex v. For every edge $e = [v, u']$ with $u' \in \{u_1, \ldots, u_d\}$, in the multitree, there is a corresponding edge $[v, u']$ in the multistar with weight equal to $X_v^2[e]$. For every branch $\mathcal{I}_k$ in G that is anchored at v, let e_1 and e_2 be the corresponding two edges in $\mathcal{I}_k$ that are incident to v. These two edges form a branch of length two and with weight $X_v^3[e_1, e_2]$ in the multistar. These are the only branches of length greater than one in the multistar. OPT_v is set to the weight of the minimum b-weight spanning tree for this multistar.

Computation of $Z_v^1[u]$: After deleting $T(u)$ the value $Z_v^1[u]$ is computed analogously to the computation of $\textsc{Opt}_v$.
Computation of $Z_v^2[e]$: If edge e is not included in the minimum b-weight spanning tree corresponding to $\textsc{Opt}_v$, then $Z_v^2[e] := \textsc{Opt}_v$. Otherwise, the computation is similar to the computation of $\textsc{Opt}_v$ above: Construct a multistar with central vertex v and introduce for every edge (with the exception of edge e) a corresponding edge in the multistar. Edges belonging to the same branch yield branches of length two in the multistar. Set $Z_v^2[e]$ to the weight of the minimum b-weight spanning tree for the multistar.

Clearly, the last vertex that is handled is the root r of the tree. Afterwards, the entry $\textsc{Opt}_r$ will contain the weight of the minimum b-weight spanning tree, i.e., the solution to the problem B-MST. Next, we analyze the overall time-complexity of the above procedure. This is done in two separate steps, first for the arrays X_v^1, X_v^2, X_v^3, and then for the arrays Z_v^1, Z_v^2, $\textsc{Opt}_v$:

It is convenient to analyze the computation of X_v^1, X_v^2 and X_v^3 via the concerned maximal branches. For every maximal branch $\mathcal{I}_k$ with r edges, the algorithm computes an overall number of $O(r^2)$ values $X_v^1[e_1, e_2]$ and each value is computed in constant time. Moreover, there have to be computed $O(r)$ entries $X_v^2[e]$, in $O(r)$ time per entry. Finally, there is exactly one entry $X_v^3[e_1, e_2]$ to be computed (where v is the anchor of $\mathcal{I}_k$) and the computation of this value takes $O(r^2)$ times. All in all, this costs $O(r^2)$ overall time for one maximal branch and $\sum_{i=1}^{b} r_i^2$ overall time for all maximal branches, where r_i is the number of edges in the maximal branch $\mathcal{I}_i$. Since $r_i \le |V| - 1$ (every maximal branch is a path) and $\sum_{i=1}^{b} r_i = |E|$ (the maximal branches form a partition of E), this overall time is in $O(|V|\,|E|)$.

Next, we investigate the cost of computing the entries in $\textsc{Opt}_v$, Z_v^1 and Z_v^2. This is done by considering the concerned vertex v. Assume that v has s sons in the rooted tree and that v is connected to its sons by an overall number of $t \ge s$ edges. The value $\textsc{Opt}_v$ can be computed in $O(s^2 + t)$ time (cf. Theorem 4.1 in Section 4). For every incident edge e that is not included in the minimum b-weight spanning tree, the value $Z_v^1[e]$ equals $\textsc{Opt}_v$ and hence can be found in constant time. For the remaining s edges, the value $Z_v^1[e]$ is computed in $O(s^2 + t)$ time. Hence, the computation of Z_v^1 can be done in $O(s^3 + ts)$ time. Summarizing, this yields $O(s^3 + ts)$ overall time for handling vertex v and $O(|V|^3 + |V|\,|E|)$ overall time for handling all vertices.

Summarizing, we formulate the following theorem.

Theorem 5.1. *Let $G = (V, E)$ be a Eulerian multigraph with $E = \{1, 2, \dots, r\}$ such that the edge sequence $1, \dots, r$ forms a Eulerian path. Let w be a weight function on the edge sets $\{i, i+1, \dots, j\}$, $1 \le i \le j \le r$, that fulfills property* (4). *Then a spanning tree for G with minimum branch weight can be computed in $O(|V|^3 + |V|\,|E|)$ time.*

6. NP-completeness for planar graphs

In this section, it is proved that the problem B-MST is NP-complete even for *planar* Eulerian multigraphs. The transformation is from the following NP-complete Hamiltonian path problem (cf. Plesnik, 1979).

Problem: Hamiltonian path in planar bipartite directed graphs with degree bound two (HP-PBD2)

Instance: A simple planar bipartite directed graph $H = (B \cup W, A)$ and two vertices $s \in B$ and $t \in W$ that fulfill the following properties:

(1) $|B| = |W| = m$ and $A \subseteq (B \times W) \cup (W \times B)$.
(2) All vertices in $B \backslash \{s\}$ have outdegree two and indegree one, s has outdegree two and indegree zero.
(3) All vertices in $W \backslash \{t\}$ have outdegree one and indegree two, t has outdegree zero and indegree two.

Question: Does H contain a Hamiltonian Path that goes from s to t?

Theorem 6.1. *For planar multigraphs $G = (V, E)$, problem* B-MST *is NP-complete.*

Proof: We will construct a Eulerian multigraph $G = (V, E)$, a weight function w on the edge sets and a bound K such that G allows a spanning tree with b-weight at most K if and only if the instance of HP-PBD2 has a Hamiltonian path.

Indeed, let $H = (B \cup W, A)$, $s \in B$ and $t \in W$ constitute an instance of HP-PBD2. Define $A_{BW} = A \cap (B \times W)$ and $A_{WB} = A \cap (W \times B)$. Two arcs $f_1 = (b_1, w_1)$ and $f_2 = (b_2, w_2)$ in A_{BW} are called *sisters* if $b_1 = b_2$ or $w_1 = w_2$ holds. Note that every arc in A_{BW} possesses exactly two sisters. The sister-relation naturally decomposes the arc set A_{BW} into cycles of even length which are called *sisterhoods*. Such a cycle alternatingly consists of clockwise and counter-clockwise directed arcs. For every sisterhood S, arbitrarily choose a *startvertex* and an orientation for the cycle and denote by $P(S)$ the undirected path that results from starting in the startvertex, running once through the cycle according to the chosen orientation and which ends again in the startvertex. Let s denote the total number of sisterhoods. Let $P_1, \dots, P_s$ be an enumeration of the undirected paths $P(S)$ coresponding to the sisterhoods. For every arc $(w, b) \in A_{WB}$, define another undirected path that contains only the edge $[w, b]$. Denote these $m - 1$ paths by $P_{s+1}, \dots, P_{s+m-1}$. Observe that the paths $P_1, \dots, P_{s+m-1}$ exactly cover A; in other words, for every edge in one of these paths there is a unique corresponding arc in the arc set A. Next, define undirected paths Q_k for $1 \le k \le s + m - 2$, that connect the endvertex of P_k to the startvertex of P_{k+1} as follows: Every Q_k contains at least one and at most $2m - 1$ edges, and it only uses edges whose directed counterpart occurs in H. The exact combinatorial structure of the Q_k is arbitrary and does not matter. Clearly, the concatenation $P^* = P_1 Q_1 P_2 \cdots Q_{s+m-2} P_{s+m-1}$ of the thus defined paths forms an undirected path that visits every vertex in $B \cup W$ at least once.

Finally, we define the graph G and the weight function w: The vertex set of $G = (V, E)$ equals $B \cup W$. The edge set E consists of all edges in P^*; these edges are numbered according to the ordering along P^*. The weights w_{ij} with $1 \le i \le j \le |E|$ are defined by $w_{ij} = s + j - i$ and thus all have value at least $s + 1$. It remains to define the weights w_{ii} for single edges. We distinguish between so-called *light* and *heavy* edges. Every edge belonging to any Q_k is a heavy edge of weight $s + 1$. Every edge belonging to any P_k is a light edge and has weight zero or one. In every P_k with $1 \leq k \leq s$, the first and the last edge (i.e., the two edges that are incident to the startvertex of the corresponding sisterhood) receive weight one. All remaining edges receive weight zero. Clearly, the so constructed graph G is a Eulerian multigraph. Since the paths P_k and Q_k only contain edges that are also present in H, the graph G is planar. It is easy to verify that the weights fulfill property (4).

We claim that G has a spanning tree with b-weight at most s if and only if the instance of HP-PBD2 has a Hamiltonian path.

(If). Assume, H contains a Hamiltonian Path P from s to t. Then P contains all $m-1$ arcs in A_{WB} (these are the only possibilities for leaving the vertices in $W \backslash \{t\}$), and from every sisterhood in A_{BW}, P either contains all arcs at the odd places or all arcs at the even places. Consider the spanning tree T_P that consists of the light edges corresponding to P. Trivially, the edge set of T_P does not contain any branches with two or more edges. Hence, the b-weight of T_P equals the weight of its single edges. From every sisterhood, P and T_P use exactly one of the two edges with weight one. Since there are s sisterhoods, this yields a total b-weight of s for T_P.

(Only if). Assume, G possesses a spanning tree T with b-weight s. Clearly, T consists of $2m-1$ light edges and it does not contain heavy edges. From any sisterhood, T can use at most half of the edges (otherwise, T would contain a branch of size at least two and of weight at least $s+1$). A straightforward counting argument yields that T uses all $m-1$ light edges corresponding to A_{WB} and m light edges corresponding to A_{BW}, and thus exactly half of the edges of every sisterhood. For no sisterhood S, the overall weight of edges in $S \cap T$ can be zero (otherwise, T contains a branch of size at least two). Hence, the overall weight of edges in $S \cap T$ equals one for all sisterhoods S. Summarizing, this yields that from every sisterhood, the tree T either contains all edges at the odd places or all edges at the even places. Now consider the arc set A_T in A that corresponds to T: A_T is connected and of cardinality $2m-1$. In A_T, every vertex in $W \backslash \{t\}$ has outdegree one and every vertex in B has outdegree one, and obviously A_T forms a Hamiltonian path from s to t. □

7. Discussion

In this paper we considered TSPs with permuted Monge matrices as cost matrices whose patching graph is a multistar or a multitree. By performing a careful analysis, we simplified Gaikov's theory and considerably accelerated the solution procedure in the case of multitrees. These considerations are not only a step towards a better understanding of the nature of the problem and its computational complexity, but they also offer a basis for new heuristics: We call a set of tours an *exponential neighborhood*, if it has an exponential size, but it is possible to find an optimal solution within this set in polynomial time. The papers Burkard and Deineko (1995) and Glover and Punnen (1995) describe some examples of exponential neighborhoods. The multitrees of this paper also give rise to a rather powerful exponential neighborhood, in the sense that an optimal solution can be found within a rather large set of tours.

Acknowledgments

This research has been supported by the Spezialforschungsbereich F003 “Optimierung und Kontrolle,” Projektbereich Diskrete Optimierung and by the START program Y43-MAT of the Austrian Ministry of Science.

References

R.E. Burkard and V.G. Deineko, “Polynomially solvable cases of the traveling salesman problem and a new exponential neighborhood,” *Computing*, vol. 54, pp. 191–211, 1995.

R.E. Burkard, V.G. Deineko, R. van Dal, J.A.A. van der Veen, and G.J. Woeginger, “Well-solvable special cases of the TSP: A survey,” *SIAM Reviews*, 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.

V.Ya. Burdyuk and V.N. Trofimov, “Generalizations of the results of Gilmore and Gomory on the solution of the travelling salesman problem (in Russian),” *Izv. Akad. Nauk SASS, Tech. Kibernet*. vol. 3, pp. 16–22, 1976. English translation in *Engineering Cybernetics*, vol. 14, pp. 12–18, 1976.

V.G. Deineko, “Applying dynamic programming to solving a speical traveling salesman problem (in Russian),” *Issledovanie operaziy i ASU* Kiev, vol. 16, pp. 47–50, 1979.

V.G. Deineko and V.L. Filonenko, “On the reconstruction of specially structured matrices (in Russian),” *Aktualnye Problemy EVMI Programmirovanie*, Dnepropetrovsk, DGU, pp. 43–45, 1979.

H.N. Gabov, “Data structures for weighted matching and nearest common ancestors with linking,” in *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, Philadelphia, pp. 434–443.

N.E. Gaikov, “On the minimization of a linear form on cycles (in Russian),” *Vestsi Akad. Navuk BSSR Ser. Fiz.-Mat. Navuk*, vol. 4, p. 128, 1980.

P.C. Gilmore and R.E. Gomory, “Sequencing a one state variable machine: a solvable case of the travelling salesman problem,” *Oper. Research*, vol. 12, pp. 655–679, 1964.

P.C. Gilmore, E.L. Lawler, and D.B. Shmoys, “Well-solved special cases,” chapter 4 in [12], pp. 87–143.

F. Glover and A.P. Punnen, “The traveling salesman problem: New solvable cases and linkages with the development of approximation algorithms,” Technical Report, University of Colorado, Boulder, 1995.

E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, *The Travelling Salesman Problem*, Wiley, Chichester, 1985.

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.

J. Plesnik, “The NP-completeness of the Hamiltonian cycle problem in planar digraphs with degree bound two,” *Information Processing Letters*, vol. 8, pp. 199–201, 1979.

V.I. Sarvanov, “On the complexity of minimizing a linear form on a set of cyclic permutations (in Russian),” *Dokl. Akad. Nauk SSSR*, vol. 253, pp. 533–534. English translation in *Soviet Math. Dokl.*, vol. 22, pp. 118–120, 1980.