Skyline Trips of Multiple POIs Categories

Saad Aljubayrin [1], Zhen He [2], and Rui Zhang [1]

[1] Department of Computing and Information Systems, The University of Melbourne
Melbourne, Australia
Aljubayrin@su.edu.sa
rui.zhang@unimelb.edu.au
[2] Department of Computer Science and Computer Engineering, Latrobe University
Melbourne, Australia
z.he@latrobe.edu.au

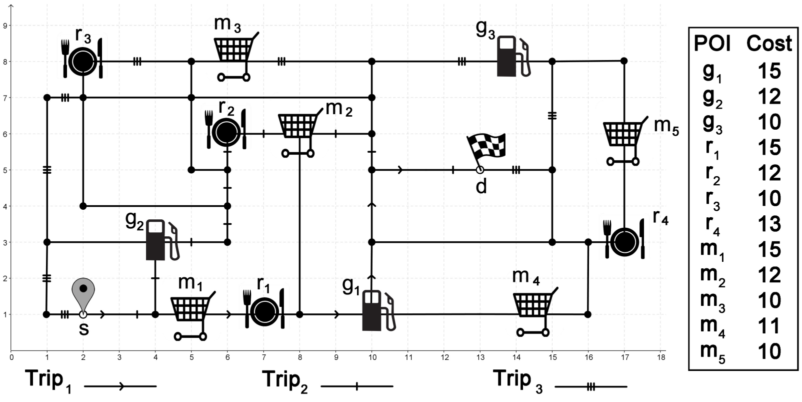
Abstract. In this paper, we introduce a new interesting path finding problem, which is the Skyline Trips of Multiple POIs Categories (*STMPC*) query. In particular, given a road network with a set of Points of Interest (POIs) from different categories, a list of items the user is planning to purchase and a pricing function for items at each related POI; find the skyline trips in term of both trip length and trip aggregated cost. This query has important applications in everyday life. Specifically, it assists people to choose the most suitable trips among the skyline trips based on two dimensions; trip total length and trip aggregated cost. We prove the problem is NP-hard and we distinguish it from existing related problems. We also proposed a framework and two effective algorithms to efficiently solve the *STMPC* query in real time and produce near optimal results when tested on real datasets.

Keywords: path finding, skyline paths, skyline trips, trip planing

1 Introduction

Over the past few decades spatial databases have been studied extensively, resulting in significant outcomes in areas such as spatial indexing, path finding and data modelling [5,11,8,21,28,31]. In this paper we focus on the path finding field and introduce a new interesting path finding problem, which is the Skyline Trips of Multiple POIs Categories *STMPC* query. In particular, given a road network graph G, source s and destination d, a set of n POIs categories $C = \{c_1, c_2, ...c_n\}$ with a set P of POIs in each category $c_i = \{p_1, p_2, ...p_n\}$, a list O of items the user is planning to purchase $O = \{o_1, o_2, ...o_n\}$ and a pricing function $f(o_i)$ for items at each related POI p_i; find the skyline trips $Sky(T) = \{t_1, t_2, ...t_n\}$ that each starts at s, pass through at least one POI p_i from each related category and ends at d, thus $t_i = \{s, p_1 \in c_1, p_2 \in c_2, ...p_n \in c_n, d\}$. $Sky(T)$ is the set of trips that are not dominated by any possible other trip in term of both trip length and trip cost aggregated from POIs creating the trip.

Since the Trip planing query (TPQ) studied in [19] is a special case of our problem, we would illustrate it first and distinguish it from the typical shortest path query. TPQ is a generalization of the Traveling Salesman problem (TSP) [9] and more inclusive than normal shortest path query. Specifically, a TPQ usually starts from a source point



POI	Cost
g_1	15
g_2	12
g_3	10
r_1	15
r_2	12
r_3	10
r_4	13
m_1	15
m_2	12
m_3	10
m_4	11
m_5	10

Fig. 1: Motivating example

and pass through a number of POIs in particular order and possibly restricted by some constraint (e.g. time, distance). In contrast, a typical shortest path query aims at finding the shortest path between source and destination by finding the smallest aggregated weight of road segments connecting the query points. For example, TPQ can be finding a path from a user's office that pass by an ATM, supermarket, restaurant and ending the trip at the user's home. While a shortest path query can be finding the path between two targets (e.g. ATM and supermarket) in a trip. The cost at each POIs is not considered in the TPQ, thus, there is only one optimal trip, which is the shortest trip that passes through at least one POI from each category.

The main difference between the TPQ query and the query studied in this paper is that, the *STMPC* query involves another optimality dimension, which is the trip total cost aggregated from POIs creating the trip. Specifically, while the TPQ query only finds the shortest trip, our query uses the trips length and cost to find the set of skyline trips. This results in the possibility of having multiple skyline optimal trips. We illustrate the *STMPC* query in the following example.

Figure 1 shows an example of a road network with source s and destination d and a number of POIs from three different categories; gas stations, restaurants and supermarkets. Based on the prices of items the user is planning to purchase and their quantity, we define the cost at each POI. When the user at s wants to visit one POI from each category on her way to d, there can be a large number of possible trips which all include one POI from each category. For instance, $Trip_1 = \{s, m_1, r_1, g_1, d\}$ has the shortest distance (15) with expensive cost (45), while $Trip_3 = \{s, r_3, m_3, g_3, d\}$ has the lowest cost (30) but with quite long distance (27). In addition, there can be a trip that is both short (17) and has a low cost (36) i.e. $trip_2 = \{s, g_2, r_2, m_2, d\}$ .Therefore, the user can choose the most preferable trip amongst the skyline trips, which are based on two dimensions; trip distance and trip total cost. In this example the trips $Trip_1, Trip_2$ and $Trip_3$ are the skyline trips because they are not dominated by any other possible trip.

Motivated by scenarios such as the previous example, we formulated the *STMPC* query and proposed two effective algorithms to efficiently answer the query in real time and produce near optimal results. Our first algorithm is generally based on defining a new weight for each POI, where this weight is a combination of both POI cost and POI distance from query points. Next, we perform a number of iterations, where in each iteration we change the weight of the two dimensions (cost and distance) in order to get the skyline candidates from each category. Our second algorithm does not only consider the distance between each POI and the query end points, it also considers clustered POIs using a suitable data structure (e.g. Quadtree, Rtree). This results in more accurate skyline trips especially if some POIs are clustered in geographical spots.

In this paper we also proposed a framework to estimate the network distance between POIs and query points. This is because using the Euclidean distance in road network does not usually provide accurate measurement while using the exact network distance between all POIs and query points would be an expensive task at the query time. Our framework is based on precomputing the network distance between POIs and a group of geographical spots in the network and then using this distance to estimate the actual network distance.

We make the following contributions:

1. We introduce the *STMPC* query, which is a novel path finding problem and has important applications in everyday life.
2. We proposed two interesting heuristic algorithms to solve the *STMPC* query and produce near optimal results.
3. We proposed an offline framework to estimate the network distance between POIs and predefined geographical regions, which can contain the query points. Our framework shows superior results compared to the Euclidean distance estimation.
4. We perform extensive experiments to evaluate the effectiveness and efficiency of the proposed algorithms, and the results are summarized as follows:
   (a) In term of effectiveness, our algorithms produce up to 0.99 optimal results based on our optimality indicter. The accuracy of the distance estimated by our framework is between 0.96 and 0.99 compared to actual network distance.
   (b) In term of efficiency, our algorithms answer *STMPC* queries in real time and up to four orders of magnitude faster than the baseline solution.

The reminder of the paper is organized as follows. Related work is discussed in Section 2. Section 3 presents the preliminaries and problem definition. Sections 4 details the proposed efficient heuristics and the distance estimation framework. The experimental results are shown in Section 5. Finally we conclude the paper in Section 6.

2 Related Work

Related work can be categorised into two categories; road network skyline query related problems and trip planing query related problems. We are unaware of any attempt to investigate the problem of finding the skyline trips of multiple POIs categories within either categories.

First, most existing studies on skyline queries (e.g.[3,29,17,27]) have focused on efficiently finding the skyline points for datasets in the traditional database systems. Only a few studies considered the skyline concept in spatial database systems, specifically, in road networks. Deng in [7], proposed to solve the "Multi-source Skyline Query", which aims at finding skyline POIs in road network based on the attribute of the POI (e.g. price) and the aggregated distance between the POI and multiple locations. For example, find the skyline hotels that are both cheap and close to the beach, the university and the botanic garden. The setting of this problem is different from ours in that it only assumes one POI category (e.g. hotels), while we consider multiple POIs categories and also consider the total trip distance aggregated from travelling between POIs from different categories. Therefore, the solution of Deng is not applicable to our problem.

Some other studies in road network skyline query [18,20,30] focus on finding the skyline paths considering multiple path attributes such as distance, driving time, gas consumption, number of traffic lights, etc. They assume different paths would have different values at each attribute and thus their goal is to find the set of skyline paths to allow the user to choose the most preferable path. Again, this problem is different than ours in that it does not include any POIs nor the distance between them, hence, their solution is not applicable.

Other road networks skyline studies such as [12,14,13], consider continuous skyline queries for POIs in road network. They continuously search for the skyline POIs for a moving object considering both the attributes of the POIs (e.g. price, rating) and their distance to the moving object. In these studies, the outcome is not a complete trip consisting of POIs from multiple categories, instead, it is a set of skyline POIs from the same category and hence, their solutions are not applicable to our problem.

Second, most trip planing studies [19,26,4,16] have one optimal trip that answer their queries, while we consider a set of skyline trips. The TPQ [19] discussed in the introduction can be considered as a special case of our problem. This is because TPQ does not consider the cost at POIs while constructing the trip. Therefore, applying the TPQ solution to our problem would only return one trip from the skyline trips, which is the one with the lowest distance. The optimal sequenced rout query studied in [26,16] aims to solve the TPQ with order constraint, is a special case of the TPQ and thus a special case of our problem. Finally, the "The multi-rule partial sequenced route query" studied in [4] is similar to both the optimal sequenced route query and the TPQ in that, it may involve some order constraint.

Although some of the above mentioned studies might seem similar to the *STMPC* query, their solutions are not applicable. This is because our problem is mainly inherited from three major queries types, which are multi-dimensional skyline queries, nearest neighbour queries and shortest path queries. On the other hand, most of the studies discussed in this section are only inherited from two queries types.

There are some existing studies on the nearest neighbor or range queries [1,32,10,33], which retrieve the set of objects with the smallest distance to the query point. However, the *STMPC* and TPQ queries uses the aggregated distance between query points and POIs to find the trip with the lowest total distance.

3 *STMPC* Query

We first formalize the *STMPC* query and present the baseline algorithm, and then we prove it is NP-hard. Table 1 summarizes our notation.

Table 1: Frequently Used Symbols

Symbol	Explanation
s	The source of a STMPC query.
d	The destination of a STMPC query.
P	A set of POIs.
C	A set of POIs categories.
O	A set of items the user wants to purchase.
t	A trip from s to d through one POI from each category.
$Dis(p_i, p_j)$	The network distance between p_i and p_j.
p^c	The cost at a POI p.
p^d	The aggregated distance from a POI p to both s and d.
t^c	A trip total cost.
t^d	A trip total distance.
p^p	The priority value of a POI p.

3.1 Problem Definition

Giving a road network graph G, source s and destination d, a set C of POIs categories $C = \{c_1, c_2, ...c_n\}$ with a set P of POIs in each category $c_i = \{p_1, p_2, ...p_n\}$, a list O of items the user is planning to purchase $O = \{o_1, o_2, ...o_n\}$. Each item in the user list o_i can be associated with different costs at different related POIs. Let $t_i = \{s, p_1 \in c_1, p_2 \in c_2, ...p_n \in c_n, d\}$ be a trip that starts from s and passes at least through one POI from each category and finishes at d. We use ${t_i}^d$ to donate the total distance of a trip, which is the sum of the network distances between s, the trip POIs and d in the travelled order; ${t_i}^d = Dis(s, p_1) + Dis(p_1, p_2) + ... + Dis(p_n, d)$. We use ${t_i}^c$ to donate the total cost of a trip, which is the sum of the cost at each POI in the trip ${t_i}^c = {p_1}^c + {p_2}^c + ... + {p_i}^c$. For example the trip distance of $trip_1$ in Figure 1 is ${t_1}^d = Dis(s, m_1) + Dis(m_1, r_1) + Dis(r_1, g_1) + Dis(g_1, d) = 15$, while the cost of the same trip is ${t_1}^c = {m_1}^c + {r_1}^c + {g_1}^c = 45$.

We can consider any trip t_i that starts at s, pass at least through one POI from each category and ends at d as a valid trip because it answers the user query. However, different trips have different distance and cost values, hence a trip with a short distance such as $trip_1$ may have a high cost and vice versa. Therefore, we leave it up to the user to decide the relative importance of trip distance travelled and cost by returning the skyline trips. The problem of the *STMPC* query is defined as follow:

Definition 1 ***Skyline Trips of Multiple POIs Categories (STMPC) Query:*** *given a road network graph G, source s and destination d, a set C of POIs categories with a set P of POIs in each category, a list O of items the user is planning to purchase and a pricing function $f(o_i)$ for items at each related POI, the STMPC query finds the skyline trips $Sky(T)$ that each starts at s, passes through at least one POI from each related category and ends at d such that any trip $t \in Sky(T)$ are not dominated by any other trip t' in term of both trip distance t^d and trip cost t^c, i.e., $\forall t', t^d \leq t'^d \vee t^c \leq t'^c$.*

Based on the above definition, a straightforward solution is as follow. First, compute all the possible POI permutations, where only one POI from each category is chosen based on the list of items the user wants to purchase. Next, add s and d to the beginning and end of each found permutation in order to construct valid trips. Finally, compute cost and network distance for each constructed trip and perform a skyline query to find the skyline trips. The problem with this solution is that, it is extremely slow and only applicable for very small dataset sizes. For example, it takes more than 24 hours to find the skyline trips when applied only to 40 POIs.

3.2 STMPC NP-hard

The *STMPC* query can be considered as a generalization of some known path finding problems such as TPQ [19] and the travelling salesman problem (TSP) [9]. We will show in the following theorems that these two problems are special cases of the *STMPC* problem.

Theorem 1. *The metric travelling salesman problem with defined start and end points is a special case of the* STMPC *query.*

Proof. According to definition 1, when we simplify the *STMPC* problem to assume that, there is only one POI in each category, all trips will have the same cost (e.g. ${t_1}^c = {t_2}^c = ... = {t_n}^c$) while visiting POIs in different order may results in trips with different distances (e.g. ${t_1}^d \neq {t_2}^d \neq ... \neq {t_n}^d$), thus, there will only be one skyline trip, which is the trip with the minimum distance. According to the TSP definition, this version of the *STMPC* problem is an instance of the TSP. Therefore, TSP is a special case of the *STMPC* problem.

Theorem 2. *The trip planning query (TPQ) is a special case of the* STMPC *query.*

Proof. According to definition 1, when we simplify the *STMPC* problem to assume that all POIs from the same category have the same cost (e.g. ${p_1}^c \in c_1 = {p_2}^c \in c_2 = ... = {p_n}^c \in c_n$), all trips will have the same cost (e.g. ${t_1}^c = {t_2}^c = ... = {t_n}^c$). While visiting different POIs in different orders may results in trips with different distances (e.g. ${t_1}^d \neq {t_2}^d \neq ... \neq {t_n}^d$), thus, there will only be one skyline trip, which is the trip with the minimum distance. According to the TPQ definition, this version of the *STMPC* problem is a an instance of the TPQ. Therefore, TPQ is a special case of the *STMPC* problem.

Corollary 1. *The Skyline Trips of Multiple POI Category query* STMPC *is NP-hard.*

Proof. According to [9] and [19] the problems TSP and TPQ are proven to be NP-hard. Therefore, since the problems TSP and TPQ are special cases of the *STMPC* problem (theorems 1 and 2), the *STMPC* query is NP-hard.

The aim of the *STMPC* query is to find a set of optimal trips, which are the skyline trips in regard to two quality dimensions; trip cost and distance. Based on theorem 2, the TPQ is a simpler version of the *STMPC* query, where only the shortest optimal trip is queried. This means, finding each skyline trip is at least as hard as the TPQ.

4 Proposed Heuristics

In this section we present our proposed heuristics algorithms and a network distance estimation framework. For ease of understanding, we first describe a simple Euclidean distance based solution, which we call Weighted POIs Algorithm (WPOIs). Next we detail the distance estimation framework, which is used by both algorithms to estimate the network distance instead of using the Euclidean distance. Finally, we cover the second algorithm; Clustered Weighted POIs Algorithm (CWPOIs), which is an improved cluster based version of the first algorithm.

4.1 Weighted POIs Algorithm (WPOIs)

Here we present an efficient algorithm to find the skyline trips for multiple POIs categories based on two dimensions (trip cost and trip distance). The WPOIs algorithm is divided into two stages; POIs nomination stage and trip construction stage. It works by repeatedly iterating through these two stages, where the outcome of each iteration is a skyline trip candidate. In the first stage of each iteration, every POI category nominates one POI as the most superior POI in the category. In the trip construction stage of each iteration, we use s, d and the nominated POI from each category to construct a trip using a greedy approach.

POI Nomination Stage: Before we start illustrating the process of this stage, we need to define new properties for both POIs and iterations. First, for each POI, we define two properties, which are POI aggregated distance p^d and POI cost p^c. As mentioned in Section 3 ,the property p^c represents the POI expected cost based on the items the user wants to purchase, while the property p^d represents the POI aggregated Euclidean distance from both s and d. Second, for each iteration, we define two dependant weighting values w^c and w^d, which are the cost weight and the distance weight, respectively, where always $w^c + w^d = 1$. These two weights represent the importance of cost and distance when nominating a POI from each category. We also define a third property for each POI, which is the POI priority value p^p. The priority value p^p is simply the weighted sum of the POI cost and distance, thus, $p^p = w^c p^c + w^d p^d$. However, before finding the value of p^p, we first need to normalise the POIs cost range to match their distance range. Data normalisation is discussed in [22]. The value of p^p represents the total weight of a POI in each iteration.

The main idea of this stage is to vary the weights of the cost and distance (w^c, w^d) during every iteration in order to nominate the POI with the lowest p^p from each category. For example in Figure 2, when $w^c = 1$, thus $w^d = 0$ ($1 - w^c$), the priority value p^p of the three gas stations (g_1, g_2, g_3) are ($18 * 0 + 14 * 1 = 14, 14 * 0 + 18 * 1 = 18, 16 * 0 + 15 * 1 = 15$) respectively. Therefore, the gas station category nominates the POI with the lowest p^p, which is g_1. Similarly m_2 and r_2 are nominated. On the other hand when $w^d = 0.5$ and thus $w^c = 0.5$, the nominated POIs from each category are is g_3, m_3, r_2 and so on. The order of the POIs is not important at this stage of the algorithm because the trip will be formed in the trip construction stage.

Based on definition 1, if a trip t_a consists of the cheapest POI from each category (e.g. $\min({p_i}^c)$), then, t_a is a skyline trip because it is the cheapest. We could also expect

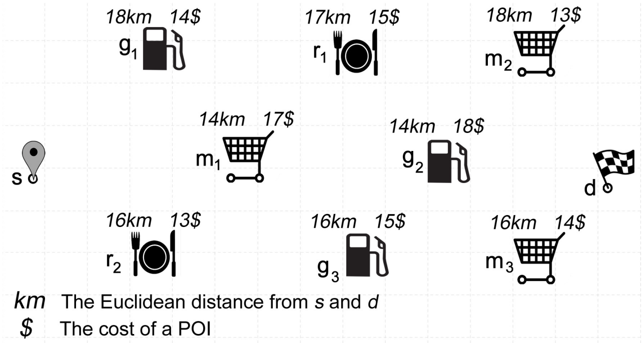


Fig. 2: WPOIs algorithm example

that, if a trip t_b consists of POIs with the least aggregated distance from s and d (e.g. $\min({p_i}^d)$), then, t_b is a skyline trip because it is the shortest. In the previous process, when $w^c = 1$, only POIs with the lowest cost will be nominated resulting in the cheapest trip (skyline). Similarly, when $w^d = 1$ the POIs of the shortest trip (skyline) will be nominated. The two skyline trips t_a and t_b are the most extreme skyline points on each dimension, thus, based on definition 1, if $t_a \neq t_b$, then, all other skyline points are between t_a and t_b.

As mentioned at the beginning of this stage, the variables w^c and w^d are dependant because $w^c + w^d = 1$. Therefore, a possible approach of trying to get all the skyline trips is to vary the cost weight w^c between 1 and 0 for a number of iterations. In every iteration, we get the best POI from each category, which together could form a skyline trip. A straightforward technique of achieving this is to have a fixed number of iterations (e.g. 100) where we gradually change the cost weight w^c between 1 and 0 in every iteration. For example the values of w^c in the 100 iterations are $(1, 0.99, 0.98...0)$ as a results the values of w^d are $(0, 0.01, 0.02...1)$. However, this approach has two main disadvantages. First, it is possible that a POI category nominates the same POI for different iterations. For example in Figure 2, the gas station g_1 is nominated when the cost weight is $1 \geq w^c \geq 7.5$. This is because the p^p of g_1 is the lowest compared to other gas stations in the specified range. Second, it is also possible to miss some skyline trips if they exist between two fractions with a small difference between them (e.g. $w^c = 0.751$ and $w^c = 0.752$). As a result, the straightforward technique is inefficient. We use an efficient iterating technique that is based on the following theorem.

Theorem 3. *If $a > b$ and p is nominated when $w^c = a$ and also nominated when $w^c = b$, then p is nominated when $a \geq w^c \geq b$.*

Proof. Since $p^p = w^c p^c + (1 - w^c)p^d$ and p is nominated based on the p^p of a and b, ${p_a}^p$ and ${p_a}^p$ respectively, where ${p_a}^p \geq {p_a}^p$. let $w^c = x$, thus, ${p_x}^p = xp^c + (1 - x)p^d$. if $a \geq x \geq b$ then, ${p_a}^p \geq {p_x}^p \geq {p_a}^p$ and hence p is nominated when $w^c = x$.

Based on the above theorem, we do not need to fix the number of iterations (e.g. 100). Instead, for each POI category, we compare the nominated POI for the two extreme weight of w^c, where $w^c = 0$ and $w^c = 1$. If the same POI is nominated, we do not need check any other value of w^c. Otherwise, we compare the nominated POI of the middle weight (e.g. $w^c = 0.5$) to both extreme values and so on. For example in Figure 2, when $w^c = 1$ the Restaurant category nominates r_2 because ${r_2}^p < {r_1}^p$ $(13 < 15)$ and also nominates r_2 when $w^c = 0$ because ${r_2}^p < {r_1}^p$ $(16 < 17)$. Therefore, we do not to iterate between 1 and 0 in order to look for a POI with less p^p in the Restaurant category.

Trip Construction Stage: As mentioned at the beginning of this subsection, the outcome of the POI nomination stage is a set of skyline candidate trips. Each consists of one nominated POI from each category. In this stage, we focus on the order of the nominated POI in order to achieve a trip that starts from s, pass through the nominated POIs and ends at d with the minimum distance. As explained in Section 3, this stage of the algorithm is NP-hard problem by itself because it is a special case of the TSP. Therefore, we use the same greedy technique used to solve the TPQ and TSP problems. It works by visiting the nearest neighbour of the last POI added to the trip starting from s and ending at d, where the Euclidean distance is used in the nearest neighbour search. In each iteration, we use the greedy technique to form the general shape of a candidate trip (POIs order) regardless of the real network distance between them.

Once all candidate trips are formed, we can use any shortest path algorithm to create the final trips using the network distance. Next, we perform a skyline query over the candidate trips using their costs and network distances as the two dimensions in order to prune any dominated trip. Finally, we get the set of skyline trips $Sky(T)$.

4.2 Distance Estimation Framework

In the previous illustration of the WPOIs algorithm, we used the Euclidean distance to estimate the aggregated distance p^d from each POI to both s and d. We also used the Euclidean distance in the greedy solution to find the next nearest neighbour of POIs forming a trip. The disadvantage of using the Euclidean distance in road networks is that, it does not reflect the actual network distance [15,25,2]. In contrast, the exact network distance is computationally expensive when computed online and hard to store when computed off-line. Therefore, we propose a network distance estimation framework to be used instead of the Euclidean distance in the WPOIs and CWPOIs algorithms. The main idea of this framework is to precompute and store the average network distance between every POI and specific geographical regions in the road network. It estimates the network distance between a POI and any network vertex by retrieving the stored distance between the POI and the region containing the queried vertex. Dividing the network into multiple geographical regions can be done using any suitable multidimensional space data structure (e.g. Quadtree, Rtree). In the settings of this study we use Quadtree for its simplicity.

The pre-computation process starts by indexing the network vertices into the Quadtree with suitable density level at each leave node. Next, for each POI we perform a single source shortest path search (Dijkstra's [8]) to find the distance from that particular POI to all network vertices. Based on these distances, we measure and store the average distance to each of the geographical regions (Quadtree leave nodes). For example in Figure

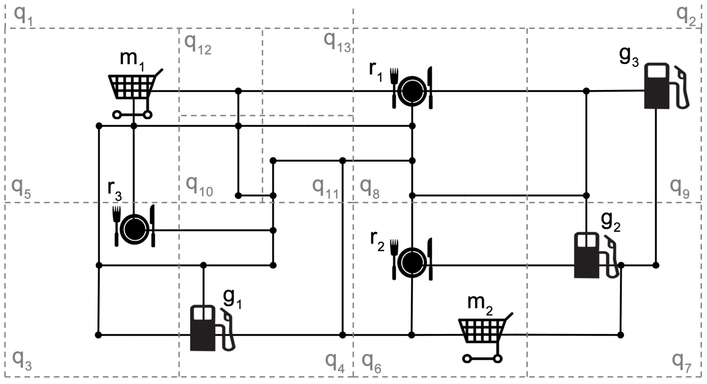


Fig. 3: WPOIs algorithm example

3, the average network distances between every POI $\{g_1, g_2, .., r_1, ..m_3\}$ and each of the Quadtree leave nodes $\{q_1, q_2, ...q_{12}\}$ are stored. At the online stage, the distance between a network vertex and a POI is estimated by retrieving the stored average distance between the POI and the geographical square containing the vertex. For example in Figure 3, we can estimate the network distance between g_1 and r_1 by retrieving the stored distance between g_1 and the Quadtree leave node q_8.

The required time to pre-compute the distances and the memory consumption of storing the distances are highly sensitive to the number of POIs and the number of Quadtree leaf nodes, which is based on the node density. However, since the framework is computed offline, the time consumption is not critical. Moreover, the memory consumption can be well managed by using the right density level in each Quadtree leaf node as will be shown in Section 5. The higher time and memory cost of pre-computing the distances is well justified by the more accurate network distance estimations.

The distance estimation framework is more suitable for static road networks. However, in order to make it applicable on time-dependent road networks [6], we can use the road network historical data to store different traveling times between POIs and the geographical regions for different times of the day.

4.3 Clustered Weighted POIs Algorithm (CWPOIs)

The CWPOIs algorithm is an improved version of the first algorithm illustrated in Section 4.1. The problem with the WPOIs algorithm is that, it only considers the aggregated distance between a POI and the query points at the nomination stage, regardless of the distance to POIs from other categories. This could result in missing some of the skyline trips candidates when their POIs are clustered far from query points. For example, the WPOIs algorithm may miss a skyline trip $t_1 = \{g_x, r_x, m_x\}$ (POIs located in the same location e.g. mall) when there is a another trip $t_2 = \{g_y, r_y, m_y\}$ (POIs located in different locations) with the same cost and worse total distance ${t_2}^d > {t_1}^d$. This is because the distance p^d of each of the three POIs $({g_x}^d, {r_x}^d, {m_x}^d)$ is large when considered individually, thus, the t_1 POIs are dominated by the POIs of t_2 (i.e.${g_x}^d > {g_y}^d, {r_x}^d >$

${r_y}^d, {m_x}^d > {m_y}^d$). Therefore, in order to overcome this obstacle we propose the CWPOIs algorithm, which considers both the aggregated distance from query points p^d and the distance between clustered POIs.

Similar to the first algorithm, the CWPOIs algorithm is based on the framework illustrated in Section 4.2 to estimate the network distance. However, it also uses the framework to cluster POIs in geographical regions (Quadtree nodes). The main idea of this algorithm is to define two properties for each Quadtree node, which are the node lowest cost n^c and the node average distance n^d. This is only applicable for nodes containing at least one POI from each related category. The first property n^c can be defined by the aggregated cost of the POI with the lowest cost p^c from each category located within the geographical area of a node n. The second property n^d is the aggregated average distance between a node n and the query points s and d, which is obtained from the pre-computation of the framework.

The process of the CWPOIS algorithm starts by defining the values for the n^c and n^d properties for each applicable Quadtree node, which can be leaf or non-leaf node. Next, we use these two values to perform a skyline query over the Quadtree nodes, where the outcome of this step is a set of non-dominated Quadtree nodes. Then, we apply the first algorithm (WPOIS) at each of the skyline Quadtree nodes to find the skyline trips candidates in each geographical region. Finally, we measure the network distance for each of the candidate trips and perform a skyline query to return the final skyline trips.

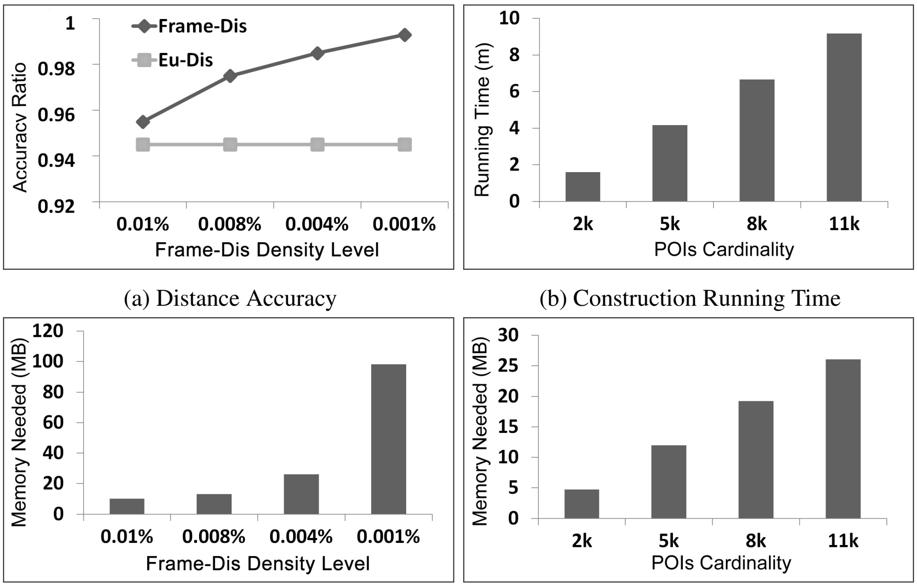
Based on the above process, the CWPOIs algorithm finds the skyline geographical regions and apply WPOIs algorithm to each of them individually. This results in separating the nomination competition (first stage of WPOIs algorithm) performed for POIs in a clustered region from other regions, thus, returning more accurate skyline trips at the final stage.

5 Experimental study

In this section we evaluate the effectiveness and efficiency of the proposed framework and algorithms. We conducted the experiments on a desktop PC with 8GB RAM and a 3.4GHz Intel$^{(R)}$ Core$^{(TM)}$ i7 CPU. The disk page size is 4K bytes. We use the London road network dataset extracted from Open Street Map[1], which contains 515,120 vertices and 838,702 edges. We also extracted the locations of 11,030 POIs in London classified into 12 different categories.

We vary parameters such as the number of POIs categories, POIs cardinality within each category, and Quadtree density level to gain insight into the performance of the framework and the algorithms in different settings. The detailed settings are given in the individual experiments.

[1] http://metro.teczno.com/# london



(a) Distance Accuracy

(b) Construction Running Time

(c) Memory Consumption (density level)

(d) Memory Consumption (cardinality)

Fig. 4: Framework Evaluation

5.1 Framework evaluation

As discussed in Section 4.2, the purpose of the framework is to provide a better estimation of the network distance than using the Euclidean distance. We validate the framework in term of both effectiveness and efficiency.

Framework accuracy: First, we compare the accuracy of the distance estimated by our framework denoted as *Frame-Dis*, to the Euclidean distance estimation denoted as *Eu-Dis*. An intuitive way to find the accuracy ratio *AR* for a distance estimation method *Est-dis*, which can be either *Frame-Dis* or *Eu-Dis*, is to compare it to the actual network distance denoted as *Act-Dis* as follow:

$$AR = \frac{\textit{Act-Dis} - (|\textit{Act-Dis} - \textit{Est-Dis}|)}{\textit{Act-Dis}}$$

For example, when $Act - Dis = 37km$ and $Frame - Dis = 37.5km$, the accuracy indicator $AR = 0.98$. The accuracy of our framework is highly sensitive to the density level at each geographical region. The less dense the Quadtree cell, the more accurate distance estimation. We vary the maximum density level inside each Quadtree node from 0.01% to 0.001% of the total network vertices. We compare the average accuracy ratio *AR* of both *Frame-Dis* and *Eu-Dis* when running 1000 queries from random POIs to random network vertices. It can be seen from Figure 4a that, the accuracy ratio of *Frame-Dis* increases up to 0.99 as the density level decreases. This is because the size of geographical regions decrease and hence, the difference between the average distance from a POI to a region and the actual distance from the POI to any vertex within that region decreases. In all density levels, our framework estimates network distance more accurately than the Euclidean distance.

Framework efficiency: The framework efficiency is evaluated using two metrics; the time taken to construct the framework and its memory consumption.

Framework construction running time: the running time of the framework is highly effected by the total number of POIs from all categories. This is because we need to find the distance to all network vertices in order to measure the average distance from each POI to every geographical region. Figure 4b illustrates the increase of framework construction time as the total number of POIs increases. When all the POIs are used, the framework is precomputed in less than 10 minutes. The graph construction running time is not affect by Quadtree density level and thus the number of the geographical locations. This is because there will be a path finding query for each POI regardless of the number of Quadtree nodes.

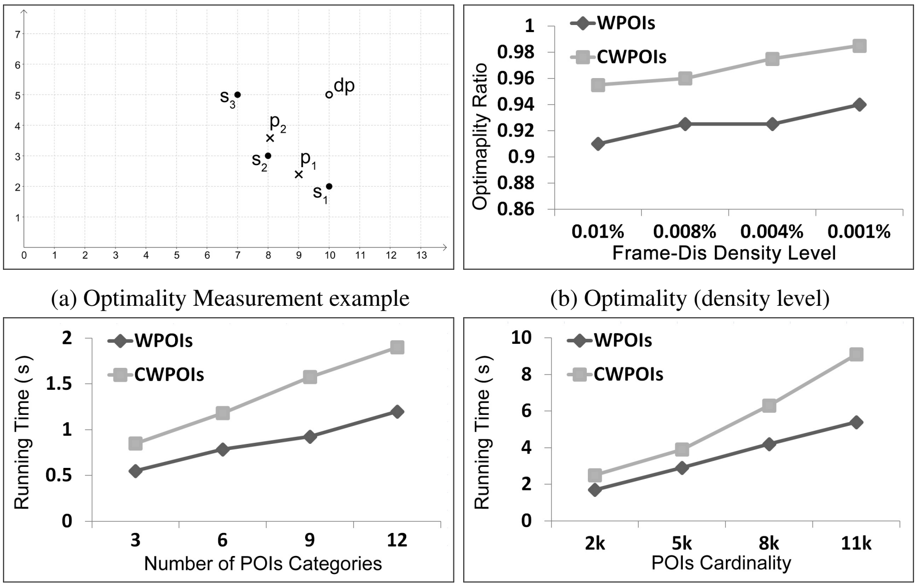
Framework memory consumption: the main purpose of the framework is to precompute and store the average distance between every pair of a POI and a geographical region. Therefore, the number of stored distances is nm, where n is the number of POIs and m is the number of geographical regions. Figure 4c illustrates the memory space needed to store the precomputed distance for different density levels in each geographical region when all of the $11{,}030$ POIs are used. When the density level decreases from 0.01% to 0.001%, the number of geographical regions increase and thus more space is needed to store the distances. At the 0.001% density, only 100 MB is needed to store the precomputed distances. Figure 4d shows different memory consumptions for different POIs cardinality when the density level is fixed to 0.004% in each geographical region. The memory consumption increases as the number of POIs increases. Based on Figures 4c and 4d, we can see the density level of the used data structure affects the memory needed more than the POIs cardinality.

5.2 WPOIs and CWPOIs Effectiveness evaluation

In this subsection, we validate the effectiveness of our proposed algorithms in finding skyline trips. However, we need to fist discuss how to measure the optimality of our results. Finally, we measure the effectiveness of our algorithms.

Effectiveness measurement: As discussed in Section 3, the baseline optimal solution is extremely slow as it takes more than 24 hours to only process 40 POIs dataset .There is no straightforward way to measure the optimality of a set of points compared to the optimal skyline set. Therefore, we propose a formula to compare the results of our algorithms to the optimal results and provide an optimality metric Opt, defined as follows:

$$Opt = 1 - \sum_{x \in Sky} = \frac{minDis(x, tp)}{minDis(x, dp)}$$



(a) Optimality Measurement example

(b) Optimality (density level)

(c) Running Time (categories number)

(d) Running Time (cardinality)

Fig. 5: Algorithms Evaluation

Where, Sky is the set of optimal skyline points, $minDis(x, y)$ is the minimum graph distance required for a point y to dominate point x, tp is the point with the minimum $minDis$ from the examined points, and dp is the best case point that is dominated by all optimal skyline points. For example in Figure 5a, when $Sky = \{s_1, s_2, s_3\}$ are the optimal skyline points, the best case point should have the coordinates $(10, 11)$ as shown in the figure to be the best point dominated by all skyline points. In addition, the $minDis(s_1, p_1)$ is 0.5 because the point p_1 needs to move down on the y axis by 0.5 in order to dominate s_1 and so on.

Effectiveness experiments: We use the optimality metric Opt to evaluate the performance of our algorithms (WPOIs, CWPOIs) compared to the optimal results for 20 different queries. We also vary the density level of the framework to reflect different performance levels. It can be seen from Figure 5b that, the optimality for both algorithms increase as the density level decreases due to higher network distance estimation. In addition, although both algorithms have high optimality metric value(over 0.9), the CWPOIs outperforms the WPOIs for all density levels. This is because CWPOIs algorithm considers clustered POIs when finding skyline trips while WPOIs algorithm only considers the distance between a POI and the query points.

5.3 Efficiency evaluation

In this subsection we validate the efficiency of both algorithms under different settings. We measured the query processing time of both WPOIs and CWPOIs with different number of POIs categories and different cardinalities within each category.

Effect of the number of POIs categories: We vary the number of related POI categories from 3 to 12 categories, where only 100 POIs from each category are considered and measure the average running time of 100 random queries. Figure 5c shows that the running time of both algorithms increases as the number of categories increases. The CWPOIs algorithm run slower than the WPOIs algorithm for all different number of categories, which is due to processing clustered POIs.

Effect of POIs cardinality: We vary the number of POIs between 2k and 11k from all 12 categories and measure the average running time of 100 random queries. Figure 5d shows that the running time of both algorithms increases as the POIs cardinality increases. The CWPOIs algorithm can take up to 9 seconds to answer a query when all POIs from all 12 categories are used. However, in reality it is not common for a user to plan to visit 12 different POI categories in one trip. In addition, the running time can be tolerated considering the near optimal results obtained by both algorithms for an NP-hard problem. Both algorithms are more than four orders of magnitude faster than the baseline solution.

6 Conclusion and Future Work

We proposed a new path finding problem, *STMPC*, which finds the skyline trips of multiple POI categories between two points based on cost and distance. We define the problem in road network settings and proved it to be NP-hard. We proposed an independent framework to estimate the network distance. This framework is based on precomputing and storing the distances between POIs and some geographical regions in the network. We also proposed two interesting heuristic algorithms, which are WPOIs and CWPOIs algorithms. The CWPOIs algorithm considers clustered POIs when nominating skyline candidate trips. As shown in the experiments section, both algorithms return skyline trips that are close to optimal trips within reasonable running time when processing a large real dataset. Our algorithms are four orders of magnitude faster than the naive optimal solution.

For future work we will consider solving the same problem when there are multiple quality dimensions (e.g. distance, cost, rating, number of stops ... etc). We might also investigate improving the memory consumption of the proposed framework by only storing network distance between different geographical regions.

7 Acknowledgment

The author Saad Aljubayrin is sponsored by Shaqra University, KSA. This work is supported by Australian Research Council (ARC) Discovery Project DP130104587 and Australian Research Council (ARC) Future Fellowships Project FT120100832.

References

1. Ali, M.E., Tanin, E., Zhang, R., Kulik, L.: A motion-aware approach for efficient evaluation of continuous queries on 3d object databases. The VLDB JournalâĂŤThe International Journal on Very Large Data Bases 19(5), 603–632 (2010)
2. Aljubayrin, S., Qi, J., Jensen, C.S., Zhang, R., He, Z., Wen, Z.: The safest path via safe zones. ICDE (2015)
3. Borzsony, S., Kossmann, D., Stocker, K.: The skyline operator. In: Data Engineering, 2001. Proceedings. 17th International Conference on. pp. 421–430. IEEE (2001)
4. Chen, H., Ku, W.S., Sun, M.T., Zimmermann, R.: The multi-rule partial sequenced route query. In: Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems. p. 10. ACM (2008)
5. Comer, D.: Ubiquitous b-tree. ACM Computing Surveys (CSUR) 11(2), 121–137 (1979)
6. Demiryurek, U., Banaei-Kashani, F., Shahabi, C.: Efficient k-nearest neighbor search in time-dependent spatial networks. In: Database and Expert Systems Applications. pp. 432–449. Springer (2010)
7. Deng, K., Zhou, X., Shen, H.T.: Multi-source skyline query processing in road networks. In: Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on. pp. 796–805. IEEE (2007)
8. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische mathematik 1(1), 269–271 (1959)
9. Dorigo, M., Gambardella, L.M.: Ant colonies for the travelling salesman problem. BioSystems 43(2), 73–81 (1997)
10. Eunus Ali, M., Zhang, R., Tanin, E., Kulik, L.: A motion-aware approach to continuous retrieval of 3d objects. In: Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on. pp. 843–852. IEEE (2008)
11. Guttman, A.: R-trees: a dynamic index structure for spatial searching, vol. 14. ACM (1984)
12. Huang, X., Jensen, C.S.: In-route skyline querying for location-based services. In: Web and Wireless Geographical Information Systems, pp. 120–135. Springer (2005)
13. Huang, Y.K., Chang, C.H., Lee, C.: Continuous distance-based skyline queries in road networks. Information Systems 37(7), 611–633 (2012)
14. Jang, S.M., Yoo, J.S.: Processing continuous skyline queries in road networks. In: Computer Science and its Applications, 2008. CSA'08. International Symposium on. pp. 353–356. IEEE (2008)
15. Jensen, C.S., Kolářvr, J., Pedersen, T.B., Timko, I.: Nearest neighbor queries in road networks. In: Proceedings of the 11th ACM international symposium on Advances in geographic information systems. pp. 1–8. ACM (2003)
16. Kanza, Y., Levin, R., Safra, E., Sagiv, Y.: Interactive route search in the presence of order constraints. Proceedings of the VLDB Endowment 3(1-2), 117–128 (2010)
17. Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: An online algorithm for skyline queries. In: Proceedings of the 28th international conference on Very Large Data Bases. pp. 275–286. VLDB Endowment (2002)
18. Kriegel, H.P., Renz, M., Schubert, M.: Route skyline queries: A multi-preference path planning approach. In: Data Engineering (ICDE), 2010 IEEE 26th International Conference on. pp. 261–272. IEEE (2010)
19. Li, F., Cheng, D., Hadjieleftheriou, M., Kollios, G., Teng, S.H.: On trip planning queries in spatial databases. In: Advances in Spatial and Temporal Databases, pp. 273–290. Springer (2005)
20. Mouratidis, K., Lin, Y., Yiu, M.L.: Preference queries in large multi-cost transportation networks. In: Data Engineering (ICDE), 2010 IEEE 26th International Conference on. pp. 533–544. IEEE (2010)

21. Nutanong, S., Zhang, R., Tanin, E., Kulik, L.: The v*-diagram: a query-dependent approach to moving knn queries. Proceedings of the VLDB Endowment 1(1), 1095–1106 (2008)
22. Pyle, D.: Data preparation for data mining, vol. 1. Morgan Kaufmann (1999)
23. Qi, J., Zhang, R., Kulik, L., Lin, D., Xue, Y.: The min-dist location selection query. In: Data Engineering (ICDE), 2012 IEEE 28th International Conference on. pp. 366–377. IEEE (2012)
24. Qi, J., Zhang, R., Wang, Y., Xue, A.Y., Yu, G., Kulik, L.: The min-dist location selection and facility replacement queries. World Wide Web 17(6), 1261–1293 (2014)
25. Shahabi, C., Kolahdouzan, M.R., Sharifzadeh, M.: A road network embedding technique for k-nearest neighbor search in moving object databases. GeoInformatica 7(3), 255–273 (2003)
26. Sharifzadeh, M., Kolahdouzan, M., Shahabi, C.: The optimal sequenced route query. The VLDB journal 17(4), 765–787 (2008)
27. Sharifzadeh, M., Shahabi, C.: The spatial skyline queries. In: Proceedings of the 32nd international conference on Very large data bases. pp. 751–762. VLDB Endowment (2006)
28. Shipman, D.W.: The functional data model and the data languages daplex. ACM Transactions on Database Systems (TODS) 6(1), 140–173 (1981)
29. Tan, K.L., Eng, P.K., Ooi, B.C., et al.: Efficient progressive skyline computation. In: VLDB. vol. 1, pp. 301–310 (2001)
30. Tian, Y., Lee, K.C., Lee, W.C.: Finding skyline paths in road networks. In: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. pp. 444–447. ACM (2009)
31. Vranken, W.F., Boucher, W., Stevens, T.J., Fogh, R.H., Pajon, A., Llinas, M., Ulrich, E.L., Markley, J.L., Ionides, J., Laue, E.D.: The ccpn data model for nmr spectroscopy: development of a software pipeline. Proteins: Structure, Function, and Bioinformatics 59(4), 687–696 (2005)
32. Zhang, R., Lin, D., Ramamohanarao, K., Bertino, E.: Continuous intersection joins over moving objects. In: Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on. pp. 863–872. IEEE (2008)
33. Zhang, R., Qi, J., Lin, D., Wang, W., Wong, R.C.W.: A highly optimized algorithm for continuous intersection join queries over moving objects. The VLDB JournalâĂŤThe International Journal on Very Large Data Bases 21(4), 561–586 (2012)