Two Algorithmic Results for the Traveling Salesman Problem

ALEXANDER I. BARVINOK *

Version of January 3, 2001

Abstract. An algebraic approach to the Traveling Salesman Problem is described which results in an algorithm for counting Hamiltonian circuits in a graph and in an approximation polynomial time algorithm for computing the longest Hamiltonian circuit with the given vertices in a normed space. For a graph with n vertices the counting algorithm has $2^{n+O(\log n)}$ time complexity whereas the space complexity is polynomial in n. For any norm in a Euclidean space and for any number $\delta > 0$ we present a polynomial time algorithm which computes a Hamiltonian circuit with the given vertices in the space whose length approximates, with relative error less than δ, the largest length of a Hamiltonian circuit with these vertices. As a by-product of our approach we prove that the permanent of a matrix can be computed in polynomial time provided the rank of the matrix is fixed.

Key words: Hamiltonian circuit, Traveling Salesman Problem, Combinatorial Optimization

1. Introduction

The Hamiltonian Circuit Problem and its weighted version known as the Traveling Salesman Problem are probably the most famous problems of Combinatorial Optimization. They represent perfect examples of so-called "NP-hard" problems and serve as a test for various computational methods (see, for example, [3,6]). In this paper we prove two new algorithmic results about these problems. We use the model of computation based on RAM (see, for example, [1]). As a measure of complexity we consider the number of performed arithmetic operations (addition, subtraction, multiplication, division, and comparison of real (rational) numbers). Sometimes it is convenient to add another operation, namely, taking a root of a non-negative number.

* Supported by the United States Army Research Office through the Army Center of Excellence for Symbolic Methods in Algorithmic Mathematics (ACSyAM), Mathematical Sciences Institute of Cornell University, Contract DAAL03-91-C0027 and Center for Discrete Mathematics and Theoretical Computer Science (DIMACS)

We begin with formulations of these problems. Let $\Gamma = (V, E)$ be a directed graph with vertex set V and set of edges E. Suppose that the graph Γ has n vertices, that is $|V| = n$. An ordering $P_1, \ldots, P_n$ of the vertices of Γ is called a *Hamiltonian circuit* if and only if the vertices P_i, P_{i+1} for $i = 1, \ldots, n-1$ and the vertices P_n, P_1 are adjacent in Γ.

Hamiltonian Circuit Problem. Given a directed graph Γ decide whether Γ has a Hamiltonian circuit and if so, construct such a circuit. We assume that Γ is given by its adjacency matrix.

The Traveling Salesman Problem is a weighted version of the above problem. We assume that the edges of Γ have certain weights $\{w(e) : \ e \in E\}$ and we are looking for a Hamiltonian circuit of the maximal (or minimal) weight $w(e_1) + \ldots + w(e_n)$, where $e_1, \ldots, e_n$ are the edges of the circuit.

Our first result deals with the worst-case complexity of the Hamiltonian Circuit Problem. In fact, we solve a more difficult problem, namely, how to count the number of Hamiltonian circuits in a graph.

(1.1) Theorem. *There exists an algorithm which for any given $n \in \mathbb{N}$ and for any given directed graph Γ with n vertices computes the number of Hamiltonian circuits in Γ with $O(n^4 \cdot 2^n)$ time complexity and $O(n^2)$ space complexity. If Γ does contain a Hamiltonian circuit then the algorithm constructs such a circuit with $O(n^6 \cdot 2^n)$ time complexity and $O(n^2)$ space complexity.*

To the best knowledge of the author, the algorithm of Theorem 1.1 is the fastest known PSPACE algorithm for the Hamiltonian Circuit Problem. The exhaustive search can be accomplished in a polynomially bounded space, however it would give us $O(n!)$ time complexity. The method of dynamic programming allows one to test a graph with the time complexity $O(n^2 \cdot 2^n)$, however it uses $O(2^n)$ memory (since the basic idea is a reduction to subgraphs, see, for example, [6]). The algorithm from the paper [5] (see the "HPA3" algorithm from [5]) uses $O(n)$ memory but has $2^{2n} n^{O(1)}$ time complexity.

The second result deals with a special version of the Traveling Salesman Problem. Assume that we have a complete graph Γ whose vertices $P_1, \ldots, P_n$ are certain points in a Euclidean space $\mathbb{R}^d$. Let us choose an arbitrary norm $\| \cdot \|$ in $\mathbb{R}^d$ and define the weight $w(e)$ of an edge $e = (P_i, P_j)$ to be the distance $\|P_i - P_j\|$ between the vertices in this norm. We are looking for a *longest* Hamiltonian circuit in Γ.

(1.2) Theorem. *Let us fix a number $d \in \mathbb{N}$, a norm $\| \cdot \|$ in the d-dimensional Euclidean space $\mathbb{R}^d$ and a number $\delta > 0$. Then there exists a polynomial time algorithm which for any given $n \in \mathbb{N}$ and for any n points $P_1, \ldots, P_n$ in $\mathbb{R}^d$, given by their coordinates, computes a Hamiltonian circuit with vertices $P_1, \ldots, P_n$ whose length approximates, with relative error less than δ, the largest length of a Hamiltonian circuit with these vertices.*

An algorithm from [2] constructs a Hamiltonian cycle in a complete weighted graph with nonegative weights whose weight is at least $2/3$ of the maximum weight of a Hamiltonian circuit. To the best knowledge of the author this was the best approximation achieved so far. Considerable attention has been paid to approximating the *smallest* length of a Hamiltonian circuit in Euclidean space where factor 3/2 has been achieved (see [6]). Theorem 1.2 clarifies the status of the Longest Euclidean Hamiltonian Circuit Problem in the computational complexity hierarchy. It was proven in [8] that the problem of computing the *smallest* weight of a Hamiltonian circuit in a complete graph whose edges have weights 1 or 2 is MAX SNP-hard, that is, if for any fixed $\delta > 0$ there exists a polynomial time approximation algorithm which achieves the relative error δ then such an algorithm exists for a large class of difficult combinatorial problems. It follows immediately that the problem of computing the *largest* weight of a Hamiltonian circuit in such a graph is also MAX SNP-hard. The last observation contrasts with Theorem 1.2 and shows, in particular, that the weights arising from a norm in a vector space are "better" than arbitrary weights satisfying the triangle inequality. In contrast, the status of the minimal weight Traveling Salesman Problem in the Euclidean space (and even in the Euclidean plane) is still unclear: it is not known to be MAX SNP-hard, and no polynomial approximation scheme is known.

Our approach is algebraic. We introduce a certain inner product in the ring of polynomials and then reduce our problem to the problem of computation of the inner product of two given polynomials. All the necessary facts about this inner product are presented in Section 2. As a by-product we prove, in particular, that the permanent of a matrix can be computed in polynomial time provided the rank of the matrix is fixed (Section 3). In Section 4 we introduce the Hamiltonian permanent of a matrix, whose difference from the usual permanent is that we take the sum over all Hamiltonian permutations. Similarly, we represent the Hamiltonian permanent as the inner product of two polynomials and prove that the Hamiltonian permanent can be computed in polynomial time provided the rank of the matrix is fixed. In Section 5 we prove Theorem 1.1. In Section 6 we introduce the notion of *combinatorial rank* of a matrix and prove that if the $n \times n$ matrix of weights has a fixed combinatorial rank then we can compute the largest weight of a Hamiltonian circuit with a relative error ϵ in time which is polynomial in ϵ^{-1} and n. In Section 7 we show how to construct a particular Hamiltonian circuit whose weight approximates the largest weight of a Hamiltonian circuit. In Section 8 we show that the matrix of pairwise distances between n points in any *polyhedral* norm in a Euclidean space has a fixed combinatorial rank independent of n. Finally, we prove Theorem 1.2.

(1.3) Notation. We let S_n denote the symmetric group, that is the group of all permutations of the set $\{1, \ldots, n\}$. By H_n we mean the subset of S_n containing all the permutations which consist of a single cycle. Thus $|H_n| = (n-1)!$ and for a complete weighted graph Γ with n vertices and the matrix $W = (w_{ij}) : \ 1 \leq i, j \leq n$ of weights, the largest weight of a Hamiltonian circuit can be written as

$$c(W) = \max\Big\{\sum_{i=1}^{n} w_{i,g(i)} : \ g \in H_n\Big\}.$$

2. Preliminaries. An Inner Product in the Ring of Polynomials

Let $\mathcal{P}_n = \mathbb{R}[x_1, \ldots, x_n]$ be the ring of polynomials in variables $x_1, \ldots, x_n$ over the reals. For a polynomial $P \in \mathcal{P}_n$ we interpret its argument $x = (x_1, \ldots, x_n)$ as a vector from the Euclidean space $\mathbb{R}^n$. We fix the standard inner product $\langle \cdot, \cdot \rangle$ in $\mathbb{R}^n$, so $\langle x, y \rangle = x_1 \cdot y_1 + \ldots + x_n \cdot y_n$ for vectors $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ from $\mathbb{R}^n$. For an index $\alpha = (\alpha_1, \ldots, \alpha_n)$ where $\alpha_i : \ i = 1, \ldots, n$ are non-negative integers by x^α we denote the monomial $x^\alpha = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$. We consider the following inner product (which we denote also by $\big\langle \cdot, \cdot \big\rangle$) defined on the real vector space $\mathcal{P}_n$:

$$\text{for} \qquad P = \sum_\alpha p_\alpha \cdot x^\alpha \quad \text{and} \quad Q = \sum_\alpha q_\alpha \cdot x^\alpha$$

we put

$$\big\langle P, Q \big\rangle = \sum_\alpha \alpha_1! \cdots \alpha_n! \cdot p_\alpha \cdot q_\alpha. \tag{2.1}$$

It is immediate that $\langle \cdot, \cdot \rangle$ is indeed an inner product. For a polynomial $P = \sum_\alpha p_\alpha x^\alpha$ let us introduce a differential operator

$$P\Big(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n}\Big) = \sum_\alpha p_\alpha \frac{\partial^{\alpha_1 + \ldots + \alpha_n}}{\partial x_1^{\alpha_1} \ldots \partial x_n^{\alpha_n}}.$$

Then formula (2.1) can be formally written as

$$\big\langle P, Q \big\rangle = P\Big(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n}\Big) Q(x_1, \ldots, x_n)\Big|_{x_1 = \ldots = x_n = 0}. \tag{2.2}$$

We also give the third equivalent definition of the inner product in $\mathcal{P}_n$. We observe that

$$Q(x) = Q\Big(\frac{\partial}{\partial y_1}, \ldots, \frac{\partial}{\partial y_n}\Big) \exp\{\langle x, y \rangle\}\Big|_{y=0}.$$

Therefore by (2.2) we get

$$\big\langle P, Q \big\rangle = P\Big(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n}\Big) Q\Big(\frac{\partial}{\partial y_1}, \ldots, \frac{\partial}{\partial y_n}\Big) \exp\{\langle x, y \rangle\}\Big|_{x=y=0}. \tag{2.3}$$

An important feature of the introduced inner product is that it "behaves nicely " with respect to a linear transformation of the variables $x_1, \ldots, x_n$. More precisely, let G be a linear operator on the Euclidean space $\mathbb{R}^n$. By G^* we denote the operator on $\mathbb{R}^n$ which is conjugate to G, so $\langle x, Gy \rangle = \langle G^* x, y \rangle$ for all $x, y \in \mathbb{R}^n$. For a polynomial $P \in \mathcal{P}_n$ let us define a polynomial $G(P) \in \mathcal{P}_n$ by the formula

$$G(P)(x) = P\big(G^*(x)\big).$$

We observe that

$$G(P+Q) = G(P) + G(Q), \quad G(P \cdot Q) = G(P) \cdot G(Q) \quad \text{and} \quad (GH)(P) = G(H(P))$$

for all polynomials $P, Q \in \mathcal{P}_n$ and linear operators G, H on $\mathbb{R}^n$. The following result is well-known, but for the sake of completeness we give its short proof here.

(2.4) Theorem. *We have*

$$\big\langle G(P), Q\big\rangle = \big\langle P, G^*(Q)\big\rangle$$

for all polynomials $P, Q \in \mathcal{P}_n$ and all linear operators G on $\mathbb{R}^n$.

Proof. We use (2.3), the identity $\langle G(x), y\rangle = \langle x, G^*(y)\rangle$ for all $x, y \in \mathbb{R}^n$, and the observation that the differential operators $P\Big(\frac{\partial}{\partial x_1}, \dots, \frac{\partial}{\partial x_n}\Big)$ and $Q\Big(\frac{\partial}{\partial y_1}, \dots, \frac{\partial}{\partial y_n}\Big)$ commute.

Thus we have

$$\big\langle P, G^*(Q)\big\rangle = P\Big(\frac{\partial}{\partial x_1}, \dots, \frac{\partial}{\partial x_n}\Big) Q\Big(\frac{\partial}{\partial y_1}, \dots, \frac{\partial}{\partial y_n}\Big) \exp\{\langle G(x), y\rangle\}\Big|_{x=y=0} =$$

$$= Q\Big(\frac{\partial}{\partial y_1}, \dots, \frac{\partial}{\partial y_n}\Big) P\Big(\frac{\partial}{\partial x_1}, \dots, \frac{\partial}{\partial x_n}\Big) \exp\{\langle x, G^*(y)\rangle\}\Big|_{x=y=0} =$$

$$= \big\langle Q, G(P)\big\rangle = \big\langle G(P), Q\big\rangle.$$

□

In particular, the inner product $\big\langle \cdot, \cdot \big\rangle$ in the ring of polynomials $\mathcal{P}_n$ is invariant under orthogonal transformations, that is $\big\langle G(P), G(Q)\big\rangle = \big\langle P, Q\big\rangle$ if G is an orthogonal transformation of $\mathbb{R}^n$.

We will see that the problems of Combinatorial Optimization that we are interested in can be reduced to the computation of the inner product of the two given polynomials. If the number n of variables is small we can use the definition (2.1) directly to efficiently compute this inner product. However, even if n is large, it might happen that by a suitable linear transformation we can significantly reduce the number of variables. Then to compute the desired inner product we use Theorem 2.4 and (2.1). As an illustration of this approach, in the next section we show that the permanent of a matrix can be computed in polynomial time provided the rank of the matrix is fixed. Finally we present a useful identity for the inner product of polynomials of a particular type.

(2.5) Lemma. *Let $P \in \mathcal{P}_n$ be a homogeneous polynomial of degree n and $Q(x) = x_1 \cdots x_n$. For a (possibly empty) subset $\omega \subset \{1, \dots, n\}$ let us denote by $x_\omega \in \mathbb{R}^n$ the vector whose i-th coordinate is equal to 1 if $i \in \omega$ and 0 otherwise. Then*

$$\langle P, Q\rangle = (-1)^n \cdot \sum_\omega (-1)^{|\omega|} P(x_\omega),$$

where the sum is taken over all subsets of $\{1, \dots, n\}$ and $|\omega|$ denotes the cardinality of ω.

Proof. This identity holds since for $P = Q$ the sum is obviously equal to 1. If P is a monomial whose support does not contain certain $i \in \{1, \dots, n\}$, then the summands corresponding to $\omega - \{i\}$ and $\omega \cup \{i\}$ annihilate each other. □

3. Example. Computing the Permanent

In this section we give an example which illustrates how our technique can be applied to combinatorial computations.

(3.1) Definition. Let $A = (a_{ij})$ be a real $n \times n$ square matrix. The expression

$$\text{per } A = \sum_{g \in S_n} \prod_{i=1}^{n} a_{i,g(i)}$$

is called the *permanent* of A (see [7]).

It is known that the problem of computing the permanent of a given matrix is $\#P$-hard. We show, however, that if we fix the rank of the matrix then its permanent can be computed in strongly polynomial time. Our method can be interpreted as a "Gaussian elimination" for the permanent.

(3.2) The algorithm.

Input: an $n \times n$ matrix A such that rank $A \le r$, where $r \in \mathbb{N}$ is a fixed number;

Output: the number per A.

Algorithm:

1. Compute an $n \times n$ matrix $B = (b_{ij})$ such that $b_{ij} = 0$ unless $i \le r$ and an $n \times n$ matrix $G = (g_{ij})$ such that $GB = A$ (such matrices exist since rank $A \le r$);

2. Define two polynomials L and R in r variables $x_1, \ldots, x_r$ of degree n by the formulas

$$L(x_1, \ldots, x_r) = \prod_{j=1}^{n} \sum_{i=1}^{r} b_{ij} \cdot x_i \ , \ R(x_1, \ldots, x_r) = \prod_{i=1}^{n} \sum_{j=1}^{r} g_{ij} \cdot x_j$$

and then expand them into the sum of $\binom{n+r-1}{r-1}$ monomials

$$L = \sum_{\alpha} \lambda_\alpha \cdot x^\alpha, \ R = \sum_{\alpha} \rho_\alpha \cdot x^\alpha$$

consecutively removing the parentheses;

3. Compute the desired value of the permanent by the formula

$$\text{per } A = \sum_{\alpha = (\alpha_1, \ldots, \alpha_r)} \alpha_1! \cdots \alpha_r! \cdot \lambda_\alpha \cdot \rho_\alpha .$$

(3.3) Theorem. *Let us fix $r \in \mathbb{N}$. Then for any given $n \in \mathbb{N}$ and for any given $n \times n$ matrix A such that rank $A \le r$ the above algorithm computes the permanent of A using a number of arithmetic operations which is bounded by a polynomial in n (the degree of* *this polynomial is linear in r). If A is a rational matrix, then the size of all the numbers involved in the algorithm is bounded by a polynomial in the input size.*

Proof. First, we estimate the complexity of the algorithm. To compute the matrices G and B one needs to perform $O(n^3)$ arithmetic operations (as to solve the problem of linear algebra). Since the number r of variables is fixed, the expansions of L and R contain $O(n^{r-1})$ terms and we can compute these expansions (consecutively removing the parenthesis) by using $O(n^{r-1})$ arithmetic operations. Similarly, since the expansions of L and R contain $O(n^{r-1})$ monomials, the third step of the algorithm requires a polynomial in n number of arithmetic operations provided r is fixed.

Now we are going to prove that the above algorithm indeed computes the permanent of A. Let us define polynomials $P, Q \in \mathcal{P}_n$ as follows

$$P(x) = \prod_{j=1}^{n} \sum_{i=1}^{n} a_{ij} \cdot x_i, \quad Q(x) = x_1 \cdots x_n$$

for $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$. Then by (3.1) and (2.1)

$$\text{per } A = \langle P, Q \rangle.$$

Now we observe that the $G(L) = P$ in the ring of polynomials $\mathcal{P}_n$. Applying Theorem 2.4 we get

$$\text{per } A = \langle G(L), Q \rangle = \langle L, G^*(Q) \rangle.$$

Let us denote $\hat{R} = G^*(Q)$. Then $\hat{R}$ is the product of linear forms

$$\hat{R}(x) = \prod_{i=1}^{n} \sum_{j=1}^{n} g_{ij} \cdot x_j.$$

Since L is a polynomial in the first r variables $x_1, \ldots, x_r$ only we get

$$\text{per } A = \langle L, \hat{R} \rangle = \langle L, R \rangle.$$

Now by the definition (2.1) we conclude that the algorithm indeed computes the value of the permanent of A. □

4. Hamiltonian Permanent

In this section we consider an analogue of permanent which we call "Hamiltonian permanent". We recall that by H_n we denote the set of all Hamiltonian cycles (i.e., permutations which consist of a single cycle) in the symmetric group S_n. For an $n \times n$ matrix A we define its Hamiltonian permanent by the formula

$$\text{hamp } A = \sum_{g \in H_n} \prod_{i=1}^{n} a_{i,g(i)}. \tag{4.1}$$

If A is the adjacency matrix of a directed graph then hamp A is the number of Hamiltonian circuits in the graph. Therefore to compute the Hamiltonian permanent of a given matrix is a $\#P$-hard problem. However, we show that one can compute the Hamiltonian permanent in strongly polynomial time if the rank of the matrix is fixed. First, we represent the Hamiltonian permanent as a certain inner product in the ring of polynomials $\mathcal{P}_n$.

(4.2) Lemma. *For $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ let us define a diagonal $n \times n$ matrix $D(x) = \text{diag}\,\{x_1, \ldots, x_n\}$. For an $n \times n$ matrix A let us put*

$$P(x_1, \ldots, x_n) = \mathrm{Tr}\,\big(D(x)A\big)^n,$$

where Tr *denotes the usual trace of an $n \times n$ square matrix. So $P \in \mathcal{P}_n$ is a homogeneous polynomial of degree n in n variables $x_1, \ldots, x_n$. Let us denote $Q(x) = x_1 \cdots x_n \in \mathcal{P}_n$. Then*

$$\text{hamp } A = \frac{1}{n} \cdot \big\langle P, Q \big\rangle.$$

Proof. Let us denote by $T_s(x)$ the s-th diagonal element of the matrix $T(x) = \big(D(x)A\big)^n$. Then

$$T_s(x) = \sum_I \prod_{j=1}^{n} x_{i_j} \cdot a_{i_j, i_{j+1}}, \tag{4.2.1}$$

where the sum is taken over all sequences $I = (1 \leq i_1, \ldots, i_n \leq n)$, where $i_1 = s$ and we put $i_{n+1} = s$. Let us denote by $t_{s,I} \in \mathcal{P}_n$ the I-th summand of (4.2.1). It is clear now that

$$\big\langle t_{s,I} \quad , \quad Q \big\rangle = \begin{cases} \prod_{j=1}^{n} a_{i_j, i_{j+1}}, & \text{if all the numbers } i_1, \ldots, i_n \text{ are pairwise distinct;} \\ 0, & \text{otherwise.} \end{cases}$$

Hence there is a bijection between the set of all strings $I = (i_1, \ldots, i_n)$ such that $\big\langle t_{s,I}, Q \big\rangle \neq 0$ and the set of all Hamiltonian permutations $g \in S_n$ which maps a sequence I to the Hamiltonian permutation g such that $g(i_j) = i_{j+1}$ where $i_1 = i_{n+1} = s$. Therefore, for all $s = 1, \ldots, n$ we have that

$$\big\langle T_s, Q \big\rangle = \text{hamp } A.$$

Since $P = T_1 + \ldots + T_n$ the proof follows. □

Now we describe an algorithm which computes the Hamiltonian permanent of a given matrix.

(4.3) Computing the Hamiltonian permanent of a generic matrix.

Input: an $n \times n$ matrix A;

Output: the number hamp A.

Algorithm:

1. Put hamp $A := 0$;

2. Consecutively generate all the subsets $\omega \subset \{1, \ldots, n\}$, for every subset ω define the $n \times n$ matrix $A(\omega) = \big(a_{ij}(\omega)\big)$ by the formula:

$$a_{ij}(\omega) = \begin{cases} a_{ij}, & \text{if } i \in \omega; \\ 0, & \text{otherwise,} \end{cases}$$

raise the matrix $A(\omega)$ to the n-th power $A(\omega)^n$ and put

$$\text{hamp } A := \text{hamp } A + (-1)^{|\omega|} \cdot \text{Tr } A(\omega)^n;$$

3. Finally put hamp $A := \dfrac{(-1)^n}{n} \cdot$hamp A.

(4.4) Theorem. *For any given $n \in \mathbb{N}$ and for any given $n \times n$ matrix A the above algorithm computes the Hamiltonian permanent of A using $O(n^4 \cdot 2^n)$ arithmetic operations and $O(n^2)$ space. If A is a rational matrix, then the size of all the numbers involved in the algorithm is bounded by a polynomial in the input size.*

Proof. First we estimate the complexity of the algorithm. We need to enumerate 2^n subsets ω. For any $\omega \subset \{1, \ldots, n\}$ the power $A(\omega)^n$ can be computed using $O(n^4)$ arithmetic operations (in fact, we can achieve $O(n^3 \cdot \log n)$ by repeated squaring and using fast matrix multiplication we can proceed even faster).

Now we have to prove that the algorithm indeed computes the Hamiltonian permanent of the matrix A. This immediately follows by Lemma 4.2 and Lemma 2.5. □

Let us show that the Hamiltonian permanent of a matrix can be computed in polynomial time provided the rank of the matrix is fixed.

(4.5) Computing the Hamiltonian permanent of a matrix when the rank is fixed.

Input: an $n \times n$ matrix A such that rank $A \leq r$.

Output: the number hamp A.

Algorithm:

Without loss of generality we assume that $n \geq r^2$ (otherwise we compute hamp A using the definition (4.1) directly).

1. Compute an $n \times n$ non-degenerate matrix U such that the last $n - r$ columns of the matrix AU are zeros. Compute the inverse matrix U^{-1};

2. For the variables $x_1, \ldots, x_n$ define the $n \times n$ diagonal matrix $D(x) =$ diag $\{x_1, \ldots, x_n\}$. Then compute the $n \times n$ matrix $S(x)$ whose entries are linear forms in $x = (x_1, \ldots, x_n)$ by the formula

$$S(x) = U^{-1} D(x) A U.$$

Thus the (i, j)-th entry of $S(x)$ is written as $\langle s_{ij}, x \rangle$ where $s_{ij} \in \mathbb{R}^n$ is a vector and we compute these vectors s_{ij};

3. Put $m = r^2$ and construct a bijection ϕ which maps the set of pairs $\{(i,j):\ 1 \leq i,j \leq r\}$ onto the set $\{1,\ldots,m\}$ and the inverse bijection ψ;

4. Define an $r \times r$ matrix $F(x)$ whose (i,j)-th entry is the variable $x_{\phi(i,j)}$. Raise the matrix $F(x)$ to the n-th power and define a polynomial L of degree n in $m = r^2$ variables $x_1,\ldots,x_m$ by the formula $L(x) = \text{Tr } F(x)^n$. Compute the expansion of $L(x)$ into the sum of monomials

$$L(x) = \sum_{\alpha} \lambda_\alpha \cdot x^\alpha$$

in the variables $x_1,\ldots,x_m$;

5. Construct an $n \times n$ matrix $G = (g_{ij}):\ 1 \leq i,j \leq n$ in the following way: for $j = 1,\ldots,m$ the j-th column of G is the vector $s_{\psi(j)}$ whereas $g_{ij} = 0$ for $j > m$. Define the polynomial R of degree n in $m = r^2$ variables $x_1,\ldots,x_m$ by the formula $R(x) = \prod_{i=1}^{n}\sum_{j=1}^{m} g_{ij} \cdot x_j$ and compute its expansion

$$R(x) = \sum_{\alpha} \rho_\alpha \cdot x^\alpha$$

into the sum of monomials in $x_1,\ldots,x_m$;

6. Compute the desired value of the Hamiltonian permanent by the formula:

$$\text{hamp } A = \frac{1}{n} \cdot \sum_{\alpha=(\alpha_1,\ldots,\alpha_m)} \alpha_1! \cdots \alpha_m! \cdot \lambda_\alpha \cdot \rho_\alpha.$$

(4.6) Theorem. *Let us fix $r \in \mathbb{N}$. Then for any given $n \in \mathbb{N}$ and for any given $n \times n$ matrix A such that* rank $A \leq r$ *the above algorithm computes the Hamiltonian permanent of A using a number of arithmetic operations which is bounded by a polynomial in n (the degree of this polynomial is linear in r^2). If A is a rational matrix then the size of all the numbers involved in the algorithm is bounded by a polynomial in the input size.*

Proof. First, we estimate the complexity of the algorithm. Steps 1 and 2 as problems of linear algebra require $O(n^3)$ arithmetic operations. Since r is fixed Step 3 requires a constant number of operations. To perform Step 4 we can consecutively raise the matrix $F(x)$ to the n-th power and then sum up its diagonal polynomials. Since the number r is fixed, the expansion of L contains $O(n^{r^2})$ terms and it can be computed using $O(n^{r^2})$ operations. Similarly, the expansion of R (Step 5) contains $O(n^{r^2})$ terms and it can be computed in $O(n^{r^2})$ time which is polynomially bounded in n when r is fixed. Finally, the Step 6 can be made in $O(n^{r^2})$ time.

Now we have to prove that the algorithm indeed computes the Hamiltonian permanent. Since

$$\text{Tr } \big(D(x)A\big)^n = \text{Tr } \big(U^{-1}D(x)AU\big)^n$$

by Lemma 4.2 we get

$$\text{hamp } A = \frac{1}{n} \cdot \Big\langle \text{Tr } S(x)^n \quad , Q \quad \Big\rangle,$$

where $Q = x_1 \cdots x_n$. Now we observe that the last $n - r$ columns of $S(x) = U^{-1}D(x)AU$ are zero vectors. Therefore Tr $S(x)^n$ = Tr $C(x)^n$, where $C(x)$ is the left upper corner $r \times r$ submatrix of $S(x)$. The operator G maps the polynomial $L(x)$ =Tr $F(x)^n$ to the polynomial Tr $C(x)^n$ in the ring of polynomials $\mathcal{P}_n$. Therefore, by Theorem 2.4 we have

$$\text{hamp } A = \frac{1}{n} \cdot \Big\langle \text{Tr } C(x)^n \ , \ Q \Big\rangle = \frac{1}{n} \cdot \big\langle L \ , \ G^*(Q) \big\rangle.$$

From now on we follow the proof of Theorem 3.3. Let us denote $\hat{R} = G^*(Q) \in \mathcal{P}_n$. Then $\hat{R}$ is the product of linear forms

$$\hat{R}(x) = \prod_{i=1}^{n} \sum_{j=1}^{n} g_{ij} \cdot x_j.$$

Since L is a polynomial in the first $m = r^2$ variables $x_1, \ldots, x_m$ only we deduce that

$$\text{hamp } A = \frac{1}{n} \cdot \big\langle L, \hat{R} \big\rangle = \frac{1}{n} \cdot \big\langle L, R \big\rangle.$$

Formula (2.1) implies that the algorithm indeed computes the value of the Hamiltonian permanent of A. □

5. Counting Hamiltonian Circuits

In this section we prove Theorem 1.1. Without loss of generality we assume that the given graph does not contain loops or multiple directed edges. Therefore, we assume that a graph Γ with n vertices $\{1, \ldots, n\}$ is given by its adjacency matrix A:

$$A_{ij} = \begin{cases} 1 & \text{if } \overrightarrow{ij} \text{ is a directed edge in } \Gamma, \\ 0 & \text{otherwise.} \end{cases}$$

(5.1) Proof of Theorem 1.1. First we show how to compute the number of Hamiltonian circuits in the graph Γ. If A is the adjacency matrix of Γ then hamp A is equal to the number of Hamiltonian circuits in Γ. To compute the Hamiltonian permanent we use Algorithm 4.3.

Now we show how to construct a Hamiltonian circuit in Γ provided such a circuit exists. We use the "divide and conquer" method. First, we test whether the graph Γ has a Hamiltonian circuit. Then we consecutively try each edge e of Γ. We check whether the graph obtained from Γ by deleting the edge e contains a Hamiltonian circuit. If it does, we modify Γ by removing e. Otherwise we conclude that a Hamiltonian circuit we are looking for must contain e. In the both cases we proceed trying the next edge. Since the number of edges does not exceed n^2 we get the desired complexity of the algorithm. □

6. Combinatorial Rank

Now we turn to the Traveling Salesman Problem. We consider a complete directed weighted graph Γ with n vertices and an $n \times n$ matrix W of non-negative weights w_{ij}. We want to compute the largest weight of a Hamiltonian circuit in Γ, that is the number

$$c(W) = \max\Bigl\{\sum_{i=1}^{n} w_{i,g(i)}\ :\ g \in H_n\Bigr\}$$

(see (1.3)). In particular, we are looking for a condition on the matrix of weights W which can ensure polynomial solvability of the corresponding problem. Algorithm 4.5 computes the Hamiltonian permanent of a matrix in polynomial time provided the matrix has a fixed rank. Let us try to find an appropriate version of the "fixed rank" for the Traveling Salesman Problem. We observe that if we formally replace "$\sum$" by "max" and "$\prod$" by "$\sum$" in the definition (4.1) of the Hamiltonian permanent then we come to the above formula for the largest weight of a Hamiltonian circuit. If we replace "$\sum$" by "max" and "$\cdot$" by "$+$" in a definition of the rank, we come to the following notion.

(6.1) Definitions. Let $W = (w_{ij})$ be an $n \times n$ real matrix. We say that the *combinatorial rank* of W does not exceed r if there exist r real n-vectors $h^s = (h_1^s, \ldots, h_n^s)$ and r real n-vectors $v^s = (v_1^s, \ldots, v_n^s):\ s = 1, \ldots, r$ such that

$$w_{ij} = \max\{h_i^s + v_j^s,\ s = 1, \ldots, r\} \tag{6.1.1}$$

for all $1 \le i, j \le n$. In this case we write

$$\operatorname{comr} W \le r.$$

We say that the representation (6.1.1) is *well-centered* if

$$\max\{h_i^s + v_j^s,\ s = 1, \ldots, r\} = \max\{|h_i^s + v_j^s|,\ s = 1, \ldots, r\}$$

for all $1 \le i, j \le n$.

Like the usual rank, the combinatorial rank of an $n \times n$ real matrix does not exceed n. However, it seems to be a difficult problem to compute the smallest r which bounds the combinatorial rank of a matrix. Therefore, in what follows we assume that the matrix W of weights is already presented in the form (6.1.1). In Section 8 we describe some natural examples of matrices having small combinatorial rank for which the well-centered representation (6.1.1) can be easily constructed. The property of "well-centeredness" is essentially used in the approximation algorithm that we present later in this section. Although it is always possible to make the representation (6.1.1) well-centered by adding a suitable large constant to every weight w_{ij}, one may loose the approximability this way.

Our main goal is to prove that for any $r \in \mathbb{N}$ there exists a polynomial time algorithm which for any $n \times n$ matrix $W = (w_{ij})$ of weights such that $\operatorname{comr} W \leq r$ and the matrix is given by its well-centered representation and for any given $\epsilon > 0$ computes a Hamiltonian circuit whose weight approximates, with relative error less than ϵ, the maximal weight (1.3) of a Hamiltonian circuit in time which is polynomial in n and ϵ^{-1}. First we compute (approximately) the maximum weight $c(W)$ (this section) and then compute a suitable circuit (next section).

(6.2) Lemma. *Suppose that $W = (w_{ij})$ is a matrix of non-negative weights given by its well-centered representation (6.1.1). For a real parameter t we define an $n \times n$ matrix $A(t) = (a_{ij}(t))$ by the formulas:*

$$a_{ij}(t) = \sum_{s=1}^{r} \exp\{t \cdot h_i^s\} \cdot \exp\{t \cdot v_j^s\}$$

for $1 \leq i, j \leq n$. For a given $\epsilon > 0$ choose a positive even integer m such that

$$m > \frac{\log(n-1)! + n \cdot \log r}{\log(1+\epsilon)},$$

and put

$$c_m = \left(\frac{1}{r^n (n-1)!} \frac{d^m}{dt^m} \operatorname{hamp} A(t) \Big|_{t=0} \right)^{\frac{1}{m}}.$$

Then c_m approximates the largest weight $c(W)$ of a Hamiltonian circuit with relative error less than ϵ, that is

$$(1+\epsilon) \cdot c_m \geq c(W) \geq c_m.$$

Proof. Let $\{1, \ldots, r\}^n$ be the set of all n-tuples $\mathbf{s} = (s_1, \ldots, s_n)$ of numbers $s_i \in \{1, \ldots, r\}$. For a Hamiltonian permutation $g \in H_n$ and for an n-tuple $\mathbf{s} \in \{1, \ldots, r\}^n$ let us denote by $\sigma(g, \mathbf{s})$ the sum

$$\sigma(g, \mathbf{s}) = \sum_{i=1}^{n} \big(h_i^{s_i} + v_{g(i)}^{s_i} \big),$$

where $h^1, \ldots, h^r$ and $v^1, \ldots, v^r$ are the vectors from Definition 6.1. Then the Hamiltonian permanent of the matrix $A(t)$ can be written as

$$\operatorname{hamp} A(t) = \sum_{g \in H_n} \sum_{\mathbf{s} \in \{1, \ldots, r\}^n} \exp\big\{ t \cdot \sigma(g, \mathbf{s}) \big\}.$$

Therefore

$$(c_m)^m = \frac{1}{r^n (n-1)!} \sum_{g \in H_n} \sum_{\mathbf{s} \in \{1, \ldots, r\}^n} \big(\sigma(g, \mathbf{s}) \big)^m. \tag{6.2.1}$$

Now we use the following inequality which holds for any set of real numbers $\sigma_1, \dots, \sigma_N$ and any positive even integer m:

$$\Big(\sum_{j=1}^{N} \sigma_j^m\Big)^{1/m} \geq \max\{|\sigma_j| : \ j = 1, \dots, N\} \geq \Big(\frac{1}{N}\sum_{j=1}^{N} \sigma_j^m\Big)^{1/m}.$$

Since the two summations in (6.2.1) contain $N = r^n(n-1)!$ summands and since our choice of m implies $N^{1/m} \leq 1+\epsilon$ we get

$$(1+\epsilon)\cdot c_m \geq \max_{g \in H_n, \mathbf{s} \in \{1,\dots,r\}^n} |\sigma(g,\mathbf{s})| \geq c_m. \tag{6.2.2}$$

For any $g \in H_n$ we have

$$\max_{\mathbf{s}\in\{1,\dots,r\}^n} \sigma(g,\mathbf{s}) \leq \max_{\mathbf{s}\in\{1,\dots,r\}^n} |\sigma(g,\mathbf{s})| \leq \max_{s_1,\dots,s_n \in \{1,\dots,r\}} \sum_{i=1}^{n} |h_i^{s_i} + v_{g(i)}^{s_i}| =$$

$$\sum_{i=1}^{n} \max\{|h_i^s + v_{g(i)}^s| : s = 1, \dots, r\} \qquad \text{(we use well-centeredness)} =$$

$$\sum_{i=1}^{n} \max\{h_i^s + v_{g(i)}^s : s = 1,\dots,r\} = \max_{\mathbf{s}\in\{1,\dots,r\}^n} \sigma(g,\mathbf{s}).$$

Therefore, for every $g \in H_n$ we have

$$\max_{\mathbf{s}\in\{1,\dots,r\}^n} |\sigma(g,\mathbf{s})| = \max_{\mathbf{s}\in\{1,\dots,r\}^n} \sigma(g,\mathbf{s}) = \sum_{i=1}^{n} \max\{h_i^s + v_{g(i)}^s : s = 1,\dots,r\} = \sum_{i=1}^{n} w_{i,g(i)}.$$

Hence from (6.2.2) we conclude

$$(1+\epsilon)\cdot c_m \geq \max_{g\in H_n} \sum_{i=1}^{n} w_{i,g(i)} \geq c_m$$

and the proof follows. □

(6.3) Computing the largest weight of a Hamiltonian circuit.

Input:

1. An $n \times n$ matrix of weights $W = (w_{ij})$ such that the combinatorial rank of W does not exceed r and the matrix W is given by its well-centered representation (6.1.1).

2. An $\epsilon > 0$.

Output: A number c approximating the largest weight $c(W)$ of a Hamiltonian circuit with relative error less than ϵ;

Algorithm: Let us choose a positive even integer m wich satisfies the inequality of Lemma 6.2. Without loss of generality we may assume that $\epsilon < 1/2$ and choose m to be the smallest even integer bigger than $4\epsilon^{-1}n^2$. For a variable t let us define an $n \times n$ matrix $\widetilde{A}(t) = (\widetilde{a}_{ij}(t))$ as follows

$$\widetilde{a}_{ij}(t) = \sum_{s=1}^{r} \Big(1 + t \cdot h_i^s + \ldots + \frac{(t \cdot h_i^s)^m}{m!}\Big) \cdot \Big(1 + t \cdot v_j^s + \ldots + \frac{(t \cdot v_j^s)^m}{m!}\Big)$$

(we have replaced the "exp" in the definition of $A(t)$ by the first m terms of its Taylor expansion).

Since $\widetilde{A}(t)$ is the sum of r matrices, each of those being the product of a row and a column, we have that rank $\widetilde{A}(t) \leq r$ for all t. Furthermore, the Hamiltonian permanent of $\widetilde{A}(t)$ is a polynomial in t of degree at most $2mn$:

$$\operatorname{hamp} \widetilde{A}(t) = \sum_{k=0}^{2mn} \alpha_k \cdot t^k.$$

Using Algorithm 4.5 we compute the values of hamp $\widetilde{A}(t)$ for $t = 0, \ldots, 2mn$ and then compute the coefficent α_m from the arising system of linear equations.

Finally we compute

$$d = \frac{m!}{r^n (n-1)!} \cdot \alpha_m$$

and put

$$c = d^{\frac{1}{m}}. \tag{6.3.1}$$

If we need to perform arithmetic operations over the rationals, we can choose m twice as large and then take the m-th root with error less than $\epsilon/2$.

(6.4) Theorem. *Let us fix $r \in \mathbb{N}$. Then for any $n \in \mathbb{N}$, for any $n \times n$ matrix W of non-negative weights such that the combinatorial rank of W does not exceed r and W is given by its well-centered representation (6.1.1) and for any positive $\epsilon > 0$ the above algorithm computes the largest weight $c(W)$ of a Hamiltonian circuit with relative error less than ϵ. The algorithm uses a number of arithmetic operations which is polynomial in n and ϵ^{-1} and takes a root once. If the input consists of rational numbers then all the computations can be performed over the rationals in time which is polynomial in the input size and ϵ^{-1}.*

Proof. By Theorem 4.6 and our choice of m we conclude that the algorithm computes the values of hamp $\widetilde{A}(t)$ in time which is polynomial in n and ϵ^{-1}. Furthermore, the size of the system of linear equations defining α_m is polynomial in n and ϵ^{-1} and thus the computation of α_m can be done in polynomial time. Finally, the polynomial estimate of the complexity of the algorithm follows by the well-known fact that a root of a rational number with a given error can be computed in polynomial time. We observe that

$$\frac{d^m}{dt^m}\text{hamp } A(t)\Big|_{t=0} = \frac{d^m}{dt^m}\text{hamp } \widetilde{A}(t)\Big|_{t=0}$$

and by Lemma 6.2 we conclude that c approximates the largest weight of a Hamiltonian circuit up to the desired relative error. □

In the next section we will need a version of Algorithm 6.3 which does not use the operation of taking a root from a non-negative number.

(6.5) Version of Algorithm 6.3. We stop before the line (6.3.1) and output a pair of numbers (d, m), such that $d^{\frac{1}{m}} = c$ is the approximate largest weight $c(W)$ of a Hamiltonian circuit and m is a natural number such that $m \leq 4\epsilon^{-1}n^2 + 2$.

7. Constructing a Hamiltonian Circuit

Now we consider how to construct a particular Hamiltonian circuit whose weight approximates the largest weight of a Hamiltonian circuit with relative error less than ϵ. Let Γ be a directed weighted graph with n vertices $\{1, \ldots, n\}$ and the matrix $W = (w_{ij})$ of weights. We will use a version of the "divide and conquer" algorithm, adding one edge to the circuit at every step. Thus at every step the circuit g will be represented by a set of vertex disjoint directed paths $\Pi = \{\pi_1, \ldots, \pi_t\}$ which cover the set $\{1, \ldots, n\}$. So every vertex i belongs to exactly one path π_j from the collection Π. We allow paths containing one vertex and no edges. We start with n paths, $\pi_i = \{i\}$, $i = 1, \ldots, n$ and end up with one Hamiltonian path π_1. At every iteration of the algorithm we merge a pair of these paths into one path.

(7.1) Definitions and notation. Let $\pi = \{i_1 \to i_2 \to \ldots \to i_m\}$ be a directed paths with m distinct vertices $\{i_1, \ldots, i_m\} \subset \{1, \ldots, n\}$. We call vertex i_1 the *tail* of π (denoted tail(π)) and vertex i_m the *head* of π (denoted head(π)). For a path π containing one vertex we let tail(π) =head(π).

For a directed path π we define the *weight* of π (denoted by w_π) as the sum of the edges of the path. In particular, we interpret w_{ij} as the weight of the directed path $\pi = \overrightarrow{ij}$. If π consists of one vertex we let $w_\pi = 0$. Let $\Pi = \{\pi_1, \ldots, \pi_t\}$ be a collection of vertex disjoint directed paths which cover the set $\{1, \ldots, n\}$. We define the *weight* of Π (denoted by w_Π) as the sum of the weights of the paths π_i : $w_\Pi = w_{\pi_1} + \ldots + w_{\pi_t}$.

For a collection $\Pi = \{\pi_1, \ldots, \pi_t\}$ of vertex disjoint paths we denote by $c(W, \Pi)$ the maximal weight of a Hamiltonian circuit which contains all the paths $\pi_1, \ldots, \pi_t$. We recall that by $c(W)$ we denote the largest weight of a Hamiltonian circuit.

The following simple result plays the crucial role in our considerations.

(7.2) Lemma. *Let $\Pi = \{\pi_1, \dots, \pi_t\}$ be a collection of vertex disjoint paths which cover the set $\{1, \dots, n\}$. Let us define a $t \times t$ matrix $\overline{W} = (\overline{w_{ij}})$ by the formula*

$$\overline{w_{ij}} = w_{ab}, \quad \text{where} \quad a = \text{head}(\pi_i) \text{ and } b = \text{tail}(\pi_j).$$

Then

$$c(W, \Pi) = w_\Pi + c(\overline{W}). \tag{7.2.1}$$

If the matrix W is given by its well-centered representation

$$w_{ij} = \max\{h^s_i + v^s_j : \; s = 1, \dots, r\}$$

for some n-dimensional vectors $h^1, \dots, h^r, v^1, \dots, v^r$ then the matrix $\overline{W}$ can be defined by a well-centered representation

$$\overline{w_{ij}} = \max\{\overline{h^s_i} + \overline{h^s_j} : \; s = 1, \dots, r\}, \tag{7.2.2}$$

where the t-dimensional vectors $\overline{h^s}, \overline{v^s} : \; s = 1, \dots, r$ are defined as follows

$$\overline{h^s_i} = h^s_a, \quad \overline{v^s_j} = v^s_b, \qquad \text{where } a = \text{head}(\pi_i) \quad \text{and} \quad b = \text{tail}(\pi_j).$$

Proof. Let us denote by $H_n(\Pi)$ the set of all Hamiltonian circuits which contain every directed paths π_i from the collection Π. We establish a bijection $\phi : H_n(\Pi) \longrightarrow H_t$. Let $g \in H_n(\Pi)$ be a Hamiltonian circuit containing the paths $\pi_1, \dots, \pi_t$. We define the Hamiltonian permutation $\gamma \in H_t$ of the set $\{1, \dots, t\}$ as follows. For $i \in \{1, \dots, t\}$ we let $\gamma(i) = j$, where $\pi_j \in \Pi$ is the unique path such that the edge which joins head(π_i) and tail(π_j) is the edge of the Hamiltonian circuit g. It is immediate that ϕ is indeed a bijection and that

$$\sum_{i=1}^{n} w_{i,g(i)} = w_\Pi + \sum_{i=1}^{t} \overline{w_{i,\gamma(i)}}.$$

Therefore (7.2.1) is proven. The equations (7.2.2) are obvious. $\square$

(7.3) Computing a Hamiltonian circuit of the maximal weight.

Input:

1. An $n \times n$ matrix of weights $W = (w_{ij})$ such that the combinatorial rank of W does not exceed r and the matrix W is given by its well-centered representation (6.1.1).

2. An $\epsilon > 0$.

Output:

A Hamiltonian circuit $g \in H_n$ whose weight $c = \sum_{i=1}^{n} w_{i,g(i)}$ approximates the largest weight $c(W)$ of a Hamiltonian circuit with relative error less than ϵ, that is,

$$(1 + \epsilon)c \geq c(W) \geq c.$$

Algorithm:

1. Without loss of generality we may assume that $n > 2$ and that $\epsilon < 1$. We compute $\epsilon_1 = \epsilon/2n$ and introduce an integer variable t. Let us put $t = n$ and define a collection $\Pi = \{\pi_1, \ldots, \pi_n\}$ of n directed paths where every path $\pi_i = \{i\}$ consists of a single vertex $\{i\}$. After the $(n-t)$-th iteration of the algorithm the Hamiltonian circuit is represented by a collection $\Pi = \{\pi_1, \ldots, \pi_t\}$ of t vertex disjoint paths which cover the set of vertices $\{1, \ldots, n\}$

2. If $t = 1$ we go to Step 3. If $t > 2$ then for every $k = 1, \ldots, t-1$ we do the following procedure:

Define a new path $\widetilde{\pi}_k$ by merging the paths π_k and π_t. To do that join the vertices head(π_t) and tail(π_k) by the edge. Let $\Pi_k = \{\pi_1, \ldots, \pi_{k-1}, \widetilde{\pi}_k, \pi_{k+1}, \pi_{t-1}\}$ be a new collection of paths. Compute the $(t-1) \times (t-1)$ matrix $\overline{W}$ and its well-centered representation for the collection Π_k as in Lemma 7.2 . Apply Version 6.5 of Algorithm 6.3 with the matrix $\overline{W}$ and relative error ϵ_1. Let (d_k, m_k) be the output of (6.5). Compute w_{Π_k} and define

$$c_k = w_{\Pi_k} + (d_k)^{\frac{1}{m_k}}.$$

Now we choose κ such that $c_\kappa = \max\{c_k : \ k = 0, \ldots, t-1\}$. We find κ without computing c_k (so to avoid taking a root) in the following way. For any pair (i,j) we can check whether the system of real inequalities

$$w_{\Pi_i} + y_i \geq w_{\Pi_j} + y_j, \ y_i^{m_i} = d_i, y_j^{m_j} = d_j, y_i \geq 0, y_j \geq 0 \tag{7.3.1}$$

in two real variables y_i, y_j is feasible using an algorithm from the existential theory of the reals (see, for example, [9]). Hence we check the inequality $c_i \geq c_j$ without computing these numbers. Thus we find κ and redefine the collection $\Pi := \Pi_\kappa$. Redefine $t := t-1$ and go to Step 2.

3. The circuit g is the unique Hamiltonian circuit containing the Hamiltonian path π_1.

(7.4) Theorem. *Let us fix $r \in \mathbb{N}$. Then for any $n \in \mathbb{N}$, for any $n \times n$ matrix W of non-negative weights such that the combinatorial rank of W does not exceed r and W is given by its well-centered representation (6.1.1) and for any $\epsilon > 0$ the above algorithm computes a Hamiltonian circuit whose weight approximates, with relative error less than ϵ, the largest weight $c(W)$ of a Hamiltonian circuit. The algorithm uses a number of arithmetic operations which is polynomial in n and ϵ^{-1}. If the input consists of rational numbers then all the computations can be performed over the rationals in time which is polynomial in the input size and ϵ^{-1}.*

Proof. First we prove that the algorithm is correct. We prove by induction on $n-t$ that after the $(n-t)$-th iteration of Step 2 the algorithm produces a collection of t vertex disjoint paths $\Pi = \{\pi_1, \ldots, \pi_t\}$ such that

$$(1+\epsilon_1)^{n-t} \cdot c(W, \Pi) \geq c(W) \geq c(W, \Pi). \tag{7.4.1}$$

This is obviously the case when $t = n$. Suppose that the inequality is satisfied after the $(n-t)$-th iteration. On the $(n-t+1)$-st iteration, the algorithm constructs $t-1$ collections $\Pi_1, \ldots, \Pi_{t-1}$ of paths such that any Hamiltonian circuit g which contains all paths from Π should also contain all paths from Π_k for exactly one $k \in \{1, \ldots, t-1\}$. Therefore

$$c(W, \Pi) = \max\{c(W, \Pi_k) : \; k = 1, \ldots, t-1\}. \tag{7.4.2}$$

By Lemma 7.2 and Theorem 6.4 we conclude that c_k approximates the value of $c(W, \Pi_k)$ with relative error less than ϵ_1, that is, $c(W, \Pi_k) \leq (1+\epsilon_1)c_k$. Therefore, from (7.4.2) we deduce that

$$c(W, \Pi) \leq (1+\epsilon_1)c(W, \Pi_\kappa),$$

and (7.4.1) follows for all t. Now we plug $t = 1$ into (7.4.1) and use the inequality $(1+\epsilon_1)^n \leq 1+\epsilon$ to see that the Hamiltonian path π_1 that we constructed satisfies the inequality

$$(1+\epsilon)c(W, \pi_1) \geq c(W) \geq c(W, \pi_1).$$

In other words, the weight of the unique Hamiltonian circuit which contains the Hamiltonian path π_1 approximates the largest weight of a Hamiltonian circuit with relative error less than ϵ. Hence the algorithm is correct.

The algorithm uses $n-1$ application of Algorithm 6.3 (Version 6.5), which in turn, uses a polynomial in n and ϵ^{-1} number of arithmetic operations (Theorem 6.4). The complexity of feasibility testing for the system (7.3.1) is polynomial in m_i, m_j as follows by the results of [9]. Since the numbers m_i and m_j do not exceed $4\epsilon^{-1}n^2+2$ the complexity of this procedure is polynomial in n and ϵ^{-1}. Therefore Algorithm 7.3 has the desired complexity. □

8. Longest Hamiltonian Circuit in Euclidean Space

In this section we consider certain natural examples of matrices having a small combinatorial rank. These matrices arise as the matrices of pairwise distances between given points in various metrics in a Euclidean space. The first such example (L^∞ norm in $\mathbb{R}^d$) was suggested by S. Onn. Inspired by this example, the author considered L^1 norm in the Euclidean space and then A. Gerards suggested a far reaching generalization.

Let $\|\cdot\|$ be a norm in the Euclidean space $\mathbb{R}^d$. Assume that we have a set $P_1, \ldots, P_n$ of points in $\mathbb{R}^d$ and let us consider the $n \times n$ matrix $W = (w_{ij})$ of pairwise distances between these points, so

$$w_{ij} = \|P_i - P_j\|.$$

Let us consider the unit ball

$$\mathbf{B} = \{x \in \mathbb{R}^d : \|x\| \leq 1\}.$$

If $\mathbf{B}$ is a polytope then the norm $\|\cdot\|$ is called *polyhedral* norm. The following result was suggested by A. Gerards.

(8.1) Lemma. *Let $\|\cdot\|$ be a polyhedral norm in the Euclidean space $\mathbb{R}^d$. Assume that $W = (w_{ij})$ is the matrix of pairwise distances between given points $P_1, \ldots, P_n \in \mathbb{R}^d$ defined by the formula*

$$w_{ij} = \|P_i - P_j\|.$$

Then the combinatorial rank of the matrix W does not exceed the number of facets of the polytope $\mathbf{B}$.

Proof. We denote by $\langle \cdot, \cdot \rangle$ the standard inner product in $\mathbb{R}^d$ and assume that the ball $\mathbf{B}$ is given by the system of linear inequalities:

$$\mathbf{B} = \big\{ x \in \mathbb{R}^d : \langle b^s, x \rangle \leq 1, \ s = 1, \ldots, r \big\},$$

for some vectors $b^1, \ldots, b^r$ in $\mathbb{R}^d$. We can choose the vectors b^s ranging over all normal vectors to the facets of $\mathbf{B}$. Since $\mathbf{B}$ is centrally symmetric, the set of normals $\{b_s : s = 1, \ldots, r\}$ consists of $r/2$ pairs of antipodal vectors $b, -b$. For $s = 1, \ldots, r$ let us define the vectors $h^s = (h_1^s, \ldots, h_n^s)$ by the formula $h_i^s = \langle P_i, b^s \rangle$ and the vectors $v^s = (v_1^s, \ldots, v_n^s)$ by the formula $v_i^s = -\langle P_i, b^s \rangle$.

Then we have

$$\|P_i - P_j\| = \min\big\{ \lambda \geq 0 : P_i - P_j \in \lambda \cdot \mathbf{B} \big\} = \tag{8.1.1}$$

$$\min\big\{ \lambda \geq 0 : \langle P_i - P_j, b_s \rangle \leq \lambda \text{ for } s = 1, \ldots, r \big\} = \max\big\{ h_i^s + v_j^s : s = 1, \ldots, r \big\}.$$

Since $\mathbf{B}$ is centrally symmetric we have that

$$\max\big\{ h_i^s + v_j^s : s = 1, \ldots, r \big\} = \max\big\{ |h_i^s + v_j^s| : s = 1, \ldots, r \big\}.$$

Thus $\operatorname{comr} W \leq r$, and, moreover, the representation (8.1.1) is well-centered. □

Thus, if $\|\cdot\|$ is the L^∞ norm then for any n points $P_1, \ldots, P_n$ in $\mathbb{R}^d$ the combinatorial rank of the matrix $W = (w_{ij})$, where $w_{ij} = \|P_i - P_j\|$, does not exceed $2 \cdot d$ and for the L^1 norm the combinatorial rank of this matrix does not exceed 2^d. The main observation is that we can bound the combinatorial rank of the matrix of pairwise distances independently of the number of points.

(8.2) Computing the largest length of a Hamiltonian circuit in a polyhedral norm.

Let us fix a Euclidean space $\mathbb{R}^d$ and a polyhedral norm $\|\cdot\|$ there. In other words, we fix a convex centrally symmetric polytope $\mathbf{B} \in \mathbb{R}^d$ containing the origin as its interior point. So

$$\mathbf{B} = \big\{ x \in \mathbb{R}^d : \langle b^s, x \rangle \leq 1, s = 1, \ldots, r \big\}$$

for some vectors $b^1, \ldots, b^r$. If the vectors $b^1, \ldots, b^r$ are rational we say that the norm $\|\cdot\|$ is *rational.*

Input:

1. Points $P_1, \dots, P_n \in \mathbb{R}^d$ given by their coordinates;

2. A number $\epsilon > 0$;

Output: A Hamiltonian circuit whose length approximates, with relative error less than ϵ, the largest length (in the norm $\| \cdot \|$) of a Hamiltonian circuit with the vertices $P_1, \dots, P_n$.

Algorithm:

1. Compute r vectors $h^s = (h_1^s, \dots, h_n^s)$ and r vectors $v^s = (v_1^s, \dots, v_n^s)$ by the formulas: $h_i^s = \langle P_i, b^s \rangle$ and $v_i^s = -\langle P_i, b^s \rangle$ for for $s = 1, \dots, r$ and $i = 1, \dots, n$.

2. Define an $n \times n$ matrix $W = (w_{ij})$ by the formula $w_{ij} = \max\{h_i^s + v_j^s : \ s = 1, \dots, r\}$ and apply Algorithm 7.3.

(8.3) Theorem. *Let us fix a polyhedral norm $\| \cdot \|$ in the Euclidean space $\mathbb{R}^d$. Then for any given $n \in \mathbb{N}$, for any n points $P_1, \dots, P_n$ in $\mathbb{R}^d$ given by their coordinates, and for any given $\epsilon > 0$ the above algorithm computes a Hamiltonian circuit whose length approximates, with relative error less than ϵ, the largest length of a Hamiltonian circuit with the vertices $P_1, \dots, P_n$. The algorithm uses a number of arithmetic operations which is polynomial in n and ϵ^{-1}. If $\| \cdot \|$ is a rational norm and the points $P_1, \dots, P_n$ are rational then all the computations can be performed over the rationals in time which is polynomial in the input size and ϵ^{-1}.*

Proof. Follows by Theorem 7.4 and Lemma 8.1 (see (8.1.1) in particular). □

Let $\| \cdot \|$ be an arbitrary norm in $\mathbb{R}^d$. Then we can approximate the unit ball $\mathbf{B}$ by a centrally symmetric convex polytope and thus approximate the norm $\| \cdot \|$ by a certain polyhedral norm. To prove Theorem 1.2 we must prove that there *exists* a polynomial time algorithm that computes an (approximate) longest Hamiltonian circuit. We will discuss how to *construct* a particular algorithm after the proof of Theorem 1.2.

(8.4) Proof of Theorem 1.2. Without loss of generality we assume that δ is sufficiently small, say $\delta < 1/3$. Let $\mathbf{B} = \{x \in \mathbb{R}^d : \|x\| \leq 1\}$ be the unit ball in the norm $\| \cdot \|$. There exists a rational centrally symmetric polytope $\mathbf{B}_1$ such that

$$(1 - \delta/3) \cdot \mathbf{B} \subset \mathbf{B}_1 \subset (1 + \delta/3) \cdot \mathbf{B}.$$

Then the Minkowski functional

$$\|x\|_1 = \min\{\lambda \geq 0 : x \in \lambda \cdot \mathbf{B}_1\}$$

is a polyhedral norm in $\mathbb{R}^d$ such that

$$(1 - \delta/3) \cdot \|x\|_1 \leq \|x\| \leq (1 + \delta/3) \cdot \|x\|_1$$

for all $x \in \mathbb{R}^d$.

The algorithm, whose existence is asserted, is the version of Algorithm 8.2 that uses the polyhedral norm $\|\cdot\|_1$ defined by the ball $\mathbf{B}_1$ and $\epsilon = \delta/3$. The length of the computed circuit approximates, with error less than δ, the largest length of a Hamiltonian circuit. Since the norm $\|\cdot\|$ and the error δ are fixed, we choose the norm $\|\cdot\|_1$ independently of the choice of the points $P_1, \ldots, P_n$. $\square$

To construct a specific algorithm whose existence is claimed by Theorem 1.2 we have to construct a rational centrally symmetric polytope $\mathbf{B}_1$ approximating the unit ball $\mathbf{B}$. A particular construction depends on how the norm $\|\cdot\|$ is given. For example, for the usual Euclidean norm for $\mathbf{B}_1$ we can take the convex hull of a sufficiently dense centrally symmetric ϵ-net on the unit sphere. A similar construction works for any L_p-norm. One can suggest a method for a norm given by an oracle. This oracle computes the norm of any given point and it should be *well guaranteed* in the sense of [4]. That is, the points with small cordinates shouldn't have too large norm whereas the points with a large coordinate shouldn't have too small norm. Then one can construct a linear operator L which transforms the ball $\mathbf{B}$ into a *well-rounded* body (see [4]), take the convex hull of a sufficiently dense ϵ-net and then apply the inverse operator L^{-1}. We don't discuss this issue in detail since it is a problem of computational convex geometry. As we mentioned earlier, the complexity of this construction does not affect the complexity of the algorithm from Theorem 1.2. However, the complexity the polyhedron $\mathbf{B}_1$ itself (the number of its facets and the size of integers in defining linear inequalities) does affect the computational complexity of the algorithm from Theorem 1.2.

9. Remarks

Similar methods can be developed for "Euclidean relaxations" of other NP-hard optimization problems. Let us consider, for example, the weighted 3-Dimensional Matching Problem. In this problem we are given an $n \times n \times n$ cubic tensor W of weights $W = (w_{ijk}): \ 1 \leq i, j, k \leq n$ and we are looking for a partition of the set $\{1, \ldots, n\}$ into $n/3$ pairwise disjoint 3-tuples so that the total weight of this partition is the largest possible (see [3]). One can construct a polynomial time approximation algorithm in the special case of this problem where the weight w_{ijk} is the perimeter of the triangle $\{i, j, k\}$ for given points $\{1, \ldots, n\}$ in a space $\mathbb{R}^d$ equipped with some fixed norm $\|\cdot\|$. The main challenge in this class of problems is to construct a polynomial time approximation algorithm for the Minimal Length Traveling Salesman Problem.

A simple modification of our methods allows one to find new special cases where the Minimal Weight Traveling Salesman Problem admits a polynomial time approximation algorithm. This is the case when the matrix $W = (w_{ij})$ is given by a representation of the type

$$w_{ij} = \min\{h_i^s + v_j^s : \ s = 1, \ldots, r\}$$

for all i, j and a fixed number r of real n-dimensional vectors h^s, v^s (one also needs an analogue of well-centeredness, cf. Definition 6.1). However, the author doesn't know any natural class of matrices having this property. Besides, it may be not easy to check whether the matrix admits such a representation.

Acknowledgements

Lemma 8.1 was proved in collaboration with Shmuel Onn and Bert Gerards. I am grateful for their valuable suggestions. I am indebted to anonymous referees for their remarks and for pointing out the papers [2] and [5].

References

1. A Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Reading, Mass: Addison-Wesley, 1974.
2. M.L. Fisher, G.L. Nemhauser and L.A. Wolsey, An analysis of approximations for finding a maximum weight hamiltonian circuit, *Operations Research*, **25**(1979), N 4, 799-809.
3. M. Garey and D. Johnson, *Computers and Intractability: a Guide to the Theory of NP-completeness*, Freeman, San Francisco, 1979.
4. M. Grötschel, L. Lovász, A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin, 1988.
5. Yu. Gurevich, S. Shelah, Expected computation time for Hamiltonian Path Problem, *SIAM J. Comput.*, **16**(1987), N 3, 486-502.
6. E. Lawler, et al., eds. *The Traveling Salesman Problem: a guided tour of combinatorial optimization*, Chichester, New York: Wiley, 1985.
7. H. Minc, *Permanents*, Reading, Mass.: Addison-Wesley, 1978.
8. C. H. Papadimitriou and M. Yannakakis, The traveling salesman problem with distances one and two, *Math. of Oper. Res.*, **18**(1993), N 1, 1-11.
9. J. Renegar, On the computational complexity and geometry of the first order theory of the reals. Part 1. Introduction. Preliminaries. The geometry of semi-algebraic sets. The decision problem for the existential theory of the reals, *Journal of Symbolic Computation* **13**, N 3 (1992), 255-299.

A.I. Barvinok
Department of Mathematics
University of Michigan
Ann Arbor, MI 48109-1003 USA

e-mail: barvinok@math.lsa.umich.edu