Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2016, Article ID 5040513, 8 pages
http://dx.doi.org/10.1155/2016/5040513

Research Article

Exact and Heuristic Algorithms for Routing AGV on Path with Precedence Constraints

Liang Xu,[1] Yao Wang,[1] Lin Liu,[1] and Jiaxing Wang[2]

[1]*School of Business Administration, The Southwestern University of Finance and Economics, Chengdu 611130, China*
[2]*Department of Economics Mathematics, The Southwestern University of Finance and Economics, Chengdu 611130, China*

Correspondence should be addressed to Yao Wang; ywangswu@163.com

Received 4 January 2016; Revised 19 April 2016; Accepted 28 June 2016

Academic Editor: Fei Liu

A new problem arises when an automated guided vehicle (AGV) is dispatched to visit a set of customers, which are usually located along a fixed wire transmitting signal to navigate the AGV. An optimal visiting sequence is desired with the objective of minimizing the total travelling distance (or time). When precedence constraints are restricted on customers, the problem is referred to as traveling salesman problem on path with precedence constraints (TSPP-PC). Whether or not it is NP-complete has no answer in the literature. In this paper, we design dynamic programming for the TSPP-PC, which is the first polynomial-time exact algorithm when the number of precedence constraints is a constant. For the problem with number of precedence constraints, part of the input can be arbitrarily large, so we provide an efficient heuristic based on the exact algorithm.

1. Introduction

A traveling salesman problem on path (TSPP for short) arises in dispatching an automated guided vehicle (AGV) to visit a set of locations along the wire which transmits signal to navigate the AGV. For the wired AGV, a slot is cut into the floor where a wire is placed, and the slot is cut along the path that the AGV is to follow. A sensor installed on bottom of the AGV detects the radio signal from the wire to navigate the AGV. The areas where AGVs are applied have increased significantly. By survey of Vis [1], typical applications of AGVs include manufacturing, distribution, and transshipment. Other contexts for TSPP can also be found in the applications of other linear service networks, such as laser drill in manufacturing, tracked crane in container terminal, and school bus shuttling in transportation.

We then illustrate the TSPP by the example in Figure 1.

In this example, the AGV is travelling along the wire (in grey) to shuttle materials between the two conveyors to the four workstations. It has in total four workstations to be visited, where conveyor 1 is the start point of salesman. We denote the start point as 0 and denote the four workstations as 1, 2, 3, and 4, respectively. There are two precedence constraints restricted on the four customers as follows:

$$\begin{aligned} P_1&: 0 \longrightarrow 3 \longrightarrow 1, \\ P_2&: 0 \longrightarrow 4 \longrightarrow 2, \end{aligned} \tag{1}$$

which means that customer 3 should be visited before customer 1 and customer 4 should be visited before customer 2.

Imbedded by the special underlying network, TSPP is a special case of the classic traveling salesman problem. Without any constraints, TSPP can be solved trivially. Though there are a deluge of researches contributing to TSP and its varieties, there is no, as far as we know, literature dealing with TSPP-PC directly. Although TSP with precedence constraints, as a special case of TSP, is known to be NP-complete (Garey and Johnson [2]), whether or not TSPP-PC is NP-complete is unknown in literature. In this paper, we design dynamic programming for the TSPP-PC, which is the first polynomial-time exact algorithm when the number of precedence constraints is a constant. For the problem with number of precedence constraints, part of the input can be

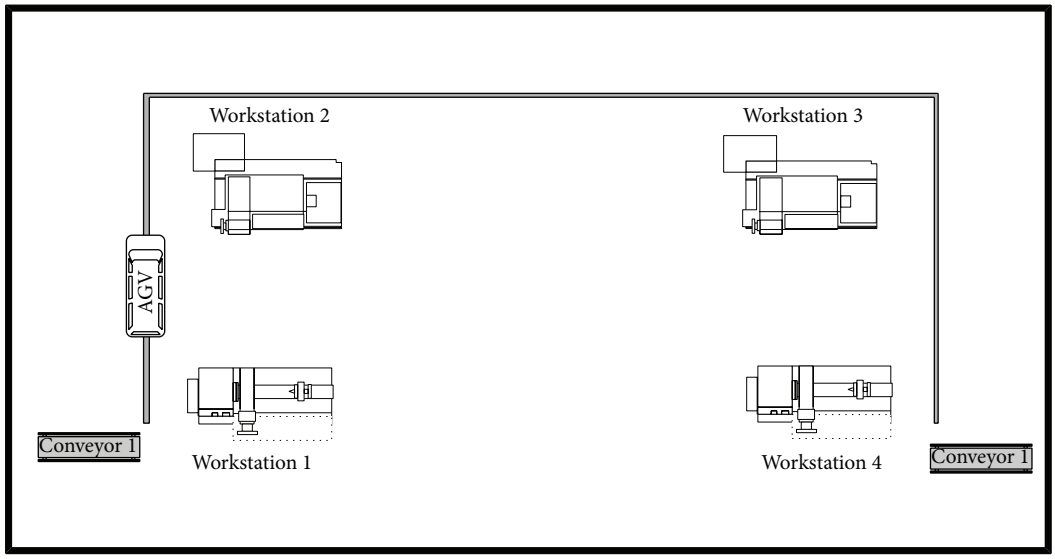

Figure 1: Example to illustrate the TSPP.

arbitrarily large, so we provide an efficient heuristic based on the exact algorithm.

The exact algorithm is based on dynamic programming, which takes into consideration the last vertices in all precedence constraints visited by the salesman rather than all the vertices visited. Since the number of precedence constraints is a constant k, then the dynamic programming is expected to achieve a polynomial-time running time.

This paper is organized as follows: Section 2 is the literature review for the related research and Section 3 introduces the formal problem definition. The dynamic programming for TSPP-PC is provided in Section 4, and the dynamic programming can be proved to be the first polynomial-time exact algorithm for TSPP-PC with number of precedence constraints being a constant. In Section 5, we study TSPP-PC with arbitrary large precedence constraints and develop a heuristic based on the exact algorithm. We have a conclusion and give suggestions for the future work in Section 6.

2. Literature Review

TSP is intensively famous and well-studied in the literature [3], which includes two main streams of varieties: time windows (Foccaci et al. [4]) and precedence constraints (Mingozzi et al. [5]). As we know, it is a NP-hard problem; hence there is no general polynomial-time exact algorithm. However, many exact but not polynomial-time algorithms are developed to solve TSP. One may refer to Lawler et al. [6] to take a glance at general TSP solution approaches.

Though TSP is not polynomial-time solvable unless NP = P, there are many special cases which are well solvable. Lawler et al. [6] and Burkard et al. [7] provide complete surveys for the well solvable special cases of TSP. The most important special case of solvable TSP is so-called pyramidally solvable TSPs. The pyramidal tours have two nice properties. First, finding a minimum cost pyramidal tour takes only $O(n^2)$ time, in terms of computation complexity. Second, there exist certain structures of distance matrix which guarantee the existence of a pyramidal optimal tour. Up to now, this distance matrix structure has been found to guarantee the shortest pyramidal tour: Monge matrices, Supnick matrices, Demidenko matrices, Kalmanson matrices, Van der Veen matrices, and generalized distribution matrices. All of these special matrices impose certain conditions on the distance matrices and, therefore, guarantee the existence of pyramidal optimal tour.

Though the general Euclidean TSP is still NP-hard (Sanjeev [8]), there are some cases that are well solvable. One of them, which is similar to TSPP, is k-line TSP introduced in Deĭneko and Woeginger [9], where the vertices lie on k parallel (or almost parallel) lines in the Euclidean plane. TSPP is a special case of k-line Euclidean TSP with $k = 1$. However, the previous research on this scope does not incorporate the considerations of precedence constraints.

The permuted Monge TSP introduced in Burkard et al. [10] also forms an extensive large class of solvable TSPs. A matrix C is called a permuted Monge matrix if there exists a permutation ϕ such that C_ϕ is a Monge matrix. Similar to Euclidean TSP, although the general TSP on permuted Monge matrices is not polynomial solvable, a number of specific forms of matrices provide the possibility of polynomial solving algorithm.

For the TSP, some works contribute to the literature by providing approximation algorithms rather than exact algorithms. Based on a tree algorithm which is used to provide an approximated TSP tour, Rathinam et al. [11] have developed a 2-approximation algorithm for multiple traveling salesmen problem. Later, Xu et al. [12] improved this result to $(2 - 1/k)$ through providing extended Christofides heuristics for this problem. Later, Xu and Rodrigues [13] have developed a 3/2-approximation algorithm called Exchange Algorithm for the same problem, which achieves the best approximation ratio unless the 3/2 Christofides heuristics for the TSP can be further improved.

Mingozzi et al. [5] proposed a dynamic programming strategy for the TSP with time windows and precedence constraints, and a nice bounding function is adopted to reduce the state space graph. In the dynamic programming of Mingozzi et al. [5], its state is defined as (S, τ, j), which can be interpreted as the family of all possible paths from the start point, visiting all customers in the customer subset S, stopping at customer j, and finishing visiting at time τ. Since the set S clearly has an exponential size to the problem scale, the dynamic programming cannot be expected to achieve a polynomial-time running time to return an optimal solution to the TSPP-PC. For TSP with precedence constraints, Moon et al. [14] presented an efficient genetic algorithm, in which a topological sort is utilized, and a new crossover operation is developed for genetic algorithm.

Though the TSP with precedence constraints has been studied in the literature, we still need to bridge the gaps for TSPP-PC. The first gap lies in lacking of an answer to the open question of whether or not the problem is NP-complete. The second gap is the absence of an efficient heuristic for the AGV industry.

3. Problem Definition

In TSPP-PC, a salesman is supposed to visit a set of vertices, which are located along a path. The distance between two vertices is measured by the Euclidean distance on the path. The objective of the TSPP-PC is to determine the optimal sequence of the vertices to be visited with the total traveling distance minimized and without violating the given precedence constraints.

Formally, we can represent the TSPP-PC as follows. Suppose the salesman is located at 0, so-called start point, originally, and he is expected to visit a set of vertices along a path, denoted by $V = \{1, 2, \ldots, n\}$. For $i = 1, 2, \ldots, n$, we use x_i to denote the coordinate of vertex i and use X to denote $\{x_1, x_2, \ldots, x_n\}$. Hence, the distance between i and j, denoted by d_{ij}, is determined by

$$d_{ij} = \left| x_i - x_j \right|. \tag{2}$$

The objective of the problem is to determine a sequence $(i_1, i_2, \ldots, i_n)$ of vertices, such that the total travel distance D where

$$D = d_{0,i_1} + d_{i_1,i_2} + \cdots + d_{i_n-1,i_n} + d_{i_n,0} \tag{3}$$

is minimized, without violating the precedence constraints imposed. That is, the salesman starts from the start point, visits all the vertices one by one as the sequence, and finally returns to the start point, subject to the precedence constraints. Let P denote the collection of all the k precedence constraints, P_i for $i = 1, 2, \ldots, k$. Strictly,

$$P = \left(P_1, P_2, \ldots, P_k\right), \tag{4}$$

where P_i is denoted as

$$\begin{array}{lccccc} \text{vertex} & v_{i,0} & v_{i,1} & v_{i,2} & \cdots & v_{i,l_i} \\ \text{priority} & 0 & 1 & 2 & \cdots & l_i. \end{array} \tag{5}$$

We have $v_{i,0} = 0$ for $i = 1, 2, \ldots, k$, which means that each precedence constraint must begin from vertex 0. For any single precedence P_i with $l_i + 1$ vertices, it is noted that $v_{i,j}$ has the jth priority. The vertex $v_{i,j}$ can be visited if all vertices $v_{i,t}$ with higher priority ($t < j$) have been visited. For simplicity of notation, we denote the precedence P_i as $(v_{i,0}, v_{i,1}, \ldots, v_{i,l_i})$. Taking P_2 in the first section as an example, $P_2 = (v_{2,0}, v_{2,1}, v_{2,2}) = (0, 4, 2)$, where $l_2 = 2$ and vertex 0 must be visited before vertex 4 and vertex 2 can be visited unless vertex 4 has been visited. Thus, any instance of TSPP-PC can be described as (V, X, P).

One can notice that, for instance (V, X, P) to the TSPP-PC, in a single precedence constraint, a vertex v appears exactly once; otherwise, multiple v in a single precedence constraint leads to a contradiction because any vertex u in between consecutive v's should be visited both after v and before v. Furthermore, if more than one precedence constraint shares a common vertex v, the instance (V, X, P) may have no feasible solution because two precedence constraints share common vertices that may be contradictive. For example, the two precedence constraints $P_1 = (0, 2, 1, 3)$ and $P_2 = (0, 1, 4, 5, 2)$ share two vertices 1 and 2, and the two constraints are contradictive because 2 should be visited before 1 in P_1 and 2 should be visited after 1 in P_2, which implies there exists no feasible solution for this instance. To guarantee existence of feasible solution, we consider only those instances that all precedence constraints in P share no common vertex, which implies that each vertex appears in all precedence constraints in P at most once. For precedence P_i where $i = 1, 2, \ldots, k$, any vertex v can be visited if and only if all vertices in P_i before v have been visited.

In this paper, we first consider the number of precedence constraints, k, is independent of the instance input and design the polynomial-time exact algorithm with time complexity $O(kn^{k+1})$ based on dynamic programming. Since the exact algorithm has an exponential running time when k is part of the problem instance and can be arbitrarily large, we design a heuristic algorithm based on the exact algorithm for the TSPP-PC with two precedence constraints.

Next, we show as follows that, for any instance of the TSPP-PC, finding an optimal visiting sequence in P suffices an optimal visiting sequence in V. Let x_R denote the coordinate which is the biggest among all the vertices in P. That is, $x_R = \arg\max\{x_i \mid i \in P\}$. We further define the set of vertices that are on the right-hand side of vertex x_R as R; that is, $R = \{i \in V \mid x_i > x_R\}$.

Furthermore, we define the set $L = \{i \in V \setminus P \mid x_i \leq x_R\}$. Clearly, the set of V is decomposed as P, R, and L; that is, $V = P \cup R \cup L$.

We are now in position to present the first property of TSPP-PC to remove the consideration of vertices in R and L.

Property 1. For any instance of TSPP-PC, an optimal solution must be in the form of $(\ldots, x_R, R, \ldots)$ where the solution visits vertices in R one by one after it visits the vertex x_R.

This is true because there is no precedence constraint imposed on the vertices in R and finally to reach the vertex

n, the distance of trip from x_R to x_n and from x_n back to x_R could not be shortened.

Due to Property 1, we can remove the consideration of the vertices in R. After we obtain the optimal visiting sequence in the vertex set which excludes R from V, we insert the vertices in R to establish the final sequence.

Furthermore, we are even able to remove the consideration of vertices in L. That is because when we schedule the vertices in P, to reach the vertex x_R, the salesman would pass by all the vertices in L as the continuity of the traveling. Because the vertices in L have no precedence constraint, the salesman could visit these vertices when passing by them.

Based on the fact above, we can simplify this problem by removing the consideration of the vertices in L and R. After an optimal visiting sequence in P is obtained, we can solve the problem with L and R as shown in Algorithm 1:

Algorithm 1 (transformation of optimal sequence in P to optimal sequence in V).

Input. An optimal visiting sequence in P is denoted as S_P.

Output. An optimal visiting sequence in V is denoted as S_V.

(1) Set $S_V = \langle 0 \rangle$.

(2) Find the sets P, L, and R, and $x_R = \arg\max\{x_i \mid i \in P\}$.

(3) Insert the vertices in L into S_P when the salesman passes by them. Afterwards, we obtain a new sequence S_R.

(4) Insert the vertices in R into S_R after x_R in a nondecreasing order to obtain S_V.

The following theorem states the correctness for Algorithm 1.

Theorem 2. *Given any optimal visiting sequence S_P for P, Algorithm 1 returns an optimal visiting sequence S_V for $V = P \cup L \cup R$.*

Proof. We prove the correctness for Theorem 2 by contradiction. Suppose there exists a visiting sequence S'_V with total distance d, whose travel distance is shorter than S_V. We can construct as follows a shorter visiting sequence S'_P. Since the distance from x_R to the farthest vertex in R is a constant, denoted as d_1, then shortcutting all vertices in R from S'_V leads to a visiting sequence S'_R with distance $d - d_1$. Note that S'_R has a shorter distance than the visiting sequence S_R obtained in Algorithm 1 because S_R has a distance d_1 shorter than S_V. Finally, notice that shortcutting vertices in L from S'_R obtains S'_P with the same travel distance and shortcutting vertices in L from S_R obtains S_P with the same travel distance. One can conclude that the travel distance for S'_P is less than S_P, which contradicts the assumption that S_P is the optimal visiting sequence in P. □

4. Exact Algorithm for TSPP-PC

In this section, we focus on the TSPP-PC with constant precedence constraints. As mentioned before, our task is to find the first polynomial-time exact algorithm for this problem.

Before introducing our dynamic programming for finding the optimal sequence in P, we need to state a new property for an optimal solution which is a basis of our dynamic programming.

Property 2. For any instance of the TSPP-PC, an optimal sequence must visit a new vertex (the vertex that has not been visited in the existing sequence) at each step of the salesman. Thus, an optimal sequence contains each vertex in P exactly once.

This property holds by the fact that, supposing the salesman visits a vertex i more than once in a sequence, we can always shortcut duplicated i to obtain a new sequence with equal or shorter total length.

We use s_i to denote the last vertex that salesman has just visited in the ith precedence, and so we can use $S = (s_1, s_2, \ldots, s_k)$ to denote the set of vertices that the salesman has just visited in the k precedence constraints.

Recall that the k precedence constraints are imposed by

$$\begin{aligned} P_1 &= \left(0, v_{1,1}, v_{1,2}, \ldots, v_{1,l_1}\right), \\ P_2 &= \left(0, v_{2,1}, v_{2,2}, \ldots, v_{2,l_2}\right), \\ &\vdots \\ P_k &= \left(0, v_{k,1}, v_{k,2}, \ldots, v_{k,l_k}\right). \end{aligned} \tag{6}$$

By definition of $S = (s_1, s_2, \ldots, s_k)$, we can conclude that if the salesman has visited s_i in P_i for $i = 1, 2, \ldots, k$, then all the vertices in P_i before s_i have been visited and all the vertices in P_i after s_i have not been visited.

Let (t, S) denote the state that the salesman has just visited $S = (s_1, s_2, \ldots, s_k)$ in all the precedence constraints and stops at the vertex s_t, where $1 \le t \le k$.

Let $f^*(t, S)$ denote the minimum travel time taken by the salesman who starts from the start point and ends up with state (t, S). We use $\Gamma(t, S)$ to denote all those states (t', S') that can reach state (t, S) by exact one step of the salesman. For any vertex v in P with $v \ge 0$, let $\sigma^{-1}(v)$ denote the immediately previous vertex before v in the precedence that contains v. Let $\sigma_t^{-1}(S)$ denote the vector obtained by replacing s_t in S by $\sigma^{-1}(s_t)$. It is noted that, for the state (t, S), since the salesman has visited all the vertices in S and stops at s_t, then before salesman's last step, he must have visited all the vertices in $\sigma_t^{-1}(S)$ and can start from any vertex of $\sigma_t^{-1}(S)$ to reach s_t in the last step. Hence, by definition of $\Gamma(t, S)$ and above analysis, we can see that

$$\Gamma^{-1}(t, S) = \left\{\left(t', S'\right) \mid S' = \sigma_t^{-1}(S),\ 1 \le t' \le k\right\}. \tag{7}$$

With the current state being (t, S), if $s_t \ge 0$ (the vertex s_t has a previous vertex in P_t), before the step which arrives at the state (t, S), the salesman must start with a state $(t', S') \in \Gamma(t, S)$ and arrives s_t via the edge $(s_{t'}, s_t)$. Meanwhile, with the current state being (t, S), if $s_t = 0$, since vertex 0 has been

visited initially, according to Property 2, we do not need to consider this state as a potential for optimal solution, and thus we let $f^*(t,S) = +\infty$ if $s_t = 0$. Then, the recursion function can be

$$f^*(t,S) = \begin{cases} \min\limits_{(t',S')\in\Gamma^{-1}(t,S)} \left\{ f^*\left(t',S'\right) + d\left(s_{t'},s_t\right)\right\}, & s_t > 0; \\ +\infty, & s_t = 0. \end{cases} \quad (8)$$

To facilitate the dynamic programming, we define boundary conditions as follows:

$$f^*\left(t,\underline{S}\right) = 0, \quad \forall t = 1,2,\ldots,k, \quad (9)$$

where $\underline{S} = (0,0,\ldots,0)$ in which the k coordinates are all equal to zero.

Based on the recursion equation, the optimal travel time could be written as

$$\min_{1\le t\le k} \left\{ f^*\left(t,\overline{S}\right) + d\left(\overline{s}_t,0\right)\right\}, \quad (10)$$

where $\overline{S} = (\overline{s}_1,\overline{s}_2,\ldots,\overline{s}_k)$ and $\overline{s}_t$ denotes the last vertex in P_t for $t = 1,2,\ldots,k$.

In summary, based on the dynamic programming, we describe the exact algorithm as in Algorithm 3. The exact algorithm is based on Property 2. Since the salesman visits a new vertex at each step, then it takes $|V|$ steps for the salesman to visit all the vertices (without the step to return to the start point). Meanwhile, it is observed that if it takes i steps ($i > 0$) for the salesman to reach a state (t,S), then the salesman must start from a state in $\Gamma^{-1}(t,S)$ which takes exact $i-1$ steps. Hence, we can iteratively calculate the state values for all states which take the salesman i steps with $i = 1,2,\ldots,|V|$.

Algorithm 3 (exact algorithm for the TSPP-PC).

Input. It is an instance (V,X,P) to the TSPP-PC.

Output. It is an optimal objective value to (V,X,P).

(1) For $t = 1,2,\ldots,k$, calculate $f^*(t,\underline{S})$ by boundary conditions (9).

(2) For $i = 1,2,\ldots,|V|$, do the following:

  (a) Find all the solutions for $\sum_{j=1}^{k} y_j = i$ with $y_j \in \{0,1,\ldots,l_j\}$ and denote the set of all solutions as Y_i.

  (b) For each element $(y_1,y_2,\ldots,y_k) \in Y_i$, calculate $f^*(t,(v_{1,y_1},v_{2,y_2},\ldots,v_{k,y_k}))$ with $1 \le t \le k$ by recursion equation (8), where v_{j,y_j} denote the y_jth vertex (excluding vertex 0) in P_j.

(3) Calculate the optimal objective value to (V,X,P) by (10).

Next, we state the correctness and time complexity of Algorithm 3 in the following theorem.

Theorem 4. *Algorithm 3 returns an optimal solution to the TSPP-PC with a time complexity in $O(kn^{k+1})$.*

Proof. Based on the dynamic programming, to prove the correctness of Algorithm 3, it suffices to prove that, for any state that can be reached by the salesman, Algorithm 3 returns its state value, that is, for $t = 1,2,\ldots,k$ and any $S = (s_1,s_2,\ldots,s_k)$ with $s_i \in P_i$ for $i = 1,2,\ldots,k$, Algorithm 3 returns $f^*(t,S)$.

We prove the correctness by induction. First, for those states that can be reached by 0 step, that is, for $t = 1,2,\ldots,k$ and $\underline{S} = (0,0,\ldots,0)$, we have $f^*(t,\underline{S}) = 0$. For any state that can be reached by the salesman with $i-1$ steps, suppose the state values have been returned by Step (2) of Algorithm 3 when it is repeated for $i-1$. We prove as follows, for any state that can be reached by the salesman with i steps, the state values are returned by Step (2) of Algorithm 3 when it is repeated for i. For any state (t,S) that can be reached with i steps, by definition of Y_i, since the salesman visits one new customer in each step and the total number of customers that have been visited is i, it can be seen that there exists $(y_1,y_2,\ldots,y_k) \in Y_i$ such that $S = (v_{1,y_1},v_{2,y_2},\ldots,v_{k,y_k})$.

According to definition of Y_{i-1}, we know that there exists $(y_1',y_2',\ldots,y_k') \in Y_{i-1}$ such that $\sum_{j=1}^{k} |y_j' - y_j| = 1$. By assumption of the induction, all the state values for (t',S') with $1 \le t' \le k$ and $S' = (v_{1,y_1'},v_{2,y_2'},\ldots,v_{k,y_k'})$ have been returned, which implies that $\Gamma^{-1}(t,S)$ have been returned. Thus, by recursion equation (8), the state value $f^*(t,S)$ can be returned.

We analyze the computation complexity of this exact algorithm as follows. The number of precedence constraints is k. In Algorithm 3, each iteration of Step (2) needs to calculate state value $f^*(t,S)$ with $t = 1,2,\ldots,k$ and $S = (s_1,s_2,\ldots,s_k)$ where $s_i \in P_i$ for $i = 1,2,\ldots,k$, which has a running time in $O(kn^k)$ ($l_i \le n$ for $i = 1,2,\ldots,k$). Then, totally, the computation effort of Algorithm 3 is in $O(kn^{k+1})$, which has a polynomial running time only when k is a constant independent of the input size. □

Example 5. Let us use the example in Section 1 to illustrate the exact algorithm for TSPP-PC.

First, by boundary conditions in (9), in Step (1) Algorithm 3, when $t = 1$ and $t = 2$ we have

$$f^*(1,(0,0)) = 0, \qquad f^*(2,(0,0)) = 0. \quad (11)$$

Then, in Step (2)(a), when $i = 1$, we have

$$Y_1 = \{(0,1),(1,0)\}. \quad (12)$$

For $(y_1,y_2) = (0,1) \in Y_1$ and $t = 1$, we have $v_{1,y_1} = 0$ and $v_{2,y_2} = 4$. Then, by recursion equation (8) with $S = (s_1,s_2) = (0,4)$ and $t = 1$, since $s_1 = 0$, we have

$$f^*(1,(0,4)) = +\infty. \quad (13)$$

For $(y_1, y_2) = (0,1) \in Y_1$ and $t = 2$, we have $v_{1,y_1} = 0$ and $v_{2,y_2} = 4$. Then, by recursion equation (8) with $S = (s_1, s_2) = (0,4)$ and $t = 2$, since $s_2 > 0$, we have

$$f^*(2,(0,4)) = \min\{f^*(1,(0,0)) + d(0,4), f^*(2,(0,0)+d(0,4))\} = 4. \quad (14)$$

Similarly, for $(y_1, y_2) = (1,0) \in Y_1$, we have

$$f^*(1,(3,0)) = 3, \quad f^*(2,(3,0)) = +\infty. \quad (15)$$

When Step (2) is repeated for $i = 2$, we have

$$Y_2 = \{(0,2),(1,1),(2,0)\}. \quad (16)$$

For $(y_1, y_2) = (0,2)$, we can obtain that

$$f^*(1,(0,2)) = +\infty, \quad f^*(2,(0,2)) = 6. \quad (17)$$

For $(y_1, y_2) = (1,1)$, we can obtain that

$$f^*(1,(3,4)) = 5, \quad f^*(2,(3,4)) = 4. \quad (18)$$

For $(y_1, y_2) = (2,0)$, we can obtain that

$$f^*(1,(1,0)) = 5, \quad f^*(2,(1,0)) = +\infty. \quad (19)$$

When Step (2) is repeated for $i = 3$, we have

$$Y_3 = \{(1,2),(2,1)\}. \quad (20)$$

For $(y_1, y_2) = (1,2)$, we can obtain that

$$f^*(1,(3,2)) = 7, \quad f^*(2,(3,2)) = 6. \quad (21)$$

For $(y_1, y_2) = (2,1)$, we can obtain that

$$f^*(1,(1,4)) = 7, \quad f^*(2,(1,4)) = 8. \quad (22)$$

When Step (2) is repeated for $i = 4$, we have

$$Y_4 = \{(2,2)\}. \quad (23)$$

For $(y_1, y_2) = (2,2)$, we can obtain that

$$f^*(1,(1,2)) = 7, \quad f^*(2,(1,2)) = 8. \quad (24)$$

Thus, in Step (3), according to (10), we have that the optimal objective value equals

$$\min\{f^*(1,(1,2)) + d(1,0), f^*(2,(1,2)) + d(2,0)\} = 8. \quad (25)$$

5. Heuristic Algorithm for the TSPP-PC

Based on the dynamic programming for the TSPP-PC, we can develop the first exact algorithm for this problem. As shown in Theorem 4, the dynamic programming has a time complexity of $O(kn^{k+1})$, which is a polynomial running time when k is a constant independent of the problem instance. The constant k precedence number commonly applies to the case when the AGV has k loading positions. For instance, when an AGV has k loading positions, it has a capacity of loading at most k items simultaneously, and thus it only needs to consider k precedence constraints where k is a constant.

In comparison, for this AGV with no fixed loading positions, its loading capacity depends on the sizes of different objects, and thus the number of precedence constraints is part of the instance input. When k is part of the problem instance and can be arbitrarily large, the dynamic programming has an exponential running time. Hence, we need to develop an efficient heuristic for the TSPP-PC with arbitrarily large k.

Although the dynamic programming has an exponential $O(kn^{k+1})$ running time when k is part of the problem instance, it is still quite efficient when applied to the instance with $k = 2$ (with a running time of $O(n^3)$). Based on this observation, our heuristic for the TSPP-PC is based on an iterative insertion of vertices in one precedence constraint and applies the dynamic programming for TSPP-PC with $k = 2$ to the current visiting sequence for the salesman (which can be taken as the first precedence) and the precedence that is to be inserted (which is the second precedence) to find an optimal visiting sequence for the TSPP-PC with two precedence constraints. The details of the heuristic algorithm can be described in Algorithm 6.

Algorithm 6 (heuristic algorithm for the TSPP-PC).

Input. It is an instance (V, X, P) to the TSPP-PC.

Output. It is a feasible solution S to (V, X, P).

(1) Let a visiting sequence $S = \langle 0 \rangle$. Do the following until $P = \emptyset$.

  (a) Suppose P currently has m precedence constraints, denoted as $P_1, P_2, \ldots, P_m$. Find the precedence P_{j^*} in P such that the first vertex in P_{j^*} has a closest distance with last vertex in $V(S)$.

  (b) Apply Algorithm 3 to obtain an optimal solution S' to the TSPP-PC with two precedence constraints S and P_{j^*}.

  (c) Let $S = S'$ and $P = P \setminus P_{j^*}$.

(2) Return S.

To test the efficiency of the heuristic algorithm for the TSPP-PC, we conduct several numerical experiments for different scale of problem instances. In the first set of numerical experiments, we focus on testing the average performance of the heuristic over the greedy insertion algorithm that is usually applied by the industry.

In the subsequent numerical experiments, we use the following settings. In total 100 problem instances of different

Table 1: Average performance: Algorithm 6 versus greedy insertion.

(a)

Group	Obj_GRE	Time_GRE (s)	Obj_HEU	Time_HEU (s)	Improvement (%)
1	677	0.04	640	0.035	5.7
2	816	0.019	739	0.15	10.5
3	1030	0.02	744	0.61	38.3
4	1363	0.02	926	7.2	47.2
5	1374	0.18	853	10.4	61.1
6	1564	0.25	885	31.3	76.6
7	1715	0.26	957	47	79.1
8	1915	0.28	889	55.4	115.5
9	2228	0.32	947	135	135.9
10	2309	0.48	976	215	136.5

(b)

Group	Obj_GRE	Time_GRE (s)	Obj_HEU	Time_HEU (s)	Improvement (%)
1	677	0.04	640	0.04	5.7
2	816	0.019	732	0.16	11.5
3	1030	0.02	749	0.62	37.5
4	1363	0.02	910	6.34	49.7
5	1374	0.18	862	8.85	59.4
6	1564	0.25	862	21	81.4
7	1715	0.26	924	42	85.4
8	1915	0.28	905	91	111.5
9	2228	0.32	977	122	127.9
10	2309	0.48	993	142	132.5

scales are generated. The problem instances contain only customers in the precedence constraints. Meanwhile, as mentioned before, we consider in the problem instances only those precedence constraints sharing no common customers. The 100 problem instances are divided into 10 groups with the ith group having i precedence constraints for $i = 1, 2, \ldots, 10$. The biggest problem instance has in total 50 customers with 10 precedence constraints. Each customer is randomly generated with the coordination from $\{1, 2, \ldots, 200\}$.

In the following, we briefly introduce the greedy insertion algorithm that is usually applied in the industry. In the greedy insertion algorithm, it starts with the visiting sequence containing only the start point. In each iteration of the greedy insertion algorithm, from the current standing point of the salesman, it finds out the closest available customer that has not been visited in all precedence constraints, adds the customer to the visiting sequence, and removes the customer from the precedence. The greedy insertion algorithm repeatedly inserts customer into visiting sequence until all the customers have been inserted.

Next, we compare the average objective value and running time for each group of the problem instances. In Table 1(a), the second column and third column indicate the objective value and the running time for greedy insertion algorithm, the fourth and fifth column indicate the objective value and the running time for the heuristic algorithm, and the final column indicates the improvement of the objective of the heuristic versus the objective of the greedy insertion algorithm.

From Table 1(a), one can see that, with growth of scale of the problem instance, the heuristic algorithm has a much better average performance than the greedy insertion algorithm. For the problem instances with two precedence constraints, the objective value of the heuristic algorithm and greedy insertion algorithm is relatively close. For the problem instance with 5 precedence constraints (Group 5), the improvement ratio can be seen to be 61.1%. For the problem instance with at most 10 precedence constraints (Group 10), the improvement ratio has been raised up to 136.5%. From the perspective of the running time, when precedence constraints in each problem instance are no more than five, running times of the heuristic algorithm and the greedy insertion algorithm are comparable. With increasing of number of precedence constraints, the running time of the heuristic algorithm rises for two reasons. The first reason is that the iterations of the heuristics go up with the number of precedence constraints and the second reason is that the running time for each iteration goes up with number of the total vertices in the two precedence constraints.

To test the robustness of the heuristic, we also run the 100 problem instances in a different manner which revises Step (1)(a) of Algorithm 6 as inserting the precedence in P with minimum index. We also compare the average performance and running time of revised Algorithm 6 to

the greedy insertion algorithm. In the following table, the second and third column indicate the average objective value and running time for the greedy insertion algorithm, the fourth and fifth column indicate the average objective value and running time for Algorithm 6, and the final column indicates the improvement of the objective of the greedy insertion algorithm versus the objective of the revised algorithm.

From Table 1(b), we can see that, firstly, the running times of Algorithm 6 and revised Algorithm 6 for each group are very close. Moreover, revised Algorithm 6 has a relatively close improvement ratio over the heuristic when compared to the greedy insertion algorithm. In conclusion, the heuristic algorithm based on the exact algorithm is robust with the polices that determines the precedence to be inserted in Step (1)(a) of Algorithm 6.

6. Conclusion

In this note, we considered the TSPP-PC, which is a variety of TSPs. Though TSP is widely studied, whether or not TSPP-PC is polynomial-time solvable remains an open question in the literature. We provide a positive answer to this problem when number of precedence constraints k is part of the input size and develop efficient heuristic for the problem with arbitrarily large k.

Future work is supposed to be taken on these directions. First, we are expecting to prove the NP-hardness for the TSPP-PC with k part of the input size. Second, for the case that different precedence constraints share common vertices, it remains to develop an exact algorithm or to prove its NP-hardness. Third, developing heuristic algorithm with searching part of the state space in the dynamic programming is supposed to be another interesting research direction.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work was supported in part by the National Science Foundation with Grant 71201127 and National Science Foundation with Grant 71201128.

References

[1] I. F. A. Vis, "Survey of research in the design and control of automated guided vehicle systems," *European Journal of Operational Research*, vol. 170, no. 3, pp. 677–709, 2006.

[2] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, WH Freeman and Company, New York, NY, USA, 1979.

[3] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research*, vol. 21, no. 2, pp. 498–516, 1973.

[4] F. Focacci, A. Lodi, and M. Milano, "A hybrid exact algorithm for the TSPTW," *INFORMS Journal on Computing*, vol. 14, no. 4, pp. 403–417, 2002.

[5] A. Mingozzi, L. Bianco, and S. Ricciardelli, "Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints," *Operations Research*, vol. 45, no. 3, pp. 365–377, 1997.

[6] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, John Wiley & Sons, Chichester, UK, 1985.

[7] R. E. Burkard, V. G. Deineko, R. van Dal, J. A. van der Veen, and G. Woeginger, "Well-solvable special cases of the traveling salesman problem: a survey," *SIAM Review*, vol. 40, no. 3, pp. 496–546, 1998.

[8] A. Sanjeev, "Polynomial time approximation schemes for Euclidean TSP and other geometric problems," in *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pp. 2–11, 1996.

[9] V. G. Deĭneko and G. J. Woeginger, "The convex-hull-and-k-line travelling salesman problem," *Information Processing Letters*, vol. 59, no. 6, pp. 295–301, 1996.

[10] R. E. Burkard, V. G. Deineko, and G. J. Woeginger, "The travelling salesman problem on permuted Monge matrices," *Journal of Combinatorial Optimization*, vol. 2, no. 4, pp. 333–350, 1998.

[11] S. Rathinam, R. Sengupta, and S. Darbha, "A resource allocation algorithm for multivehicle systems with nonholonomic constraints," *IEEE Transactions on Automation Science and Engineering*, vol. 4, no. 1, pp. 98–104, 2007.

[12] Z. Xu, L. Xu, and B. Rodrigues, "An analysis of the extended Christofides heuristic for the k-depot TSP," *Operations Research Letters*, vol. 39, no. 3, pp. 218–223, 2011.

[13] Z. Xu and B. Rodrigues, "A 3/2-approximation algorithm for the multiple TSP with a fixed number of depots," *INFORMS Journal on Computing*, vol. 27, no. 4, pp. 636–645, 2015.

[14] C. Moon, J. Kim, G. Choi, and Y. Seo, "An efficient genetic algorithm for the traveling salesman problem with precedence constraints," *European Journal of Operational Research*, vol. 140, no. 3, pp. 606–617, 2002.