Collective Travel Planning in Spatial Networks

Shuo Shang, Lisi Chen, Zhewei Wei, Christian S. Jensen, *Fellow, IEEE,* Ji-Rong Wen, and Panos Kalnis

Abstract—Travel planning and recommendation are important aspects of transportation. We propose and investigate a novel Collective Travel Planning (CTP) query that finds the lowest-cost route connecting multiple sources and a destination, via at most k meeting points. When multiple travelers target the same destination (e.g., a stadium or a theater), they may want to assemble at meeting points and then go together to the destination by public transport to reduce their global travel cost (e.g., energy, money, or greenhouse-gas emissions). This type of functionality holds the potential to bring significant benefits to society and the environment, such as reducing energy consumption and greenhouse-gas emissions, enabling smarter and greener transportation, and reducing traffic congestions. The CTP query is Max SNP-hard. To compute the query efficiently, we develop two algorithms, including an exact algorithm and an approximation algorithm. The exact algorithm is capable finding the optimal result for small values of k (e.g., $k = 2$) in interactive time, while the approximation algorithm, which has a 5-approximation ratio, is suitable for other situations. The performance of the CTP query is studied experimentally with real and synthetic spatial data.

Keywords—Collective Travel Planning, Location Based Services, Spatial Networks, Spatial Databases

1 Introduction

The continued proliferation of GPS-equipped mobile devices (e.g., vehicle navigation systems and smart phones) and the proliferation of online map-based services (e.g., Google Maps[1], Bing Maps[2], and MapQuest[3]) enable people to acquire their current geographic positions in real time and to retrieve spatial information relevant to their travel. In this paper, we aim to provide fundamental geographic functionality that is relevant to a range of services. Specifically, we propose and investigate a novel query, the Collective Travel Planning (CTP) query, that finds the lowest-cost route connecting multiple query sources and a destination via at most k meeting points. For example, when multiple travelers target the same destination (e.g., a stadium or a theater), they may want to assemble at their nearest meeting points, and then travel together to the destination by collective transport (e.g., shuttle bus or taxi) to reduce their global travel cost (e.g., energy, money, or greenhouse-gas emissions). This type of query is useful in organizing large events, and it can bring significant benefit to society and the environment: it can help optimize the allocation of transportation resources, reduce resource (e.g., energy and money) consumption, and enable smarter and greener transportation; and it can help reduce greenhouse-gas emissions and traffic congestion. The EU targets a 50% reduction in CO_2 emissions by 2050. This work is motivated in part by the EU project Reduction[4].

Given the current locations Q of a set of travelers, a set of meeting points S, a destination d, and an integer threshold k ($1 \leq k \leq \min\{|S|, |Q|\}$), we aim to identify a subset A of S with at most k elements that when used as meeting points results in the minimum global travel cost. The global travel cost includes two parts: a local travel cost and a connection travel cost. The local travel cost is the sum of the travel cost from each traveler's current location to their closest meeting point, and the connection travel cost is the sum of the travel costs from each meeting point to the destination. The meeting point count k is expected to be set according to the resources that can be used (e.g., the number of shuttle buses and drivers). For example, an event organizer my choose 100 meeting points for the event due to the limitation of resources.

An example of the CTP query is shown in Figure 1, where p_1, p_3, p_4, and p_5 are selected meeting points and d is the destination. Let $k = 5$ and subset $A = \{p_1, p_3, p_4, p_5\}$. First, travelers go to their closest meeting point by private transport. Then the travelers at the same meeting point go together to the destination by collective transport. For example, for the traveler at q_1, p_1 is the closest meeting point, so the traveler will follow the shortest path from q_1 to p_1 by private transport (local travel cost of q_1). A total of five travelers, q_1, q_2, q_3, q_4, and q_5, meet at p_1. They then follow the shortest path from p_1 to the destination d by collective transport (connection travel cost of p_1).

When a collective travel route is planned, we can set a meeting time for each selected meeting point, where the meeting time depends on the travel distance from the meeting point to the destination and the event time at the destination.

- *Shuo Shang is with Department of Computer Science, China University of Petroleum, Beijing, P.R.China. E-mail: jedi.shang@gmail.com*
- *Lisi Chen is with School of Computer Engineering, Nanyang Technological University, Singapore. E-mail: lchen012@e.ntu.edu.sg*
- *Zhewei Wei and Ji-Rong Wen are with School of Information, Renmin University of China, P.R.China. E-mail: zhewei@ruc.edu.cn, jirong.wen@gmail.com*
- *Christian S. Jensen is with Department of Computer Science, Aalborg University, DK-9220 Aalborg East, Denmark. E-mail: csj@cs.aau.dk*
- *Panos Kalnis is with King Abdullah University of Science and Technology, Saudi Arabia. E-mail: panos.kalnis@kaust.edu.sa*

1. http://maps.google.com/
2. http://www.bing.com/maps/
3. http://www.mapquest.com
4. http://www.reduction-project.eu/

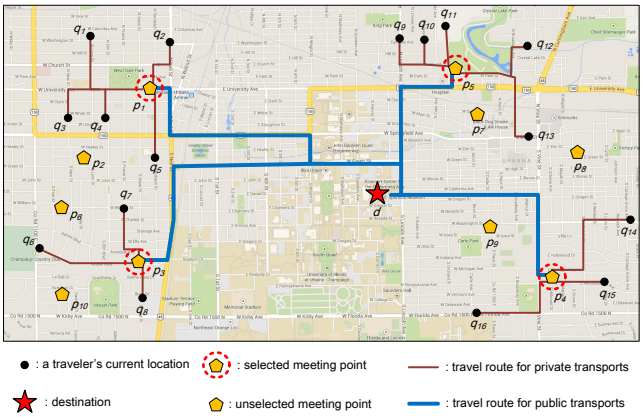


Fig. 1. An example of the CTP query

For example, assume that an event starts at 7 p.m. and that the estimated travel time from a meeting point p to the destination is one hour. We can then set the meeting time at meeting point p to 6 p.m. Then every individual should arrive at p before 6 p.m. The departure time is the minimum of the arrival time of the latest arriving individual and 6 p.m. Thus, it is not necessary for any individuals to wait beyond 6 p.m.

To the best of our knowledge, this is the first study of the collective travel planning query in spatial networks. Some existing multi-source trip planning queries (e.g., the group nearest neighbor query [20] [21] and the group trip planning query [14] that also aim to minimize all travelers' global travel cost) assume that each traveler goes to the destination individually and do not take into account collective travel. We contend that it is of societal interest to provide solutions that take into account means of collective transportation, as this may contribute to reducing energy consumption, pollution, global warming, and congestion.

Next, the CTP query is also different from most existing ridesharing (carpooling) services [1] [2] [18] [24]. Generally, such services aim to plan a travel route with pick-up and drop-off locations for a small number of users with similar destination, while the CTP query aims to plan a collective trip for many users (e.g., tens or hundreds of users or more) located all over a city and targeting the same destination. In fact, the CTP query can be viewed as a variant of the *metric k uncapacitated facility location problem* (k-UFL) [17], as it asks for an optimal meeting point set A ($A \subseteq S \wedge |A| \leq k$). Existing ridesharing techniques do not address this problem.

The CTP query is applied in spatial networks, since in a large number of practical scenarios, users (e.g., pedestrians and vehicles) move in such networks (e.g., roads and railways). Exact search is a straightforward method to compute the CTP query that evaluates each potential subset A ($A \subseteq S \wedge |A| \leq k$), of which there are $\sum_{i=1}^{k} \binom{|S|}{i} = \sum_{i=1}^{k} \frac{|S|!}{(|S|-i)!i!}$ possible combinations. We define a pair of an upper and a lower bound to prune the search space during query processing. For a small threshold k (e.g., $k = 2$), the exact algorithm is capable of finding the optimal result of the CTP query in interactive time. However, $\sum_{i=1}^{k} \frac{|S|!}{(|S|-i)!i!}$ is exponential in $|S|$, and the CTP query cannot be computed in polynomial time. In fact, the CTP query is a variant of the k-UFL problem [17] and is Max SNP-hard. To the best of our knowledge, no existing method can compute the CTP query efficiently.

To achieve better performance than does the exact algorithm, an approximation algorithm is developed with a 5 approximation ratio. Initially, we arbitrarily select a subset A ($|A| \leq k$) from S. Then we define three operations based on *local search* [7] [29]: ***add*** (add a new item $p \in (S \setminus A)$ to A, if $|A| < k$), ***drop*** (drop an item from A, if $|A| > 1$), and ***swap*** (swap an item in A with another items in $(S \setminus A)$). We repeatedly apply a randomly selected operation to improve the global travel cost by a factor of $1 + \varepsilon$, where ε is an arbitrary small constant. The search process terminates when no new operation can produce an improved result. The cost of the obtained result is guaranteed to be at most 5 times worse than that of the global optimum. The experimental results show that the approximate results are generally very close to the global optimum (less than 1.15 times larger).

The main contribution in relation to the approximation algorithm is to "bridge theory and practice." There exist several theoretical methods for the k-UFL problem (e.g., modify one item [7] or modify multiple items at one time [29]), and their targets are to achieve a lower approximation ratio. However, the CTP query has to balance accuracy and efficiency. Although some theoretical methods can achieve a lower approximation ratio, their query efficiency is very low. Thus, our target is to select a suitable theoretical method for the CTP query and then make it practical. Through theoretical analysis, we only allow one item to be modified in an operation. We propose two effective pruning techniques that accelerate the approximation algorithm while retaining its approximation ratio; experimental results show that the query efficiency is improved by at least an order of magnitude. It is worth noting that the theoretical method cannot be used by itself due to its low efficiency.

We further extend the approximation algorithm to two practical scenarios where (1) the connection travel cost is dependent on the number of travelers, and (2) where a traveler close to the destination can go to the destination directly. We develop a series of new metrics and bounds for these scenarios. The theoretical approximation ratio does not work here, and we conduct extensive experiments to show that our extension is usable in the new scenarios.

To sum up, the contributions of the paper are as follows.

- We propose a novel type of query to plan the lowest-cost routes connecting multiple query sources and a destination via at most k meeting points.
- We define a series of distance metrics to evaluate the travel cost under different conditions (Section 2).
- We prove that the CTP query is SNP-hard that can be reduced to the k-UFL problem (Section 3).
- We develop an exact algorithm with effective pruning techniques to find the optimal result for small k (e.g., $k = 2$) (Section 4).
- We develop an approximation algorithm with two effective pruning techniques to compute the CTP query efficiently (Sections 5).
- We define new distance measures for practical scenarios and further extend the algorithms correspondingly (Section 6).

- We conduct extensive experiments on real and synthetic spatial data to investigate the performance (efficiency and effectiveness) of the developed algorithms (Section 7).

The rest is organized as follows. Section 2 introduces the spatial networks and distance metrics used in the paper and defines the problem. Section 3 analyzes the problem. The exact algorithm is introduced in Section 4, while the approximation algorithm is covered in Section 5. The developed algorithms are further extended to practical scenarios in Section 6, which is followed by a coverage of experimental results in Section 7. Related work is covered in Section 8, and conclusions are drawn in Section 9.

2 Preliminaries

2.1 Spatial Networks

A spatial network is modeled as a connected and undirected graph $G(V, E, F, W)$, where V is a vertex set and $E \subseteq V \times V$ is an edge set. A vertex $v_i \in V$ represents a road intersection or an end of a road. An edge $e_k = (v_i, v_j) \in E$ is defined by two vertices and represents a road segment that enables travel between vertices v_i and v_j. Function $F : V \cup E \rightarrow Geometries$ records geometrical information of the spatial network G. In particular, it maps a vertex and an edge to the point location of the corresponding road intersection and to a polyline representing the corresponding road segment, respectively.

Function $W : E \rightarrow R$ is a function that assigns a real-valued weight to each edge. The weight $W(e)$ of an edge e represents the corresponding road segment's length or some other relevant property such as its travel time [9] or fuel consumption [12], [28], which may be obtained from historical traffic data. Given two vertices p_a and p_b in a spatial network, the network shortest path between them (i.e., a sequence of edges linking p_a and p_b where the accumulated weight is minimal) is denoted by $SP(p_a, p_b)$, and its length is denoted by $sd(p_a, p_b)$. When weights model aspects such as travel time and fuel consumption, the lower bound of network distance is not necessarily the corresponding Euclidean distance; thus, spatial indexes such as the R-tree [13] are not effective. For simplicity, we assume that the data points considered (e.g., meeting points and query points) are located on vertices. It is straightforward to also support data points on edges.

In this work, we study static spatial networks. To enhance the efficiency of CTP query processing, we assume that all-pair shortest path distances have been pre-computed. The time complexity of this pre-computation process is $O(V^2 lg(V) + VE)$ when using by Dijkstras algorithm [8], and the running time is studied experimentally in Section 7.

2.2 Travel Cost Functions

Given a set of vertices A and a vertex q in a spatial network, the minimum network distance between q and A is defined by

$$d(q, A) = \min_{p_i \in A} \{sd(q, p_i)\}, \tag{1}$$

where p_i is a vertex belonging to A.

Given a set of query points Q, a set of meeting points S, and a destination d, let A be an arbitrarily-selected set of meeting points ($A \subseteq S$ and $|A| \leq k$). The local travel cost (LTC) and the connection travel cost (CTC) of A are defined by Equations 2 and 3, respectively.

$$LTC(A) = \alpha \cdot \sum_{q \in Q} d(q, A) \tag{2}$$

$$CTC(A) = \beta \cdot \sum_{p \in A} sd(p, d) \tag{3}$$

Here, α and β represent the energy consumption (or greenhouse-gas emissions) per unit distance for individual travel and collective travel, respectively. By combining Equations 2 and 3, the global travel cost (GTC) of subset A is defined by

$$GTC(A) = LTC(A) + CTC(A). \tag{4}$$

2.3 Problem Definition

The CTP problem is defined as follows. Given a set of query points Q, a set of meeting points S, a destination d, and an integer threshold k ($1 \leq k \leq \min\{|S|, |Q|\}$), the Collective Travel Planning (CTP) query finds a subset A ($A \subseteq S \wedge |A| \leq k$) such that $GTC(A)$ is minimized, i.e., $\forall A' \subseteq S(|A'| \leq k \Rightarrow GTC(A) \leq GTC(A'))$.

Extensions: We future extend the CTP query into two practical scenarios. First, the connection travel cost of meeting point p is proportional to the number $p.t$ of travelers that meet at p. For each point p, we need $\lceil \frac{p.t}{c} \rceil$ taxis, where c is the capacity of a taxi. For example, assuming the capacity of a taxi is 5 and 7 people are waiting at p, we need 2 taxis there. Second, if a traveler is very close to the destination, he/she can go to the destination directly. We define a series of new practical distance measures to capture this aspect and further extend the developed algorithms accordingly in Section 6. The theoretical approximation ratio does not work here, and we conduct extensive experiments to show that our extension is usable in the new scenarios.

3 Problem Analysis

The CTP problem is related to the *metric k uncapacitated facility location problem* (k-UFL) [17], where the aim is to find a minimum cost solution to connect a set of cities to a set of open facilities. More precisely, we are given a set of cities $\mathcal{C}$, a set of facility locations $\mathcal{F}$, a transportation cost c_{ij} for connecting city i to facility j, and a cost f_j of opening facility $j \in \mathcal{F}$. The goal is to identify a subset of $\mathcal{F}$ with at most k facilities and to connect each city to an open facility so that the total cost is minimized. In the metric version of the k-UFL problem, we assume that the connection costs are metric, meaning that they are non-negative, symmetric, and satisfy the triangle inequality.

The study of the k-UFL problem dates back to the early 90s [19]. Guha and Khuller [10] proved that k-UFL cannot be approximated within a factor of 1.463, if we

assume $\mathbf{NP} \subseteq \mathrm{DTIME}[n^{O(\log\log n)}]$. Charikar et al. [3] gave the first constant factor approximation algorithm with approximation ratio 9.8. Later, Jain and Vazirani [17] improved this ratio to 6 using a primal-dual scheme and Lagrangian relaxation techniques. This was further improved to 4 by Jain et al. [16], using the Lagrangian Multiplier Preserving property of the greedy dual growth algorithms. These algorithms use either the prime-dual scheme or linear programming rounding techniques, and they are hard to implement in real applications. Another and more practical line of study of the k-UFL problem is based on *local search*. In a local search algorithm, we repeatedly add, drop, or swap a facility to reduce the total cost, and we stop when a local optimum is found. Devanur et. al. [7] offered a 5-approximation ratio based on local search. Later, Zhang [29] proved that if the algorithm can add, drop, or swap a constant number of facilities in an operation then the approximation ratio becomes $2+\sqrt{3}+\varepsilon$, for a small constant ε. However, if only one facility is allowed to be modified in an operation, Zhang's technique yields a ratio of $3+\sqrt{6}$, which is worse than Devanur's 5 approximation ratio.

Reduction to the k-UFL problem: The CTP problem can be reduced to the k-UFL problem. More precisely, given an instance of the CTP problem, we can construct a corresponding instance of the k-UFL problem as follows. We construct a city i_q in the k-UFL instance for each query point q in the CTP instance and a facility j_p for each meeting point p in the CTP instance. The cost of facility i_p serving city j_q is set to $c_{i_p j_q} = \alpha \cdot sd(q,p)$, which is the cost of transportation from query point q to meeting point p in the CTP problem. The cost for opening facility j_p is set to be $f_{j_p} = \beta \cdot sd(p,d)$, which is the cost of transportation from meeting point p to the destination d in the CTP problem. A solution to this instance of the k-UFL problem identifies a subset A of at most k facilities and connects each city i_q to a facility $j_q \in A$, such that that the total cost

$$\sum_{q\in Q} \alpha \cdot d(q,p) + \sum_{p\in A} \beta \cdot sd(p,d)$$

is minimized. Once the subset A is determined, the solution is obtained by connecting each city to its nearest facility in A. Thus, the k-UFL problem is to minimize the total cost

$$\alpha \cdot \sum_{q\in Q} d(q,A) + \beta \cdot \sum_{p\in A} sd(p,d) = GTC(A),$$

which is the objective function in the CTP problem. In other words, we can take any instance of the CTP problem and transform it to an instance of the k-UFL problem. We can then solve that problem and transform the solution into a solution to the original CTP problem. This implies that we can achieve a local search algorithm for the CTP problem with an approximation ratio of 5 if only a single meeting point can be modified in an operation and an approximation ratio of $2+\sqrt{3}+\varepsilon$ if multiple meeting points can be modified.

In the CTP query, we only modify a single meeting point in an operation to achieve higher query efficiency. A detailed discussion is given in Section 5.4.

Hardness of the CTP Problem: To show the hardness of the CTP problem, we need to reduce a known hard problem to the CTP problem. Based on above analysis, an obvious choice is the k-UFL problem. However, it is not clear that this direction of the reduction is true. A subtle difference between the CTP problem and the k-UFL problem is that the k-UFL allows arbitrary facility opening costs, while the CTP problem requires the distances between the meeting points and the destination to satisfy the triangle inequality. Therefore, if we map the opening cost of a facility in the k-UFL problem to the distance between its corresponding meeting point and the destination in the CTP problem, the shortest path between the meeting point and the destination may go through some other meeting points, which violates the facility cost assignments of the k-UFL problem. So we cannot straightforwardly apply the hardness result of the k-UFL problem to the CTP problem.

Instead, we reduce the *B-vertex cover problem* to the CTP problem. In the B-vertex cover problem, we are given a graph $G=(V,E)$ with the degree of each node bounded by a constant B, and the goal is to find a minimum vertex cover, which is a minimum subset of vertices such that each edge in E is covered by at least one vertex in the cover. The B-vertex cover was shown to be Max SNP-hard by Papadimitriou and Yannakakis [22]. We present a L-reduction from the B-vertex cover problem ti the CTP problem, which implies that the CTP problem is also Max SNP-hard.

Theorem 1: The CTP problem is Max SNP-hard.

Max-SNP-hardness essentially implies that the CTP problem also cannot be approximated within a factor of $1+\varepsilon$, unless P=NP, for some small constant ε. The proof of Theorem 1 follows the same framework as the proof of Theorem 1 in reference [11].

Proof of Theorem 1: To design an L-reduction, we assume that there is an polynomial algorithm $\mathcal{A}$ that solves the CTP problem with approximation ratio $1+\frac{\varepsilon}{1+B}$, and we will show that this algorithm solves the B-vertex cover problem with approximation ratio $1+\varepsilon$.

Given an instance of the B-vertex cover $G=(V,E)$, we construct an instance of the CTP problem as follows. We construct a query point q_e for each edge e in E and a meeting point p_v for each vertex v in V. Each query point q_e is connected to each meeting point p_v. The distance $d(q_e,p_v)$ is set to 1 if edge e is incident to vertex v and 2, otherwise. We also construct a destination point d and connect each meeting point to d with distance 1. By this setup, the shortest path from a meeting point to d is the edge that connects them. Finally, we assign energy consumption parameters $\alpha=1$ and $\beta=\frac{|E|}{Bs}$, where s is the size of the optimal vertex cover. Here we assume that the algorithm knows s in advance, which is not realistic. We will show how to remove this assumption later.

We claim that there is a solution to the CTP problem with global travel cost $|E|+\frac{|E|}{B}$. Recall that the algorithm outputs a subset A of the meeting points, which corresponds to a subset A of the vertices in the B-vertex cover instance. We set A to be the subset of meeting points that corresponds to a minimum

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2015.2509998, IEEE Transactions on Knowledge and Data Engineering

vertex cover of G. We have $|A| = s$. By the property of a vertex cover, each edge has a vertex in A, and it follows that each query point can be assigned to a meeting point within distance 1. Thus, the local travel cost for each point is $\alpha = 1$, and the total local travel cost is $|E|$. We also note that the connection travel cost for each meeting point in A is $\beta = \frac{|E|}{Bs}$ and that the total connection travel cost is $s \cdot \frac{|E|}{Bs} = \frac{|E|}{B}$. Therefore the global cost is $|E| + \frac{|E|}{B}$, and the claim holds.

The claim implies that the optimal solution for the CTP problem achieves a global travel cost at most $|E| + \frac{|E|}{B}$. Since the algorithm $\mathcal{A}$ solves the CTP problem with approximation ratio $1 + \frac{\varepsilon}{1+B}$, it follows that $\mathcal{A}$ provides a solution to the CTP problem with global travel cost at most

$$\left(1 + \frac{\varepsilon}{1+B}\right) \cdot \left(|E| + \frac{|E|}{B}\right) = |E| + (1+\varepsilon)\frac{|E|}{B}.$$

Let $r = |A|$ be the number of meeting points returned by $\mathcal{A}$, and let l be the set of query points that are assigned to a meeting point with distance 2. Then the local travel cost is $|E|+l$, and the connection travel cost is $r \cdot \frac{|E|}{Bs}$. It follows that

$$|E| + (1+\varepsilon)\frac{|E|}{B} \geq |E| + l + r\frac{|E|}{Bs},$$

which implies that

$$(1+\varepsilon)\frac{|E|}{B} \geq l + r\frac{|E|}{Bs} \geq (l+r)\frac{|E|}{Bs}.$$

The last inequality holds because a vertex cover exists with size s and the degree of each vertex is bounded by B; thus, the number of edges $|E|$ is at most Bs. We have

$$(1+\varepsilon)s \geq l + r. \quad (5)$$

For the final vertex cover, we first select the vertices that correspond to the meeting points in A. Note that there are l query points that are assigned to a meeting point with distance 2, so there are exactly l vertices that are not covered by A. We will select a vertex for each of the l vertices. By equation (5), the total size of the vertex cover is $l + r \leq (1+\varepsilon)s$, which is within a factor $(1+\varepsilon)$ of the size of the optimal vertex cover. This proves that the resulting vertex cover is an ε-approximation of the optimal vertex cover.

Finally, as we do not know s in advance, we can run the above reduction for all possible values of s in the range from 1 to $|V|$ to produce $|V|$ vertex covers, one of which has size at most $(1+\varepsilon)s$. Thus, we can return the vertex cover with the minimum size, which is guaranteed to have size at most $(1+\varepsilon)s$. □

4 Exact Algorithm

4.1 Basic Idea

Exact search is a straightforward method to compute the CTP query. Given a set of meeting points S, a set of query locations Q, an integer threshold k, and a destination d, we select and evaluate each potential subset A ($A \subseteq S \wedge |A| \leq k$) iteratively. A pair of an upper and a lower bound on the global travel cost is developed to prune the search space. By combining the computation results, the subset with the minimum cost is found.

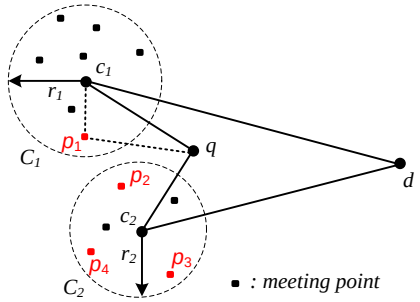


Fig. 2. An example of the exact search when $k = 4$

To enhance query efficiency, we assume that the meeting points in S are indexed by the iDistance indexing method [15]. The well-known iDistance method enables efficient computation of nearest neighbors in spatial networks. Other spatial indexes can also be adopted. In iDistance, a data partition/clustering method (e.g., k-means, k-medoids) is used to group the points into m clusters. For each cluster C_i, a reference point c_i is selected. Then, we compute and record the network distance between c_i and every meeting point $p \in C_i$. A B^+-tree is adopted to index the points using the network distance to the corresponding reference point as a key. In our implementation, to find a suitable number m of clusters and to achieve high performance, we conducted extensive experiments when establishing the iDistance index (as [15]).

4.2 Upper and Lower Bounds

To prune the search space, we define and use the upper and lower bounds of the global travel cost $GTC(A)$.

Consider the example in Figure 2. The meeting points are grouped into clusters C_1 and C_2, and points c_1 and c_2 are the corresponding reference points. Point $q \in Q$ is a query point, and d is the destination. Let $k = 4$. Subset $A = \{p_1, p_2, p_3, p_4\}$ is a 4-subset selected from S, where p_1 belongs to C_1, and p_2, p_3, and p_4 belong to C_2. Next, we estimate the lower and upper bounds of the network distance between query point q and meeting point p_1 based on the triangle inequality of the shortest-path distance. The triangle inequality in spatial networks is represented as follows.

$$sd(v_1, v_2) + sd(v_2, v_3) > sd(v_1, v_3)$$

$$sd(v_1, v_2) - sd(v_2, v_3) < sd(v_1, v_3)$$

Here, v_1, v_2, and v_3 are vertices in $G.V$, and any one of them is not on the shortest path between the others. Otherwise, we may have that $sd(v_1, v_2) + sd(v_2, v_3) = sd(v_1, v_3)$ and $sd(v_1, v_2) - sd(v_2, v_3) = sd(v_1, v_3)$.

In Figure 2, meeting point p_1 belongs to cluster C_1, and c_1 is its reference point. According to the triangle inequality, we have the following inequalities.

$$sd(q, p_1) < sd(q, c_1) + sd(c_1, p_1)$$

and

$$\begin{cases} sd(q, p_1) > sd(q, c_1) - sd(c_1, p_1) \\ sd(q, p_1) > sd(c_1, p_1) - sd(q, c_1) \end{cases}$$

$$\Rightarrow sd(q,p_1) > |sd(q,c_1) - sd(c_1,p_1)|$$

According to Equation 1, the lower and upper bound of the minimum network distance between query point q and subset A are computed as follows, respectively.

$$d(q,A) = \min_{p\in A}\{sd(q,p)\} > \min_{p\in A}\{|sd(q,c_i) - sd(c_i,p)|\}$$

and

$$\min_{p\in A}\{|sd(q,c_i) - sd(c_i,p)|\} > \min_{C_i\cap A\neq\varnothing}\{|sd(q,c_i) - r_i|\}$$

$$\Rightarrow d(q,A) > \min_{C_i\cap A\neq\varnothing}\{|sd(q,c_i) - r_i|\} = d(q,A).lb$$

$$d(q,A) = \min_{p\in A}\{sd(q,p)\} < \min_{p\in A}\{sd(q,c_i) + sd(c_i,p)\}$$

and

$$\min_{p\in A}\{sd(q,c_i) + sd(c_i,p)\} < \min_{C_i\cap A\neq\varnothing}\{sd(q,c_i) + r_i\}$$

$$\Rightarrow d(q,A) < \min_{C_i\cap A\neq\varnothing}\{sd(q,c_i) + r_i\} = d(q,A).ub$$

Here, c_i is the reference point of cluster C_i, and r_i is the radius of C_i (network distance from c_i to the cluster boundary). Thus, we have that $r_i > sd(c_i,p)$, and we can replace $sd(c_i,p)$ by r_i in the inequalities.

For the example in Figure 2, the values of $d(q,A).lb$ and $d(q,A).ub$ are computed as follows.

$$d(q,A).lb = \min\{(|sd(q,c_1) - r_1|),(|sd(q,c_2) - r_2|)\}$$

$$d(q,A).ub = \min\{(sd(q,c_1) + r_1),(sd(q,c_2) + r_2)\}$$

According to Equation 2, the lower and upper bounds of local travel cost $LTC(A)$ are defined as follows.

$$LTC(A).lb = \alpha \cdot \sum_{q\in Q} d(q,A).lb \tag{6}$$

$$LTC(A).ub = \alpha \cdot \sum_{q\in Q} d(q,A).ub \tag{7}$$

Then, we estimate the lower and upper bounds of the connection travel cost $CTC(A)$ in Equations 8 and 9, respectively.

$$\begin{cases} CTC(A) = \beta \cdot \sum_{p\in A} sd(p,d) \\ sd(p,d) \geq \min_{C_i\cap A\neq\varnothing}\{sd(c_i,d) - r_i\} \end{cases}$$

$$\Rightarrow \sum_{p\in A} sd(p,d) \geq \sum_{p\in A} \min_{C_i\cap A\neq\varnothing}\{sd(c_i,d) - r_i\}$$

$$\Rightarrow CTC(A).lb = \beta \cdot |A| \cdot \min_{C_i\cap A\neq\varnothing}\{sd(c_i,d) - r_i\} \tag{8}$$

$$\begin{cases} CTC(A) = \beta \cdot \sum_{p\in A} sd(p,d) \\ sd(p,d) \leq \max_{C_i\cap A\neq\varnothing}\{sd(c_i,d) + r_i\} \end{cases}$$

$$\Rightarrow \sum_{p\in A} sd(p,d) \leq \sum_{p\in A} \max_{C_i\cap A\neq\varnothing}\{sd(c_i,d) + r_i\}$$

$$\Rightarrow CTC(A).ub = \beta \cdot |A| \cdot \max_{C_i\cap A\neq\varnothing}\{sd(c_i,d) + r_i\} \tag{9}$$

Here, p is a meeting point in subset A ($|A| \leq k$). For all $p \in A$, we have that $sd(p,d) \geq \min_{C_i\cap A\neq\varnothing}\{sd(c_i,d) - r_i\}$ and $sd(p,d) \leq \max_{C_i\cap A\neq\varnothing}\{sd(c_i,d) + r_i\}$.

For the example in Figure 2, the lower and upper bounds of connection travel cost $CTC(A)$ are computed as follows.

$$CTC(A).lb = \beta \cdot \min\{(sd(c_1,d) - r_1),(sd(c_2,d) - r_2))\}$$

$$CTC(A).ub = \beta \cdot \max\{(sd(c_1,d) + r_1),(sd(c_2,d) + r_2))\}$$

By combining Equations 6 and 8 and 7 and 9, respectively, the lower and upper bounds of the global travel cost $GTC(A)$ are computed as follows.

$$GTC(A).lb = LTC(A).lb + CTC(A).lb \tag{10}$$

$$GTC(A).ub = LTC(A).ub + CTC(A).ub \tag{11}$$

To find the subset with the minimum global travel cost, we evaluate each potential subset A. Among all scanned subsets, we define a global upper bound UB as

$$UB = \min_{A\in S}\{GTC(A).ub\}, \tag{12}$$

where S contains all scanned subsets. For a subset A, if $GTC(A).lb$ exceeds the global upper bound UB, A cannot be the subset with the minimum global travel cost; thus, A can be pruned safely. Otherwise, we match each query point $q \in Q$ to its closest meeting point $p \in A$, and compute the exact value of $GTC(A)$.

4.3 Algorithm

Algorithm 1: Exact Algorithm

Data: meeting point set S, query point set Q, destination d, threshold k
Result: subset A ($A \subseteq S \wedge |A| \leq k$) with the minimum $GTC(A)$

```
1  H ← ∅; UB ← +∞; i = 1;
2  while i ≤ k do
3      for each i-subset A ⊆ S do
4          compute GTC(A).lb and GTC(A).ub;
5          if GTC(A).ub < UB then
6              UB ← GTC(A).ub;
7          if GTC(A).lb ≤ UB then
8              H.push(A);
9      while H ≠ ∅ do
10         A ← H.min.pop();
11         if GTC(A).lb > UB then
12             record UB and the corresponding i-subset;
13             H.clear();
14             break;
15         compute GTC(A);
16         if GTC(A) < UB then
17             UB ← GTC(A);
18     i ← i + 1;
19 return UB and the corresponding subset;
```

The exact algorithm is detailed in Algorithm 1. Here, we evaluate all possible subsets of cardinality from 1 to k (lines 1–2). For each potential i-subset A, we compute and record its lower bound $GTC(A).lb$ and upper bound $GTC(A).ub$ (lines 3–4). If the value of $GTC(A).ub$ is less than that of UB, the value of UB is set to $GTC(A).ub$ (lines 5–6). If the value of $GTC(A).lb$ is no greater than that of UB, A is put into a heap H sorted according to the value of $GTC(A).lb$

(line 7–8). At each time, the i-subset A with the minimum $GTC(A).lb$ is selected from H (lines 9–10). If $GTC(A).lb$ exceeds UB, we record UB and the corresponding i-subset and break the loop (lines 11-14). Otherwise, we compute the exact value of $GTC(A)$, and we check whether $GTC(A)$ is less than UB. If so, UB is updated to the value of $GTC(A)$ (lines 15–17). After evaluating all subsets of cardinality from 1 to k, the subset with the minimum global travel cost is found (line 19).

4.4 Complexity Analysis

The exact algorithm evaluate each subset A ($A \subseteq S \wedge |A| \leq k$), and there is a total of $\sum_{i=1}^{k} \binom{|S|}{i} = \sum_{i=1}^{k} \frac{|S|!}{(|S|-i)!i!}$ possible combinations. For each subset A, we match query points to their closest meeting point $p \in A$. The time complexity is $O(|Q||A|) = O(|Q|)$ because $|A|$ is a constant no larger than k. Thus, the time complexity of the exact algorithm is

$$O\left(|Q| \sum_{i=1}^{k} \frac{|S|!}{(|S|-i)!i!}\right) = O\left(|Q| \cdot |S|^k\right).$$

The last equation uses Stirling's approximation.

5 Approximation Algorithm

5.1 Basic Idea

The CTP query is Max SNP-hard. To find the optimal result, the exact algorithm has to evaluate each subset A. For small threshold k, the exact algorithm is able to do so in interactive time. To compute the CTP query efficiently, we develop an approximation algorithm based on the *local search scheme* [7] [29] that can compute the query in polynomial time and that guarantees a 5-approximation ratio. Initially, we arbitrarily select a subset A ($|A| \leq k$) from S. Then we define three operations: add, $drop$, and $swap$. In a sequence of iterations, we randomly use an operation to optimize the global travel cost. The process terminates when no operation can produce a better result. The obtained result has a cost that is at most 5 times that of the global optimum [7]. The experimental results (refer to Section 7) show that algorithm achieves greatly improved running time and that the costs of the results are close to those of the optimal results (i.e., less than 1.15 times larger).

In Section 5.2, we introduce the operations as well as the pruning rules during query processing. The approximation algorithm is detailed in Section 5.3.

5.2 Operations and Pruning Rules

5.2.1 Operations

In this section, we introduce three types of operations and corresponding pruning rules. The approximation algorithm initially selects an arbitrary subset A ($|A| \leq k$) from S and matches each traveler $q \in Q$ to its closest meeting point $p \in S$ and computes the exact value of $GTC(A)$ (refer to Equation 4). Then, to optimize the global travel cost, the approximation algorithm uses three operations:

- ***add***(p): add a new meeting point $p \in (S \setminus A)$ to A if $|A| < k$;
- ***drop***(p): drop a meeting point p from A if $|A| > 1$;
- ***swap***(p, p'): swap a meeting point $p \in A$ with another meeting point $p' \in (S \setminus A)$.

We repeatedly apply a randomly selected operation to improve the global travel cost by a factor of $1 + \varepsilon$, where ε is an arbitrary small constant. The search process terminates when no new operation can produce a better result. The main reason that we compare the ratio $GTC(A)/GTC(A')$ with $1 + \varepsilon$ is to make sure the algorithm runs in polynomial time. If each operation improves the total cost by at least a factor of $(1 + \varepsilon)$, the number of operations is at most $log_{1+\varepsilon} W$, where W is the maximum possible global travel cost. If we do not impose the $(1+\varepsilon)$ constraint, each operation may result in a very small improvement, and the number of operations could be as large as W. Note that we only need log W bits to represent the cost, so this is not a polynomial time algorithm.

5.2.2 Add Operation

Figures 3(a), 3(b), and 3(c) illustrate examples of the add, $drop$, and $swap$ operations, where q_1, q_2, q_3, q_4, and q_5 are query points and d is the destination. In Figure 3(a), $A = \{p_1, p_2\}$ is a subset of meeting points, and query points q_1, q_2, q_3 are matched to p_1, and q_4 and q_5 are matched to p_2. According to Equation 4, the global travel cost $GTC(A)$ is computed as follows.

$$GTC(A) = \alpha \cdot (\sum_{i=1}^{3} sd(q_i, p_1) + \sum_{j=4}^{5} sd(q_j, p_2)) + \beta \cdot (sd(p_1, d) + sd(p_2, d))$$

Then we add a new meeting point p_3 to A and get a new subset $A' = \{p_1, p_2, p_3\}$. As each query point is matched to its nearest meeting point, query points q_1 and q_2 are matched to p_1, and q_3 and q_4 are matched to p_3, and q_5 is matched to p_2. The global travel cost $GTC(A')$ is computed as follows.

$$GTC(A') = \alpha \cdot (\sum_{i=1}^{2} sd(q_i, p_1) + sd(q_5, p_2) + \sum_{j=3}^{4} sd(q_j, p_3)) + \beta \cdot (sd(p_1, d) + sd(p_2, d) + sd(p_3, d))$$

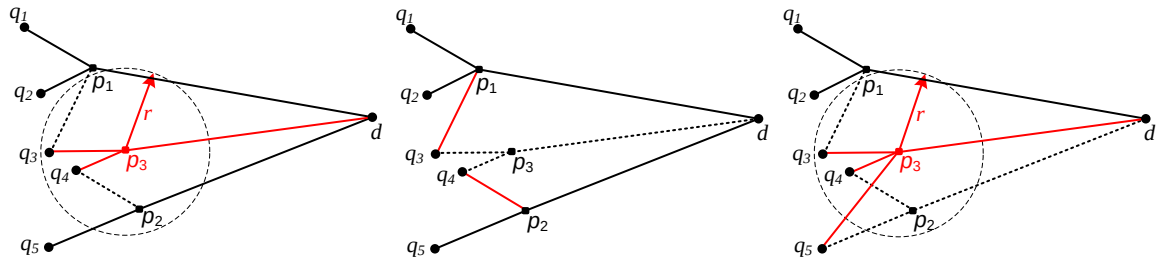
If the cost is improved by a factor of $(1 + \varepsilon)$ (i.e., $\frac{GTC(A)}{GTC(A')}$ exceeds $(1 + \varepsilon)$), operation $add(p_3)$ is valid. Otherwise, the operation is invalid, and p_3 is not added to A.

Each time when conducting the operation $add(p')$, the existing matching of each query point $q \in Q$ to meeting points in A has to be updated. To accelerate the rematching process, we propose the following pruning rule.

Given a subset A and the matching between Q and A, we define a search radius r as follows.

$$r = \max_{q \in Q, p \in A} \{sd(q, p)\} \quad (13)$$

Thus, r is the maximum network distance between a query point $q \in Q$ and a meeting point $p \in A$. When conducting the operation $add(p')$, we only need to rematch the query points



(a) add (b) $drop$ (c) $swap$

Fig. 3. Examples of add, $drop$, and $swap$ operations

covered by region $C(p', r)$, the circular region with center p' and radius r.

As an example in Figure 3(a), a new meeting point p_3 is added to $A = \{p_1, p_2\}$ ($A' = \{p_1, p_2, p_3\}$). We compute r according to Equation 13. To establish a new matching between each query point $q \in Q$ and each meeting point $p \in A'$, we only need to rematch the query points covered by the region $C(p_3, r)$. In Figure 3(a), query points q_3 and q_4 are covered by $C(p_3, r)$, and are rematched. Because $sd(q_3, p_1) > sd(q_3, p_3)$ and $sd(q_4, p_2) > sd(q_4, p_3)$, both q_3 and q_4 are matched to p_3. The matchings of other query points remains unchanged.

***Lemma 1**: When adding a new meeting point p' to A, the matchings of the query points outside the region $C(p', r)$ are unaffected.*

Proof: For a query point q outside region $C(p', r)$, we have that $sd(q, p') > r$. Assume that p is the meeting point closest to q in A. We have that $\forall p'' \in A \setminus \{p\}$ $(sd(q, p) < sd(q, p''))$. According to the definition of r (Equation 13), we have that $sd(q, p) \leq r$. Thus, we have that $sd(q, p) \leq r < sd(q, p') \Rightarrow sd(q, p) < sd(q, p')$, so p is still the meeting point closest to q in A'.

5.2.3 Drop Operation

For the example in Figure 3(b), $A = \{p_1, p_2, p_3\}$ is a subset of meeting points. Initially, query points q_1 and q_2 are matched to p_1, q_3 and q_4 are matched to p_3, and q_5 is matched to p_2. We compute $GTC(A)$ according to Equation 4. Then the operation $drop(p_3)$ is conducted, and the meeting point p_3 is dropped from subset $A = \{p_1, p_2, p_3\}$ ($A' = \{p_1, p_2\}$). Each query point $q \in Q$ is matched to its closest meeting point $p \in A'$. As shown in Figure 3(b), query points q_1, q_2, and q_3 are matched to p_1, and q_4 and q_5 are matched to p_2. After that, we compute $GTC(A')$ and check the validity of operation $drop(p_3)$. If $\frac{GTC(A)}{GTC(A')} < 1+\varepsilon$, operation $drop(p_3)$ is valid, and meeting point p_3 can be dropped from A. Otherwise, $drop(p_3)$ is invalid.

Finding a valid meeting point $p \in A$ to drop is time-consuming. In the worst case, all points in A have to be checked. To improve performance, we propose a pruning rule to pre-check the validity of operation $drop(p)$. First, we define the connection travel cost of meeting point p as follows (refer to Equation 3).

$$CTC(p) = \beta \cdot sd(p, d) \tag{14}$$

Here, β represents the energy consumption per unit distance for collective travel.

***Lemma 2**: If $\frac{GTC(A)}{GTC(A)-CTC(p)} < 1 + \varepsilon$, operation $drop(p)$ is invalid.*

Proof: Assume that query points $\{q_1, q_2, ..., q_n\} \subseteq Q$ are matched to meeting point p initially. For any q_i, we have that $\forall p' \in A \setminus \{p\}$ $(sd(q_i, p) < sd(q_i, p'))$. If p is dropped from A, query points $\{q_1, q_2, ..., q_n\}$ will be matched to other meeting points, and their local travel costs will increase (refer to Equation 2). Thus, we have that $GTC(A) - CTC(p) < GTC(A')$. Therefore, $\frac{GTC(A)}{GTC(A')} < \frac{GTC(A)}{GTC(A)-CTC(p)}$. This in turn means that

$$\frac{GTC(A)}{GTC(A')} < \frac{GTC(A)}{GTC(A) - CTC(p)} < 1 + \varepsilon.$$

Therefore, operation $drop(p)$ is invalid.

For example, in Figure 3(b), if we want to drop meeting point p_3, we compute of $CTC(p_3)$ (refer to Equation 14), and pre-check the validity of $drop(p_3)$ by comparing the value of $\frac{GTC(A)}{GTC(A)-CTC(p_3)}$ to $1 + \varepsilon$. If $\frac{GTC(A)}{GTC(A)-CTC(p_3)} < 1 + \varepsilon$, operation $drop(p_3)$ is invalid and p_3 cannot be dropped from A. Otherwise, we rematch the query points that were matched to p_3 to their closest meeting points in A', respectively (i.e., q_3 to p_1, and q_4 to p_2), and compute the exact value of $GTC(A')$.

If the condition of $\frac{GTC(A)}{GTC(A)-CTC(p)} < 1 + \varepsilon$ does not hold, we will do further validity checking for operation $drop(p)$.

5.2.4 Swap Operation

The $swap$ operation can be viewed as a combination of a $drop$ and an add. However, a swap operation may be valid even when its drop and add operations are each invalid. Figure 3(c) gives an example of the $swap$ operation. Initially, subset $A = \{p_1, p_2\}$, and query points q_1, q_2, and q_3 are matched to meeting point p_1, and q_4 and q_5 are matched to p_2. Then we conduct operation $swap(p_2, p_3)$ and we get $A' = \{p_1, p_3\}$. Accordingly, we match the query points to the meeting points in A' (q_1 and q_2 to p_1 and q_3, q_4, and q_5 to p_3) and compute $GTC(A')$. The process of operation $swap(p_2, p_3)$ is detailed as follows.

First, we drop meeting point p_2 from A, and we label the query points that were matched to p_2 as "unmatched". For example, in Figure 3(c), when we drop p_2 from A, q_3 and q_4 are labeled as "unmatched". Then, we add meeting point p_3 to the subset $(A \setminus \{p_2\})$. We label the query points

covered by circular region (p_3, r) as "unmatched". As an instance, q_3 is labeled as "unmatched" (q_4 is already labeled). After that, we match all "unmatched" query points to their closest meeting points in A', and the matching of other query points remains unchanged. For example, in Figure 3(c), q_3, q_4, and q_5 are matched to p_3, and the matchings of q_1 and q_2 remains unchanged. Finally, we compute $GTC(A')$ according to Equation 4 and compare $\frac{GTC(A)}{GTC(A')}$ to $(1+\varepsilon)$ to check the validity of the $swap(p_2, p_3)$ operation.

5.3 Algorithm

Algorithm 2: Approximation Algorithm

Data: meeting point set S, query point set Q, destination d, threshold k
Result: subset A ($A \subseteq S \wedge |A| \leq k$) with the minimum $GTC(A)$

1 select a subset A from S;
2 compute $GTC(A)$;
3 **while** *true* **do**
4 **for** *each point* $p \in (S \setminus A)$ **do**
5 **if** $|A| < k$ **then**
6 $A' \leftarrow A.add(p)$;
7 $r \leftarrow \max_{q \in Q, p \in A}\{sd(q,p)\}$;
8 rematch query points in region $C(p, r)$;
9 compute $GTC(A')$;
10 **if** $\frac{GTC(A)}{GTC(A')} > 1+\varepsilon$ **then**
11 $A.add(p)$ is valid, and $A \leftarrow A'$;
12 continue;
13 break;
14 **for** *each pair* (p, p'), $p \in A$, $p' \in (S \setminus A)$ **do**
15 $A' \leftarrow A.swap(p, p')$;
16 rematch all "unmatched" query points;
17 compute $GTC(A')$;
18 **if** $\frac{GTC(A)}{GTC(A')} > 1+\varepsilon$ **then**
19 $A.swap(p, p')$ is valid, and $A \leftarrow A'$;
20 continue;
21 **for** *each point* $p \in A$ **do**
22 **if** $|A| > 1$ **then**
23 $A' \leftarrow A.drop(p)$;
24 compute $CTC(p)$;
25 **if** $\frac{GTC(A)}{GTC(A)-CTC(p)} > 1+\varepsilon$ **then**
26 rematch the query points that were matched to p;
27 compute $GTC(A')$;
28 **if** $\frac{GTC(A)}{GTC(A')} > 1+\varepsilon$ **then**
29 $A.drop(p)$ is valid, and $A \leftarrow A'$;
30 continue;
31 break;
32 **if** *add, swap, and drop are all invalid* **then**
33 **return** A *and* $GTC(A)$*;*

The approximation algorithm is detailed in Algorithm 2. Initially, we arbitrarily select a subset A ($A \subseteq S \wedge |A| \leq k$), and we match query points in Q to meeting points in A. Then, we compute the global travel cost $GTC(A)$ according to Equation 4 (lines 1–2). Next, we try to use three operations add, $swap$, and $drop$ to optimize the global travel cost.

add operation: if the size of subset A is less than k, when adding a point p to A, we first compute the value of search radius r (Equation 13). Then, we rematch the query points covered by circular region $C(p, r)$ to meeting points in A', and we compute $GTC(A')$ (lines 4–9). Next, we check the validity of operation $A.add(p)$ by comparing the value of $\frac{GTC(A)}{GTC(A')}$ to $(1+\varepsilon)$. If $A.add(p)$ is valid, we update A to A'. Otherwise, $A.add(p)$ is invalid and A remains unchanged (lines 10–13).

swap operation: we try to swap meeting point $p \in A$ with another point $p' \in (S \setminus A)$. According to Section 5.2.4, we find all "unmatched" query points and match them to meeting points in A', and we compute $GTC(A')$ (lines 14–17). Finally, we check the validity of $A.swap(p, p')$. If $A.swap(p, p')$ is valid, A is updated to A' (lines 18–20).

drop operation: if the size of A is greater than 1, when dropping a point p, we compute $CTC(p)$ (lines 21–24). Then, we precheck the validity of $A.drop(p)$ by comparing the value of $\frac{GTC(A)}{GTC(A)-CTC(p)}$ to $1+\varepsilon$. If $drop(p)$ is valid, we rematch the query points that were matched to p, and we compute $GTC(A')$ (lines 25–27). Finally, we recheck the validity of $A.drop(p)$ by comparing the value of $\frac{GTC(A)}{GTC(A')}$ to $1+\varepsilon$. If $A.swap(p)$ is valid, A is updated to A' (lines 28–31).

If no operations can optimize the global travel cost by a factor of $1+\varepsilon$, the search process terminates, and the current subset A and the value of $GTC(A)$ are returned (lines 32–33).

5.4 Complexity Analysis

Assume that C_0 is the global travel cost of the initially selected subset A and that C is the global optimum. The maximum number of operations m depends on the ratio of C_0 to C. The value of m is computed as follows.

$$C_0 \cdot (1+\varepsilon)^{-m} \leq C$$
$$\Rightarrow\ m = \lfloor log_{(1+\varepsilon)}\ \frac{C_0}{C} \rfloor$$

To find a valid operation, we check all possibilities of add, $drop$, and $swap$, which has time complexity $O(|Q|(|S| - |A|)) + O(|Q||A|) + O(|Q|(|S| - |A|)|A|) = O(|Q||S|)$ because $|A|$ is a constant no larger than k. The total number $\lfloor log_{(1+\varepsilon)}\ \frac{C_0}{C} \rfloor$ of operations is a constant. Thus, the time complexity of the approximation algorithm is $O(\lfloor log_{(1+\varepsilon)}\ \frac{C_0}{C} \rfloor |Q||S|) = O(|Q||S|)$.

In our setting, only a single meeting point can be modified in an operation, and the algorithm achieves an approximation ratio of 5. The experimental results show that the approximate results are generally very close to the global optimum (less than 1.05 times larger). If multiple meeting points can be modified, the time complexity of the approximation algorithm is $O(|Q|\binom{|S|-|A|}{n}) + O(|Q|\binom{|A|}{n}) + O(|Q|\binom{|S|-|A|}{n}\binom{|A|}{n}) = O(|Q| \cdot |S|^n)$, where n is the number of modified meeting points in an operation. Although this can be done with the slightly better approximation ratio $2+\sqrt{3}+\varepsilon$, the query efficiency is affected badly.

6 EXTENSIONS

We future extend the CTP query to include two practical scenarios. First, it is of interest to consider connection travel costs of meeting points that are dependent on the number of travelers that travel from each meeting point to the destination. A new practical connection travel cost $CTC_p(A)$ is defined as:

$$CTC_p(A) = \beta \cdot \sum_{p \in A} \lceil \frac{p.t}{c} \rceil \cdot sd(p,d). \quad (15)$$

Here, $p.t$ is the number of travelers at meeting point p, and c is the capacity of a shuttle bus (collective travel). By combining Equations 2 and 15, the practical global travel cost $GTC_p(A)$ is computed as

$$GTC_p(A) = LTC(A) + CTC_p(A). \quad (16)$$

Next, we extend the exact and approximation algorithms to support the new distance measure. In the exact algorithm (refer to Section 4), we estimate the lower and upper bounds of the practical connection travel cost $CTC_p(A)$ in Equations 17 and 18, respectively.

$$\begin{cases} CTC_p(A) = \beta \cdot \sum_{p \in A} \lceil \frac{p.t}{c} \rceil \cdot sd(p,d) \\ sd(p,d) \geq \min_{C_i \cap A \neq \varnothing} \{sd(c_i,d) - r_i\} \end{cases}$$

$$\Rightarrow \sum_{p \in A} \lceil \frac{p.t}{c} \rceil \cdot sd(p,d) \geq \sum_{p \in A} \lceil \frac{p.t}{c} \rceil \cdot \min_{C_i \cap A \neq \varnothing} \{sd(c_i,d) - r_i\}$$

$$\Rightarrow \sum_{p \in A} \lceil \frac{p.t}{c} \rceil \cdot sd(p,d) \geq \frac{|Q|}{c} \cdot \min_{C_i \cap A \neq \varnothing} \{sd(c_i,d) - r_i\}$$

$$\Rightarrow CTC_p(A).lb = \beta \cdot \frac{|Q|}{c} \cdot \min_{C_i \cap A \neq \varnothing} \{sd(c_i,d) - r_i\} \quad (17)$$

$$\begin{cases} CTC_p(A) = \beta \cdot \sum_{p \in A} \lceil \frac{p.t}{c} \rceil \cdot sd(p,d) \\ sd(p,d) \leq \max_{C_i \cap A \neq \varnothing} \{sd(c_i,d) + r_i\} \end{cases}$$

$$\Rightarrow \sum_{p \in A} \lceil \frac{p.t}{c} \rceil \cdot sd(p,d) \leq \sum_{p \in A} \lceil \frac{p.t}{c} \rceil \cdot \max_{C_i \cap A \neq \varnothing} \{sd(c_i,d) + r_i\}$$

$$\Rightarrow \sum_{p \in A} \lceil \frac{p.t}{c} \rceil \cdot sd(p,d) \leq (\frac{|Q|}{c} + |A|) \cdot \max_{C_i \cap A \neq \varnothing} \{sd(c_i,d) + r_i\}$$

$$\Rightarrow CTC_p(A).ub = \beta \cdot (\frac{|Q|}{c} + |A|) \cdot \max_{C_i \cap A \neq \varnothing} \{sd(c_i,d) + r_i\} \quad (18)$$

Here, p is a meeting point in subset A. Integer $p.t$ is the number of travelers at meeting point p, $|Q|$ is the total number of travelers, and integer c is the capacity of a shuttle bus. In the example in Figure 2, the lower and upper bounds of the practical connection travel cost $CTC_p(A)$ are computed as follows.

$$CTC_p(A).lb = \beta \cdot \frac{|Q|}{c} \cdot \min\{(sd(c_1,d) - r_1), (sd(c_2,d) - r_2))\}$$

$$CTC_p(A).ub = \beta \cdot \frac{|Q|}{c} \cdot \max\{(sd(c_1,d) + r_1), (sd(c_2,d) + r_2))\}$$

By combining Equations 6 and 17 and 7 and 18, the lower and upper bounds of the practical global travel cost $GTC_p(A)$ are computed as follows.

$$GTC_p(A).lb = LTC(A).lb + CTC_p(A).lb \quad (19)$$

$$GTC_p(A).ub = LTC(A).ub + CTC_p(A).ub \quad (20)$$

The exact algorithm with the practical connection travel cost is obtained by substituting Equations 17, 18, 19, and 20 into Algorithm 1.

In the approximation algorithm, the practical connection travel cost of meeting point p is updated as follows.

$$CTC_p(p) = \beta \cdot \lceil \frac{p.t}{c} \rceil \cdot sd(p,d) \quad (21)$$

Lemma 2 in the *drop* operation (refer to Section 5.2.3) is not affected by the new measure. Assume that query points $\{q_1, q_2, ..., q_n\} \subseteq Q$ are matched to meeting point p initially. For any query point $q_i \in \{q_1, q_2, ..., q_n\}$, we have that $\forall p' \in A \setminus \{p\}$ $(sd(q_i,p) < sd(q_i,p'))$. If meeting point p is dropped from A, query points $\{q_1, q_2, ..., q_n\}$ are matched to other meeting points $\{p_1, p_2, ..., p_m\} \subseteq A \setminus \{p\}$, and their local travel costs increase (refer to Equation 2). For meeting points $\{p_1, p_2, ..., p_m\}$, their practical connection travel costs also increase because they will serve more query points (refer to Equation 15). Thus, we have that $GTC_p(A) - CTC_p(p) < GTC_p(A')$. Therefore, $\frac{GTC_p(A)}{GTC_p(A')} < \frac{GTC_p(A)}{GTC_p(A) - CTC_p(p)}$. This in turn means that

$$\frac{GTC(A)}{GTC(A')} < \frac{GTC(A)}{GTC(A) - CTC(p)} < 1 + \varepsilon.$$

In that case, the operation $drop(p)$ is invalid. Therefore, Lemma 2 is not affected by the new measure.

The approximation algorithm with the practical connection travel cost is achieved by substituting Equations 17, 18, 19, 20, and 21 into Algorithm 2.

Notice that $CTC_p(A)$ and $GTC_p(A)$ are not metrics. Thus, the approximation ratio 5 is not valid. We conduct extensive experiments to verify the performance of the practical CTP query, and the experimental results show that the approximate results are also very close to the global optimum.

Second, it is of interest to study the scenario where travelers are allowed to proceed directly to the destination if they are closer to the destination than to any meeting point. We extend the approximation algorithm (Algorithm 2) to support this scenario. Initially, the destination can be viewed as a meeting point, and it is put into the meeting point set A. During the search process, the destination will not be removed from A by *drop* and *swap* operations. When the search terminates, each traveler is matched to his/her closest meeting point, and the travelers that are matched to the destination can go to the destination directly. In Figure 11, when $\varepsilon = 0.03$ (default value), the approximation algorithm can also achieve a very good approximation ratio (less than 1.2) and low CPU time (less than 140 ms for BRN and less than 300 ms for NRN).

7 EXPERIMENTS

We report on extensive experiments with real and synthetic data that offer insight into the properties of the developed algorithms.

Table I: Pre-Processing Time and Required Disk Space

Road Networks	Pre-Processing Time	Required Disk Space
BRN (28,342 vertices and 38,577 edges)	193 seconds	3.5 GB
NRN (175,813 vertices and 179,179 edges)	1145 seconds	125 GB

Table II: Parameter Settings

	NRN	**BRN**
Query point count $\|Q\|$	5,000–15,000/default 5,000	1,000–9,000/default 1,000
Meeting point count $\|S\|$	50–1,600/default 100	50–1,600/default 100
Integer threshold k	5–160/default 10	5–160/default 10
ε	0.03–0.15/default 0.03	0.03–0.15/default 0.03
shuttle bus capacity c	5–20/default 5	5–20/default 5

7.1 Settings

We use graphs extracted from two spatial networks, namely the Beijing Road Network (BRN) and the North America Road Network (NRN)[5]. The graphs are stored using adjacency lists. Query points are randomly selected vertices and the meeting points are generated according to random (default), uniform, and Gaussian distributions.

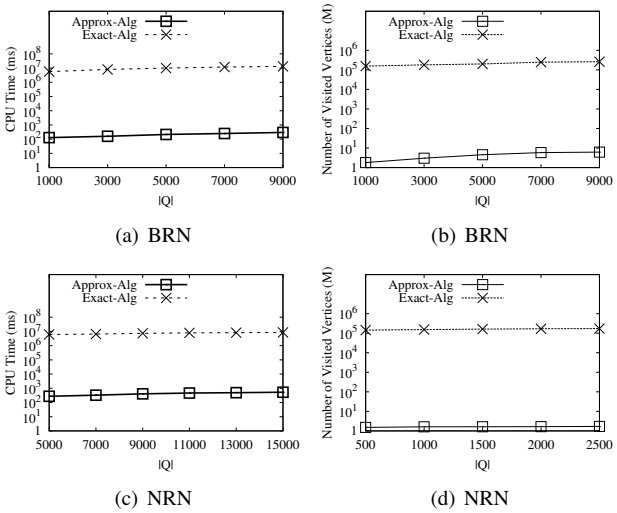
We pre-compute the all-pair shortest path distances using Dijsktra's algorithm [8] and store the pre-computed results on disk. Parallel computing techniques are easily adopted to accelerate the pre-computation because the all-pair shortest path computation can be viewed as $|G.V|$ single-source shortest path computations that are independent of each other. If we divide the task into N subtasks for N different workers and each worker computes $\frac{|G.V|}{N}$ single-source shortest path distances, the running time is expected to be improved by approximately a factor of N. The pre-computation algorithm was implemented in Java and run on a cluster with 16 servers. Each server has a 16-core CPU (2.40GHz) and 300GB memory. The pre-computation results are stored on disk. The pre-processing times and the required disk space are listed in Table I.

In the experiments, the graphs were memory resident when running Dijkstra's algorithm [8], as the memory occupied by BRN/NRN was less than 20MB. All algorithms were implemented in Java and run on a Windows 7 platform with an Intel i7-4770k processor (3.50GHz) and 16GB memory. Unless stated otherwise, experimental results are averaged over 20 independent trails with different query inputs. The main performance metrics are CPU time and the number of visited vertices. The number of visited vertices is used as a metric since it describes the number of data accesses.

The parameter settings are listed in Table II. In subsequent figures, the exact algorithm (Section 4) is denoted by "Exact-Alg," and the approximation algorithm (Section 5) is denoted by "Approx-Alg." We extend the query into two practical scenarios (Section 6). For the first (the connection travel cost is dependent on the number of travelers), the exact algorithm is denoted by "Exact-Alg-p1," and the approximation algorithm is denoted by "Approx-Alg-p1." In the second scenario (a traveler can go to the destination directly), the exact algorithm is denoted by "Exact-Alg-p2," and the approximation algorithm is denoted by "Approx-Alg-p2."

7.2 Effect of Query Point Count $|Q|$

First, we investigate the effect of the query point count $|Q|$ on the performance of the two algorithms with the default settings. Intuitively, a larger $|Q|$ causes more query points to be processed (being matched to their closest meeting point) and has a larger search space. Thus, the CPU time and the number of visited vertices are expected to be higher for both algorithms. However, from Figure 4, it is clear that with the help of its heuristic search strategy and pruning rules, the approximation algorithm outperforms the exact algorithm by almost a factor of 10^5 (for both CPU time and visited vertices). It is worth noting that (i) the number of visited vertices may exceed the number of vertices in the graph $|G.V|$ since a vertex may be visited several times by network expansions from different expansion centers; (ii) the CPU time is not fully aligned with the number of visited vertices. To prune the search space, the algorithms need more computational effort to maintain their bounds. In some cases, the increased computation cost may offset the benefits of the reduction in the number of visited vertices.



(a) BRN (b) BRN

(c) NRN (d) NRN

Fig. 4. Effect of query point count $|Q|$

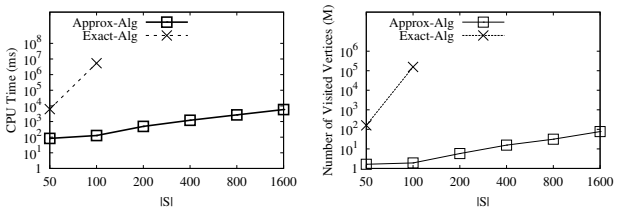
7.3 Effect of Meeting Point Count $|S|$

Figure 5 considers the effect of varying the meeting point count $|S|$. There exist $\sum_{i=1}^{k} \frac{|S|!}{(|S|-i)!i!}$ combination possibilities for subset A ($A \subseteq S \wedge |A| \leq k$). With a fixed value of k, a larger meeting point count $|S|$ leads to more computation. In Figure 5, the CPU time and the number of visited vertices for both algorithms increase with $|S|$, and the increase of the exact algorithm is much faster than that of the approximation algorithm. The CPU time and the number of visited vertices required by the exact algorithm are at least 10^2 times higher than those needed by the approximation algorithm.

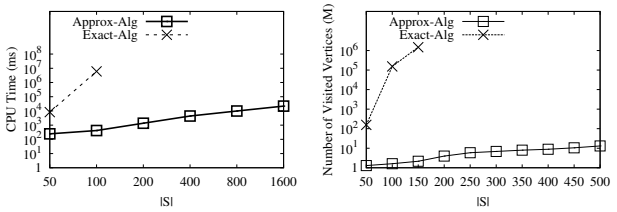
7.4 Effect of k

Next, we vary threshold k. For the exact algorithm, with a fixed value of the meeting point count $|S|$, a larger k leads

5. http://www.cs.utah.edu/~lifeifei/SpatialDataset.htm

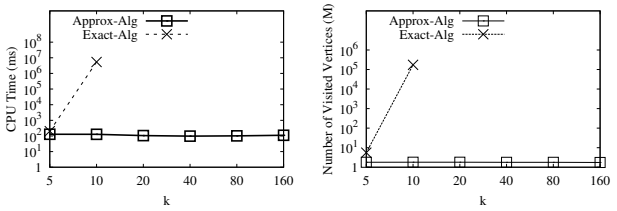


(a) BRN (b) BRN

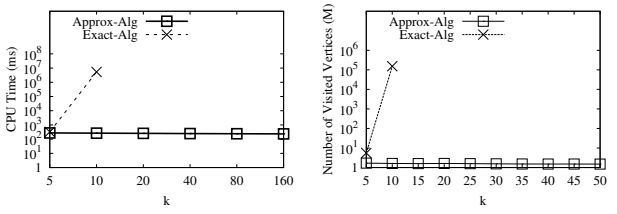


(c) NRN (d) NRN

Fig. 5. Effect of meeting point count $|S|$



(a) BRN (b) BRN



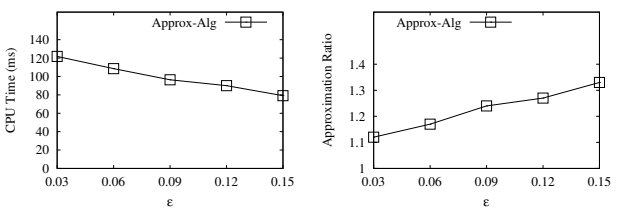
(c) NRN (d) NRN

Fig. 6. Effect of k

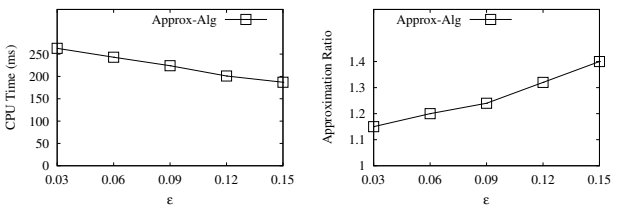
to more combination possibilities ($\sum_{i=1}^{k} \frac{|S|!}{(|S|-i)!i!}$). Intuitively, the larger k becomes, the larger the required search space becomes, and thus the required CPU time and the number of visited vertices are expected to increase correspondingly. On the other hand, the approximation algorithm is not sensitive to the value of k because the total number of operations is indirectly affected by the value of k (refer to Section 5.2). In Figure 6, the approximation algorithm outperforms the exact algorithm by factors of more than 10^5 in terms of both CPU time and the number of visited vertices when k exceeds 10.

7.5 Effect of ε

Figure 7 shows the effect of varying parameter ε on the efficiency and accuracy of the approximation algorithm. An operation (add, $drop$, and $swap$) will improve the global travel cost by a factor of $1+\varepsilon$; thus, the total number of operations $\lfloor log_{(1+\varepsilon)} \frac{C_0}{C} \rfloor$ is inversely proportional to the value of ε (refer to Section 5.2). A larger value of ε means fewer operations; thus, the query efficiency is improved and less CPU time is required. However, a larger value of ε also leads to a



(a) BRN (b) BRN

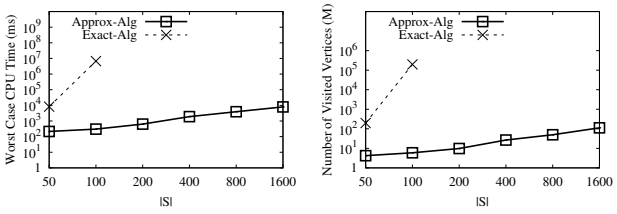


(c) NRN (d) NRN

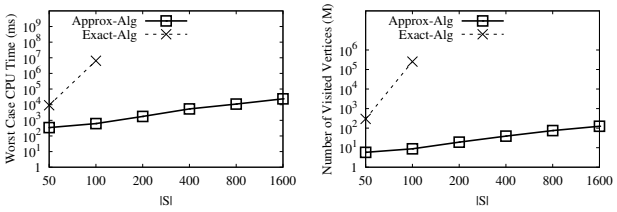
Fig. 7. Effect of ε

lower accuracy of query result. It is worth noting that when $\varepsilon = 0.03$, the approximation algorithm achieves a very good approximation ratio (less than 1.1 in BRN and less than 1.15 in NRN) and low CPU time (less than 120 ms in BRN and less than 260 in NRN).

7.6 Query Performance, Worst-Case Measurements



(a) BRN (b) BRN



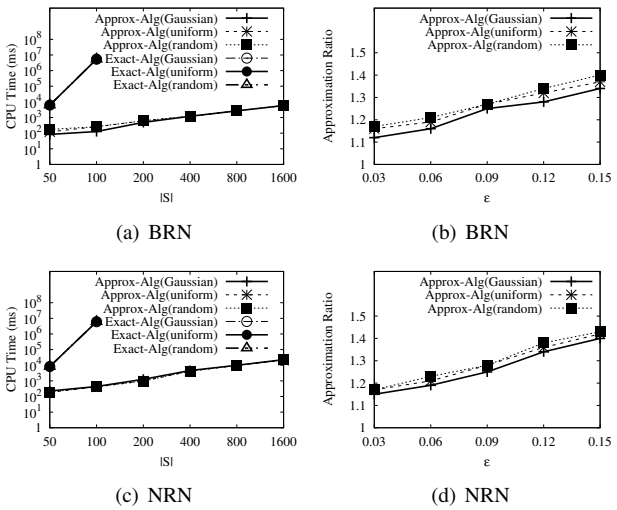
(c) NRN (d) NRN

Fig. 8. Query performance, worst-case measurements

In our settings, the worst-case measurement is the worst experimental result among 20 independent experiments. Figure 8 reports worst-case measurements for the CTP query. It is clear that the approximation algorithm is able to compute the CTP query in interactive time.

7.7 Effect of Meeting Point Distributions

We study the effect of meeting point distributions on query performance. We consider three types of distributions: Gaussian, uniform, and random. From Figure 9, it is clear that the time cost of the approximation algorithm is not affected significantly by meeting point distributions, and compared to the



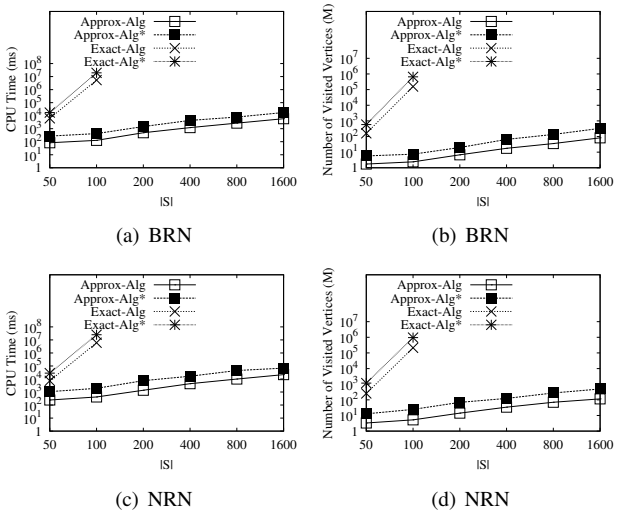
(a) BRN (b) BRN

(c) NRN (d) NRN

Fig. 9. Effect of meeting point distributions

random distribution, the Gaussian and uniform distributions may lead to a bit lower approximation ratios.

7.8 Effect of Pruning Techniques



(a) BRN (b) BRN

(c) NRN (d) NRN

Fig. 10. Effect of pruning techniques

We study the effect of pruning techniques in the exact and approximation algorithms. The exact algorithm without pruning techniques (without upper and lower bounds, refer to Section 4.2) is denoted by "Exact-Alg-p*," and the approximation algorithm without pruning techniques (Lemmas 1 and 2, refer to Sections 5.2.2 and 5.2.3) is denoted by "Approx-Alg-p*." In Figure 10, it is clear that with the help of pruning techniques, the performance of the exact algorithm is improved by 5–8 times, and the performance of the approximation algorithm is improved by at least an order of magnitude.

7.9 Performance of the Practical CTP Query

We study the performance (efficiency and effectiveness) of the practical CTP query processing. The results are shown in Figure 11. Compared to the original CTP query, practical CTP query processing takes longer to compute the upper and lower bounds, due to the more complex distance measures (refer to Section 6). The approximation algorithm can still outperform the exact algorithm by a factor of more than 10^5 in terms of both CPU time and the number of visited vertices. Neither of the algorithms is efficiency-sensitive to the value of c (shuttle-bus capacity). When $\varepsilon = 0.03$, the approximation algorithms can also achieve a very good approximation ratio (less than 1.2) and low CPU time (less than 140 ms for BRN and less than 300 ms for NRN).

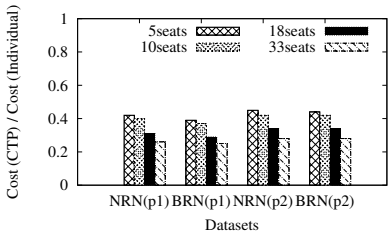


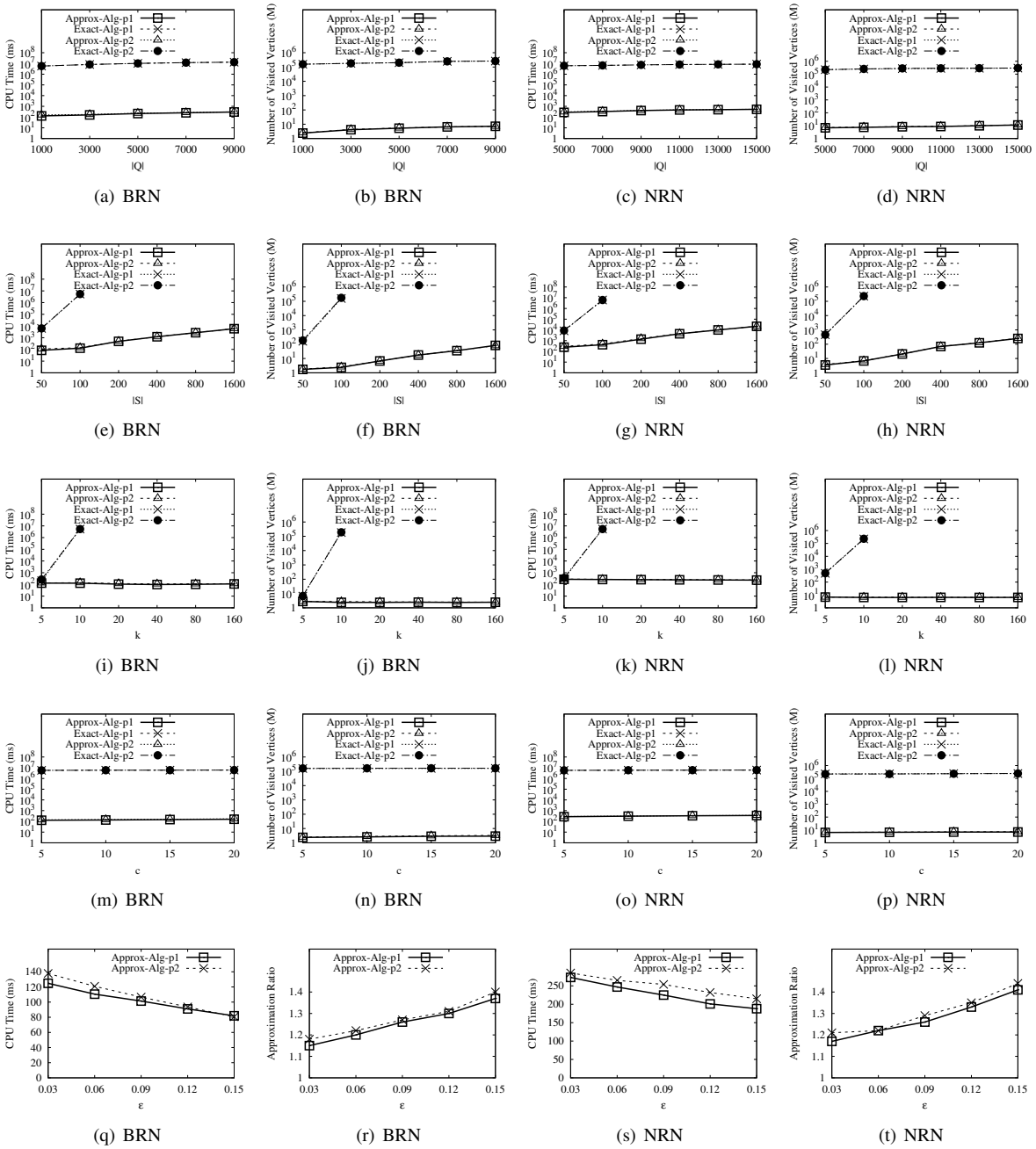
Fig. 12. Travel cost reduction

7.10 Travel Cost Reduction

We conduct a case study to gain insight into the travel cost reduction that can be achieved by using the CTP query. We select four types of vehicles for the collective travel: Toyota Corolla (5 seats, petrol cost: 6.3L/100km), Benz Minibus (10 seats, 11.3L/100km), Benz Minibus (18 seats, 12L/100km), and Yutong Bus (33 seats, 16L/100km). We assume that travelers drive the Toyota Corolla to meeting points. Figure 12 shows the travel cost reduction of using the CTP query. Compared to individual travel, the collective travel can reduce the cost by more than 60%.

8 Related Work

Group nearest neighbor [20] and aggregate nearest neighbor [21] queries are typical queries that have multiple sources and a single destination. They assume that each traveler goes to the destination individually and the queries aim to find the optimal location of the destination to minimize the travelers' global travel cost. A group travel planning query [14] extends the group nearest neighbor query to the multiple-destinations scenario, and it assumes that a group of travelers assemble at the first destination and then go together to the next destination. The group nearest group query [6] is another type of query with multiple sources and multiple destinations. It matches each traveler to their closest destination, and the query aims to find the optimal locations of the destinations to minimize the travelers' global travel cost. Unlike the existing studies, the Collective Travel Planning query has multiple query sources, multiple meeting points, and single destination. It can be viewed as a combination of the group nearest neighbor query and the group nearest group query. It assumes that a group of travelers meet at their closest meeting point and then go together to the destination, and its target is also to minimize the global travel cost.



(a) BRN (b) BRN (c) NRN (d) NRN

(e) BRN (f) BRN (g) NRN (h) NRN

(i) BRN (j) BRN (k) NRN (l) NRN

(m) BRN (n) BRN (o) NRN (p) NRN

(q) BRN (r) BRN (s) NRN (t) NRN

Fig. 11. Performance for the practical CTP query

The trajectory-based travel planning can be further divided into trajectory-to-object search and trajectory-to-trajectory search. In the trajectory-to-object search, queries aim to find objects spatially close to a query path according to some distance metrics. For example, the in-route nearest neighbor (IRNN) query [27] is designed for travelers following a fixed route. The path nearest neighbor (PNN) query [4] [23] [25] is an extension of the IRNN query that maintains an up-to-date path nearest neighbor result as the user is moving along a predefined route. Moreover, the path nearby cluster query [26] further extends the PNN query to find the POI clusters spatially close to a given path. In trajectory-to-trajectory search, queries retrieve the trajectories that have similar curve and are spatially close to a query trajectory. Travelers can use the travel history of other travelers to guide their own trips. Chen et al. study the trajectory similarity search in Euclidean space [5], and Shang et al. study the problem in spatial networks [24].

9 Conclusion and Research Directions

We propose and study a novel problem, the Collective Travel Planning (CTP) query that finds the lowest-cost travel route that connects multiple sources and a destination via at most k meeting points. The query is designed for ridesharing by a large population of travelers going to the same destination. The solution aims to offer societal and environmental benefits, such as reducing energy consumption and greenhouse gas emissions, enabling smarter and greener transportation, and

reducing traffic congestion. The CTP query is Max SNP-hard. To compute the query efficiently, an approximation algorithm was developed with a 5-approximation ratio. The performance of CTP query processing was investigated by means of extensive experiments on real and synthetic data.

Six interesting directions for future research exist. First, it is of interest to study the CTP query without the k constraint. The query then finds the least-cost travel route connecting multiple query sources and a destination via unconstrained meeting points. A key challenge is to design an approximation algorithm with a suitable approximation ratio to compute the query in interactive time. Second, it is of interest to use travel time in the query and take the travelers' transfer times into account, as this makes the problem more practical. The resulting query then aims to find the travel route with the minimum total travel time that connects multiple travelers and a destination, via at most k meeting points. Third, it is of interest to take the changes of travel costs of road segments into account and further extend the developed algorithms to dynamic spatial networks. Fourth, it is of interest to study a continuous CTP query for the scenario where one or more travelers fail to reach the meeting points on time and then may need to catch up with the group. Fifth, in the approximation algorithm, it is of interest to study how to identify a good initial answer efficiently, to further enhance the query efficiency. Sixth, to find the optimal solution, the exact algorithm should be conducted k times to test all possible values of k. It is of interest to study a more integrated exact algorithm to achieve a higher query efficiency.

REFERENCES

[1] R. Baldacci, V. Maniezzo, and A. Mingozzi. An exact method for the car pooling problem based on lagrangean column generation. *Operation Research*, 52:422–439, 2004.

[2] R. W. Calvo, F. de Luigi, P. Haastrup, and V. Maniezzo. A distributed geographic information system for the daily carpooling problem. *Computer Operation Research, Elsevier Science Ltd.*, 2004.

[3] M. Charikar, S. Guha, É. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem. In *STOC*, pages 1–10, 1999.

[4] Z. Chen, H. T. Shen, X. Zhou, and J. X. Yu. Monitoring path nearest neighbor in road networks. In *SIGMOD*, pages 591–602, 2009.

[5] Z. Chen, H. T. Shen, X. Zhou, Y. Zheng, and X. Xie. Searching trajectories by locations: an efficiency study. In *SIGMOD*, pages 255–266, 2010.

[6] K. Deng, S. W. Sadiq, X. Zhou, H. Xu, G. P. C. Fung, and Y. Lu. On group nearest group query processing. *IEEE Trans. Knowl. Data Eng.*, 24(2):295–308, 2012.

[7] N. Devanur, N. Garg, R. Khandekar, V. Pandit, A. Saberi, and V. Vazirani. Price of anarchy, locality gap, and a network service provider game. In *Internet and Network Economics*, pages 1046–1055. Springer, 2005.

[8] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Math*, 1:269–271, 1959.

[9] H. Gonzalez, J. Han, X. Li, M. Myslinska, and J. Sondag. Adaptive fastest path computation on a road network: A traffic mining approach. In *VLDB*, pages 794–805, 2007.

[10] S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. In *ACM-SIAM symposium on Discrete algorithms*, pages 649–657, 1998.

[11] S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of Algorithms*, 31(1):228–248, 1999.

[12] C. Guo, Y. Ma, B. Yang, C. S. Jensen, and M. Kaul. Ecomark: evaluating models of vehicular environmental impact. In *ACM GIS*, pages 269–278, 2012.

[13] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[14] T. Hashem, T. Hashem, M. E. Ali, and L. Kulik. Group trip planning queries in spatial databases. In *SSTD*, pages 259–276, 2013.

[15] H. Jagadish, B. Ooi, K.-L. Tan, C. Yu, and R. Zhang. idistance: An adaptive b+-tree based indexing method for nearest neighbour search. *TODS*, 30(2):364–397, 2005.

[16] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing lp. *JACM*, 50(6):795–824, 2003.

[17] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *JACM*, 48(2):274–296, 2001.

[18] S. Ma, Y. Zheng, and O. Wolfson. T-share: A large-scale dynamic taxi ridesharing service. In *ICDE*, pages 410–421, 2013.

[19] P. B. Mirchandani and R. L. Francis. *Discrete location theory*.

[20] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis. Group nearest neighbor queries. In *ICDE*, pages 301–312, 2004.

[21] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui. Aggregate nearest neighbor queries in spatial databases. *TODS*, 30(2):529–576, 2005.

[22] C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 229–234. ACM, 1988.

[23] S. Shang, K. Deng, and K. Xie. Best point detour query in road networks. In *ACM GIS*, pages 71–80, 2010.

[24] S. Shang, R. Ding, K. Zheng, C. S. Jensen, P. Kalnis, and X. Zhou. Personalized trajectory matching in spatial networks. *VLDB J.*, 23(3):449–468, 2014.

[25] S. Shang, B. Yuan, K. Deng, K. Xie, K. Zheng, and X. Zhou. Pnn query processing on compressed trajectories. *GeoInformatica*, 16(3):467–496, 2012.

[26] S. Shang, K. Zheng, C. S. Jensen, B. Yang, P. Kalnis, G. Li, and J.-R. Wen. Discovery of path nearby clusters in spatial networks. *IEEE Trans. Knowl. Data Eng.*, online first:1–14, 2014.

[27] S. Shekhar and J. S. Yoo. Processing in-route nearest neighbor queries: a comparison of alternative approaches. In *ACM GIS*, pages 9–16, 2003.

[28] B. Yang, C. Guo, C. S. Jensen, M. Kaul, and S. Shang. Stochastic skyline route planning under time-varying uncertainty. In *ICDE*, pages 136–147, 2014.

[29] P. Zhang. A new approximation algorithm for the k-facility location problem. *Theoretical Computer Science*, 384(1):126–135, 2007.

Shuo Shang is a professor of computer science at China University of Petroleum-Beijing. He was a research assistant professor at department of computer science, Aalborg University. He obtained his Ph.D. in computer science from University of Queensland. His research interest is efficient query processing in spatiotemporal databases. He has served as session chair and invited reviewer for many prestigious conferences and journals, including ICDE, TKDE, The VLDB Journal and ACM TIST.

Lisi Chen is a Ph.D. candidate with Nanyang Technological University. His research interests include geo-textual data management, spatial keyword query evaluation, and location based social networks.

Zhewei Wei is an associate professor at Renmin University of China. He obtained his Ph.D in Computer Science and Engineering from The Hong Kong University of Science and Technology in 2012. His research interests include streaming algorithms and data structures.

Christian S. Jensen is Obel Professor of Computer Science at Aalborg University, Denmark. His research concerns data management and data-intensive systems, and its focus is on temporal and spatiotemporal data management. Christian is an ACM and an IEEE fellow, and he is a member of the Academia Europaea, the Royal Danish Academy of Sciences and Letters, and the Danish Academy of Technical Sciences. He has received several national and international awards for his research. He is editor-in-chief of ACM Transactions on Database Systems (TODS) and was an editor-in-chief of The VLDB Journal from 2008 to 2014.

Ji-Rong Wen is a professor at Renmin University of China. His main research interest lies on Web big data management, information retrieval, data mining and machine learning. He was a senior researcher at MSRA, and he has 50+ U.S. patents in Web search and related areas. He is currently the associate editor of ACM Transactions on Information Systems (TOIS). He is an IEEE senior member.

Panos Kalnis is an associate professor at KAUST. He received his Diploma in Computer Engineering from the Computer Engineering and Informatics Department, University of Patras, and PhD from HKUST. His research interests include Database outsourcing and Cloud Computing, Mobile Computing, and Spatiotemporal and High-dimensional Databases. He serves on associate editors of TKDE, and The VLDB Journal.