Chapter 11

POLYNOMIALLY SOLVABLE CASES OF THE TSP

Santosh N. Kabadi
Faculty of Administration
University of New Brunswick
Fredericton, New Brunswick
Canada E3B 5A3
kabadi@unb.ca

This chapter is dedicated to the memory of my parents
Mr. Narayan D Kabadi and Mrs Sushilabai N Kabadi.

1. Introduction

In this chapter we use the permutation definition of TSP introduced in Chapter 1. Given a cost matrix C of size n, the Traveling Salesman Problem (TSP) is thus to find a tour γ on $N = \{1, 2, \ldots, n\}$ such that $c(\gamma) = \sum_{i \in N} c_{i,\gamma(i)}$ is minimum. We allow some non-diagonal elements of C to be ∞. For $E = \{(i, j) : c_{ij}$ is finite$\}$, $G = (N, E)$ is called the digraph associated with C and we call C *compatible* with G. If C is symmetric, then we define G as an undirected graph. As in previous chapters, TSP can be equivalently stated as the problem of finding a tour $\mathcal{H}$ in G such that $c(\mathcal{H}) = \sum_{(i,j) \in \mathcal{H}} c_{ij}$ is minimum. We denote this problem by *TSP(G, C)*. When G is a complete digraph or graph (or equivalently, all the non-diagonal entries of C are finite), we denote the problem by *TSP(C)*.

2. Constant TSP and its generalizations

We start with a very special subclass of the TSP called *constant-TSP*. An instance *TSP(G, C)* is called a *constant-TSP* if and only if all the

G. Gutin and A.P. Punnen (eds.), The Traveling Salesman Problem and Its Variations, 489–583.

tours in G have the same cost with respect to C. We call a matrix C with finite non-diagonal entries a *constant tour matrix (CT-matrix)* if and only if *TSP(C)* is a constant-TSP.

A complete characterization of matrices C for which *TSP(G,C)* is constant-TSP is an open problem. For any digraph $G = (N, E)$ and arbitrary values $\{a_i, b_i : i \in N\}$, if C is defined as $c_{ij} = a_i + b_j$ for all $(i,j) \in E$, and $c_{ij} = \infty$ for all $(i,j) \notin E$, then *TSP(G, C)* is obviously a constant-TSP. Gabovich [341] proved that the reverse implication holds when G is complete. He attributes the special case of this result for the case of symmetric cost matrices to Rublineckii [733] and Leontev [557]. Several other independent proofs of this result have since then been reported [176, 361, 489, 558]. An independent proof for only the symmetric case is also reported in [524].

Theorem 1 *A cost matrix C is a CT-matrix if and only if there exist $\{a_i, b_i : i \in N\}$ such that $c_{ij} = a_i + b_j$ for all i, j, $i \neq j$. If C is symmetric, then it is a CT-matrix if and only if there exist $\{a_i : i \in N\}$ such that $c_{ij} = a_i + a_j$ for all i, j, $i \neq j$.*

Proof. The sufficiency part of the theorem is easy to prove.

The following simple proof of necessity of the condition is taken from [489]. Let C be a CT-matrix and let G be a complete digraph on node set N. Define $p = c_{1,2} - c_{1,n} - c_{n,2}$; $a_i = c_{i,n} + p/2$ and $b_i = c_{n,i} + p/2$ for $i = 1, \dots, n-1$; and $a_n = b_n = -p/2$. When the matrix C is symmetric, we have $a_i = b_i$ for all i. Define matrix C' as $c'_{ij} = c_{ij} - a_i - b_j$ for all i, j. Then $c'_{1,2} = 0$ and for all $1 \leq i < n$, $c'_{i,n} = c'_{n,i} = 0$. Let G' be the directed subgraph of G obtained by deleting node n and let C'' be obtained from C' by deleting its nth row and column. Then every Hamiltonian path in G' has the same cost, say u, with respect to the cost matrix C''. Let γ be an arbitrary tour in G'. For each $i \in N - \{n\}$, γ defines a unique Hamiltonian path in G' from $\gamma(i)$ to i with total cost $c''(\gamma) - c''_{i,\gamma(i)} = u$. Hence $c''_{i,\gamma(i)} = u/(n-2)$ for all $i \in N - \{n\}$. Since the tour γ was selected arbitrarily, it follows that each non-diagonal element of C'' has value $u/(n-2)$. Since $c''_{1,2} = 0$, this implies that each non-diagonal element of C'', and therefore each non-diagonal element of C', has a value of 0. This proves the theorem. ■

The above result can be equivalently stated in terms of the dimension of the TSP polytope of a complete digraph (graph). This is discussed in Chapter 2.

Suppose $G = (N, E)$ is the undirected tour $(1, 2, \dots, n, 1)$. Then *TSP(G, C)* is a constant-TSP for any symmetric matrix C compatible with G. It was observed by Krynski [524] that when n is an even integer, there exist symmetric matrices C compatible with G for which there do not exist $\{a_i : i \in N\}$ such that $c_{ij} = a_i + a_j$ for all $(i, j) \in E$. Thus, the above characterization of CT-matrices does not extend to matrices compatible with an arbitrary graph or digraph. We call a digraph $G = (N, E)$ a *constant tour digraph (CT-digraph)* if and only if whenever *TSP(G, C)* is a constant-TSP, there exist $\{a_i, b_i : i \in N\}$ such that for all $(i, j) \in E$, $c_{ij} = a_i + b_j$. An undirected graph $G = (N, E)$ is called CT-graph if and only if for any symmetric matrix C compatible with G such that *TSP(G, C)* is a constant-TSP, there exist $\{a_i : i \in N\}$ such that for all $(i, j) \in E$, $c_{ij} = a_i + a_j$. A complete characterization of CT-graphs and CT-digraphs is an open problem. Note that this problem is different from the problem, mentioned earlier, of characterizing matrices C for which *TSP(G, C)* is constant TSP. We give below results on subclasses of CT-graphs and CT-digraphs reported in [489].

Let (V_1, V_2) be a pair of disjoint, finite sets. For any two digraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the *join* of G_1 and G_2 is the digraph $G_1 + G_2 = (V_1 \cup V_2, E')$ where $E' = E_1 \cup E_2 \cup \{(i, j), (j, i) : i \in V_1, j \in V_2\}$. For any two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the *join* of G_1 and G_2 is the undirected graph $G_1 + G_2 = (V_1 \cup V_2, E')$, where $E' = E_1 \cup E_2 \cup \{(i, j) : i \in V_1, j \in V_2\}$. If $|V_2| = 1$ then we call $G_1 + G_2$ the *1-extension* of G_1 [489].

The only properties of the digraph G that are used in the proof of Theorem 1 are : (i) $\{(i, n), (n, i)\} \subseteq E$ for all $i \in \{1, \dots, n-1\}$ and (ii) the digraph G' obtained from G by deleting the node n is strongly Hamiltonian (that is, every arc in G' lies on some tour). The same argument can therefore be used to prove Theorem 2 below.

Theorem 2 *[489] The* 1*-extension of a strongly Hamiltonian digraph is a CT-digraph.*

For the undirected case, we can get the following slightly stronger result.

Theorem 3 *[489] The* 1*-extension of a Hamiltonian graph is a CT-graph.*

It may be noted that for $n \geq 3$, the complete digraph (graph) on node set N is 1-extension of the complete digraph (graph) on node set $N - \{n\}$. Theorem 1 is thus a simple corollary of theorems 2 and 3.

For two bipartite digraphs (graphs) $G_1 = (S_1 \cup T_1, E_1)$ and $G_2 = (S_2 \cup T_2, E_2)$ on disjoint node sets, *B-join* of G_1 and G_2, denoted by $G_1 +_B G_2$, is the bipartite digraph (graph) with node set $(S_1 \cup S_2) \cup (T_1 \cup T_2)$ and arc set (edge set) $E_1 \cup E_2 \cup \{(i, j), (j, i) : i \in S_1, j \in T_2\} \cup \{(i, j), (j, i) : i \in S_2, j \in T_1\}$ $(E_1 \cup E_2 \cup \{(i, j) : i \in S_1; j \in T_2\} \cup \{(i, j) : i \in S_2; j \in T_1\})$. If

G_2 is the complete digraph (graph) on two nodes then we call $G_1 +_B G_2$ the *1-1 extension* of G_1.

Theorem 4 *[489] The $1 - 1$ extension of a strongly Hamiltonian bipartite digraph is a CT-digraph. The $1 - 1$ extension of a Hamiltonian bipartite graph is a CT-graph.*

Since the complete bipartite digraph (graph) $\overset{\leftrightarrow}{k}(n,n)$ $(K(n,n))$ is the $1 - 1$ extension of $\overset{\leftrightarrow}{k}(n-1,n-1)$ $(K(n-1,n-1))$, it follows from Theorem 4 that $\overset{\leftrightarrow}{k}(n,n)$ $(K(n,n))$ is a CT-digraph (graph) for all n.

The following additional classes of CT-graphs and digraphs are identified in [489].

Theorem 5 *[489] Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two digraphs (graphs) on disjoint node sets. Then under each of the following conditions, $G_1 + G_2$ is a CT-digraph (graph).*

(i) G_1 *and* G_2 *have at least one arc (edge) each and* $3 \leq |V_1| = |V_2|$.

(ii) $2 \leq |V_1| < |V_2|$, $|E_1| \neq \emptyset$ *and* G_2 *is Hamiltonian.*

(iii) $3 \leq |V_1| < |V_2|$, $|E_1| \neq \emptyset$, *and* G_2 *has* k *node-disjoint simple paths which cover all the nodes in* V_2 *for some* $k < |V_1| - 1$.

At first sight, it may seem that a necessary condition for an undirected graph G to be a CT-graph would be that every edge in G lies in some tour in G [524]. A counter-example to this can be produced using the following observation.

Observation 6 *[489] Let $G = (S \cup T, E)$ be a CT-graph. Then $G' = (S \cup T, E \cup \{(u,v)\})$ is a CT-graph, where u, v either both belong to S or they both belong to T.*

For additional results on subclasses of CT-digraphs (graphs) as well as interesting classes of digraphs (graphs) that are not CT-digraphs (graphs), the reader is referred to [489].

The significance of the class constant-TSP in the study of polynomially solvable cases of the TSP lies in the fact that if $TSP(G, C')$ is a constant-TSP, then for any other matrix C compatible with G and for arbitrary tours γ and ψ in G, $c(\gamma) - c(\psi) = c''(\gamma) - c''(\psi)$, where $C'' = C + C'$. For any cost matrix C with associated digraph G, let us define the equivalence class

$$\begin{aligned} const(C) = \; & \{C + C' : C' \text{ is compatible with } G \text{ and} \\ & \quad TSP(G, C') \text{ is a constant-TSP}\}. \end{aligned}$$

We conjecture that the class of polynomially solvable cases of TSP is closed with respect to this equivalence relation, (that is, if $TSP(C)$ is polynomially solvable then $TSP(\bar{C})$ is polynomially solvable for any $\bar{C} \in const(C)$).

We define below the concept of *density matrix* of a matrix with finite entries[1], which is used extensively throughout this chapter.

Definition 7 : *For any $n \times n$ matrix A with finite entries (including the diagonal entries), the density matrix D of A is an $(n-1) \times (n-1)$ matrix defined as*

$$d_{ij} = a_{i,j+1} + a_{i+1,j} - a_{ij} - a_{i+1,j+1} \quad \forall\ 1 \leq i, j < n.$$

For example, if $A = \begin{bmatrix} 4 & 5 & 4 \\ 5 & 3 & 6 \\ 4 & 1 & 2 \end{bmatrix}$ then its density matrix is $D = \begin{bmatrix} 3 & -4 \\ 1 & 2 \end{bmatrix}$.

Observation 8 *[51, 479, 483] Two $n \times n$ matrices A and B have the same density matrix if and only if there exist $\{u_i, v_i : i \in N\}$ such that $a_{ij} = b_{ij} + u_i + v_j$ for all i, j.*

For any finite matrix C, let us define the equivalenceclass $dens(C)$ as the set of all matrices with the same density matrix as C. From Observation 8, it follows that $dens(C) \subseteq const(C)$. As we shall see later, many of the known polynomially solvable classes of TSP are closed under this stronger equivalence relation.

We end this section with an interesting generalization of Theorem 1 given in [785]. We call a matrix C with finite non-diagonal elements a *bi-constant tour matrix (BCT-matrix)* if and only if $|\{c(\gamma) : \gamma \text{ is a tour on } N\}| \leq 2$.

Theorem 9 *[785] A matrix C is a BCT-matrix if and only if $C \in const(C')$, where non-diagonal entries of C' have at the most two distinct values, say 0 and a, and either all non-diagonal entries with value a lie in a single row or column, or there are only two non-diagonal entries of value a and these are of the type c'_{ij} and c'_{ji}.*

A different characterization of BCT-matrices with an elementary proof is reported in [488].

[1]Though diagonal elements of the cost matrix C do not play any role in the definition of TSP, strangely, many of the algorithms for solvable cases of TSP (for example the Gilmore-Gomory algorithm) require diagonal elements to be finite and satisfy specific properties.

3. The Gilmore-Gomory TSP (GG-TSP)

The Gilmore-Gomory TSP [360] is one of the most celebrated polynomially solvable cases of the TSP. Besides being one of the first known, non-trivial polynomially solvable cases, it is also the first such case with significant real-world applications. A very simple polynomial time algorithm for this case, with a simple proof of its validity, is given by Ball et al [83] and we shall discuss it in Section 5. The algorithm that Gilmore and Gomory developed for the problem is however more efficient and it is also non-trivial with a fairly non-trivial proof of its validity. We discuss this algorithm in this section and in Section 4 we show that a slight generalization of this algorithm produces an optimal solution to a fairly large subclass of the TSP.

Gilmore and Gomory [360] considered the following case. A set of n given jobs are to be heat-treated in a furnace and only one job can be treated in the furnace at a time. The treatment of the ith job involves introducing it into the furnace at a given temperature a_i and heating/cooling it in the furnace to a given temperature b_i. The costs of heating and cooling the furnace are given by functions $f(.)$ and $g(.)$, respectively. Thus, for any u, v in $\mathbb{R}$, $u < v$, the cost of heating the furnace from temperature u to temperature v is $\int_u^v f(x)dx$, while the cost of cooling the furnace from v to u is $\int_u^v g(x)dx$. Gilmore and Gomory impose the realistic condition that

$$\text{for any } x \in \mathbb{R}, \quad f(x) + g(x) \geq 0.$$

For each ordered pair (i, j) of jobs, if we decide to heat-treat job j immediately after job i, then the furnace temperature has to be changed from b_i to a_j. This cost, which we call the change-over cost and denote by c_{ij}, is given by

$$c_{ij} = \begin{cases} \int_{b_i}^{a_j} f(x)dx & \text{if } b_i \leq a_j \\ \int_{a_j}^{b_i} g(x)dx & \text{if } a_j < b_i. \end{cases}$$

Starting with the furnace temperature of a_1 and processing job 1 first, we want to sequentially heat-treat all the jobs and end by returning the furnace temperature to a_1. The problem is to decide the order in which the jobs should be treated in the furnace so as to minimize the total change-over cost.

Gilmore and Gomory point out that if the starting temperature of the furnace is some other temperature a_0 and after processing all the jobs we want the ending temperature of the furnace to be say b_0, then the problem can be converted to the above case by introducing an $(n+1)$th job with $a_{n+1} = b_0$ and $b_{n+1} = a_0$.

3.1. Gilmore-Gomory scheme for GG-TSP

Let us now discuss the Gilmore-Gomory patching algorithm [360] (with minor modifications) for their special case of TSP.

We associate with any permutation π on N a digraph $G_\pi = (N, E_\pi)$, where $E_\pi = \{(i, \pi(i)) : i \in N\}$. Let $G_1, G_2, \dots, G_r$ be the connected components of G_π on node sets $N_1, N_2, \dots, N_r$, respectively. Then each G_i defines a subtour $\mathfrak{C}_i$ on the node set N_i. We call $\mathfrak{C}_1, \mathfrak{C}_2, \dots, \mathfrak{C}_r$ the *subtours of* π. If $r = 1$ then π defines a *tour* on N and we call such a permutation a *tour.* If $|N_i| > 1$ then the subtour $\mathfrak{C}_i$ is called a *non-trivial subtours of* π. A permutation with a single non-trivial subtour is called a *circuit.* A circuit with its only non-trivial subtour of the form (i, j, i) is called a *transposition* and is denoted by α_{ij}. A transposition of the form $\alpha_{i,i+1} = \alpha_{i+1,i}$ is called an *adjacent transposition* and is denoted by β_i. We denote by ξ the identity permutation, (that is, $\xi(i) = i$ for all i in N). For any two permutations π and ψ on N, we define $\pi \circ \psi$, *product of* π *with* ψ, as $\pi \circ \psi(i) = \pi(\psi(i))$ for all $i \in N$.

For any cost matrix C and any two permutations π and ψ on N. we define the *permuted cost matrix* $C^{\pi,\psi}$ as

$$c_{ij}^{\pi,\psi} = c_{\pi(i),\psi(j)} \quad \forall\, i, j.$$

We denote $C^{\xi,\psi}$ by C^ψ. Thus, $c(\pi \circ \psi) = c^\pi(\psi) = \sum_{i \in N} c^\pi_{i,\psi(i)}$.

Definition 10 *Suppose the digraph G_π associated with a permutation π has m connected components on node sets $N_1, N_2, \dots, N_m$. Then $G_p^\pi = (N_p^\pi, E_p^\pi)$, the patching pseudograph of π, is defined as $N_p^\pi = \{1, \dots, m\}$, and $E_p^\pi = \{e_i = (u, v) : i \in \{1, 2, \dots, n-1\}, i \in N_u, (i+1) \in N_v\}$. For any $S \subseteq \{1, 2, \dots, n-1\}$ we denote by $E_p^\pi[S]$ the set $\{e_i \in E_p^\pi : i \in S\}$.*

It may be noted that the edges $\{e_1, e_2, \dots, e_{n-1}\}$, when traversed in that order, form an Eulerian trail in G_p^π. Figure 11.1 shows the patching pseudograph for the case : $n = 8$, $\pi(1) = 4$, $\pi(2) = 3$, $\pi(3) = 5$, $\pi(4) = 1$, $\pi(5) = 2$, $\pi(6) = 8$, $\pi(7) = 7$, $\pi(8) = 6$.

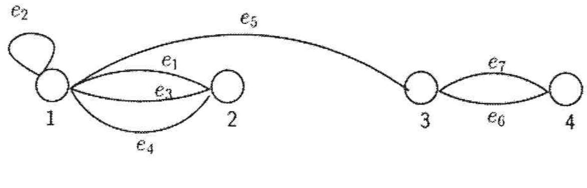

Figure 11.1. Patching pseudograph G_p^π

An optimal tour γ^* is an optimal assignment of a successor job to each job i which results in a cyclic ordering of the jobs. This can be expressed as an optimal matching of the values $\{b_i : i \in N\}$ to values $\{a_i : i \in N\}$ which results in a cyclic job ordering. The Gilmore-Gomory scheme starts by optimally matching of the b_i's with the a_i's, disregarding the requirement that the resultant job ordering should be cyclic. If the resultant permutation π on the job set N is a tour, then it is obviously an optimal tour. If we get subtours, then these subtours are patched together to get a tour as follows: a pair $\{i, i+1\} \subseteq \{1, \dots, n-1\}$ is chosen such that jobs i and $i+1$ lie in different subtours of π. (It may be noted the choice of patchings depends on the numbering of the jobs.) The permutation π is then modified to $\pi \circ \beta_i$. This results in the two subtours of π containing jobs i and $i+1$ being combined into one, while the other subtours of π remain unaffected. This operation, which we call *adjacent patching,* is then repeated until we get a tour. The patching pseudograph of π plays an important role in identifying an optimal set of adjacent patchings. A formal description of the algorithm is given below.

Gilmore-Gomory Scheme for the GG-TSP

Step 1: Renumber the jobs such that $b_1 \leq b_2 \leq \cdots \leq b_n$. Let π be a permutation on the set N such that $a_{\pi(1)} \leq a_{\pi(2)} \leq \cdots \leq a_{\pi(n)}$. If π is a tour then output π and stop.

Step 2: Suppose the digraph G_π has m connected components. Construct the patching pseudograph $G_p^\pi = (N_p^\pi, E_p^\pi)$. Assign to each edge $e_i \in E_p^\pi$ a cost $w_i = d_{ii}$, where D is the density matrix of C^π. Find a minimum cost spanning tree $E_p^\pi[T^*]$ in G_p^π, (that is, the edge set of the optimal spanning tree is $\{e_i : i \in T^*\}$).

Step 3: Construct a digraph $G_O = (T^*, E_O)$ where E_O is defined as follows: for all $\{i-1, i\} \subseteq T^*$,

$$\begin{aligned} (i-1, i) &\in E_O \text{ if } d_{ij} = 0 \text{ for all } j < i \\ (i, i-1) &\in E_O \text{ otherwise.} \end{aligned}$$

Find an ordering $(i_1, i_2, \dots, i_k)$ of its node set T^* such that for any $e = (i_u, i_v) \in E_O$, $u < v$. (Since the digraph G_O is acyclic, such an ordering exists.) Then $\gamma^* = \pi \circ \beta_{i_1} \circ \beta_{i_2} \circ \cdots \circ \beta_{i_k}$ is a tour and is the desired output. Stop.

It is not difficult to verify that the complexity of the above scheme is $O(n \log n)$.

Let us now illustrate the above algorithm with an example. Let $n = 8$, $(a_1, b_1) = (250, 100)$, $(a_2, b_2) = (160, 140)$, $(a_3, b_3) = (350, 200)$, $(a_4, b_4) = (100, 300)$, $(a_5, b_5) = (120, 450)$, $(a_6, b_6) = (550, 500)$, $(a_7, b_7) =$

$(500, 600)$, $(a_8, b_8) = (400, 700)$; $f(x) = 2$ and $g(x) = -1$ for all $x \in \mathbb{R}$. In this case, the cost matrix is

$$C = \begin{bmatrix} 300 & 120 & 500 & 0 & 40 & 900 & 800 & 600 \\ 220 & 40 & 420 & -40 & -20 & 820 & 720 & 520 \\ 100 & -40 & 300 & -100 & -80 & 700 & 600 & 400 \\ -50 & -140 & 100 & -200 & -180 & 500 & 400 & 200 \\ -200 & -290 & -100 & -350 & -330 & 200 & 100 & -50 \\ -250 & -340 & -150 & -400 & -380 & 100 & 0 & -100 \\ -350 & -440 & -250 & -500 & -480 & -50 & -100 & -200 \\ -450 & -540 & -350 & -600 & -580 & -150 & -200 & -300 \end{bmatrix}.$$

Step 1: We already have $b_1 \leq b_2 \leq \cdots \leq b_8$. Hence, no renumbering of jobs is necessary. We have $\pi(1) = 4$, $\pi(2) = 5$, $\pi(3) = 2$, $\pi(4) = 1$, $\pi(5) = 3$, $\pi(6) = 8$ $\pi(7) = 7$ $\pi(8) = 6$. The permutation π is not a tour. The graph G_π is shown in Figure 11.2.

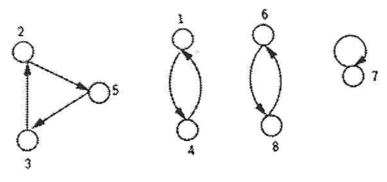

Figure 11.2. The digraph G_π

The matrix C^π and its density matrix D are as given below.

$$C^\pi = \begin{bmatrix} 0 & 40 & 120 & 300 & 500 & 600 & 800 & 900 \\ -40 & -20 & 40 & 220 & 420 & 520 & 720 & 820 \\ -100 & -80 & -40 & 100 & 300 & 400 & 600 & 700 \\ -200 & -180 & -140 & -50 & 100 & 200 & 400 & 500 \\ -350 & -330 & -290 & -200 & -100 & -50 & 100 & 200 \\ -400 & -380 & -340 & -250 & -150 & -100 & 0 & 100 \\ -500 & -480 & -440 & -350 & -250 & -200 & -100 & -50 \\ -600 & -580 & -540 & -450 & -350 & -300 & -200 & -150 \end{bmatrix},$$

and

$$D = \begin{bmatrix} 20 & 20 & 0 & 0 & 0 & 0 & 0 \\ 0 & 20 & 40 & 0 & 0 & 0 & 0 \\ 0 & 0 & 50 & 50 & 0 & 0 & 0 \\ 0 & 0 & 0 & 50 & 50 & 50 & 0 \\ 0 & 0 & 0 & 0 & 0 & 50 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 50 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Step 2 : $m = 4$; $N_1 = \{2, 3, 5\}$, $N_2 = \{1, 4\}$, $N_3 = \{6, 8\}$, $N_4 = \{7\}$. The patching pseudograph G_p^π is precisely the one shown in Figure 11.1. The weights of the edges of G_p^π are $\{w_1 = d_{11} = 20,\ w_2 = d_{22} = 20,\ w_3 =$

$d_{33} = 50,\ w_4 = d_{44} = 50,\ w_5 = d_{55} = 0,\ w_6 = d_{66} = 0,\ w_7 = d_{77} = 0\}$. Hence, $\{e_1, e_5, e_6\}$ is the edge set of a minimal cost spanning tree, (that is, $T^* = \{1, 5, 6\}$).

Step 3 : The corresponding Order digraph G_O is shown in Figure 11.3.

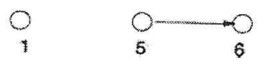

Figure 11.3. Order digraph $G_O = (T^*, E_O)$

Thisgives us ordering (1, 5, 6) of elements of T^*. Hence, $\gamma^* = \pi \circ \beta_1 \circ \beta_5 \circ \beta_6$. Thus, the output is $\gamma^* = (1, 5, 8, 6, 7, 3, 2, 4, 1)$.

The key operation used in the above algorithm is *post-multiplication of the starting permutation π by a sequence of adjacent transpositions.* We now present some basic results that will help us develop a better feel for this operation and therefore the algorithm. This will simplify explanation of the extensions of the scheme that will be discussed in the next section.

3.2. Some basic results

Recall that a circuit is a permutation with only one non-trivial subtour.

Observation 11 *[360] For $i \in \{1, 2\}$, let φ_i be a circuit on N with its unique non-trivial subtour $\mathfrak{C}_i$ on node set N_i. If $N_1 \cap N_2 = \emptyset$, then $\varphi_1 \circ \varphi_2 = \varphi_2 \circ \varphi_1$.*

For example, for any $\{i, j\} \subseteq \{1, 2, \ldots, n\}$ such that $|i - j| > 1$, $\beta_i \circ \beta_j = \beta_j \circ \beta_i$. However, $\beta_i \circ \beta_{i+1} \neq \beta_{i+1} \circ \beta_i$.

Observation 12 *[360] Let π be a permutation on N with non-trivial subtours $\mathfrak{C}_1, \mathfrak{C}_2, \ldots, \mathfrak{C}_r$ on node sets $N_1, N_2, \ldots, N_r$, respectively. For each $j \in \{1, \ldots, r\}$, let φ_j be the circuit on N with $\mathfrak{C}_j$ as its unique non-trivial subtour. Then $\pi = \varphi_{i_1} \circ \varphi_{i_2} \circ \cdots \circ \varphi_{i_r}$ for any ordering $(i_1, i_2, \ldots, i_r)$ of the elements of the set $\{1, 2, \ldots, r\}$.*

Observation 13 *[360] Let π be an arbitrary permutation on N and let $\{i, j\} \subseteq N$. (i) If i and j both belong to the same subtour $\mathfrak{C}$ of π then in $\pi \circ \alpha_{ij}$, the subtour $\mathfrak{C}$ is decomposed into two subtours, one containing i and the other containing j, while all the other subtours of $\pi \circ \alpha_{ij}$ are precisely the same as the other subtours of π. (ii) If i and j belong to*

Chapter 15

THE BOTTLENECK TSP

Santosh N. Kabadi
Faculty of Administration
University of New Brunswick-Frederiction
New Brunswick, Canada
kabadi@unb.ca

Abraham P. Punnen
Department of Mathematical Sciences
University of New Brunswick-Saint John
New Brunswick, Canada
punnen@unbsj.ca

1. Introduction

In this chapter we study the *bottleneck traveling salesman problem* (BTSP) introduced in Chapter 1. BTSP is a variation of the classical traveling salesman problem (TSP) that differs from the TSP only in the objective function. Let us first restate the problem.

Let $G = (N, E)$ be a (directed or undirected) graph on node set $N = \{1, 2, \ldots, n\}$ and let $\mathbb{F}$ be the family of all Hamiltonian cycles (tours) in G. For each edge $e \in E$, a cost c_e is prescribed. For any $\mathcal{H} \in \mathbb{F}$, let $c_{\max}(\mathcal{H}) = \max\{c_e : e \in \mathcal{H}\}$. Then the BTSP is to find a Hamiltonian cycle $\mathcal{H} \in \mathbb{F}$ such that $c_{\max}(\mathcal{H})$ is as small as possible.

Without loss of generality we replace G by K_n in the undirected case and $\overset{\leftrightarrow}{K}_n$ in the directed case by adding the missing edges with cost ∞. Thus, the edge costs will be given in the form of an $n \times n$ matrix C, called the cost matrix, where any non-diagonal entry c_{ij} corresponds to the cost c_e of the edge $e = (i, j)$. As indicated in Chapter 1and Chapter 11, we can also represent a tour in $\overset{\leftrightarrow}{K}_n$ as a cyclic permutation γ on $N = \{1, 2, \ldots, n\}$. Let $c_{\max}(\gamma) = \max\{c_{i,\gamma(i)} : i \in N\}$. Then the BTSP is to find a tour γ^* on N such that $c_{\max}(\gamma^*) = \min\{c_{\max}(\gamma) : \gamma$ is a tour on $N\}$. When the underlying cost matrix needs to be emphasized, we sometimes denote the BTSP with cost matrix C as BTSP(C). The BTSP can also be formulated as an integer programming problem:

$$\begin{aligned}
&\text{Minimize} && \max\{c_{ij}x_{ij},\ 1 \le i,j \le n,\ i \ne j\} && (1)\\
&\text{Subject to} && && (2)\\
&\sum_{i=1}^{n} x_{ij} &=& \ 1, j \in N && (3)\\
&\sum_{j=1}^{n} x_{ij} &=& \ 1, i \in N && (4)\\
&x_{ij} &=& \ 0 \text{ or } 1 && (5)\\
&\sum_{i\in S}\sum_{j\in \overline{S}} x_{ij} &\ge& \ 1\ \forall S \subset N, && (6)
\end{aligned}$$

where $\overline{S} = N \setminus S$.

In fact each of the (mixed) integer programming formulations of TSP discussed in Chapter 1 leads to a corresponding (mixed) integer programming formulation of the BTSP.

To the best of our knowledge, the bottleneck TSP was introduced by Gilmore and Gomory [360] assuming special structure of elements of C. Garfinkel and Gilbert [349] considered the general BTSP model and discussed an application of the problem in the context of machine scheduling. Meaningful interpretations of the BTSP model and its variations can be given in the context of some route planning problems and transportation of goods perishable in time.

Another problem closely related to the BTSP is the *maximum scatter traveling salesman problem* (MSTSP) [33]. Here, each edge e in G is assigned a weight w_e and we seek a tour $\mathcal{H}$ in G such that the smallest edge weight in $\mathcal{H}$ is as large as possible. Let $w_{\min}(\mathcal{H}) = \min\{w_e : e \in \mathcal{H}\}$. Then, the MSTSP is to find a Hamiltonian cycle $\mathcal{H} \in \mathbb{F}$ such that $w_{\min}(\mathcal{H})$ is as large as possible.

Applications of MSTSP and its variations include medical image processing [666], obtaining revetting sequence in joining metals in the aircraft industry [745, 746] etc.

The problems MSTSP and BTSP are equivalent in the sense that an optimal solution to the MSTSP with weight matrix W can be obtained by solving the BTSP with cost matrix $C = -W$ and vice versa. If we require the edge costs/weights to be positive, a large enough constant could be added to each of the edge costs/weights. However, this transformation may not be useful for some approximation algorithms that use special structure of C or W, since the addition of a constant and/or multiplication of edge costs/weights by -1 may violate key properties of the matrix that are used by these algorithms. Thus in such cases, the characteristics of the two problems are different and warrant separate treatment.

When the cost matrix C is symmetric, (equivalently the underlying graph is symmetric), we refer to the BTSP as *symmetric bottleneck traveling salesman problem* (SBTSP); otherwise it is referred to as *asymmetric bottleneck traveling salesman problem* (ABTSP). A special case of SBTSP, where the vertices of G correspond to points in the Euclidean plane and edge costs are Euclidean distances, is referred to as *Euclidean bottleneck traveling salesman problem*(EBTSP). Similarly we have symmetric, asymmetric, and Euclidean versions of the MSTSP.

It is well known that the Hamiltonian cycle problem on a grid graph is NP-complete [466]. As an immediate consequence we have that EBTSP is NP-hard and hence BTSP is NP-hard [466]. In fact BTSP is NP-hard even if we restrict the graph G to a grid graph or a planar bipartite graph in which degree of each node is 3 or less. As in the case of TSP, this follows immediately by a reduction from the Hamiltonian cycle problem on these graphs which is known to be NP-complete [466] (see Appendix B). Similarly, MSTSP can be shown to be NP-hard on grid graphs and planar bipartite graphs in which degree of each node is 3 or less. Fekete [284] recently proved that MSTSP under Euclidean distances in $\mathbb{R}^d$ is NP-hard for any fixed $d \geq 3$. More complexity results are discussed is Section 3.

2. Exact Algorithms

Recall that an exact algorithm for an optimization problem is guaranteed to produce an optimal solution for any instance of the problem or declare that a feasible solution does not exist. Since BTSP and MSTSP are NP-hard, such algorithms are generally of implicit enumeration type. Exact algorithms for MSTSP have not been discussed explicitly in literature. However, the transformation discussed in Section 1 can be used to solve MSTSP as BTSP.

2.1. BTSP as a TSP

In the preceding chapters, we have seen several interesting properties of and solution approaches for the TSP. Let us now examine how the TSP is related to the BTSP. We will first show that BTSP can be formulated as a TSP in the sense that an optimal solution to this TSP gives an optimal solution to the BTSP.

Note that in solving BTSP to optimality, the numerical values of the edge costs are unimportant; only the ordering of edge costs matters. Let the edges of G be labeled as $\{e_1, e_2, \ldots, e_m\}$ such that $c_{e_1} \leq c_{e_2} \leq \cdots \leq c_{e_m}$. Let $\{d_e : e \in E\}$ be another set of costs for the edges of G. Let us denote by BTSP(C) and BTSP(D) the instances of BTSP with edge costs c_e's and d_e's respectively. Let L and U respectively be known lower and upper bounds on the optimal objective function value of BTSP(C).

Lemma 1 *If $d_{e_1} \leq d_{e_2} \leq \cdots \leq d_{e_m}$ with $d_{e_i} < d_{e_{i+1}}$ whenever $c_{e_i} < c_{e_{i+1}}$ for all i such that $L \leq c_{e_i} \leq U$, then every optimal solution to BTSP(D) is also an optimal solution to BTSP(C).*

The proof of the above lemma is straightforward.

Let $\alpha_1 < \alpha_2 < \cdots < \alpha_t$ be an ascending arrangement of distinct costs c_e of edges of G such that $L \leq c_e \leq U$. Define $F_r = \{\mathcal{H} \in \mathbb{F} : c_{\max}(\mathcal{H}) = \max\{c_e : e \in \mathcal{H}\} = \alpha_r\}$ for $r = 1, \ldots, t$, $F_{t+1} = \{\mathcal{H} \in \mathbb{F} : c_{\max}(\mathcal{H}) > \alpha_t\}$, and $U_r = \cup_{i=1}^{r} F_i$, $r = 1, \ldots, t+1$. Consider new edge weights $\{d_e : e \in E\}$ satisfying,

$$\min\{\sum_{e \in \mathcal{H}} d_e : \mathcal{H} \in F_r\} > \min\{\sum_{e \in \mathcal{H}} d_e : \mathcal{H} \in U_{r-1}\} \tag{7}$$

for all $2 \leq r \leq (t+1)$. Here, minimum over empty set is taken as $-\infty$. Let TSP(D) denote the TSP with edge weights d_e for $e \in E$.

Theorem 2 *Every optimal solution to TSP(D) is also an optimal solution to BTSP(C).*

Proof. Clearly, if k is the smallest index such that F_k is non-empty then any $\mathcal{H} \in F_k$ is an optimal solution to BTSP(C) with the optimal objective function value α_k. If $\mathcal{H}'$ is an optimal solution to TSP(D) and $\mathcal{H}' \in F_p$, then,

$$\sum_{e \in \mathcal{H}'} d_e = \min\{\sum_{e \in \mathcal{H}} d_e : \mathcal{H} \in F_p\} > \min\{\sum_{e \in \mathcal{H}} d_e : \mathcal{H} \in U_{p-1}\}.$$

Thus, U_{p-1} must be empty and hence $\mathcal{H}'$ is optimal to BTSP(C). ∎

Corollary 3 *Let $b_1 = 0$ and for $j = 2, \ldots, t+1$, let $b_j = nb_{j-1} + 1$. Let the edge costs d_e's be defined as :*

$$d_e = \begin{cases} 0 & \text{if } c_e \leq \alpha_1 \\ b_j & \text{if } c_e = \alpha_j,\ j = 2, \ldots, t \\ b_{t+1} & \text{if } c_e > \alpha_t \end{cases}$$

Then every optimal solution to TSP(D) is also an optimal solution to BTSP(C).

Corollary 4 *Let p be the largest index such that $c_{e_p} \leq U$ and $m = |E|$. Define the edge costs $d_{e_j} = 2^{j-1}$ for $j = 1, 2, \ldots, p$ and $d_{e_j} = 2^{p+1}$ for $j = p+1, p+2, \ldots, m$. Then every optimal solution TSP(D) is also an optimal solution to BTSP(C).*

Let k_i be the number of edges e with $c_e = \alpha_i$, $i = 1, \ldots, t$.

Corollary 5 *Let $b_1 = 0$. For $j = 2, \ldots, t+1$, let u_j be the smallest positive integer such that $\sum_{i=u_j}^{j-1} k_i < n$. Define $b_j = \sum_{i=u_j}^{j-1} k_i b_i + (n - \sum_{i=u_j}^{j-1} k_i) b_{u_j - 1} + 1$. Let the edge costs d_e's be defined as :*

$$d_e = \begin{cases} 0 & \text{if } c_e \leq \alpha_1 \\ b_j & \text{if } c_e = \alpha_j, \; j = 2, \ldots, t \\ b_{t+1} & \text{if } c_e > \alpha_t \end{cases}$$

Then every optimal solution to TSP(D) is also an optimal solution to BTSP(C).

The proofs of corollaries 3, 4 and 5 follow from the fact that the special edge costs defined satisfy the condition (7).

Results analogous to Theorem 2 in the context of various bottleneck problems including the BTSP are well known [146, 410, 472, 691]. Although, Theorem 2 shows that BTSP can be solved as a TSP, it is not of much practical value since the costs d_e used in TSP(D) grow exponentially. However, when the number of distinct edge costs between given lower and upper bounds on the optimal objective function value is relatively small (say ≤ 10), then the edge costs defined in corollaries 3 and 5 are sometimes useful. This is exploited in the generalized threshold algorithm given in the next section.

A result similar to Theorem 2 can be obtained for the case of MSTSP showing that MSTSP can be solved as a MAX TSP or as a TSP.

2.2. Generalized Threshold Algorithm

The generalized threshold algorithm [691] is a modification of the well known threshold algorithm for solving bottleneck problems [269]. It solves BTSP as a sequence of TSP's with relatively smaller edge costs utilizing Corollary 3. Without loss of generality, we assume the edge costs are positive. Thus $L > 0$. Let $S_1, S_2, \ldots, S_r$ be an ordered partition of the index set $\{1, 2, \ldots, t\}$, (that is, $p \in S_i$, $q \in S_j$, $i < j$ implies $\alpha_p <$

α_q). We say that an edge e of G *corresponds to* S_i if $c_e = \alpha_p$ for some $p \in S_i$. For each edge e that corresponds to S_i, define $c_e^{'} = i$, $1 \leq i \leq r$. Define $c_e^{'} = 0$ if $c_e < L$ and $c_e^{'} = r + 1$ if $c_e > U$. Let BTSP (C') represent the BTSP with edge costs $\{c_e^{'} : e \in E\}$. Let $\mathcal{H}'$ be an optimal solution to BTSP(C') with optimal objective function value k. (Note that $k \in \{1, \ldots, r\}$.) Let $\mathcal{H}^*$ be an optimal solution to BTSP.

Theorem 6 *[691]* $\min_{i \in S_k}\{\alpha_i\} \leq c_{\max}(H^*) \leq \max_{i \in S_k}\{\alpha_i\}$.

Note that BTSP (C') is an approximation to the problem BTSP(C). If r is small (say ≤ 10), then BTSP(C') could be solved as a TSP with edge costs of moderate size using Corollary 3 or 5. Further, the solution to BTSP (C') provides new lower and/or upper bounds for the BTSP as guaranteed by Theorem 6. Using these new bounds, a 'better' approximation to BTSP can be constructed. Continuing this process, we eventually get an optimal solution to the BTSP. Our generalized threshold algorithm is precisely this. A formal description of the algorithm is given below.

The Generalized Threshold Algorithm

Step 1: Construct a lower bound L and an upper bound U for the optimal objective function value. Set $q = 1$, (q is the iteration counter).

Step 2: Let $\alpha_1 < \alpha_2 < \cdots < \alpha_{t_q}$ be an ascending arrangement of distinct edge costs c_e such that $L \leq c_e \leq U$.

Step 3: Construct the ordered partition $S_1, S_2, \ldots, S_{r(q)}$ of $\{1,2,\ldots, t_q\}$. Let every edge e with cost $c_e < L$ correspond to S_0 and every edge e with $c_e > U$ correspond to $S_{r(q)+1}$. Let $c_e^{'} = i$ for edges e that corresponds to S_i, $0 \leq i \leq r(q) + 1$.

Step 4: Solve the BTSP with costs $c_e^{'}$. (This is done by solving an equivalent TSP indicated in Theorem 2.) Let $\mathcal{H}'$ be the optimal solution produced and let the optimal objective function value be k. (Note that $k \in \{1, \ldots, r(q)$.)

Step 5: If $|S_k| = 1$, then output $\mathcal{H}'$ and stop. Else update the lower and upper bounds
$L = \min_{i \in S_k}\{\alpha_i\}$ and
$U = \max_{e \in \mathcal{H}'}\{c_e\}$.
Set $q = q + 1$ and go to Step 2.

The validity of the generalized threshold algorithm follows from the preceding discussions. Note that the number of partitions, r (indicated as $r(q)$ in the algorithm), is a function of the iteration counter and can be changed from iteration to iteration. For a large value of r, the TSP in Step 4 will have very large edge costs, leading to overflow errors. A reasonable choice of r is a number less than or equal to 10. The complexity of the algorithm depends on the number of iterations, which in turn depends on the way the partitions $S_1, S_2, \ldots, S_r$ are constructed. If $S_1, S_2, \ldots, S_{r-1}$ are selected as singletons and S_r contains all the remaining edges with cost c_e between L and U, the algorithm could take $O(n^2/r)$ iterations in worst case, since $t = O(n^2)$. We call this the *incremental search version* of the generalized threshold algorithm. If $|S_i|$ is approximately equal to $|S_j|$ for all $1 \leq i, j \leq r$, the number of iterations of the generalized threshold algorithm is $O(\log_r n)$. We call this *the r-section version* of the generalized threshold algorithm. If the lower bound L is expected to be tight, the incremental search version may work well in practice and often terminates in one iteration, although its worst case complexity is too high. If $r = 2$ (which is desirable for very large n), Step 4 of the generalized threshold algorithm can be implemented as testing hamiltonicity of an appropriate spanning subgraph of G. This special case of the generalized threshold algorithm is precisely the adaptation of the well known *threshold algorithm* to the BTSP.

The generalized threshold algorithm could take advantage of existing powerful TSP codes to get a reasonably fast algorithm for the BTSP without much programming effort. It is easy to develop a corresponding generalized threshold algorithm for the MSTSP which solves the problem as a sequence of MAX TSP's. We could also use the generalized threshold algorithm for BTSP to solve MSTSP using the transformation discussed in the Section 1.

2.3. Branch and Bound Algorithms

Branch and bound algorithms are classical approaches for solving 'hard' combinatorial optimization problems. The power of a branch and bound algorithm depends on the ability to generate good lower and upper bounds on the optimal objective function value and establishing an efficient branching strategy to generate the search tree. Garfinkel and Gilbert [349], Carpaneto et al [165], and Sergeev and Chernyshenko [757] developed specialized branch and bound algorithms to solve BTSP. Computational results based on problems of size less than or equal 200 are reported in [165, 349]. There is no recent experimental study published on the branch and bound algorithms for BTSP. The branching strategies used by Carpaneto et al [165] and Garfinkel and Gilbert [349] are similar to those studied for the case of TSP and will not be discussed here. For details we refer to the original papers. (See also Chapter 4.)

2.3.1 Lower Bounds. We now discuss some good lower bounds for the BTSP that can be obtained efficiently.

2-max Bound: This lower bound is very simple and easy to compute. It is valid for the symmetric version of BTSP only. For each node i of G, let $\Delta(i)$ be the set of edges incident on i and μ_i be the second smallest cost (counting multiplicity) of edges in $\Delta(i)$. Then $\max_{i \in N}\{\mu_i\}$ is a lower bound on the optimal objective function value of the BTSP. In a graph with m edges, this bound can be identified in $O(m)$ time. An asymmetric version of 2-max bound is introduced by Carpaneto et al [165].

Biconnected Spanning Subgraph Bound: We denote this lower bound as *BSS bound* and it is defined for the symmetric version of BTSP. Since every tour is biconnected, the optimum objective function value of a Bottleneck biconnected spanning subgraph problem (BBSSP) on the graph G is a lower bound for the BTSP. Several algorithms are available to solve the BBSSP. For example, in a graph with m edges and n nodes, Manku [578] proposed an $O(m)$ algorithm, Punnen and Nair [685] an $O(m + n \log n)$ algorithm, Timofeev [793] an $O(n^2)$ algorithm, and Parker and Rardin [661] an $O(n^2 \log n)$ algorithm to solve BBSSP.

Strongly Connected Spanning Subgraph Bound: We denote this bound as SCSS bound and is used for the asymmetric version of the BTSP. Since every directed Hamiltonian cycle is strongly connected, the optimal objective function value of a bottleneck strongly connected spanning subgraph problem (BSSSP) on the digraph G is a lower bound for the ABTSP on G. In a digraph with m arcs, BSSSP can be solved in $O(m)$ time [677].

Assignment Bound: The assignment problem is used to compute lower bounds for the traveling salesman problem. In the same way, the bottleneck assignment problem (BAP) can be used to compute a lower bound for the BTSP. Carpaneto et al [165] used the assignment bound, among other lower bounds, in their branch and bound algorithm for BTSP. They also provided a heuristic search scheme to find alternate optimal solutions for the BAP that correspond to cyclic permutations (tours). If the heuristic is successful in getting such a tour, it is indeed an optimal tour. BAP can be solved in $O(n^{2.5})$ time using an algorithm of Punnen and Nair [686]. For the case of sparse cost matrix (several edge costs are very large) an algorithm due to Gabow and Tarjan [343] runs faster. Other algorithms for BAP include Derigs and Zimmerman [253], and Carpaneto and Toth [167].

2.4. Branch and Cut Algorithms

Branch and cut algorithms are the state-of-the-art exact algorithms for the TSP (see Chapters 4and 2) that could solve reasonably large TSPs to optimality. A branch and cut algorithm for the TSP is based on a partial linear programming representation of the TSP. The success of the algorithm depends on the ability to generate facets or high dimensional faces of the TSP polytope that are violated by a 'current' infeasible solution. The BTSP can be formulated as a bottleneck linear programming problem (BLP) [410]

$$\begin{array}{ll} \text{Minimize} & \max\{c_e : x_e > 0\} \\ \text{Subject to} & X \in T_n, \end{array}$$

where T_n is the TSP polytope (symmetric or asymmetric depending on whether BTSP is symmetric or asymmetric) and entries of $X = (x_1, x_2, \ldots, x_m)$ correspond to edges of G. Since the objective function of the BLP is concave and quasi-convex, it can be shown that there exists an optimal solution to this BLP that is an extreme point of T_n and a local minimum is a global minimum. Thus branch and cut algorithms similar to those for TSP can be developed for the BTSP using cutting planes and a BLP solver. However, no implementation of such an algorithm is available in literature.

A dynamic programming approach for the BTSP was proposed by Sergeev [756]. As in the case of TSP, this algorithm is primarily of theoretical importance only.

3. Approximation Algorithms

In Chapters 5 and 6 we have studied approximation algorithms for the TSP where a *priori* mathematical guarantee could be obtained on the performance of an approximation algorithm. Chapters 8, 9, and 10 discussed implementation aspects of various practical approximation algorithms (heuristics) for TSP. The literature on approximation algorithms for BTSP is not as extensive as that of the TSP. In this section we study approximation algorithms for BTSP and MSTSP. We assume throughout this section that the edge costs are positive.

Recall that an algorithm yields a factor δ approximation for a minimization problem, if the algorithm is guaranteed to produce a solution whose objective function value is at most δ times the optimal objective function value. An algorithm yields a factor δ-approximation for a maximization problem if the algorithm is guaranteed to produce a solution whose objective function value is at least $1/\delta$ times the optimal objective function value.

Note that transformation used earlier between BTSP and MSTSP may not map a δ-approximate solution of BTSP to a δ-approximate solution of MSTSP. Thus, for worst case analysis of approximation algorithms we treat these two problems separately.

Theorem 7 *[661, 33] Unless P = NP, there is no polynomial time δ-approximation algorithm for the BTSP or MSTSP for any constant δ, $1 \leq \delta < \infty$.*

Proof. Let us first consider the case of symmetric BTSP. We prove the theorem by showing that any such approximation algorithm $\mathcal{A}$, if exists, can be used to solve the Hamiltonian cycle problem in polynomial time, implying P = NP. Suppose that we are given a graph G^* and we wish to test if G^* contains a Hamiltonian cycle. Convert the graph G^* to a complete graph K_n by adding the missing edges and assign a cost of $\delta + 1$ to each of these new edges. Assign a cost 1 to each of the original edges. Now we have an instance of SBTSP on K_n. If G^* contains a Hamiltonian cycle, the optimal objective function value, OPT, of our SBTSP is 1 and if the algorithm $\mathcal{A}$ is applied to this instance, it must produce a tour of value δ or less (in fact exactly equal to 1). If G^* has no Hamiltonian cycle, then *OPT* is $\delta + 1$. Thus the objective function value of the solution produced by $\mathcal{A}$ is less than or equal to δ precisely when G^* contains a Hamiltonian cycle. The result now follows from the NP-completeness of Hamiltonicity testing [347]. The proof for the case of ABTSP or MSTSP (symmetric and asymmetric) can be obtained in a similar way. ∎

In view of Theorem 7, it is not very likely that we shall succeed in obtaining meaningful performance bound for polynomial time approximation algorithms for BTSP or MSTSP with arbitrary edge costs. However, by assuming special properties of edge costs, heuristics with guaranteed performance bounds can be obtained.

3.1. Worst Case Analysis of Heuristics for BTSP

Recall that for any $\tau \geq 1/2$, the edge costs c_e of a complete graph K_n satisfy the τ-triangle inequality [22], if for any three nodes i, j, k of K_n, $c_{ij} \leq \tau(c_{ik} + c_{kj})$. If $\tau = 1$, τ-triangle inequality reduces to the triangle inequality. $\tau = 1/2$ forces all the edges of K_n to be of same cost. If $\tau > 1$, τ-triangle inequality can be viewed as a relaxation of the triangle inequality, where as for $1/2 < \tau < 1$, it is a restriction on the triangle inequality. The following lemma is an immediate consequence of the τ-triangle inequality.

Lemma 8 *Suppose the edge costs c_e of K_n satisfy the τ-triangle inequality for some $\tau \geq 1/2$. Let $P(i,j) = (e_1, e_2, \ldots, e_r)$ be a path in K_n joining vertices i and j. Then, $c_{ij} \leq \min\{S_1, S_2\}$, where $S_1 = \tau^{r-1}c_{e_r} + \sum_{k=1}^{r-1} \tau^k c_{e_k}$ and $S_2 = \tau^{r-1}c_{e_1} + \sum_{k=1}^{r-1} \tau^k c_{e_{r+1-k}}$.*

The concept called the *power of a graph* plays a central role in our approximation algorithms for SBTSP

Definition 9 *Let $G = (N, E)$ be a graph (not necessarily complete) and t be a positive integer. The t^{th} power of G is the graph $G^t = (N, E^t)$, where there is an edge $(u,v) \in E^t$ whenever there is a path from u to v in G with at most t edges.*

For any graph $G = (N, E)$ with edge costs c_e for $e \in E$, $c_{\max}(G)$ denotes $\max\{c_e : e \in E\}$. Similarly, $c_{\min}(G) = \min\{c_e : e \in E\}$. A similar notation will be used if G is replaced by a collection S of edges of G. We use the phrase $e \in G$ and $e \in E$ interchangeably.
G^2 is called the square of the graph G and G^3 is the cube of G.

Lemma 10 *Suppose the edge costs c_e of K_n satisfy the τ-triangle inequality for some $\tau \geq 1/2$. Let G be a subgraph of K_n. Then*

$$c_{\max}(G^t) \leq \begin{cases} t\ c_{\max}(G) & \text{if } \tau = 1 \\ \frac{\tau}{\tau-1}(2\tau^{t-1} - \tau^{t-2} - 1)c_{\max}(G) & \text{if } \tau > 1 \\ \frac{\tau}{\tau-1}(\tau^{t-1} + \tau - 2)c_{\max}(G) & \text{if } \tau < 1 \end{cases}$$

Proof. Let (i,j) be an arbitrary edge of G^t. By definition of G^t, there exists a path $P(i,j) = (e_1, e_2, \ldots, e_r)$ in G^t from i to j of length at most t. By Lemma 8,

$$c_{ij} \leq \tau^{r-1}c_{e_r} + \sum_{k=1}^{r-1} \tau^k c_{e_k} \leq t\ c_{\max}(G) \text{ for } \tau = 1.$$

Similarly, if $\tau > 1$, then by Lemma 8 we have,

$$\begin{aligned} c_{ij} &\leq \tau^{r-1} c_{e_r} + \sum_{k=1}^{r-1} \tau^k c_{e_k} \\ &\leq (\tau^{r-1} + \sum_{k=1}^{r-1} \tau^k) c_{\max}(G) \\ &\leq (\tau^{t-1} + \sum_{k=1}^{t-1} \tau^k) c_{\max}(G) \\ &= \frac{\tau}{\tau - 1}(2\tau^{t-1} - \tau^{t-2} - 1) c_{\max}(G) \end{aligned}$$

The case $\tau < 1$ can be proved in a similar way. ■

Theorem 11 *Suppose the edge costs c_e of K_n satisfy the τ-triangle inequality for some $\tau \geq 1/2$. Let S be a spanning subgraph of K_n such that $c_{\max}(S)$ is a lower bound for the optimal objective function value of BTSP and let $\mathcal{H}^*$ be an optimal solution to BTSP on K_n. If S^t, $1 \leq t < n$ and integer t, contains a Hamiltonian cycle $\mathcal{H}$, then*

$$c_{\max}(\mathcal{H}) \leq \begin{cases} t\ c_{\max}(\mathcal{H}^*) & \text{if } \tau = 1 \\ \frac{\tau}{\tau-1}(2\tau^{t-1} - \tau^{t-2} - 1) c_{\max}(\mathcal{H}^*) & \text{if } \tau > 1 \\ \frac{\tau}{\tau-1}(\tau^{t-1} + \tau - 2) c_{\max}(\mathcal{H}^*) & \text{if } \tau < 1 \end{cases}$$

Proof. Assume that $\tau = 1$. By definition of power of a graph, $c_{\max}(\mathcal{H}) \leq \max(S^t) \leq t\ c_{\max}(S)$. The last inequality follows from Lemma 10. From the optimality of $\mathcal{H}^*$, $c_{\max}(S) \leq c_{\max}(H^*)$. The result now follows immediately. The case $1/2 \leq \tau < 1$ and $\tau > 1$ can be proved in a similar way. ■

Theorem 11 and Lemma 10 are generalizations of corresponding results by Hochbaum and Shmoys [449] proved for the case $\tau = 1$.

Let us now discuss a simple heuristic for BTSP called *the bottleneck double tree heuristic,* which is an adaptation of the double tree heuristic for the TSP.

Bottleneck Double Tree Heuristic

Step 1: Compute a bottleneck spanning tree T of K_n.

Step 2: Duplicate the edges of T to form an Eulerian multigraph T^*.

Step 3: Identify an Eulerian tour in T^*. Traverse T^* along this Eulerian tour and introduce shortcuts whenever a previously visited node is encountered, to produce a tour in K_n.

Note that Step 3 of the algorithm allows a lot of flexibility. It is possible to construct examples where the double tree heuristic produces the worst solution. The quality of the solution produced however depends on the order in which the edges are traversed in the Eulerian tour. The solution generated by different orders of traversal may be different. Among all such solutions, identifying the best one can be shown to be a NP-hard problem. It may be noted that the cube of any connected graph is Hamiltonian connected (and hence Hamiltonian) [448, 449]. It has been shown by Hobbs [448] that a tour can be generated in Step 3 of the algorithm by introducing shortcuts to only paths of T^* of length 3 or less and such a tour, say $\mathcal{H}'$, belongs to T^3. (See also [100].) By Theorem 11, if the edge costs satisfy τ-triangle inequality, then

$$c_{\max}(\mathcal{H}') \leq \begin{cases} 3\ c_{\max}(\mathcal{H}^*) & \text{if } \tau = 1 \\ \frac{\tau}{\tau-1}(2\tau^2 - \tau - 1)c_{\max}(\mathcal{H}^*) & \text{if } \tau > 1 \\ \frac{\tau}{\tau-1}(\tau^2 + \tau - 2)c_{\max}(\mathcal{H}^*) & \text{if } 1/2 \leq \tau < 1 \end{cases}$$

where $\mathcal{H}^*$ is an optimal solution to SBTSP. The short-cutting phase as described above can be done in $O(n)$ time, (see for example Hobbs [448]). Also, the bottleneck spanning tree in Step 1 can be obtained in $O(m)$ time, where m is the number of edges in G [153]. Thus we have the following theorem.

Theorem 12 *If the edge costs satisfy the τ-triangle inequality, then the double tree algorithm produces a solution to the SBTSP in $O(n^2)$ time with a performance bound δ, where*

$$\delta = \begin{cases} 3 & \textit{if } \tau = 1 \\ \frac{\tau}{\tau-1}(2\tau^2 - \tau - 1) & \textit{if } \tau > 1 \\ \frac{\tau}{\tau-1}(\tau^2 + \tau - 2) & \textit{if } 1/2 \leq \tau < 1 \end{cases}$$

Let us now consider a general heuristic algorithm for BTSP based on the concept of power of a graph.

Algorithm Power(S,t)

Step 1: Construct a connected spanning subgraph S of K_n such that $c_{\max}(S)$ is a lower bound on the optimal objective function value of the BTSP.

Step 2: Find the smallest positive integer t such that S^t is Hamiltonian.

Step 3: Output any Hamiltonian cycle in S^t.

When S is a bottleneck spanning tree, and $t = 3$, Power(S,t) is identical to the double tree algorithm if Step 3 of power(S,t) is implemented using Steps 2 and 3 of the double tree algorithm. The complexity and performance bound in this case are the same as those of the double tree algorithm. We now show that by investing the same amount of time a better performance bound can be achieved.

Recall that the objective function value of the bottleneck biconnected spanning subgraph problem (BBSSP) is a lower bound on the optimal objective function value of BTSP. Thus, we could use a bottleneck biconnected spanning subgraph for S in Step 1 of algorithm *Power(S,t)*. As we have seen earlier, BBSSP on K_n can be solved in $O(n^2)$ time, (see Timofeev [793], Parker and Rardin [661], Punnen and Nair [685], and Manku [578]).

Now, for a biconnected graph, S, what is the smallest value of t such that S^t is Hamiltonian? Fleischner [312] proved that this value of t is 2. We state this elegant result in the following theorem.

Theorem 13 *The square of every biconnected graph is Hamiltonian.*

We are now left with the task of generating a tour in the square of a biconnected graph. Lau [542, 543] suggested an $O(n^2)$ algorithm to accomplish this. (See also Rardin and Parker [697].) Thus, by Theorem 11, the performance bound δ of this algorithm is 2τ (using $t = 2$) for $\tau \geq 1/2$. We thus have the following theorem.

Theorem 14 *Ifthe edge costs of K_n satisfy τ-triangle inequality, then a 2τ-approximate solution to the BTSP can be obtained in $O(n^2)$ time.*

Algorithm Power(S,t) for $t = 2$, and 3, with $\tau = 1$ was published first by Doroshko and Sarvanov [260] in a Russian paper in 1981. The case $t = 2$, $\tau = 1$ was obtained independently by Parker and Rardin [661]. Hochbaum and Shmoys [449] considered the case for general t with $\tau = 1$. We now show that improving the performance bound of Theorem 14 for any polynomial time algorithm and any $\tau >$ 1/2 amounts to P=NP. This result was proved by Doroshko and Sarvanov [260], Parker and Rardin [661], and Hochbaum and Shmoys [449] for the case $\tau = 1$.

Theorem 15 *Unless P = NP, there is no polynomial time $2\tau - \epsilon$ approximation algorithm for BTSP on K_n, with edge costs satisfying τ-triangle inequality, for any $\epsilon > 0$, $\tau > 1/2$.*

Proof. We prove the theorem by showing that for any $\epsilon > 0$, $\tau > 1/2$, a polynomial time $2\tau - \epsilon$ approximation algorithm $\mathcal{A}$ for the BTSP on K_n, with edge costs satisfying τ-triangle inequality, can be used to test Hamiltonicity of an arbitrary graph (digraph), establishing P=NP. Let