Improved Approximation Algorithms for Clustered TSP and Subgroup Planning

Jingyang Zhao, Mingyu Xiao*, Junqiang Peng, Ziliang Xiong

University of Electronic Science and Technology of China
jingyangzhao1020@gmail.com, myxiao@gmail.com, jqpeng0@foxmail.com, forgottencosecant@outlook.com

Abstract

In the Clustered TSP (CTSP), we are given an edge-weighted graph satisfying the triangle inequality property, and a family of pairwise disjoint vertex groups. The goal is to find a minimum weight tour that includes all vertices, ensuring that the vertices within each group appear consecutively on the tour. The subgroup planning problem (SGPP) is an extension of CTSP by relaxing some triangle inequality requirements on edge weights. CTSP and SGPP have plentiful applications in AI and robotics. In this paper, we design three improved approximation algorithms for SGPP and CTSP. First, we propose a polynomial-time 2.167-approximation algorithm for SGPP, improving the previous ratio of 3 (IJCAI 2017). Second, we give an FPT 2.072-approximation algorithm for SGPP parameterized by the maximum group size, improving the previous ratio of 2.5 (IJCAI 2017). Third, we prove an FPT ($\beta < 1.5$)-approximation algorithm for SGPP parameterized by the number of groups, which even improves the previous ratio 1.667 for CTSP (ORL 1999). We also conduct experiments to evaluate the performance of our algorithms.

Introduction

The famous traveling salesman problem (TSP) as well as some other path planning problems, are going to find a route of a vehicle to visit a group of locations under some requirements. These problems play an important role in operations research and have also attracted certain attention in AI and robotics (Nash, Koenig, and Likhachev 2009; Jaillet and Porta 2013; Ninomiya et al. 2015; Surynek 2015).

One application scenario for these path planning problems is that the group of locations can be divided into smaller subgroups based on different properties, and each subgroup has its own rules and requirements for the route. Motivated by warehouse routing and production planning, the Clustered TSP (CTSP) was proposed (Chisman 1975; Lokin 1979). In CTSP, the group of locations is divided into several subgroups, and locations within the same subgroup must be visited consecutively (in any order).

To answer more application scenarios, CTSP was further generalized to the subgroup path planning problem (SGPP) in the literature (Sumita et al. 2017). The goal of SGPP is still to find a minimum weight tour visiting all vertices (locations) such that the vertices in the same subgroup are visited consecutively, as in CTSP. An illustration of SGPP, along with CTSP is shown in Figure 1. However, the edge weight function in SGPP is more general. In CTSP, the distances among any three locations always satisfy the triangle inequality relationship. However, in SGPP, the edge weight function w *partially* satisfies the triangle inequality. Specifically, it requires that $w(a, b) + w(b, c) \geq w(a, c)$ holds only if a, b, c belong to the same subgroup or a, c do not belong to the same subgroup. This property is called the *partial metric*, which was defined to meet the needs of AI applications (Sumita et al. 2017). Note that the targets in different subgroups may be different and then the routing cost (edge weight) in each subgroup may be different. We may even add other expenses caused when executing tasks in each subgroup to the edge weights in the subgroup. Thus, in most application scenarios, the weight of edges within each subgroup may be much higher than that of other edges, which may lead to the partial metric. Although many applications of SGPP and CTSP have been mentioned in the literature (Sumita et al. 2017; Laporte and Palekar 2002), we introduce one more application for SGPP below.

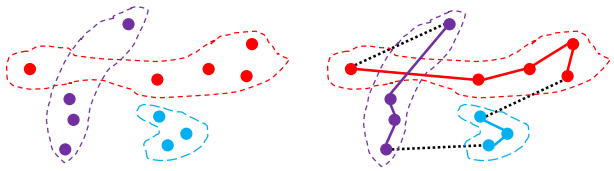

Figure 1: An example of SGPP and CTSP, where (a) represents three subgroups of vertices, and (b) represents a feasible tour

In electronic printing, a robotic arm needs to process various types of holes, such as through-holes, blind vias, and buried vias, which can be considered as distinct subgroups of vertices. To reduce switching costs and minimize error accumulation, the robotic arm must process holes of the same type consecutively. Moreover, different types of holes require varying levels of precision, resulting in higher routing costs within groups containing more complex or precision-

*Corresponding author
Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

demanding holes. Thus, minimizing the total routing cost in this scenario can be modeled as an instance of SGPP.

When each subgroup contains exactly two locations, SGPP reduces to the subpath planning problem (SPP) (Gyorfi et al. 2010; Safilian et al. 2016). SPP is to find a tour passing through several edges. One application is in polishing robots (Sumita et al. 2017). The target is to polish scratches on a surface of a workpiece, which can be regarded as edges that need to be traversed. Another application in electronic printing can be found in (Gyorfi et al. 2010).

In this paper, we focus on approximation algorithms for SGPP and CTSP. For any minimization problem Π, an algorithm is called an α-approximation algorithm if it can produce a solution with an objective value not exceeding α times the optimal value in polynomial time, where α is called the approximation ratio. For a problem Π with a parameter k, an algorithm is called an FPT α-approximation algorithm if it runs in $\mathcal{O}(f(k) \cdot poly(|\Pi|))$ time with an approximation ratio of α, where $f(k)$ is a computable function on k only. FPT algorithms and FPT approximation algorithms are particularly useful in practice when the parameter k has a small value. We highlight that the metric property in CTSP and the partial metric property in SGPP play an important role in designing approximation algorithms. If the edge function is general (with no triangle inequality requirement on any edges), then it may be impossible to design any algorithm with a constant approximation ratio. TSP with a general edge weight function is inapproximable within any constant approximation ratio unless P=NP (Sahni and Gonzalez 1976), which is also a special case of our problem under the general edge weight function.

Related Works

First, we review the results for TSP and Path TSP (PTSP) since PTSP is a related problem that will be frequently used in our algorithms. PTSP requires finding a minimum weight path visiting all vertices between two given end vertices. For TSP, PTSP, and CTSP, we assume the weight function satisfies the triangle inequality.

TSP admits an approximation ratio of 1.5 (Christofides 1976), which was recently improved to $1.5 - 10^{-36}$ (Karlin, Klein, and Gharan 2021). For PTSP, Hoogeveen (1991) proposed a $5/3$-approximation algorithm. This ratio has similarly been improved to about $1.5-10^{-36}$ (Karlin, Klein, and Gharan 2021), as any α-approximation algorithm for TSP can be adapted to yield an $(\alpha+\varepsilon)$-approximation algorithm for PTSP with any fixed constant $\varepsilon > 0$ (Traub, Vygen, and Zenklusen 2022). In what follows, we use β (resp., $f(n)$) to denote the approximation ratio (resp., running time) of the current best algorithm for PTSP.

CTSP has been extensively studied in the literature, such as exact algorithms (Ahmed 2013), approximation algorithms (Guttmann-Beck et al. 2000; Bao and Liu 2012; Kawasaki and Takazawa 2020), local search heuristics (Mestria 2018), and metaheuristics (Ding, Cheng, and He 2007; Zhang et al. 2018; Baniasadi et al. 2020).

In terms of approximation algorithms, Arkin, Hassin, and Klein (1997) proposed the first 3.5-approximation algorithm for CTSP, which was subsequently improved to 2.75 (Guttmann-Beck et al. 2000) and then further to $13/6$ (Bao and Liu 2012). Moreover, some approximation algorithms (Guttmann-Beck et al. 2000; Kawasaki and Takazawa 2020) focus on variants of CTSP, where each group may have a specified starting or ending vertex.

Guttmann-Beck et al. (2000) proposed a β-approximation algorithm with a running time of $\mathcal{O}(n^m \cdot f(n))$. However, this algorithm is not an FPT approximation algorithm with respect to the parameter m. Anily, Bramel, and Hertz (1999) studied the Ordered CTSP (OCTSP), where the order of visiting the groups in the tour is specified as part of the input, and proposed a $5/3$-approximation algorithm with a running time of $\mathcal{O}(n^3)$. Since the order of groups can be enumerated in $\mathcal{O}(m!)$ time, their algorithm directly implies an FPT $5/3$-approximation algorithm for CTSP with a running time of $\mathcal{O}(m! \cdot n^3)$, which remains the best-known FPT approximation algorithm with respect to m to date. Moreover, if there are $m = 3$ groups and one of the group consists of only one vertex, Gendreau, Laporte, and Hertz (1997) proposed a $3/2$-approximation algorithm. For CTSP parameterized by t, the group size, to our knowledge, there is no special study on FPT approximation algorithms.

SGPP is a generalization of CTSP, where we relax the triangle inequality requirements on the edges. Thus, the approximation results for CTSP may not apply to the more general problem SGPP. The best-known approximation algorithms for SGPP were provided by Sumita et al. (2017). They introduced a polynomial-time 3-approximation algorithm for SGPP and an FPT 2.5-approximation algorithm parameterized by the group size t with a running time of $\mathcal{O}(nt \cdot 2^t + n^3)$. The later result also applies to CTSP since CTSP is a special case of SGPP. For polynomial-time approximation ratios, there is a gap between SGPP and CTSP. Whether the gap can be erased by improving the the 3-approximation ratio for SGPP to that for CTSP was asked in some references (Sumita et al. 2017; Kawasaki and Takazawa 2020).

For SPP, the special case of SGPP where each subgroup contains two locations, there is a polynomial-time 1.5-approximation algorithm (Sumita et al. 2017). More approximation algorithms for TSP-related problems can be found in the recent survey (Saller, Koehler, and Karrenbauer 2023).

Our Contributions

In this paper, we study approximation algorithms for SGPP, which also hold for CTSP. Our contributions are as follows.

- For SGPP, we propose a polynomial-time 2.167-approximation algorithm, significantly improving the best-known approximation ratio of 3 (Sumita et al. 2017), which also positively addresses the question asked in (Sumita et al. 2017; Kawasaki and Takazawa 2020).
- For SGPP parameterized by the group size t, we introduce an FPT 2.072-approximation algorithm, which improves upon the previous FPT 2.5-approximation algorithm (Sumita et al. 2017).
- For SGPP parameterized by the number of groups m, we propose an FPT β-approximation algorithm with a running time of $\mathcal{O}(n^2 \cdot f(n) + 2^m \cdot n^4)$, which even improves

	Problems	**Poly. time**	**FPT w.r.t.** t	**FPT w.r.t.** m
Previous results	CTSP	2.167 (Bao and Liu 2012)	-	1.667 in time $\mathcal{O}^*(m!)$ (Anily et al. 1999)
	SGPP	3 (Sumita et al. 2017)	2.5 in time $\mathcal{O}^*(2^t)$ (Sumita et al. 2017)	-
Our results	Both problems	$\mathbf{2.167}$	$\mathbf{2.072}$ in time $\boldsymbol{\mathcal{O}^*(2^t)}$	$\boldsymbol{\beta < 1.5}$ in time $\boldsymbol{\mathcal{O}^*(2^m)}$

Table 1: A summary of previous approximation ratios and our approximation ratios for CTSP and SGPP

the previous FPT 1.667-approximation algorithm with a running time of $\mathcal{O}(m! \cdot n^3)$ for CTSP (Anily, Bramel, and Hertz 1999). We improve both the approximation ratio and running time bound. Moreover, our method can be applied to two other problems, called OCTSP and OSGPP, to improve their approximation ratios.

- At last, we conduct experiments to evaluate the performance of our algorithms for SGPP.

A summary of previous results and our results is presented in Table 1. Note that for SGPP and CTSP parameterized by the number of groups, the ratio β should be the best possible, as Guttmann-Beck et al. (2000) proved that any β'-approximation algorithm for CTSP, even with four groups, would imply a β'-approximation algorithm for PTSP.

Due to limited space, the proofs of lemmas and theorems marked with "*" were omitted and they can be found in the full version of this paper.

Notations

In SGPP, we are given an edge-weighted complete graph $G = (V, E, w)$, where the vertex set V is split into m pairwise disjoint and non-empty groups: $S_1, ..., S_m$, and the edge weight function w satisfies the *partial metric* property defined below. We use $ab \in E$ to denote an edge in E between vertices a and b, and its weight is denoted by $w(a, b)$. The weight function w satisfies the *partial metric* property:

1. $w(a, b) + w(b, c) \geq w(a, c) = w(c, a)$ for any $a, b, c \in S_i$ with any $1 \leq i \leq m$;
2. $w(a, b)+w(b, c) \geq w(a, c) = w(c, a)$ for any $a, b, c \in V$ with $S_i \neq S_j$, where $a \in S_i$ and $c \in S_j$.

If w fully satisfies the triangle inequality, it is called a *metric*. Let n denote the number of vertices in G. Let t denote the maximum group size, i.e., $t = \max_{1 \leq i \leq m} |S_i|$. If there is a group S_i whose size is smaller than t, we may take a vertex in S_i and make several copies of it to increase the group's size to t. It is easy to see that the new instance is equivalent to the original one. Hence, we can always assume that the size of every group is t, and then we have $n = mt$.

For any positive integer x, let $[x] := \{1, 2, ..., x\}$. For any $V' \subseteq V$, we let $G[V']$ denote the complete graph induced by V'. For any multi-graph G', we use $E(G')$ (resp., $V(G')$) to denote the multi-set of edges (resp., the set of vertices) contained in G'. The *degree* of a vertex $v \in V(G')$ is the number of edges in $E(G')$ that are incident to v, and a vertex is called an *odd-degree* vertex if its degree is odd. We use $Odd(V(G'))$ to denote the set of odd-degree vertices in $V(G')$. Let $\mathcal{S} = \{S_1, ..., S_m\}$. For any $\mathcal{S}' \subseteq \mathcal{S}$, we define $V(\mathcal{S}') = \bigcup_{S \in \mathcal{S}'} V(S)$. By contracting all vertices in the same group S_i into a single vertex v'_i, we obtain a new complete graph, denoted by $G/\mathcal{S} = (V' = \{v'_1, ..., v'_m\}, E', w')$. There is one edge between each pair of vertices in V', and the function w' may be defined later.

A *walk* W, denoted by a sequence of vertices $v_1 v_2 ... v_k$, is a set of edges $\{v_1 v_2, ..., v_{k-1} v_k\}$. Its weight is denoted by $w(W) = w(v_1, v_2) + \cdots + w(v_{k-1}, v_k)$. It is a *path* if all vertices are different. It is a *tour* if the first vertex is the same with the last vertex. An *Eulerian tour* in G is a tour that contains every vertex in V at least once. A *TSP tour* in G is a tour that contains n edges and every vertex in V at least once. A *half-tour* in G is a set of m edges that form a TSP tour in $G/\mathcal{S}$.

We use B^* to denote an optimal solution (which is a TSP tour) of SGPP. It enters each group, visits all vertices within the group, and then leaves the group. We assume that the path visiting all vertices in V_i is $P_i^* = s_i \dots t_i$, where the first and the last visited vertices in S_i are s_i and t_i, respectively. We also denote the maximum weight edge in $G[S_i]$ by $f_i g_i$. Clearly, we have $w(s_i, t_i) \leq w(f_i, g_i)$. Also, since w is a partial metric, we can get $w(f_i, g_i) \leq w(P_i^*)$ by the triangle inequality. For the sake of analysis, we define parameters $\tau, \gamma, \delta \in \mathbb{R}_{\geq 0}$ such that

$$\tau = \frac{\sum_{i \in [m]} w(P_i^*)}{w(B^*)}, \quad \gamma = \frac{\sum_{i \in [m]} w(s_i, t_i)}{w(B^*)}, \quad \text{and} \quad \delta = \frac{\sum_{i \in [m]} w(f_i, g_i)}{w(B^*)}. \tag{1}$$

Note that if $w(B^*) = 0$, we define $\tau, \gamma, \delta = 0$. Since it holds $w(s_i, t_i) \leq w(f_i, g_i) \leq w(P_i^*)$ for each $i \in [m]$, we have

$$0 \leq \gamma \leq \delta \leq \tau \leq 1. \tag{2}$$

We also use T_i^* (resp., C_i^*) to denote the minimum weight spanning tree (resp., tour) in $G[S_i]$. Since P_i^* is a spanning tree in $G[S_i]$ and adding the edge $s_i t_i$ to P_i^* forms a tour in $G[S_i]$, we have

$$w(T_i^*) \leq w(P_i^*) \quad \text{and} \quad w(C_i^*) \leq w(P_i^*) + w(s_i, t_i). \tag{3}$$

Problem Definitions

Definition 1 (SGPP). *Given a complete graph $G = (V = S_1 \cup \cdots \cup S_m, E, w)$ with a partial metric weight function w, the goal is to find a minimum weight TSP tour T visiting all vertices in each group S_i consecutively.*

As mentioned, if the edge weight function fully satisfies the triangle inequality, SGPP reduces to CTSP.

Definition 2 (SPP). *Given a complete graph $G = (V = S_1 \cup \cdots \cup S_m, E, w)$ with a partial metric weight function w and $|S_i| = 2$ for each $i \in [m]$, the goal is to find a minimum weight TSP tour T that includes every edge given by S_i.*

SPP can be regarded as a special case of SGPP with $|S_i| = 2$ for each $i \in [m]$.

Next, we first introduce our polynomial-time approximation algorithms for SGPP. Then, we consider FPT approximation algorithms for SGPP parameterized by the group size and the number of groups, respectively.

A Poly-Time Approximation Algorithm

In this section, we propose a polynomial-time 2.167-approximation algorithm for SGPP. Our algorithm follows the idea of the approximation algorithm for CTSP in (Bao and Liu 2012). We will present two algorithms: Alg.1 and Alg.2. The final ratio is obtained by doing some trade-off between them. Intuitively, Alg.1 handles the case where δ is large, while Alg.2 is designed for cases where δ is small.

Our algorithms will frequently use the approximation algorithm for PTSP (Hoogeveen 1991). Given a graph G with vertices s and t, the algorithm first computes a minimum weight spanning tree in G, then finds a minimum weight matching on $Odd(V(T) \cup \{s,t\})$, and finally constructs a TSP path P between s and t in G by taking shortcuts. We use the following well-known result.

Lemma 3. (Hoogeveen 1991). *Given a graph $G = (V, E, w)$ with $s, t \in V$ and w being a metric, there exists an $\mathcal{O}(n^3)$-time algorithm that computes a TSP path P between vertices s and t in G such that $w(P) \leq \min\{2w(T^*) - w(s,t), w(T^*) + \frac{1}{2}w(C^*)\}$, where T^* (resp., C^*) denotes the minimum weight spanning tree (resp., TSP tour) in G.*

The First Algorithm

For each group S_i, if the edge f_ig_i has a large weight, a short TSP path between the vertices f_i and g_i in $G[S_i]$ can be computed using Hoogeveen's algorithm, as shown in Lemma 3. Building on this observation, Alg.1 begins by computing an approximate TSP path P_i between f_i and g_i in $G[S_i]$ by applying Hoogeveen's algorithm for each $i \in [m]$. To complete these TSP paths into a solution for SGPP, we first construct an instance $G[V']$ of SPP, where $V' = S'_1 \cup \cdots \cup S'_m$ and $S'_i = \{f_i, g_i\}$ for each $i \in [m]$. We then apply the 1.5-approximation algorithm for SPP (Sumita et al. 2017) to obtain a tour B and replace each edge f_ig_i in B with the corresponding path P_i. The details are outlined in Algorithm 1.

Lemma 4. *For SGPP, Alg.1 can compute a solution B_1 with $w(B_1) \leq (1.5 + 2 \cdot \tau - 2 \cdot \delta) \cdot w(B^*)$ in $\mathcal{O}(n^3)$ time.*

Proof. By Lemma 3 and (3), the TSP path P_i in Step 2 holds

$$w(P_i) \leq 2w(T_i^*) - w(f_i, g_i) \leq 2w(P_i^*) - w(f_i, g_i). \quad (4)$$

Consider the optimal solution B^*. Recall that the segment contained in S_i is the path $P_i^* = s_i...t_i$. Assume that the previous vertex of s_i on B^* is t_{i-1} and the subsequent vertex of t_i on B^* is s_{i+1}, and $P_i^* = s_i...f_i...g_i...t_i$. Since w is a partial metric, we can get $w(t_{i-1}, s_i) + w(P_i) + w(t_i, s_{i+1}) \geq w(t_{i-1}, f_i) + w(f_i, g_i) + w(g_i, s_{i+1})$. Therefore, by replacing P_i^* with the edge f_ig_i, we can get a solution B' of SPP in G' with a weight of at most $w(B^*)$.

Algorithm 1: The first algorithm for SGPP (Alg.1)

Input: An instance $G = (V = S_1 \cup \cdots \cup S_m, E, w)$ of SGPP.
Output: A feasible solution B_1 to SGPP.

1: **for** $i \in [m]$ **do**
2: Find a TSP path P_i between the vertices f_i and g_i in $G[S_i]$ using Hoogeveen's algorithm in Lemma 3. ▷ Recall that f_ig_i denotes the maximum weight edge in $G[S_i]$.
3: **end for**
4: Obtain a graph $G' = G[V']$, where $V' = S'_1 \cup \cdots \cup S'_m$ and $S'_i = \{f_i, g_i\}$ for each $i \in [m]$. ▷ This graph can be regarded as an instance of SPP.
5: Apply the 1.5-approximation algorithm (Sumita et al. 2017) for SPP to G' to obtain a solution B.
6: Obtain a solution B_1 by replacing the edge f_ig_i in B with the TSP path P_i for each $i \in [m]$.

By Step 5, B is a 1.5-approximate solution for SPP in G', and hence we have

$$w(B) \leq 1.5 \cdot w(B^*). \quad (5)$$

By Step 6, we have $w(B_1) = w(B) + \sum_{i\in[m]}(w(P_i) - w(f_i, g_i))$. Moreover, by (4) and (5), we can get

$$\begin{aligned} w(B_1) &= w(B) + \sum_{i\in[m]} (w(P_i) - w(f_i, g_i)) \\ &\leq 1.5 \cdot w(B^*) + \sum_{i\in[m]} (2w(P_i^*) - 2w(f_i, g_i)) \quad (6) \\ &= (1.5 + 2 \cdot \tau - 2 \cdot \delta) \cdot w(B^*), \end{aligned}$$

where the last equality follows from (1).

For the running time, Step 5 takes $\mathcal{O}(m^3)$ time (Sumita et al. 2017), and Steps 1-3 take $\mathcal{O}(mt^3)$ time by Lemma 3. Since $n = mt$, the total running time is at most $\mathcal{O}(n^3)$. □

The Second Algorithm

Our second algorithm first computes a half-tour $\widetilde{B}$ in G by finding a TSP tour in $G/\mathcal{S}$, where we define a metric *augmented weight* function w' for $G/\mathcal{S}$ that allows us to use an approximation algorithm for TSP. The half-tour intersects two vertices p_i and q_i within each S_i. Then, we find a TSP path P_i between p_i and q_i in $G[S_i]$ for each $i \in [m]$, and combine these paths with the half-tour to form a complete solution for SGPP. The details are provided in Algorithm 2.

Since we use the 1.5-approximation algorithm for TSP (Christofides 1976) in Step 2, we need to prove that the weight function w' satisfies the triangle inequality.

Lemma 5 (*). *The augmented weight function w' in Step 1 satisfies the triangle inequality property.*

Lemma 6 (*). *The half-tour $\widetilde{B}$ obtained by Alg.2 has a weight of $w(\widetilde{B}) \leq (1.5 + 0.5 \cdot \delta - 1.5 \cdot \tau) \cdot w(B^*)$.*

Lemma 7 (*). *For SGPP, Alg.2 can compute a solution B_2 with $w(B_2) \leq (1.5 + 0.5 \cdot \gamma + 0.5 \cdot \delta) \cdot w(B^*)$ in $\mathcal{O}(n^3)$ time.*

Algorithm 2: The second algorithm for SGPP (Alg.2)

Input: An instance $G = (V = S_1 \cup \cdots \cup S_m, E, w)$ of SGPP.
Output: A feasible solution B_2 to SGPP.

1: Obtain the graph $G/\mathcal{S} = (V' = \{v'_1, ..., v'_m\}, E', w')$, and define the *augmented weight* $w'(v'_i, v'_j)$ between vertices $v'_i, v'_j \in V'$ as $\frac{w(f_i,g_i)+w(f_j,g_j)}{2} + \min_{u\in S_i, v\in S_j} w(u, v)$. ▷ The augmented weight function satisfies the triangle inequality as proved in Lemma 5.
2: Obtain a TSP tour B in G' by using the 1.5-approximation algorithm for TSP (Christofides 1976). Denote the corresponding half-tour in G by $\widetilde{B}$, and $V(\widetilde{B}) \cap S_i = \{p_i, q_i\}$.
3: **for** $i \in [m]$ **do**
4: Find a TSP path P_i between the vertices f_i and g_i in $G[S_i]$ using Hoogeveen's algorithm in Lemma 3.
5: **end for**
6: Obtain a solution B_2 by combining the half-tour $\widetilde{B}$ with the TSP path P_i for each $i \in [m]$.

By doing a trade-off between Alg.1 and Alg.2, we can get our first main result.

Theorem 8. *For SGPP, there is a* 2.167*-approximation algorithm with a running time of* $\mathcal{O}(n^3)$.

Proof. We can execute both Alg.1 and Alg.2, and return the better solution, $\arg\min\{w(B_1), w(B_2)\}$. Since the running times of both Alg.1 and Alg.2 are $\mathcal{O}(n^3)$ by Lemmas 4 and 7, the running time of this algorithm is also $\mathcal{O}(n^3)$.

By (2), Lemmas 4 and 7, the better solution has an approximation ratio of $\rho = \frac{\min\{w(B_1),w(B_2)\}}{w(B^*)}$, where ρ can be calculated using the following linear program.

$$\begin{aligned} \max \quad & \rho \\ \text{s.t.} \quad & \rho \le 1.5 + 2 \cdot \tau - 2 \cdot \delta, \\ & \rho \le 1.5 + 0.5 \cdot \gamma + 0.5 \cdot \delta, \\ & 0 \le \gamma \le \delta \le \tau \le 1. \end{aligned}$$

Solving this linear program shows that the approximation ratio is at most 2.167, where in the worst case we have $\gamma = \delta = 3/5$ and $\tau = 1$. □

FPT Approximation Algorithms for SGPP

In this section, we further improve the approximation ratio by considering FPT running time. This is a hot research topic in parameterized complexity. CTSP and SGPP have also been considered from this point of view. Here, we consider two important parameters: the group size t and the number of groups m.

Parameterized by the Group Size

In this section, we propose an FPT 2.072-approximation algorithm, referred to as Alg.3, for SGPP parameterized by the group size t.

Our algorithm first computes a TSP path $P_i = p_i...q_i$ in $G[S_i]$, minimizing $w(P_i) - w(p_i, q_i)$ using dynamic programming. Then, we apply the 1.5-approximation algorithm for SPP to concatenate these TSP paths into a complete solution for SGPP, following the idea in Alg.1. The details of Alg.3 are presented in Algorithm 3.

Algorithm 3: The third algorithm for SGPP (Alg.3)

Input: An instance $G = (V = S_1 \cup \cdots \cup S_m, E, w)$ of SGPP.
Output: A feasible solution B_3 to SGPP.

1: **for** $i \in [m]$ **do**
2: Find a TSP path $P_i = p_i...q_i$ with minimized $w(P_i) - w(p_i, q_i)$ in $G[S_i]$ using the dynamic programming method in (Held and Karp 1962; Bellman 1962).
3: **end for**
4: Obtain a graph $G' = G[V']$, where $V' = S'_1 \cup \cdots \cup S'_m$ and $S'_i = \{p_i, q_i\}$ for each $i \in [m]$. ▷ This graph can be regarded as an instance of SPP.
5: Apply the 1.5-approximation algorithm for SPP (Sumita et al. 2017) to G' to obtain a 1.5-approximate solution B.
6: Obtain a solution B_3 by replacing the edge p_iq_i in B with the TSP path P_i for each $i \in [m]$.

It is worth noting that our Alg.3 is inspired by the previous FPT 2.5-approximation algorithm for SGPP (Sumita et al. 2017). The key difference lies in our approach: instead of minimizing $w(P_i)$, as done in their method, we minimize $w(P_i) - w(p_i, q_i)$. One main reason for this adjustment is that minimizing this objective allows us to get a better solution for SGPP, as suggested in (6) in the analysis of Alg.1.

Lemma 9 (*)**.** *For SGPP parameterized by the group size* t*, Alg.3 can compute a solution* B_3 *in FPT time with* $w(B_3) \le (1.5 + \tau - \gamma) \cdot w(B^*)$.

Although Alg.3 by itself may only achieve an approximation ratio of 2.5, by combining Alg.1, Alg.2, and Alg.3, we can establish an FPT 2.072-approximation ratio for SGPP, parameterized by the group size.

Theorem 10 (*)**.** *For SGPP parameterized by the group size, there is an FPT* 2.072*-approximation algorithm with a running time of* $\mathcal{O}^*(2^t)$.

Parameterized by the Number of Groups

In this section, we propose an FPT β-approximation algorithm, referred to as Alg.4, for SGPP parameterized by the number of groups m. Moreover, we explore the Ordered SGPP (OSGPP), where the order of visiting the groups in the tour is specified as part of the problem. OSGPP generalizes OCTSP (Anily, Bramel, and Hertz 1999) and has applications in CTSP, where the groups may not be disjoint (Guttmann-Beck, Knaan, and Stern 2018).

Our algorithm, detailed in Algorithm 4, begins by computing a set $\mathcal{P}_i$ of β-approximate TSP paths between each pair of distinct vertices $u, v \in S_i$ in $G[S_i]$ for each $i \in [m]$. Under the constraint that only one TSP path is used from $\mathcal{P}_i$, the algorithm then employs a dynamic programming approach to determine the optimal way to concatenate m approximate TSP paths into a complete solution for SGPP.

Before we analyze the solution, we first prove a key property of our dynamic programming approach.

Algorithm 4: The fourth algorithm for SGPP (Alg.4)

Input: An instance $G = (V = S_1 \cup \cdots \cup S_m, E, w)$ of SGPP.
Output: A feasible solution B_4 to SGPP.

1: **for** $i \in [m]$ **do**
2: For any distinct vertices $u, v \in S_i$, compute a TSP path P_{uv} between them in $G[S_i]$ by calling a β-approximation algorithm for PTSP. Denote the set of all such TSP paths by $\mathcal{P}_i$.
3: **end for**
4: For any $\mathcal{S}' \subseteq \mathcal{S}$ and $s, t \in V(\mathcal{S}')$, define $dp[\mathcal{S}'][s][t]$ as the weight of the minimum weight walk starting from s and ending at t, where the walk contains only one TSP path from $\mathcal{P}_i$ for each $S_i \in \mathcal{S}'$, and all vertices in S_i appear consecutively on the walk. If no such walk exists, set $dp[\mathcal{S}'][s][t] = \infty$.
5: Initialize $dp[\mathcal{S}'][s][t] = \infty$ for any $\mathcal{S}' \subseteq \mathcal{S}$, and $s, t \in V(\mathcal{S}')$.
6: **for** $\mathcal{S}' \subseteq \mathcal{S}$ in increasing order of $|\mathcal{S}'|$ **do**
7: Define $\overline{\mathcal{S}''} = \mathcal{S}' \setminus \mathcal{S}''$ for any $\mathcal{S}'' \subseteq \mathcal{S}'$.
8: **if** $|\mathcal{S}'| = 1$ **then**
9: For distinct $s, t \in V(\mathcal{S}')$, let $dp[\mathcal{S}'][s][t] = w(P_{st})$.
10: **else**
11: For any $s, t \in V(\mathcal{S}')$, let $dp[\mathcal{S}'][s][t]$ equal

$$\min_{\mathcal{S}_s \subseteq \mathcal{S}' : |\mathcal{S}_s| = |\mathcal{S}'|-1, s \in V(\mathcal{S}_s), t \in V(\overline{\mathcal{S}_s})} \quad \min_{x \in V(\mathcal{S}_s), y \in V(\overline{\mathcal{S}_s})} (dp[\mathcal{S}_s][s][x] + w(x, y) + dp[\overline{\mathcal{S}_s}][y][t]).$$

12: **end if**
13: **end for**
14: Get a solution $B_4 = \arg\min_{s,t \in V(\mathcal{S})}(dp[\mathcal{S}][s][t] + w(s, t))$.

Lemma 11 (*). *The dynamic programming approach in Alg.4 can optimally concatenate m TSP paths, each selected from $\mathcal{P}_i$, into a solution B_4 for SGPP.*

Theorem 12. *For SGPP parameterized by the number of groups m, Alg.4 is an FPT β-approximation algorithm with a running time of $\mathcal{O}^*(2^m)$.*

Proof. First, we analyze the weight of the solution B_4.

Recall that the path of the optimal solution B^* visiting all vertices in S_i is $P_i^* = s_i ... t_i$. For each $i \in [m]$, we select the TSP path P_i between s_i and t_i from $\mathcal{P}_i$. Since each P_i is computed using a β-approximation algorithm (as described in Step 2), the optimal way to concatenate these m TSP paths results in a β-approximate solution B for SGPP.

By Lemma 11, since the dynamic programming approach in Alg.4 can optimally concatenate m TSP paths from $\mathcal{P}_1 \cup \cdots \cup \mathcal{P}_m$ into a solution B_4, we have $w(B_4) \leq w(B)$. Thus, B_4 also yields a β-approximate solution for SGPP.

Next, we analyze the running time of Alg.4.

Steps 1-3 take $\mathcal{O}(mt^2 \cdot f(n))$ time. For the dynamic programming part, there are $\mathcal{O}(2^m \cdot n^2)$ states, and each state transition takes $\mathcal{O}(n^2)$ time. Since $n = mt$, the running time of Alg.4 is at most $\mathcal{O}(n^2 \cdot f(n) + 2^m \cdot n^4) \subseteq \mathcal{O}^*(2^m)$. □

Since CTSP is a special case of SGPP, our result in Theorem 12 directly improves the previous FPT approximation ratio from 1.667 (Anily, Bramel, and Hertz 1999) to β for CTSP parameterized by the number of groups.

By modifying the dynamic programming approach in Alg.4, we can derive a polynomial-time β-approximation algorithm for OSGPP.

Theorem 13 (*). *For OSGPP, there is a β-approximation algorithm with a running time of $\mathcal{O}(n^2 \cdot f(n) + n^3)$.*

Similarly, since OCTSP is a special case of OSGPP, our result in Theorem 13 improves the 1.667-approximation algorithm for OCTSP (Anily, Bramel, and Hertz 1999).

Moreover, Guttmann-Beck, Knaan, and Stern (2018) considered a variant of CTSP, where groups are not necessarily disjoint. Specifically, if the intersection of groups forms a path structure, they proposed a 1.667-approximation algorithm by applying the algorithm for OCTSP (Anily, Bramel, and Hertz 1999). Thus, our result in Theorem 13 also improves the approximation ratio for CTSP when the intersection of groups is a path, reducing it to β.

Experiments

We conduct experiments to evaluate the performance of our algorithms. There are mainly two experiments.

The first experiment is to compare our first FPT approximation algorithm with the previous FPT 2.5-approximation algorithm (Sumita et al. 2017), which is denoted as SYKK. Our FPT approximation algorithm indeed combines three algorithms Alg.1, Alg.2, and Alg.3. So, we will show all the results of the three algorithms. For Alg.3 and SYKK, the running time is exponentially related to t. So, the tested instances have small values of t but a large value of m. In the second experiment, we test our second FPT algorithm, Alg.4, and compare it with our polynomial-time algorithms Alg.1 and Alg.2. Alg.4's running time is exponentially related to m. So, the tested instances have small values of m but a large value of t. This is also the reason why we do not compare Alg.4 with Alg.3 (or SYKK) directly.

Since the β-approximation algorithm for PTSP (Karlin, Klein, and Gharan 2021) is difficult to implement, in our Alg.4, we use the $5/3$-approximation algorithm for PTSP (Hoogeveen 1991) instead, and hence this implementation gives Alg.4 an approximation ratio of $5/3$.

Our algorithm is implemented in C++ and tested on a server with an Intel Xeon Gold 6226R CPU @ 2.90GHz and 512 GB of memory, running Ubuntu Server 20.04 LTS. Our code is available at https://github.com/forgottencsc/TheSubgroupPlanningProblem.

Next, we introduce our datasets and present our results.

Instances. Similarly to (Sumita et al. 2017), we use VLSI datasets[1] for TSP. The datasets consist of 102 instances with the size ranging from 131 to 744,710. Note that the number in each instance's name indicates its size.

In the first experiment, we test on 35 instances with sizes almost evenly distributed between 1000 to 60000. For the second experiment, we only consider some small instances, as Alg.4 runs slowly and takes longer than 3600 seconds even for instances with only hundreds of vertices.

For each instance, we generate a set of pairwise disjoint groups of the same size t by randomly creating groups: we first sort all vertices uniformly at random, and then sequentially select t vertices to create a group. Note that a random method was also used in (Sumita et al. 2017).

[1] http://www.math.uwaterloo.ca/tsp/vlsi/index.html

Data	Alg.1	Alg.2	Alg.3	SYKK
dca1389	**158649**	259412	158655	158655
bva2144	105630	161990	**105593**	105610
dke3097	143842	224850	**143829**	**143829**
ltb3729	242874	381668	**242848**	242866
bgd4396	853665	1293878	**853647**	**853647**
fea5557	**406736**	631491	406737	406737
xsc6880	740614	1126064	**740576**	**740576**
lap7454	488905	742931	**488904**	**488904**
ida8197	1358647	1969823	**1358638**	**1358638**
dga9698	1633100	2408966	**1633071**	**1633071**
xmc10150	581597	891720	581539	**581535**
xvb13584	1604595	2466539	**1604594**	**1604594**
xrb14233	3360566	5033119	**3360435**	3360438
xia16928	1742807	2693858	1742697	**1742527**
pjh17845	2023990	3184989	**2023982**	**2023982**
frh19289	3287629	5082786	**3287424**	**3287424**
fnc19402	2150276	3265284	**2150273**	**2150273**
fma21553	3050663	4539072	**3050335**	3050466
lsb22777	**1847721**	2800760	1847727	1847727
xrh24104	3070288	4732847	**3070264**	**3070264**
bbz25234	4686555	7026817	4686553	**4686397**
irx28268	**3240560**	4969132	3240609	3240600
icx28698	**9538161**	14175921	9538263	9538263
ird29514	5543269	8435772	**5543244**	**5543244**
pbh30440	11882167	17586020	**11881972**	**11881972**
xib32892	9770557	14595323	**9770511**	9770668
fry33203	10187976	15382922	**10187932**	**10187932**
bby34656	**6919121**	10690416	6919221	6919221
pba38478	9968027	15207870	**9967925**	**9967925**
ics39603	16197682	23956408	**16197603**	**16197603**
rbz43748	13350082	20153934	13350028	**13349984**
fht47608	14214372	21805702	**14214246**	**14214246**
fna52057	11297860	17137564	**11297611**	11297619
bna56769	**20418680**	30939118	20418696	20418696
dan59296	**17004416**	25759252	17004424	17004424

Table 2: The results of Alg.1, Alg.2, Alg.3, and SYKK under $t = 10$ on 35 instances, where the best solution for each instance is highlighted in bold

Moreover, as in (Sumita et al. 2017), to ensure that the weight function is a partial metric, we set the weight of each edge uv within each group to $X \cdot d(u, v)$, where $d(u, v)$ is the original distance between u and v, and X is an integer drawn uniformly at random from the range $[2, 10]$.

The First Experiment. In the first experiment, we need to set t as small values since the running of Alg.4 is $\mathcal{O}^*(2^t)$. Note that if $t \in \{1, 2, 3\}$, both Alg.1 and Alg.3 are the same with the previous SYKK, as they will find the same TSP path within each group. Therefore, we consider $t \in \{5, 10, 15\}$.

Table 2 shows the solution quality for different algorithms under $t = 10$ on the 35 selected instances. The results for $t \in \{5, 15\}$ are similar to those for $t = 10$, which are omitted due to limited space. We can observe that for most instances, Alg.3 and SYKK give the best results among the four algorithms. They have similar performances. Alg.1 can achieve the best performance only for a small number of instances, but generates close-to-best results for the other instances. Alg.2, however, has the worst performance. Recall that Alg.1 hands the case where δ is large, whereas Alg.2 is designed for the case where δ is small. Under the setting of these instances, δ defined in (1) may have a large value. This may explain why Alg.1 significantly outperforms Alg.2.

In terms of running time, for small instances, all algorithms run quickly with no significant differences. For larger instances, Alg.1 is the fastest, Alg.2 is the slowest, and SYKK and Alg.3 are of the same level but SYKK is slightly faster than Alg.3. For example, for the instance with 59296 vertices, Alg.1, Alg.2, Alg.3, and SYKK take 395s, 977s, 424s, and 418s, respectively. The slower performance of Alg.2 is mainly due to Step 2, where it needs to find a minimum-weight spanning tree with over 2000 odd-degree vertices to obtain a 1.5-approximation solution for TSP (Christofides 1976), which is time-consuming. The detailed running time is omitted due to limited space.

m	Data	Alg.1	Alg.2	Alg.4	Running time for Alg.4 (s)
4	xqf131	3855	4106	**3603**	18
	xqg237	10569	10777	**9926**	291
	pma343	9481	9454	**9103**	1898
	pka379	13681	13607	**13196**	3080
	bcl380	6187	5961	**5606**	3161
6	xqf131	3729	4149	**3540**	52
	xqg237	3808	3866	**3468**	927
8	xqf131	3569	4562	**3485**	165
	xqg237	7405	7675	**6911**	3009
10	xqf131	5443	7116	**5346**	565
12	xqf131	1877	2488	**1861**	1948

Table 3: The results of Alg.1, Alg.2, and Alg.4, and the running time (in seconds) of Alg.4 on instances that Alg.4 can solve within 3600s, where the best solution for each instance is highlighted in bold

The Second Experiment. In this experiment, we set m as small values on small instances. In detail, we consider $m \in \{4, 6, 8, 10, 12\}$. Since we generate groups based on t, this can be done by setting $t = \lfloor \frac{n+(m-1)}{m} \rfloor$ in our experiments.

Table 3 shows the results for different algorithms under $m \in \{4, 6, 8, 10, 12\}$. Indeed, we only list the instances that Alg.4 can solve within 3600s. Note that for these instances, both Alg.1 and Alg.2 take only about 0.01s, so their running times are omitted in the table. Alg.4 achieves the best performance for every instance. This is expected, as Alg.4 must perform no worse than Alg.1. While Alg.1 computes only one TSP path within each group and then approximately completes the m TSP paths into a solution for SGPP, Alg.4 computes a set of TSP paths within each group and then optimally completes m TSP paths into a solution. However, the running time of Alg.4 is worse, increasing dramatically with the size of the instances or the number of groups.

Conclusion

In this paper, we study approximation algorithms for SGPP and CTSP. First, for polynomial-time approximation algorithms, we reduce the gap between SGPP and CTSP by improving the approximation ratio for SGPP. Second, we improve FPT approximation algorithms for SGPP as well as CTSP parameterized by the group size and the number of groups, respectively. We improve not only the approximation ratio but also the running time bound. We remark here again that for general edge weight functions, it is impossible to get any constant approximation ratio for SGPP and CTSP unless P=NP. Thus, it is meaningful to distinguish SGPP from CTSP by relaxing more on the edge weight functions.

Acknowledgments

The work is supported by the National Natural Science Foundation of China, under the grants 62372095, 62172077, and 62350710215.

References

Ahmed, Z. H. 2013. An exact algorithm for the clustered travelling salesman problem. *Opsearch*, 50: 215–228.

Anily, S.; Bramel, J.; and Hertz, A. 1999. A 5/3-approximation algorithm for the clustered traveling salesman tour and path problems. *Oper. Res. Lett.*, 24(1-2): 29–35.

Arkin, E. M.; Hassin, R.; and Klein, L. 1997. Restricted delivery problems on a network. *Networks: An International Journal*, 29(4): 205–216.

Baniasadi, P.; Foumani, M.; Smith-Miles, K.; and Ejov, V. 2020. A transformation technique for the clustered generalized traveling salesman problem with applications to logistics. *European Journal of Operational Research*, 285(2): 444–457.

Bao, X.; and Liu, Z. 2012. An improved approximation algorithm for the clustered traveling salesman problem. *Information Processing Letters*, 112(23): 908–910.

Bellman, R. 1962. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9(1): 61–63.

Chisman, J. A. 1975. The clustered traveling salesman problem. *Computers & Operations Research*, 2(2): 115–119.

Christofides, N. 1976. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon University.

Ding, C.; Cheng, Y.; and He, M. 2007. Two-level genetic algorithm for clustered traveling salesman problem with application in large-scale TSPs. *Tsinghua Science & Technology*, 12(4): 459–465.

Gendreau, M.; Laporte, G.; and Hertz, A. 1997. An Approximation Algorithm for the Traveling Salesman Problem with Backhauls. *Oper. Res.*, 45(4): 639–641.

Guttmann-Beck, N.; Hassin, R.; Khuller, S.; and Raghavachari, B. 2000. Approximation Algorithms with Bounded Performance Guarantees for the Clustered Traveling Salesman Problem. *Algorithmica*, 28(4): 422–437.

Guttmann-Beck, N.; Knaan, E.; and Stern, M. 2018. Approximation Algorithms for Not Necessarily Disjoint Clustered TSP. *J. Graph Algorithms Appl.*, 22(4): 555–575.

Gyorfi, J. S.; Gamota, D. R.; Mok, S. M.; Szczech, J. B.; Toloo, M.; and Zhang, J. 2010. Evolutionary path planning with subpath constraints. *IEEE Transactions on Electronics Packaging Manufacturing*, 33(2): 143–151.

Held, M.; and Karp, R. M. 1962. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied mathematics*, 10(1): 196–210.

Hoogeveen, J. A. 1991. Analysis of Christofides' heuristic: Some paths are more difficult than cycles. *Oper. Res. Lett.*, 10(5): 291–295.

Jaillet, L.; and Porta, J. M. 2013. Path Planning Under Kinematic Constraints by Rapidly Exploring Manifolds. *IEEE Trans. Robotics*, 29(1): 105–117.

Karlin, A. R.; Klein, N.; and Gharan, S. O. 2021. A (slightly) improved approximation algorithm for metric TSP. In *53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2021*, 32–45. ACM.

Kawasaki, M.; and Takazawa, K. 2020. Improving approximation ratios for the clustered traveling salesman problem. *Journal of the Operations Research Society of Japan*, 63(2): 60–70.

Laporte, G.; and Palekar, U. 2002. Some applications of the clustered travelling salesman problem. *Journal of the operational Research Society*, 53(9): 972–976.

Lokin, F. 1979. Procedures for travelling salesman problems with additional constraints. *European Journal of Operational Research*, 3(2): 135–141.

Mestria, M. 2018. New hybrid heuristic algorithm for the clustered traveling salesman problem. *Computers & Industrial Engineering*, 116: 1–12.

Nash, A.; Koenig, S.; and Likhachev, M. 2009. Incremental Phi*: Incremental Any-Angle Path Planning on Grids. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009*, 1824–1830.

Ninomiya, K.; Kapadia, M.; Shoulson, A.; Garcia, F. M.; and Badler, N. I. 2015. Planning approaches to constraint-aware navigation in dynamic environments. *Comput. Animat. Virtual Worlds*, 26(2): 119–139.

Safilian, M.; Hashemi, S. M.; Eghbali, S.; and Safilian, A. 2016. An Approximation Algorithm for the Subpath Planning Problem. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016*, 669–675. IJCAI/AAAI Press.

Sahni, S.; and Gonzalez, T. 1976. P-complete approximation problems. *Journal of the ACM*, 23(3): 555–565.

Saller, S.; Koehler, J.; and Karrenbauer, A. 2023. A Systematic Review of Approximability Results for Traveling Salesman Problems leveraging the TSP-T3CO Definition Scheme. *CoRR*, abs/2311.00604.

Sumita, H.; Yonebayashi, Y.; Kakimura, N.; and Kawarabayashi, K. 2017. An Improved Approximation Algorithm for the Subpath Planning Problem and Its Generalization. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017*, 4412–4418. ijcai.org.

Surynek, P. 2015. Reduced Time-Expansion Graphs and Goal Decomposition for Solving Cooperative Path Finding Sub-Optimally. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015*, 1916–1922. AAAI Press.

Traub, V.; Vygen, J.; and Zenklusen, R. 2022. Reducing Path TSP to TSP. *SIAM J. Comput.*, 51(3): 20–24.

Zhang, T.; Ke, L.; Li, J.; Li, J.; Huang, J.; and Li, Z. 2018. Metaheuristics for the tabu clustered traveling salesman problem. *Computers & Operations Research*, 89: 1–12.