An Approximation Algorithm for the Subpath Planning Problem

Masoud Safilian[†], S. Mehdi Hashemi[†], Sepehr Eghbali[†*], Aliakbar Safilian[‡]
[†]Amirkabir University of Technology, Tehran, Iran
[†*]University of Waterloo, Ontario, Canada
[‡]McMaster University, Ontario, Canada
{m.safilian, hashemi}@aut.ac.ir, sepehr.eghbali@uwaterloo.ca, safiliaa@mcmaster.ca

Abstract

The subpath planning problem (SPP) is a branch of path planning problem, which has widespread applications in automated manufacturing process as well as vehicle and robot navigation. This problem aims to find the shortest path or tour subject to covering a set of given subpaths. By casting SPP to a graph routing problem, we propose a deterministic 2-approximation algorithm finding near optimal solutions, which runs in $O(n^3)$ time.

1 Introduction

The *path planning* problem is a fundamental problem in artificial intelligence and robotics with applications also in other areas such as computer animation and computer games. Due to its applications, several variants of this problem, including planning subject to spatial constraints, have been proposed over the past few years [Nash *et al.*, 2009; Kapadia *et al.*, 2013; Jaillet and Porta, 2013; Surynek, 2015].

In this paper, we focus on *Subpath Planning Problem* (SPP), which is a constraint path planning problem. Given a set of subpaths, the goal of SPP is to find the shortest tour that travels all given subpaths. SPP is an **NP**-hard problem with widespread applications. For example, in polishing robot [Tong-ying *et al.*, 2004] the target is to polish nicks on a surface of a workpiece. Nicks, which can be considered as the subpaths, are often extracted by using image processing techniques and the goal is to polish all nicks with minimum arm movement. Also, SPP has applications in electronic printing [Gyorfi *et al.*, 2010], where the continuous-spray processes are used to deposit electrically functional material onto flexible substrate. Moreover, SPP can be used in search and rescue operations as well as aerial image using UAV, where the robot must traverse some specific paths. Fig. 1.a and Fig. 1.b show a toy example of SPP workspace and its optimal solution, respectively. A formal definition of SPP can be found in [Gyorfi *et al.*, 2010].

However, not much attention has been paid to solving this special widely used problem. Only a few approaches have been proposed, which are all meta-heuristic based. For instance, in [Tong-ying *et al.*, 2004] and [Gyorfi *et al.*, 2010] two algorithms based on *Genetic Algorithm* (GA) [Goldberg, 1989] are proposed for solving SPP in polishing robots and electronic printing, respectively. The major deficiency of such techniques is that, they cannot guarantee any bound on their final output due to their meta-heuristic (and random) nature. More importantly, GA needs considerable amount of time in order to converge to a sub optimal solution, especially for large-scale problems.

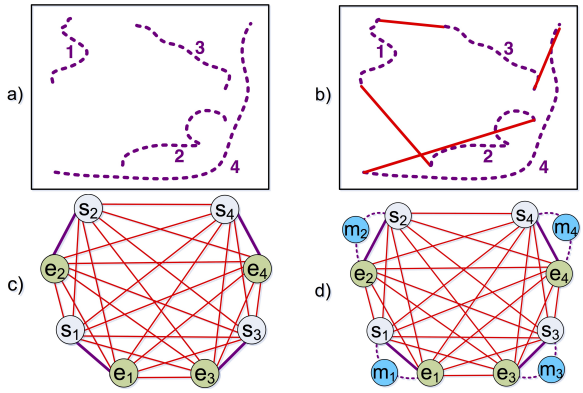


Figure 1: a) A toy example of SPP workspace (subpaths are shown in dashed lines), b) The optimal solution for the given workspace, c) Graph G, d) Graph G'.

In its graph theoretical abstraction, SPP is a graph routing problem with close connections to the *Travelling Salesman with Neighbours* (TSPN) [Papadimitriou, 1977; Safra and Schwartz, 2006], *Rural Postman Problem* (RPP) [Eiselt *et al.*, 1995b; Orloff, 1974], and *Stacker Crane Problem* (SCP) [Coja-Oghlan *et al.*, 2006; Frederickson, 1979]. However, SPP is different from these graphical problems in some aspects, which makes their solutions not applicable to the SPP.

The present paper addresses the deficiencies of meta-heuristic methods for solving SPP by proposing an approximation algorithm with a constant approximation bound and efficient polynomial time complexity.

As the first step, given n subpaths, we transform the SPP to the general *Travelling Salesman Problem* (TSP). By transforming SPP to TSP, it may seem easy to apply the existing constant-factor approximation algorithms of TSP for solving SPP. However, this is not feasible, as the edge weights in the graph of SPP are not metric, i.e., they do not necessarily sat-

isfy the triangle inequality. To mitigate this problem, in the second step, we propose an algorithm, called *Imperfectly Establish the Triangle Inequality* (IETI), which establishes the triangle inequality for a large subset of *violating triangles* (the triangles that violates the triangle inequality in the graph of SPP). We show that solving SPP on the original graph would be equivalent to solving TSP on the modified graph. The IETI algorithm runs in $O(n^2)$ time and is a fundamental step for designing an algorithm with a constant approximation factor for solving SPP.

Nonetheless, a subset of triangles may still violate the triangle inequality in the graph returned by IETI algorithm. To tackle this problem, in the third step, we propose a 2-approximation algorithm, called *Christofides for SPP* (CSPP) that is in a $O(n^3)$ time complexity. The CSPP bears some similarities to Christofides algorithm [Christofides, 1976] and is adapted to work for all outputs of the IETI algorithm. Christofides algorithm is a well-known 1.5-approximation algorithm for solving TSP due to its small approximation factor and polynomial time complexity ($O(n^3)$).

In addition to complexity analysis and proving the approximation bound of CSPP, we empirically compare it with the method proposed in [Gyorfi *et al.*, 2010] over various workspaces with different numbers of subpaths. The results indicate that CSPP is more efficient than the other existing methods both in terms of cost of the solution and running time.

The rest of the paper is organized as follows: Sections 2, 3, and 4 discuss the transformation, IETI, and CSPP algorithms, respectively. Section 5, presents the experimental results and analysis. Section 6 concludes the paper and presents some potential future directions. Due to the space limitation, some details including the full proofs of the theorems are not described in the paper. We refer the interested reader to the accompanying techinical report [Safilian *et al.*, 2016] for more details.

2 Transformation of SPP to TSP

The solutions of an SPP problem are the tours that travel all subpaths and an optimal solution is a tour with the minimum length. Each solution is a sequence of connected subpaths. Since each pair of subpaths can be connected to each other in two ways, the total number of feasible solutions would be $n!2^n$ where n is the number of subpaths.

In the recent decades, various approximation and combinatorial optimization methods have been proposed for solving TSP [Arora, 1996; Dumitrescu and Mitchell, 2001; Lin and Kernighan, 1973]. Thus, transforming of SPP to TSP facilitates applying such methods for solving SPP.

Consider an SPP instance with n subpaths indexed with the set $I = \{1, ..., n\}$. The transformation procedure of SPP to TSP, includes two stages. In the first stage, a complete graph G is constructed using the following steps:

1. For each i-th subpath ($i \in I$), consider two nodes s_i and e_i corresponding to its starting and end points, respectively.
2. For each $i \in I$, consider an edge between s_i and e_i with the weight equal to the length of i-th subpath in the workspace. Let us call this edge the *i-th subpath edge.*
3. For each pair of subpaths such as i and j ($i \neq j$), we add edges s_ie_j, e_is_j, s_is_j and e_ie_j to the graph. We also consider the weights of these newly added edges equal to the corresponding Euclidean distances in the workspace. Let us call these edges the *connecting edges*.

The TSP tour of G is not necessarily a solution for the given SPP instance, since the TSP tour of G may not cover all subpath edges (such as s_ie_i). To make sure that the TSP tour would travel all subpaths, a complete graph, denoted by G', is constructed based on G as follows:

4. For each i-th subpath, a node m_i is added to the graph, called the *middle node* of i-th subpath.
5. For each i-th subpath, two edges s_im_i and m_ie_i are added to the graph, the weights of which are equal to the half of the subpath length. These edges are called *i-th double subpath edges*.
6. For each middle node m_i, the edge m_iv, where $v \notin \{s_i, e_i\}$, is added to the graph with infinite weight.

Fig. 1.c and Fig. 1.d show the graphs G and G' for a problem with 4 subpaths, respectively.

Theorem 1 *A solution of SPP on a given instance corresponds to a solution of TSP in G'.*

Proof: It is clear that there is a finite Hamiltonian tour in G' (see Fig. 1.d). The TSP tour over G' is a Hamiltonian tour with the minimum weight. Therefore, the TSP tour over G' is finite. The TSP tour of G' visits all the middle nodes since for each i, it contains two edges crossing the node m_i. There are only two finite edges, namely s_im_i and m_ie_i, connected to m_i. Hence, the TSP tour over G' must contain s_im_i and m_ie_i for each i. Since s_im_i and m_ie_i together are equivalent to the i-th subpath in the workspace, the TSP tour of G' is the minimum weight which travels all subpaths. We can conclude that solving SPP is equivalent to finding the TSP tour in G'. It is easy to show that the cost of constructing graph G and G' is $O(n^2)$. □

3 Imperfectly Establish the Triangle Inequality

Using any existing approximation method for the TSP, requires triangle inequality to be satisfied over the entire graph. Two sets of triangles in G' may violate triangle inequality:
$\mathcal{V}_1$: The set of triangles with a subpath edge as one of their edges, i.e, triangles in the form of $\triangle s_ie_iv$, where $v \neq m_i$.
$\mathcal{V}_2$: The set of triangles with only one infinite edge. In such triangles, one of the edges is either s_im_j or e_im_j ($i \neq j$).

Other triangles in G' do not violate the triangle inequality condition and can be grouped into the following sets:
$\mathcal{V}_3$: Triangles that have more than one infinite edge.
$\mathcal{V}_4$: Triangles in which all of the edge weights are equal to their corresponding Euclidean distances.
$\mathcal{V}_5$: Triangles that are of the form $\triangle s_im_ie_i$.

As mentioned in the Introduction, we tackle the triangle inequality violation in G' in two steps. First, we propose an algorithm to establish the inequality for some triangles. Then,

we propose another algorithm that finds the approximate solution in the resulting non-Eulcidean graph.

3.1 IETI: The Procedure

The IETI algorithm takes the output of the transformation procedure (G') as its input and deals with the violating triangles in $\mathcal{V}_1$. This algorithm makes a set of modifications to the weight of edges in G' to construct the graph G'' such that TSP tours in G' and G'' are the same (see Theorem 3) and $\mathcal{V}_1 = \varnothing$, i.e., there is no violating triangles in the set $\mathcal{V}_1$ in G'' (see Theorem 2).

IETI is an iterative algorithm which loops over all subpaths. In each iteration, say i, we define a variable called *i-th degree of violation*, denoted by dv_i, to find triangles in $\mathcal{V}_1$ that violate the triangle inequality. The value of dv_i is defined as:

$$\begin{aligned} &dv_i = 0.5(\min_d(w(s_i,d)+w(e_i,d)) - w(s_i,e_i)), \\ &\forall\, d \in V(G') \setminus \{s_i, e_i\}, \end{aligned} \tag{1}$$

where $V(H)$ and $w(a,b)$ denote the set of vertices of graph H and the weight of the edge ab, respectively. Negative values of dv_i implies that at least one of the triangles with $e_i s_i$, as one of their edges, violates the triangle inequality. In such cases, the weight of edges are updated by the following equations:

$$\begin{aligned} &w(s_i,e_i) \leftarrow w(s_i,e_i) - |dv_i| \\ &w(s_i,m_i) \leftarrow w(s_i,m_i) - |dv_i|/2 \\ &w(e_i,m_i) \leftarrow w(e_i,m_i) - |dv_i|/2 \end{aligned} \tag{2}$$

$$\begin{aligned} &w(q,e_i) \leftarrow w(q,e_i) + \frac{|dv_i|}{2}, \quad w(q,s_i) \leftarrow w(q,s_i) + \frac{|dv_i|}{2}, \\ &\forall q \in V(G') \setminus \{s_i, e_i, m_i\} \end{aligned} \tag{3}$$

These changes satisfy the triangle inequality in triangles containing the i-th subpath. The appeal of using the proposed updates is that it does not make other triangles, which already satisfy the triangle inequality, violating the condition. This claim is proven in Theorem 2.

Theorem 2 *After the execution of IETI, all triangles in G'' that are not in $\mathcal{V}_2$ satisfy the triangle inequality .*

Proof: Let n be the number of subpaths of the original workspace. Consider the i-th step of IETI. Let G''_i denote the resulting graph after execution of the $i-th$ step of IETI on G'. It is clear that G''_0 and G''_n are equal to G' and G'', respectively. Now, we prove the following statement:

(*statement*): A violating triangles in G''_i either: 1) is a member of $\mathcal{V}_2$ or 2) is a triangle such as $\triangle s_j e_j v$ in $\mathcal{V}_2$ with $j > i$.

We use an inductive reasoning to prove the above statement:

(*base case*): It is evident that the above statement holds in G''_0 (which is equal to G').

(*hypothesis*): Assume that, for some t with $1 \leq t < i$, the statement holds for $G''_0, ..., G''_{t-1}$, now it suffices to show that the statement also holds for G''_t. This is shown in the inductive step.

(*inductive step*): G''_{t-1} can be in one of following forms. We show in each of them the G''_t satisfies the statement.

1. "In G''_{t-1}, the triangles with the edge $s_t e_t$ do not violate the triangle inequality condition."

It means that for any $j \leq t$, the triangles in G''_{t-1} with an edge $s_j e_j$ do not violate the triangle inequality. In this case, during the $t-th$ step, no modification will be made to the graph G''_{t-1}. Thus, G''_t would be equal to G''_{t-1}. This means that for any $j \leq t$, the triangles in G''_t with an edge $s_j e_j$ do not violate the triangle inequality. Therefore, the statement holds for G''_t.

2. "There are some triangles in G''_{t-1} with an edge $s_t e_t$, which violates the triangle inequality."

In this case the weights of edges are updated according to equations (2) and (3). Each triangle $\triangle abc$ in G''_t falls into one of the seven following categories. The validity of triangle inequality in all categories will be investigated. In particular, we consider the inequalities $w_t(a,c) + w_t(b,c) \geq w_t(a,b)$ and $w_t(a,b) + w_t(b,c) \geq w_t(a,c)$, where $w_t(a,b)$ denotes the weight of the edge ab in G''_t [1].

(a) "The edge ab is equal to $s_t e_t$."
In this case, the weight of the edge ab decreases by $|dv_t|$ and the weights of the edges ac and bc increases by $\frac{|dv_t|}{2}$. The equations (5) and (9) show that the triangles in this category satisfy the triangle inequality after modifications:

$$\begin{aligned} &w_t(a,c) + w_t(b,c) \\ &= w_t(s_t,c) + w_t(e_t,c) \\ &= w_{t-1}(s_t,c) + |dv_t|/2 + w_{t-1}(e_t,c) + |dv_t|/2 \\ &= w_{t-1}(s_t,c) + w_{t-1}(e_t,c) + 2|dv_t| - |dv_t|. \end{aligned} \tag{4}$$

By replacing $2|dv_t|$ with q where $q = w_{t-1}(s_t,e_t) - \min_{\forall d}[w_{t-1}(s_t,d) + w_{t-1}(e_t,d)]$, we would have:

$$\begin{aligned} &w_t(a,c) + w_t(b,c) \\ &= w_{t-1}(s_t,c) + w_{t-1}(e_t,c) + q - |dv_t| \\ &\geq w_{t-1}(s_t,e_t) - |dv_t| = w_t(s_t,e_t) = w_t(a,b). \end{aligned} \tag{5}$$

Also, we have:

$$\begin{aligned} &w_t(a,b) + w_t(b,c) \\ &= w_t(s_t,e_t) + w_t(e_t,c) \\ &= w_{t-1}(s_t,e_t) - |dv_t| + w_{t-1}(e_t,c) + |dv_t|/2. \end{aligned} \tag{6}$$

By replacing $|dv_t|$ with $q/2$, we would have:

$$\begin{aligned} &w_t(a,b) + w_t(b,c) = w_{t-1}(s_t,e_t) + w_{t-1}(e_t,c) \\ &- q/2 + |dv_t|/2 \\ &= w_{t-1}(s_t,e_t)/2 + |dv_t|/2 \\ &+ \min_d[w_{t-1}(s_t,d) + w_{t-1}(e_t,d)]/2 + w_{t-1}(e_t,c). \end{aligned} \tag{7}$$

[1] Validity of equations $w_t(a,b) + w_t(a,c) \geq w_t(b,c)$ is similar to of $w_t(a,b) + w_t(b,c) \geq w_t(a,c)$ can be investigated in a similar fashion.

The weight of the edge s_te_t is not modified before the t-th step. Thus, $w_{t-1}(s_t, e_t) = w_0(s_t, e_t)$. In other words, it is equal to the length of $t-th$ subpath. Hence, $w_{t-1}(s_t, e_t) \geq ed(s_t, e_t)$, where $ed(s_t, e_t)$ is the Euclidean distance between s_t and e_t in the workspace.

Also, any node such as d satisfies the inequality $w_{t-1}(s_t, d) + w_{t-1}(e_t, d) \geq ed(s_t, d) + ed(e_t, d) > ed(s_t, e_t)$.[2] As a consequence, the inequality $\min_{\forall d}[w_{t-1}(s_t, d) + w_{t-1}(e_t, d)] > ed(s_t, e_t)$ holds, which results in:

$$w_t(a, b) + w_t(b, c) \geq ed(s_t, e_t) + w_{t-1}(e_t, c) + |dv_t|/2. \quad (8)$$

If the weights of subpaths, which pass through the node c change before the $t - th$ step, then the values of $w_{t-1}(s_t, c)$ and $w_{t-1}(e_t, c)$ would be equal to $ed(s_t, c) + \frac{\alpha}{2}$ and $ed(c, e_t) + \frac{\alpha}{2}$, respectively, where α is equal to the parameter dv (degree of violation) of the subpath which passes through the node c.
Otherwise (if the value of such subpaths are not updated), the values of $w_{t-1}(s_t, c)$ and $w_{t-1}(e_t, c)$ would be $ed(s_t, c)$ and $ed(c, e_t)$, respectively. Hence, the inequality $ed(s_t, e_t) + w_{t-1}(c, e_t) > w_{t-1}(s_t, c)$ turns to $ed(s_t, e_t) + ed(c, e_t) > ed(s_t, c)$ that always holds. Consequently, the following inequality holds:

$$\begin{aligned} w_t(a, b) + w_t(b, c) &\geq w_{t-1}(s_t, c) + \frac{|dv_t|}{2} \\ &\geq w_t(s_t, c) \geq w_t(a, c). \end{aligned} \quad (9)$$

(b) "The edge ab is s_je_j (i.e., $a = s_j$ and $b = e_j$), where $j < t$" (see [Safilian *et al.*, 2016] for the proof).

(c) "The edge ab is s_je_j, where $j > t$".
In this case, the triangle $\triangle abc$ may violate the triangle inequality in the graph G''_{t-1}. Hence, it may violate the triangle inequality in G''_t as well.

(d) "$\triangle abc$ does not contain any edge in form of s_je_j for some j (i.e., it does not have any subpath edge), and the weights of the edges ac and bc change during the $t - th$ step"[3] (see [Safilian *et al.*, 2016] for the proof).

(e) "$\triangle abc$ does not contain any edge in the form of s_je_j for some j and the weight of none of the edges of the triangle changes during the $t - th$ step".
Since $\triangle abc$ satisfies the triangle inequality in G''_{t-1}, it also satisfies the inequality in G''_t.

(f) "$\triangle abc$ has one edge with the infinite weight".
The triangle violates the triangle inequality condition in both G''_{t-1} and G''_t.

(g) "$\triangle abc$ has two or three edges with the infinite weight".
It is clear that $\triangle abc$ satisfies the triangle inequality in both G''_{t-1} and G''_t

[2]The weights of edges s_td and e_td either have not changed before $t - th$ step or are modified by equation (3). Therefore, $w_{t-1}(s_t, d) \geq w_0(s_t, d) \geq ed(s_t, d)$ and $w_{t-1}(e_t, d) \geq w_0(s_t, d) \geq ed(e_t, d)$.

[3]Note that during each step of IETI for each triangle either no weight is updated (category e) or two weights are updated (category d).

Thus, after the execution of the t-th step, only the triangles with one infinite edge (category f) or with edges in the form of s_je_j for some j greater than t (category c) may violate the triangle inequality in G''_t. Therefore, the statement holds in G''_t (which is equal to G''). This concludes the proof. □

Theorem 3 *The TSP tours of G' and G'' are the same.*

Proof: To prove this theorem, we show that the TSP tours of G' and G'' are the same in terms of length and sequence of nodes. In particular, we show that the length of each finite Hamiltonian tour of G' is equal to the length of its corresponding finite Hamiltonian tour of G'' (two Hamiltonian tours in G' and G'' corresponds to each other, if they have the same sequence of nodes).

Any finite Hamiltonian tour in both G' and G'' includes all edges in the form of s_im_i and m_ie_i (for any possible index i). Note that in such a tour, for any i, m_ie_i and s_im_i appear consecutively and are connected through m_i. Let us call these two consecutive edges *"pair of edges of the i-th subpath"*. As a result, each finite Hamiltonian tour in either G' or G'' is a sequence of pairs of edges connected via some other edges.

Let H be a finite Hamiltonian tour over G' and let H' be its corresponding finite Hamiltonian tour over G''. Without loss of generality, suppose that the nodes in H and H' are ordered as follows (where any node such as p in G' corresponds to the node p' in G''):

$$H : e_1m_1, m_1s_1, s_1e_2, ..., s_{i-1}e_i, e_im_i, m_is_i, s_ie_{i+1}, ..., e_{n-1}s_n, s_nm_n, m_ne_n$$

$$H' : e'_1m'_1, m'_1s'_1, s'_1e'_2, ..., s'_{i-1}e'_i, e'_im'_i, m'_is'_i, s'_ie'_{i+1}, ..., e'_{n-1}s'_n, s'_nm'_n, m'_ne'_n.$$

During the i-th iteration of the IETI procedure, only the weights of the edges in H, which are in the form of e_im_i, m_is_i, s_ie_i, e_is_{i-1}, and s_ie_{i+1}, may change and others remain the same. Accordingly, the following equations hold:

$$w(e_im_i) + w(m_i, s_i) = w(e'_i, m'_i) + w(m'_i, s'_i) + |dv_i|. \quad (10)$$

$$\begin{aligned} w(s_{i-1}, e_i) &= w(s'_{i-1}, e'_i) - |dv_i|/2 \\ w(s_i, e_{i+1}) &= w(s_i, e_{i+1}) - |dv_i|/2. \end{aligned} \quad (11)$$

The above equations show that even after updating the weights in each step of IETI the length of the tours H and H' are equal to each other. Therefore, the length of each finite Hamilton tour in G' is equal to its corresponding tour in G''. □

By analysing the steps involved in constructing G'', it is easy to show that the total complexity of the IETI is $O(n^3)$. (see [Safilian *et al.*, 2016] for a detailed analysis.)

4 A 2-approximation Algorithm

According to Theorem 3, the TSP tour in G'' is a solution for SPP. However, some triangles in G'' still violate the triangle inequality. Therefore, it is not possible to use the Christofides algorithm (or any other existing constant-factor approximation algorithm) over G'' to find the TSP tour in G''. Nonetheless, the triangles that violate the triangle ineqiality in G'' are not arbitrary. By exploiting their characteristics, we propose an approximation algorithm for finding the TSP tour over G'', called *Christofides for SPP* (CSPP), with $O(n^3)$ time complexity and approximation bound of 2.

4.1 CSPP: the Procedure and Cost Analysis

Algorithm 1 is a pseudocode for CSPP, which takes G'' (which has $3n$ nodes and $9n^2$ edges) as the input and returns a Hamiltonian tour called h-*trail*. This algorithm consists of the following five steps:

Step 1: Finding the Minimum Spanning Tree
The first step finds the minimum spanning tree (MST) of G''. This step is the same as the first step of the Christofides algorithm whose time complexity would be in $O(n^2)$ by using either Kruskal [Kruskal, 1956] or Prim [Prim, 1957] algorithms.

Step 2: Modify MST by Adding Subpath Edges
The MST does not include any edges with infinite weight. As a consequence, any middle node m_i in G'' can be either a two-degree node in the MST connected to s_i and e_i or a leaf node in the MST connected to s_i or e_i. In this step, for each leaf middle node m_i (suppose s_i is the parent of m_i), the edge m_ie_i is added to the MST to form a graph, denoted by G^*. In G^*, the degree of any middle node is even. The computational cost of this step is of $O(n)$ using Fleury algorithm [Pemmaraju and Skiena, 2003].

Step 3: Perfect Matching over Odd-degree Nodes
This step is identical to the step of the Christofides algorithm that performs the *minimum perfect matching*. Similarly, CSPP performs minimum perfect matching in G'' between all odd-degree nodes in G^* and adds the edges involved in perfect matching to G^* to construct a graph denoted by $\hat{G}$. Since there is no odd-degree middle node in G^*, no middle node is involved in perfect matching. As a result, no edge with infinite weight is added to G^*. Thus, $\hat{G}$ still does not include any infinite edges. This step can be done in $O(n^3)$ using [Micali and Vazirani, 1980].

Step 4: Finding an Eulerian Tour Over $\hat{G}$
The output of Step 3 ($\hat{G}$), is an Eulerian graph. The current step (similar to Christofides) finds an Eulerian tour in $\hat{G}$. We call the output of this step $trail$. The computational cost of this step is of $O(n)$ using [Pemmaraju and Skiena, 2003].

Step 5: Confined Shortcut on *trail*
It is possible for an Eulerian tour (such as *trail*) to visit some nodes more than once. In order to turn an Eulerian tour into a Hamiltonian tour, the extra occurrences of nodes have to be removed. An operation, called *shortcut*, is used in Christofides algorithm to do such a transformation. However, G'' contains some infinite edges and so it is not feasible to do shortcuts like in the Christofides. To address this problem, we introduce a new operation, called *confined shortcut*. The procedure of this operation is discussed in the following.

Consider a node such as v that is visited more than once in *trail*. Let w and u denote the predecessor and successor in one of v's occurrences in the tour, respectively. In this case, it is feasible to add an edge uw to the tour and remove the edges uv and vw to decrease the number of occurrences of v by one. We keep doing this process until the number of occurrences of v in the tour is equal one. If one of the nodes u and w is a middle node, then the weight of the edge uw is infinite and performing the confined shortcut operation would result in adding an infinite edge to the tour. To resolve this problem, we avoid performing the confined shortcut operation over u-v-w, instead, confined shortcut is performed over other occurrences of v. Lemma 1 shows that performing confined shortcut over at most one of the occurrences of v in *trail* may lead to adding an infinite edge. Thus, in order to build a finite Hamiltonian tour, confined shortcuts can be performed over other occurrences to avoid adding infinite weight edges. In fact, differences between shortcut in Christofides and confined shortcut are in: 1) performing confined shortcuts in only three consecutive nodes in the tour; 2) avoiding performing confined shortcut whenever it leads to adding an infinite weight. The computational cost of this step is also of $O(n)$.

Lemma 1 *For each node v in* trail*, doing confined shortcut adds an edge with infinite weight for at most one of the v's occurrences.*

Proof: Please see [Safilian *et al.*, 2016]. □

According to time complexity of each step, the total complexity of CSPP would be of $O(n^3)$.

4.2 Approximation Bound of CSPP

In this section, we prove that the ratio bound of CSPP is 2. Lemmas 2 and 4 are adopted from [Christofides, 1976].

Lemma 2 *The weight of MST is less than the weight of T^*, where T^* is the optimal TSP tour in G''.*

Let $W(H)$ denote the sum of edge weights of a given graph H. Also, let T^* and PM^* denote the TSP tour in G'' and edges of perfect matching in step 3, respectively.

Lemma 3 *The total added weight to MST during the second step is less than the half of the weight of T^*.*

Proof: During Step 2, for each middle leaf node m_i in MST, $w(e_i, m_i)$ is added to the weight of MST. Therefore, $W(E_2) \leq \sum_{i=1}^{n} w(e_i, m_i) = \frac{1}{2}\sum_{i=1}^{n} w(s_i, m_i) + w(e_i, m_i)$, where $W(E_2)$ is the total weights of added edges to MST during this step. T^* does not include any infinite edge. Thus, for each i-th subpath, T^* includes the edges s_im_i and e_im_i, which implies $W(T^*) > \sum_{i=1}^{n} w(s_i, m_i) + w(e_i, m_i)$. Therefore, the inequality $W(E_2) < 0.5W(T^*)$ holds. This concludes the proof. □

Lemma 4 *PM^*'s weight is less than half of T^*'s.*

Theorem 4 *The ratio bound of CSPP is 2.*

Proof: The graph $\hat{G}$ is composed of the edges of the MST and the edges added in steps 2 and 3 (perfect matching). According to Lemmas 2, 3 and 4,

$$W(\hat{G}) = W(\text{MST}) + W(E_2) + W(PM^*) \leq 2W(T^*), \tag{12}$$

where $\hat{G}$ denotes the graph generated in step 3 of CSPP, MST denotes the output of step 1, E_2 denotes the set of edges added to MST in step 2, and PM^* denotes edges of perfect matching in step 3. Thus, the length of *trail* over $\hat{G}$ (found in step 4) is less than 2 times of the optimal TSP tour.

$trail$ does not contain any edge with infinite weight because there is no such edge in $\hat{G}$. Moreover, based on Lemma 1, performing shortcuts during step 5 does not add

Algorithm 1 CSPP

1: Find the minimum spanning tree, MST
2: **for all** subpaths such as i **do**
3: **if** m_i is a leaf node and s_i (or e_i) is the parent of m_i **then**
4: Add edge $m_i e_i$ (or $m_i s_i$)
5: **end if**
6: **end for**
7: Do perfect matching over odd-degree nodes in G^*
8: Add perfect matching edges to G^* and construct $\hat{G}$
9: Find Eulerian tour, $trail$, in $\hat{G}$
10: *h-trail* $\leftarrow$ *trail*
11: Make $visit$ zero for all nodes in G''
12: **while** has not reached the end of *h-trail* **do**
13: select the next node in t in forward order
14: $visit(t) \leftarrow visit(t) + 1$
15: **if** $visit(t) \geq 2$ **then**
16: $t_0 \leftarrow$ the node appears before t in *h-trail*
17: $t_1 \leftarrow$ the node appears after t in *h-trail*
18: **if** $w(t_0, t_1)$ is finite **then**
19: Delete node t
20: $visit(t) \leftarrow visit(t) - 1$
21: **end if**
22: **end if**
23: **end while**
24: **while** has not reached the end of *h-trail* **do**
25: Select the next node t in *h-trail* forward order
26: **if** $visit(t) == 2$ **then**
27: Delete node t
28: $visit(t) \leftarrow visit(t) - 1$
29: **end if**
30: **end while**
31: **return** *h-trail*

any infinite edge. Accordingly, the triangles used in performing confined shortcut (each confined shortcut execution replaces two edges of tour corresponding to a triangle with the third edge of the triangle) do not include any infinite edge. Thus, based on Theorem 2, triangle inequality is not violated in these triangles and in each execution of confined shortcuts (in step 5), the length of the tour decreases. It means that the length of the Hamiltonian tour, *h-trail*, returned by CSPP is less than the length of *trail*:

$$W(\textit{h-trail}) \leq W(\textit{trail}) = W(\hat{G}) \leq 2W(T^*). \qquad (13)$$

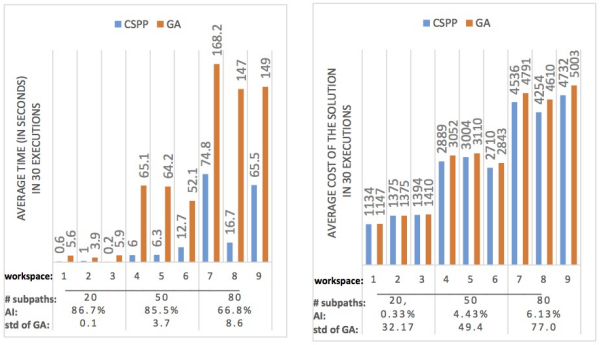
Hence, the solution produced by CSPP is within 2 of optimum and the ratio bound of CSPP is 2. □

5 Experiments and Results

In this section, we empirically gauge the performance of CSPP by comparing it with the GA-based method proposed in [Gyorfi *et al.*, 2010]. Throughout this section, we refer to this method by GA. In our experiments, we use different sets of workspaces including 3 environments with 20 subpaths, 3 environments with 50 subpaths and 3 environments with 80 subpaths. The length and the location of the subpaths are chosen randomly.

Parameter setting for the operation rates of GA are chosen to be 0.5 for crossover, 0.25 for inversion, 0.25 for rotation, 0.5 for mutation and 0.5 for subpath reversal. These parameters are fixed during all experiments and seem to be near optimal set of parameters, according to the results of different parameter settings.

The bar charts in Fig. 2 show the average results of 30 executions (per environment) of CSPP and GA. Fig. 2 also shows the average execution time, average cost (length) of



(a) Execution time (b) Cost of the solution

Figure 2: The execution time and cost of the returned solution for GA and CSPP. AI: Average Improvement (over 90 executions in 3 environments) of CSPP in comparison with GA, std of GA: average of standard deviation (over 90 executions in 3 environments)

the returned solution, average standard deviations of execution time and cost of the solution for GA, and the average improvement (AI) of the CSPP in comparison with GA. CSPP is a deterministic algorithm therefore the variance of its execution time and variance of solution cost are zero for all experiments and thus not reported. According to this figure, CSPP improves the results both in terms of the execution time and cost of the solution.

Comparing the outputs of the two algorithms, we get the following results:

1. Increasing the number of subpaths causes large deviations in the output of the GA technique. This is often not appealing since one would require multiple executions of GA to avoid sub-optimal results.
2. Fig. 2 indicates that the CSPP algorithm can find better solutions than GA with significantly less computation. More importantly, the differences between two algorithms are intensified for larger graphs, making CSPP a more reliable technique for large-scale problems.
3. The CSPP algorithm is a faster alternative than GA algorithm for solving SPP in all experiments.

Our experimental evaluations clearly show the superiority of CSPP over GA for different synthesized environments.

6 Conclusion and Future Work

We presented a 2-approximation algorithm to solve SPP. We addressed the deficiencies of existing meta-heuristic methods for solving SPP by designing a fixed-ratio bound approximation algorithm. Both theoretical and empirical analyses show superiority of CSPP over other existing solutions. We plan to use IETI algorithm with a modified version of RPP [Eiselt *et al.*, 1995a] algorithms to enhance the ratio bound.

Acknowledgment

We are grateful to anonymous reviewers for their useful comments.

References

[Arora, 1996] Sanjeev Arora. Polynomial time approximation schemes for euclidean tsp and other geometric problems. *Foundations of Computer Science*, pages 2–11, 1996.

[Christofides, 1976] N. Christofides. Worst-Case analysis of a new heuristic for the travelling salesman problem. Technical report, DTIC Document, 1976.

[Coja-Oghlan *et al.*, 2006] Amin Coja-Oghlan, Sven O Krumke, and Till Nierhoff. A heuristic for the stacker crane problem on trees which is almost surely exact. *Journal of Algorithms*, 61(1):1–19, 2006.

[Dumitrescu and Mitchell, 2001] Adrian Dumitrescu and Joseph SB Mitchell. Approximation algorithms for tsp with neighborhoods in the plane. *Annual ACM-SIAM symposium on Discrete algorithms*, pages 38–46, 2001.

[Eiselt *et al.*, 1995a] Horst A Eiselt, Michel Gendreau, and Gilbert Laporte. Arc routing problems, part i: The chinese postman problem. *Operations Research*, 43(2):231–242, 1995.

[Eiselt *et al.*, 1995b] Horst A Eiselt, Michel Gendreau, and Gilbert Laporte. Arc routing problems, part ii: The rural postman problem. *Operations Research*, 43(3):399–414, 1995.

[Frederickson, 1979] Greg N Frederickson. Approximation algorithms for some postman problems. *Journal of the ACM (JACM)*, 26(3):538–554, 1979.

[Goldberg, 1989] D.E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-wesley, 1989.

[Gyorfi *et al.*, 2010] Julius S Gyorfi, Daniel R Gamota, Swee Mean Mok, John B Szczech, Mansour Toloo, and Jie Zhang. Evolutionary path planning with subpath constraints. *Electronics Packaging Manufacturing, IEEE Transactions on*, 33(2):143–151, 2010.

[Jaillet and Porta, 2013] Léonard Jaillet and Josep M Porta. Path planning under kinematic constraints by rapidly exploring manifolds. *Robotics, IEEE Transactions on*, 29(1):105–117, 2013.

[Kapadia *et al.*, 2013] Mubbasir Kapadia, Kai Ninomiya, Alexander Shoulson, Francisco Garcia, and Norman Badler. Constraint-aware navigation in dynamic environments. *Proceedings of Motion on Games*, pages 111–120, 2013.

[Kruskal, 1956] J.B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48—50, 1956.

[Lin and Kernighan, 1973] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[Micali and Vazirani, 1980] Silvio Micali and Vijay V Vazirani. An $O(\sqrt{|V|}.|E|)$ algoithm for finding maximum matching in general graphs. *Foundations of Computer Science*, pages 17–27, 1980.

[Nash *et al.*, 2009] Alex Nash, Sven Koenig, and Maxim Likhachev. Incremental phi*: Incremental any-angle path planning on grids. *International Jont Conference on Artifical Intelligence*, 2009.

[Orloff, 1974] CS Orloff. A fundamental problem in vehicle routing. *Networks*, 4(1):35–64, 1974.

[Papadimitriou, 1977] Christos H Papadimitriou. The euclidean travelling salesman problem is np-complete. *Theoretical Computer Science*, 4(3):237–244, 1977.

[Pemmaraju and Skiena, 2003] S.V. Pemmaraju and S.S. Skiena. *Computational discrete mathematics: combinatorics and graph theory with Mathematica*. Cambridge Univ Pr, 2003.

[Prim, 1957] R.C. Prim. Shortest connection networks and some generalizations. *Bell system technical journal*, 36(6):1389—1401, 1957.

[Safilian *et al.*, 2016] Masoud Safilian, S Mehdi Tashakkori, Sepehr Eghbali, and Aliakbar Safilian. An approximation approach for solving the subpath planning problem. *arXiv preprint arXiv:1603.06217*, 2016.

[Safra and Schwartz, 2006] Shmuel Safra and Oded Schwartz. On the complexity of approximating tsp with neighborhoods and related problems. *computational complexity*, 14(4):281–307, 2006.

[Surynek, 2015] Pavel Surynek. Reduced time-expansion graphs and goal decomposition for solving cooperative path finding sub-optimally. *International Conference on Artificial Intelligence*, pages 1916–1922, 2015.

[Tong-ying *et al.*, 2004] G. Tong-ying, Q. Dao-kui, and D. Zai-li. Research of path planning for polishing robot based on improved genetic algorithm. *International Conference on Robotics and Biomimetics*, pages 334—338, 2004.