ELSEVIER

Operations Research Letters 16 (1994) 245–254

operations research letters

Monge matrices make maximization manageable☆

Ulrich Pferschy, Rüdiger Rudolf*, Gerhard J. Woeginger

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

Received 1 January 1994; revised 1 July 1994

Abstract

We continue the research on the effects of Monge structures in the area of combinatorial optimization. We show that three optimization problems become easy if the underlying cost matrix fulfills the Monge property: (A) The balanced max-cut problem, (B) the problem of computing minimum weight binary k-matchings and (C) the computation of longest paths in bipartite, edge-weighted graphs. In all three results, we first prove that the Monge structure imposes some special combinatorial property on the structure of the optimum solution, and then we exploit this combinatorial property to derive efficient algorithms.

Keywords: Monge property; Monge matrix; Distribution matrix; Balanced cut; k-matching; Longest path

1. Introduction

Since many problems in combinatorial optimization turned out to be NP-hard, much attention has been paid to polynomial time solvable special cases. Basically, we distinguish two main types of such special cases: we may either restrict the *combinatorial structure* of the input (e.g. if we consider only special graph classes, if we forbid certain subconfigurations in the input or if we restrict ourselves to geometric versions of the problem) or we may restrict the *numerical structure* of the input (e.g. if we put constraints on certain cost coefficients). The contributions in this paper belong to the latter type of special cases.

A typical example for special cost matrices are the Monge matrices: An $n \times m$ matrix C with integer entries is called a *Monge matrix* iff C satisfies the *Monge property*

$$c_{ij} + c_{rs} \leqslant c_{is} + c_{rj} \quad \text{for all } 1 \leqslant i < r \leqslant n,\ 1 \leqslant j < s \leqslant m. \tag{1}$$

This property dates back to Gaspard Monge [14] and it was rediscovered in the more general form of *Monge sequences* by Hoffman [9]. Since then, much research has been done on the effects of Monge structures in combinatorial optimization. Monge matrices are also known as *distribution matrices* (cf. [8, 18]).

☆ This research was partially supported by the Fonds zur Förderung der wissenschaftlichen Forschung, Project P8971-PHY and by the Christian Doppler Laboratorium für Diskrete Optimierung.

* Corresponding author.

SSDI 0167-6377(94)00052-2

Many problems in combinatorial optimization become easier, if their underlying cost matrix fulfills the Monge property.

- As a consequence of a general result of Hofman [9], the well-known north-west-corner rule yields an optimal solution for all feasible demand and supply vectors to the classical Hitchcock transportation problem if (and only if) the cost matrix is a Monge matrix.
- Assignment problems on Monge matrices become trivial, since the sum of the diagonal elements always yields an optimal solution.
- Another example is the traveling salesman problem. It is solvable in linear time if the distance matrix is a Monge matrix [16].

The latter example also demonstrates that not only polynomially solvable problems admit faster solution algorithms, but even NP-hard problems become efficiently solvable if a Monge matrix is at hand.

In this paper, we will enlarge the class of combinatorial problems which become easier if the underlying cost matrix is a Monge matrix. Our three main results concern (1) the balanced cut problem, (2) the problem of computing a minimum weighted binary k-matching and (3) the computation of longest paths in bipartite edge-weighted graphs. We shortly introduce these three problems and delay most exact definitions to the main part of the paper.

1.1. Balanced cut problem

Cut and partitioning problems are frequently discussed problems in combinatorial optimization. Typical applications can be found in the book by Lengauer [13]. In some cases, it is required that the cardinalities of all subsets defined by the cut are equal or within some small range; such cuts are called *balanced cuts*. In a more mathematical formulation, given an edge-weighted undirected graph $G = (V, E)$, the goal is to find a partition of the node set into k equal-sized subsets such that the sum of the weights of all edges between nodes in different subsets (clusters) is maximized (*max-cut problem*) resp. minimized (*min-cut problem*).

In general, the computation of balanced max-cuts and min-cuts is NP-hard (cf. [6]). We will show that balanced cuts possess a very special combinatorial structure, if the matrix defined by the edge-weights is a Monge matrix. Similarly as in the assignment problems mentioned above, this special combinatorial structure will make the problem trivial to solve.

1.2. Weighted perfect binary k-matching

For an integer $k \geqslant 1$, the problem of finding a minimum weighted perfect binary k-matching in a given $n \times n$ matrix C can be formulated as an integer program in the following way (cf. [15]):

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\text{subject to} \sum_{j=1}^{n} x_{ij} = k \quad \forall i = 1, \ldots, n,$$

$$\sum_{i=1}^{n} x_{ij} = k \quad \forall j = 1, \ldots, n,$$

$$x_{ij} \in \{0, 1\} \quad \forall 1 \leqslant i, j \leqslant n.$$

In words, we want to find a subset $\mathcal{K}$ of the entries of C which contains exactly k elements of each row and each column and which has minimum sum under this condition. Sometimes, k-matchings are also called *k-factors*.

A good approach to find a minimum weight k-matching in an arbitrary $n \times n$ matrix C is to consider a corresponding minimal cost flow problem. Therefore we construct a network N which consists of the original bipartite graph with an additional source and sink. The capacities of all arcs leaving the sink or entering the source are fixed to k and all other capacities to 1. It is well known that a minimal cost flow problem in a network N with n vertices and m edges with flow value v can be solved by an augmenting path method in $O(vS(n, m))$ time where $S(n, m)$ denotes the time to find a shortest path in N. Using the algorithm of Fredman and Tarjan [5] for finding shortest paths and taking into account that $v = kn$ the minimum weight k-matching can be solved in $O(kn(m + n \log n))$ time. For fixed k and $m = O(n^2)$, this time complexity is $O(n^3)$. We will show how to improve this complexity down to $O(n)$, if C fulfills the Monge property.

1.3. Longest path problem

One of the first results in the area of NP-completeness was that the problem of computing a longest path in a graph is NP-hard [11]. The NP-completeness proof easily carries over to the case of bipartite graphs (cf. [6]). In the third main result in this paper, we will show that once more the problem becomes solvable in polynomial time if the associated distance matrix $C = (c_{ij})$ of the bipartite graph is a Monge matrix.

Although *longest* path problems with underlying Monge costs have not been investigated until now, there exist several papers dealing with the comptutation of *shortest* paths with costs fulfilling certain Monge-type properties (for details see [1–3]).

1.4. Organization of the paper

In Section 2 we investigate the problem of finding maximum weight balanced cuts in Monge matrices. Section 3 derives a fast algorithm for computing a minimum weighted perfect binary k-matching in Monge matrices. In Section 4 we present a polynomial time algorithm for computing the longest path in a bipartite graph if the cost matrix is Monge.

2. Balanced cut problems

In this section we show that the *balanced k-cut problem* is efficiently solvable if the underlying cost matrix is a Monge matrix. Balanced k-cut problems are defined in the following way.

Balanced k-Cut Problem

Given an undirected graph $G = (V, E)$, $|V| = n = kp$, with edge weights c_{ij}, find a partition of the nodes into k subsets (clusters) $S_1, \ldots, S_k$ such that $|S_\ell| = p$ for $1 \leqslant \ell \leqslant k$ and such that the sum of all weights of edges between nodes in different sets is maximized.

The general version of this problem is usually called *max-cut* problem. It is known to be NP-complete even if all edge weights are equal to 0 or 1, if $k = 2$ and if no condition on the size of the clusters has to be satisfied [7]. In all special cases of the max-cut problem that have been examined till today, the *combinatorial structure* of the input was restricted to special graph classes (a summary on several polynomial and NP-hard special cases of the max-cut problem is given in [10]).

Until now, no restrictions on the numerical structure of balanced cut problems have been investigated. In the sequel we will show that the problem becomes easy if the underlying cost matrix C is a Monge matrix. First we prove the theorem for $k = 2$ (i.e. we solve the so-called *bisection problem*) and afterwards we will use this result to solve the case for arbitrary k.

Theorem 2.1. *Let $G = (V, E)$ be a complete, edge-weighted, undirected graph with node set $V = \{v_1, \dots, v_n\}$, where $n = 2p$ for some integer $p \geq 1$. Let $C = (c_{ij})$ denote the associated $n \times n$ cost matrix.*

In case C is a Monge matrix, then the maximal balanced 2-cut is constituted by the sets $S_1 = \{v_1, \dots, v_p\}$ and $S_2 = \{v_{p+1}, \dots, v_n\}$.

Proof. We will show that an arbitrary bipartition of V into two balanced sets A and B can always be transformed into the bipartition $S_1 = \{v_1, \dots, v_p\}$ and $S_2 = \{v_{p+1}, \dots, v_n\}$ without decreasing the objective value by exchanging appropriate pairs of elements between sets A and B. For two disjoint subsets $U, W \subseteq V$, we denote by $d(U, W) = \sum_{v_i \in U, v_j \in W} c_{ij}$ the overall weight of all edges between U and W.

Since A and B define a balanced partition of V, the two sets $X_A := \{v_i \mid v_i \in A, i > p\}$ and $X_B := \{v_i \mid v_i \in B, i \leq p\}$ have the same cardinality (these sets contain all elements that violate the desired solution structure). We fix an arbitrary bipartite perfect matching $\mathcal{M}_x$ between the elements of X_A and X_B and an arbitrary bipartite perfect matching $\mathcal{M}_y$ of the elements of $Y_A := A \setminus X_A$ with the elements of $Y_B := B \setminus X_B$. Note that all vertices in X_B have smaller indices than all vertices in X_A, and that all vertices in Y_A have smaller indices than all vertices in Y_B. Moreover observe that if we exchange each pair of matched elements in X_A and X_B, we transform the bipartition (A, B) into (S_1, S_2).

We claim that the value of the induced cut is not decreased by applying this exchange step, i.e. that

$$d(X_B, Y_A) + d(X_A, Y_B) \leq d(X_B, Y_B) + d(X_A, Y_A) \tag{2}$$

holds (edges whose endpoints are both exchanged or whose endpoints are both not exchanged are irrelevant for the change of the cut value and therefore are not considered in the above inequality).

Consider some pair $(v_i, v_r) \in X_B \times X_A$ that is connected by an edge in the matching $\mathcal{M}_x$. All edges connecting v_i to nodes in Y_A are removed from the cut and all edges between v_i and nodes in Y_B are added to the cut. The reverse observation holds for node v_r. Since the cost matrix is a Monge matrix and since by the definition of X_A and X_B $i < r$ holds, we have

$$c_{ij} + c_{rs} \leq c_{is} + c_{rj}$$

for every pair j, s such that $v_j \in Y_A$ and $v_s \in Y_B$ are connected by an edge in $\mathcal{M}_y$. Summing this inequality over all edges in $\mathcal{M}_x$ and all edges in $\mathcal{M}_y$ yields inequality (2). Thus our theorem is proven. □

The following conclusion establishes our general result.

Corollary 2.2. *Let $G = (V, E)$ be a complete, edge-weighted, undirected graph with node set $V = \{v_1, \dots, v_n\}$, where $n = kp$ for some integers $p, k \geq 1$. Let $C = (c_{ij})$ denote the associated $n \times n$ cost matrix.*

If C is a Monge matrix then the maximal balanced k-cut consists of the sets $S_1 = \{v_1, \dots, v_p\}$, $S_2 = \{v_{p+1}, \dots, v_{2p}\}$ up to $S_k = \{v_{(k-1)p+1}, \dots, v_n\}$.

Proof. It is sufficient to show that in any maximal balanced k-cut with clusters $A_1, \dots, A_k$, every pair A_i, A_j represents a maximal balanced 2-cut on the submatrix induced by $A_i \cup A_j$. Otherwise, the value of this 2-cut could be increased by repartitioning $A_i \cup A_j$ according to Theorem 2.1, and without affecting the edges in the cut that concern the nodes in $V \setminus \{A_i \cup A_j\}$). □

We close this section with two final remarks.

Remark 2.3. (1) The optimal solution of a max-cut problem on a Monge cost matrix can be found without any computational effort and just has to be reported.

(2) Since the entries on the main diagonal cannot contribute to the value of the cut, their values are irrelevant for the solution. Therefore, our results also hold for Supnick matrices (A Supnick matrix is a matrix

with unspecified diagonal entries where the specified entries fulfill some kind of Monge property. For more information, especially on the relation between Monge and Supnick matrices, the reader is referred to [17, 4].)

3. Weighted perfect binary k-matchings

In this section we investigate the problem of finding a minimum weighted perfect binary k-matching in a given $n \times n$ Monge matrix C.

Minimum Weight Perfect Binary k-Matching Problem
Given an $n \times n$ matrix $C = (c_{ij})$ and an integer $k \geq 1$, the goal is to find a subset of entries with minimum total weight such that every row and every column of C contains exactly k entries of this subset.

First of all we note that – as already mentioned in the introduction – the optimum value of a minimum weight 1-matching (=minimum weight assignment) in Monge matrices can be computed in $O(n)$ time by simply summing up the diagonal entries of C. Therefore, we only investigate the problem of finding minimum weight k-matchings in Monge matrices for $k \geq 2$. We will show that there always exists an optimum solution with a very special structure. Exploiting this structure algorithmically yields an $O((2k)^{4k}n)$ time algorithm for finding the value of a minimum weight k-matching. For constant k this linear time algorithm is an improvement over the general $O(n^3)$ algorithm.

Now let C be an $n \times n$ Monge matrix. We call a subset S of the entries in C a *nice* set iff the following configuration does not occur: There are four indices $i_1 < i_2$ and $j_1 < j_2$ such that the entries $c_{i_1 j_2}$ and $c_{i_2 j_1}$ both are in S and $c_{i_1 j_1}$ and $c_{i_2 j_2}$ both are not in S. This configuration is called the *forbidden configuration.*

Observation 3.1. *In a Monge matrix, there always exists a minimum weight k-matching that is nice.*

Proof. We define $\text{ENERGY}(c_{ij}) = ij$ as the *energy* of entry c_{ij}. The energy of a subset of entries is the sum of the energies of its elements. Consider the minimum weight k-matching $\mathcal{K}$ that has maximum energy among all minimum weight k-matchings. We claim that $\mathcal{K}$ is nice.

Suppose, the forbidden configuration occurs in $\mathcal{K}$ at indices $i_1 < i_2$ and $j_1 < j_2$. Removing the entries $c_{i_1 j_2}$ and $c_{i_2 j_1}$ from $\mathcal{K}$ and inserting instead $c_{i_1 j_1}$ and $c_{i_2 j_2}$ cannot increase the weight of $\mathcal{K}$, but yields a k-matching with larger energy. □

Observation 3.2. *In a Monge matrix, there always exists a minimum weight k-matching that only contains entries c_{ij} with $|i - j| < 2k$.*

Proof. We show that the nice minimum weight k-matching $\mathcal{K}$ whose existence is guaranteed by Observation 3.1 has the required property.

Suppose, $\mathcal{K}$ would contain an entry $c_{i_0 j_0}$ with $|i_0 - j_0| \geq 2k$. W.l.o.g. we have $j_0 - i_0 \geq 2k$. Since $\mathcal{K}$ contains exactly $i_0 k$ entries in the first i_0 rows of C, $\mathcal{K}$ contains at most $i_0 k - 1$ entries c_{ij} with $1 \leq i \leq i_0$ and $1 \leq j < i_0 + 2k \leq j_0$. Since $\mathcal{K}$ contains exactly $(i_0 + 2k - 1)k$ entries in the first $i_0 + 2k - 1$ columns, it contains at least

$$(i_0 + 2k - 1)k - i_0 k + 1 = 2k^2 - k + 1 > 2(k-1)^2$$

entries with $i_0 < i \leq n$ and $1 \leq j < i_0 + 2k \leq j_0$. All these entries lie to the left and below $c_{i_0 j_0}$ or they lie somewhere else. It is easy to check that one of these entries together with $c_{i_0 j_0}$ yields the forbidden configuration, a contradiction. □

Next, we present our algorithm for computing minimum weight k-matchings. It is a simple dynamic programming algorithm based on the two observations above. Observation 3.2 tells us that there always exists an optimum solution using only elements in a stripe of width $4k-1$ around the diagonal of C. The main idea is to construct the optimum solution by a scan through this stripe from the upper left corner down to the lower right corner.

For $d \geq 1$, we define a *partial k-matching above d* to be a subset $\mathcal{K}'$ of the entries in C with the following properties:

(i) All elements c_{ij} in $\mathcal{K}'$ fulfill $|i-j| < 2k$.

(ii) Every row $i \leq d$ contains exactly k elements of $\mathcal{K}'$, and every row $i > d$ does not contain any elements of $\mathcal{K}'$.

(iii) Every column $j \leq d-2k$ contains exactly k elements of $\mathcal{K}'$, and every column $d-2k < j < d+2k$ contains at most k elements of $\mathcal{K}'$. Columns j with $j \geq d+2k$ do not contain elements of $\mathcal{K}'$.

Next, we introduce a $(4k)$-dimensional integer-valued array $\text{AR}[d, i_1, \ldots, i_{4k-1}]$. Index d runs from 1 to n, indices $i_1, \ldots, i_{4k-1}$ all go from 0 to k. If at the end of our computation the value of entry $\text{AR}[d, i_1, \ldots, i_{4k-1}]$ equals w, then the minimum weight partial k-matching above row d with exactly i_ℓ entries in column $d-2k+l$ has weight w.

We initialize $\text{AR}[\cdot] \equiv \infty$ in the beginning and compute the values of $\text{AR}[\cdot]$ level by level, going from $d=1$ up to $d=n$.

For $d=1$ this is easy: If one of the indices $i_1, \ldots, i_{4k-1}$ is ≥ 2 or if $\sum_{\ell=1}^{4k-1} i_\ell \neq k$ holds, then we set $\text{AR}[1, i_1, \ldots, i_{4k-1}] = \infty$ ($\mathcal{K}'$ contains exactly k entries from the first row; hence, every column contains at most one entry). Otherwise, every i_ℓ is either 0 or 1 and

$$\text{AR}[1, i_1, \ldots, i_{4k-1}] = \sum_{\ell=1}^{4k-1} c_{1\ell} \cdot i_\ell$$

holds. Thus the first level of $\text{AR}[\cdot]$ can be computed in $O(k)$ time per element.

For $d \geq 2$, there are at most $\binom{4k}{k}$ possibilities to choose k entries from row d. We check all these possibilities and combine them with all the minimum weight partial k matchings above $d-1$. This gives $O((2k)^{4k-1})$ combinations. For every combination, we do the following: If one of the columns contains more than k entries, we disregard this combination. Otherwise, we check whether the combination has smaller weight than the current entry of $\text{AR}[\cdot]$ corresponding to this combination, and in case it has, we update $\text{AR}[\cdot]$. Clearly, this procedure fills level d of $\text{AR}[\cdot]$ correctly in $O((2k)^{4k})$ time (we check $O((2k)^{4k-1})$ combinations with $O(k)$ time per combination).

After $O((2k)^{4k}n)$ steps, the whole array $\text{AR}[\cdot]$ is filled and the value of the minimum weight k-matching can be found in $\text{AR}[n, k, k, \ldots, k]$. Summarizing, we formulate the following theorem.

Theorem 3.3. *For an $n \times n$ Monge matrix C, the minimum weight k-matching can be computed in $O((2k)^{4k}n)$ time.*

Remark 3.4. (1) If in addition to the optimum value also the optimum k-matching is desired, this can be found easily by a backtracking procedure.

(2) In a bipartite graph with n vertices and m edges, the minimum weight k-matching can be found in $O(nk(m + n\log n))$ time by solving a corresponding minimum cost flow problem. Since Observation 3.2 restricts the number of interesting arcs to $m = O(nk)$ (all arcs linking node i and j such that $|i-j| \geq 2k$ need not to be considered), we thereby obtain another fast algorithm with time complexity $O(n^2k^2 + kn^2 \log n)$ for minimum weight k-matchings in Monge matrices.

4. Longest path in a bipartite graph

In this section we consider the problem of computing the longest path in an undirected, bipartite, edge-weighted graph G where the distances satisfy the Monge property.

Problem LBP
Given two sets $V = \{v_1, v_2, \dots, v_n\}$ and $W = \{w_1, w_2, \dots, w_m\}$ and the bipartite graph $G = (V \cup W, E)$ with $W = V \times W$; each edge $(i, j) \in E$ has nonnegative cost c_{ij}. The goal is to find a path going from v_1 to v_n maximizing total cost.

In the following it is shown that problem *LBP* is solvable in polynomial time, if the costs c_{ij} fulfill the Monge property. Once more, this result is based on additional structural properties of an optimal path from v_1 to v_n imposed by the Monge property. Since problem *LBP* is only reasonable for *simple* paths, all paths used in this section are supposed to be simple.

We start with some more notations and definitions. We will identify a path P in G with the sequence of its nodes (which lie alternatively in the sets V and W). Thus P is uniquely determined by two sequences of indices of nodes in the corresponding sets V and W, called $\mathcal{V}_P$ and $\mathcal{W}_P$. By $c(P)$ we denote the total length of path P.

Furthermore, we introduce the parameter $s(P)$ which measures the maximum length of a subpath P_1 of P going from v_1 to some v_i such that $\mathcal{V}_{P_1}$ is an increasing sequence and such that $\mathcal{W}_{P_1}$ is a decreasing sequence: Given a path P determined by $\mathcal{V}_P = \langle x_1, \dots, x_r \rangle$ and $\mathcal{W}_P = \langle y_1, \dots, y_{r-1} \rangle$, we define the numbers

$$s_{\mathcal{V}}(P) = \max\{\ell \mid x_1 < x_2 < \cdots < x_\ell < x_i \ \forall i > \ell\},$$

$$s_{\mathcal{W}}(P) = \max\{\ell \mid y_1 > y_2 > \cdots > y_\ell > y_i \ \forall i > \ell\}.$$

With this, $s(P)$ is defined by $s(P) = \min\{2s_{\mathcal{V}}(P) - 1, 2s_{\mathcal{W}}(P)\}$.

Lemma 4.1. *Let P be any longest path from v_1 to v_n with node sets $\mathcal{V}_P = \langle x_1, \dots, x_r \rangle$ and $\mathcal{W}_P = \langle y_1, \dots, y_{r-1} \rangle$. Then there exists an optimal path P^* using the same nodes as P with $s(P^*) = 2r - 1$.*

Proof. We assume that $s(P) < 2r - 1$ (since otherwise we may choose $P^* = P$). In the following we will show that there always exists another optimal path P^* from v_1 to v_n that uses the same nodes as P and fulfills $s(P^*) > s(P)$. Repeating this argument until $s(P^*) = 2r - 1$ completes the proof.

We distinguish two cases depending on the parity of $s(P)$.

Case 1: If $s(P)$ is odd (i.e. $s(P) = 2s_{\mathcal{V}}(P) - 1$), then we define a number $k = (s(P) + 3)/2$ and an index ℓ with $x_\ell = \min\{x_i \in \mathcal{V}_P \mid i > k\}$. Note that the definition of $s(P)$ implies $x_k > x_\ell$. Denote by P_1 the subpath of P from v_{x_1} to $w_{y_{k-1}}$, by P_2 the path from $w_{y_{k+1}}$ to $w_{y_{\ell-1}}$ and by P_3 the path from $w_{y_{l+1}}$ to v_{x_r}. With this, path P can be written as $P = P_1 - v_{x_k} - P_2 - v_{x_\ell} - P_3$. We construct a new path P^* by reversing the subpath from v_{x_k} to v_{x_ℓ}. Thus P^* can be written as $P^* = P_1 - v_{x_\ell} - \bar{P}_2 - v_{x_k} - P_3$, where $\bar{P}_2$ denotes the reverse path according to P_2. The total length of this new path P^* is given by

$$c(P^*) = c(P) + c[x_\ell, y_{k-1}] + c[x_k, y_\ell] - c[x_\ell, y_\ell] - c[x_k, y_{k-1}].$$

Since $x_\ell < x_k$, $y_{k-1} > y_\ell$, $2s_{\mathcal{W}}(P) > 2s_{\mathcal{V}}(P) - 1$ and since C is a Monge matrix, we know that

$$c[x_\ell, y_{k-1}] + c[x_k, y_\ell] - c[x_\ell, y_\ell] - c[x_k, y_{k-1}] \geq 0,$$

which implies that $c(P^*) \geq c(P)$, i.e. P^* is also optimal. Since x_ℓ was chosen to be the minimum of all indices x_i with $i > k$, we have that $s_{\mathcal{V}}(P^*) > s_{\mathcal{V}}(P)$ and $s_{\mathcal{W}}(P^*) \geq s_{\mathcal{W}}(P)$. Hence $s(P^*) > s(P)$ holds and the first case is settled.

Case 2: If $s(P)$ is even, we can find another optimal path P^* with $s(P^*) > s(P)$ on the same node set as P in an analogous way to Case 1. Thus the lemma is proven. □

Lemma 4.1 shows that in order to compute the longest path it is sufficient to inspect all those paths P from v_1 to v_n with $2r-1$ nodes for which $s(P) = 2r-1$ holds. The next lemma further reduces the number of potential solution paths.

Lemma 4.2. *Let P be a path from v_1 to v_n with node sets $\mathcal{V}_P = \langle x_1, \ldots, i, p, \ldots, x_r\rangle$ and $\mathcal{W}_P = \langle y_1, \ldots, q, j, \ldots, y_{r-1}\rangle$ and $s(P) = 2r-1$. Let $n \geqslant p > i+1 > 1$ and $m \geqslant q > j+1 > 1$. If P contains a subpath P' with*
(i) $P' = v_i - w_q - v_p - w_j$ *or*
(ii) $P' = w_q - v_i - w_j - v_p$,
then there exists a path P^* with node set $\mathcal{V}_{P^*} = \langle x_1, \ldots, i, i+1, p, \ldots, x_r\rangle$ and $\mathcal{W}_{P^*} = \langle y_1, \ldots, q, j+1, j, \ldots, y_{r-1}\rangle$ such that $s(P^*) = 2r+1$ and $c(P^*) \geqslant c(P)$.

Proof. We will only prove part (i); part (ii) can be settled by analogous arguments. Hence, assume that we are given a path P with $s(P) = 2r-1$ which contains a subpath $P' = v_i - w_q - v_p - w_j$ with $q > j+1$ and $p > i+1$. Denote by P_1 and P_2 those subpaths of P such that P can be written as $P = P_1 - P' - P_2$. Since $q > j+1$ and $p > i+1$ holds, we can define the following path $P^* = P_1 - v_i - w_q - v_{i+1} - w_{j+1} - v_p - P_2$. Since $s(P) = 2r-1$ holds, we have $s(P^*) = 2r+1$. Since all costs are nonnegative and since C is a Monge matrix, we see that

$$c[i+1, j+1] + c[p, j+1] + c[i+1, q] \geqslant c[p, q].$$

This implies that $c(P^*) \geqslant c(P)$. □

In the following we present a dynamic algorithm for solving *LBP* in $O(nm)$ time. This algorithm exploits the structural properties of an optimal path given by Lemmas 4.1 and 4.2 by successively computing partial longest paths. We introduce two $n \times m$ arrays Vwv and Wvw. The entry $\mathrm{Vwv}[i,j]$ contains the length of the longest path starting at node v_1 and ending with the subpath $v_{i-1} - w_j - v_i$. Similarly, $\mathrm{Wvw}[i,j]$ stores the value of the longest path starting in v_1 and ending with the sequence $w_{j+1} - v_i - w_j$.

With this, solving problem *LBP* essentially reduces to computing all entries of Wvw and Vwv as given by the following recurrence relation: For $3 \leqslant i \leqslant n$ and $1 \leqslant j < m$,

$$\mathrm{Vwv}[i,j] = c[i,j] + \max \begin{cases} \mathrm{Wvw}[i-1, j], \\ c[i-1, j] + \max_{j+1 < q \leqslant m} \mathrm{Vwv}[i-1, q] \end{cases} \tag{3}$$

and $\mathrm{Vwv}[i,m] := c[i,m] + c[1,m]$ for $i \geqslant 2$ and $\mathrm{Vwv}[2,j] = c[2,j] + c[1,j]$ for all j.

For $i \geqslant 2$ and $j < m$ we have

$$\mathrm{Wvw}[i,j] = c[i,j] + \max \begin{cases} c[i, j+1] + c[1, j+1], \\ \mathrm{Vwv}[i, j+1], \\ c[i, j+1] + \max_{1 < p < i-1} \mathrm{Wvw}[p, j+1]. \end{cases} \tag{4}$$

Theorem 4.3. *Let* Vwv *and* Wvw *be given as above. Then the optimum value of a longest path from* v_1 *to* v_n *is given by*

$$\max\left\{\max_{\substack{1<j\leqslant m\\ 1\leqslant i<n-1}}\{\mathrm{Wvw}[i,j]+c[n,j]\},\ \max_{1\leqslant j\leqslant m}\mathrm{Vwv}[n,j]\right\}. \tag{5}$$

Proof. First of all we show that the above recurrence relation indeed describes a legal rule how to compute all entries of Vwv and Wvw. Consider an arbitrary entry $\mathrm{Vwv}[i,j]$, $i\geqslant 2$ and $1\leqslant j\leqslant m$. This entry should contain the value of an optimal path P starting at v_1 and ending with the three vertices $v_{i-1}-w_j-v_i$. In case $i=2$, it is easy to verify that $\mathrm{Vwv}[2,j]=c[1,j]+c[2,j]$. If $i\geqslant 3$ then there exist at least two nodes preceding v_{i-1}, such that the optimal path P ends with $v_p-w_q-v_{i-1}-w_j-v_i$. Because of Lemma 4.1, we may further assume $q>j$ and $p<i-1$. We distinguish two cases.

(i) $q>j+1$: This means that P contains a subpath $P'=v_p-w_q-v_{i-1}-w_j-v_i$. If $p<i-2$ then P' is a subpath fulfilling the requirements of Lemma 4.2 and for that reason path P need not be considered (there always exists another optimal path distinct from P). In case $p=i-2$, P can be decomposed into a path P^* starting at v_1 and ending with $v_{i-2}-w_q-v_{i-1}$ and into the two edges connecting v_{i-1} to w_j and w_j to v_i. Since the value of an optimal path P^* is given by $\mathrm{Vwv}[i-1,q]$, the total length of path P is $\mathrm{Vwv}[i-1,q]+c[i-1,j]+c[i,j]$. It remains to take the maximum over all paths depending on q.

(ii) $q=j+1$: Note that in this case path P may be written as P^*-v_i, where P^* is a path starting in v_1 and ending with $w_{j+1}-v_{i-1}-w_j$. Since the optimal value of all possible paths P^* is determined by $\mathrm{Wvw}[i-1,j]$, the optimum value of path P is given by $\mathrm{Wvw}[i-1,j]+c[i,j]$.

Putting both cases together we see that the optimal path P starting in v_1 and ending with $v_{i-1}-w_j-v_i$ is exactly described by (3). The correctness of $\mathrm{Wvw}[i,j]$ can be seen in a similar way.

It remains to be shown that the optimum length of a path P from v_1 to v_n is given by (5): Every solution path P ends in v_n. Now, either P ends with $v_{n-1}-w_j-v_n$ (then its value is determined by $\mathrm{Vwv}[n,j]$), or it ends with $w_{j+1}-v_i-w_j-v_n$ (then its value is given by $\mathrm{Wvw}[i,j]+c[n,j]$). Evaluating this maximum yields expression (5). □

Summarizing, for solving *LBP* it is sufficient to compute the arrays Vwv and Wvw as described by (3) and (4). We show that all entries of both arrays can be computed in total in $O(nm)$ time. We introduce two additional $n\times m$ arrays A and B.

$$A[i,j]:=\max_{j+1<q\leqslant m}\mathrm{Vwv}[i-1,q],\qquad B[i,j]:=\max_{1<p<i-1}\mathrm{Wvw}[p,j+1].$$

Note that $A[i,j-1]$ is determined by

$$A[i,j-1]=\max\{A[i,j],\mathrm{Vwv}[i-1,j+1]\}$$

and that $B[i+1,j]$ is given by

$$B[i,j]=\max\{B[i+1,j],\mathrm{Wvw}[i-1,j+1]\}.$$

Thus, evaluating (3) and (4) by increasing first index and decreasing second index, we can compute all entries of Vwv, Wvw, A and B in constant time per entry from already known values. Since the initialization step and the evaluation of expression (5) can also be performed in $O(nm)$ time, we have proven the following theorem.

Theorem 4.4. *In a complete bipartite, edge-weighted graph* $G=(V,E)$ *with bipartition* $V=V_1\cup V_2$ *where all edge-weights are nonnegative and fulfill the Monge property, problem LPB can be solved in* $O(|V_1||V_2|)$ *time.*

Acknowledgement

We would like to thank Bettina Klinz and Günter Rote for helpful discussions in the early stages of this research.

References

[1] A. Aggarwal, B. Schieber and T. Tokuyama, "Finding a minimum weight K-link path in graphs with Monge property and applications", in: *Proc. 9th Ann. ACM Symp. on Computational Geometry*, San Diego, California, 19–21 May, 1993, pp. 189–197.
[2] W. Bein, P. Brucker and J.K. Park, "Applications of an algebraic Monge property", unpublished manuscript, presented at the 3rd Twente Workshop on Graphs and Combinatorial Optimization, Enschede, The Netherlands, June 1993.
[3] W. Bein, L. Larmore and J. Park, "The d-edge shortest-path problem for a Monge graph", preprint, 1992.
[4] R.E. Burkard, "Special cases of travelling salesman problems and heuristics", *Acta Math. Appl. Sinica* **6**, 273–288 (1990).
[5] M.L. Fredman and R.E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms", *J. Assoc. Comput. Mach.* **34**, 596–615 (1987).
[6] M.R. Garey and D.S. Johnson, *Computers and Intractability*, Freeman, San Francisco, 1979.
[7] M.R. Garey and D.S. Johnson and L. Stockmeyer, "Some simplified NP-complete graph problems", *Theory Comput. Sci.* **1**, 237–267 (1976).
[8] P.C. Gilmore, E.L. Lawler and D.B. Shmoys, Well-Solved Special Cases, Ch. 4 in [12], pp. 87–143.
[9] A.J. Hoffman, "On simple linear programming problems", in: V. Klee (ed.), *Convexity, Proc. Symposia in Pure Mathematics*, Vol. 7, American Mathematical Society, Providence, RI, 1961, pp. 317–327.
[10] D.S. Johnson, "The NP-completeness column: an ongoing guide", *J. Algorithms* **6**, 434–451 (1985).
[11] R.M. Karp, "Reducibility among combinatorial problems", in: R.E. Miller and J.W. Thatcher (eds.), *Complexity of Computer Computations*, Advances in Computing Research, Plenum Press, 1972, pp. 85–103.
[12] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester, 1985.
[13] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, Wiley, Chichester, 1990.
[14] G. Monge, "Mémoires sur la théorie des déblais et des remblais", in: *Histoire de l'Academie Royale des Sciences, Année M. DCCLXXXI, avec les Mémoires de Mathématique et de Physique, pour la même Année, Tirés des Registres de cette Académie*, Paris, 1781, pp. 666–704.
[15] G.L. Nemhauser and L.A. Wolsey, *Integer and Combinatorial Optimization*, Wiley, New York, 1988.
[16] J.K. Park, "A special case of the n-vertex traveling salesman problem that can be solved in $O(n)$ time", *Inform. Process. Lett.* **40**, 247–254 (1991).
[17] F. Supnick, "Extreme Hamiltonean lines", *Ann. Math.* **66** (1), 179–201 (1957).
[18] J. van der Veen, "Solvable cases of the traveling salesman problem with various objective functions", Ph.D. Thesis, University Groningen, The Netherlands, 1992.