Towards A Personal Shopper’s Dilemma: Time vs Cost *

Samiul Anwar
University of Alberta, Canada
samiul@ualberta.ca

Francesco Lettich
Università Ca’ Foscari Venezia, Italy
francesco.lettich@gmail.com

Mario A. Nascimento
University of Alberta, Canada
mario.nascimento@ualberta.ca

ABSTRACT

Consider a customer who has a shopping list and a personal shopper who is willing to buy and resell goods in a customer’s shopping list. It is in the personal shopper’s best interest to find shopping routes that minimize two competing criteria: the time needed to serve a customer and the price paid for the goods. In this short paper we present an efficient solution to this problem based on finding an approximate linear skyline set of such shopping routes. (An extended version of this paper can be found at [1]).

CCS CONCEPTS
- Information systems → Location based services.

KEYWORDS
Spatial Crowdsourcing, Road Networks, Skyline.

ACM Reference Format:
Samiul Anwar, Francesco Lettich, and Mario A. Nascimento. 2020. Towards A Personal Shopper’s Dilemma: Time vs Cost . In 28th International Conference on Advances in Geographic Information Systems (SIGSPATIAL ’20), November 3–6, 2020, Seattle, WA, USA. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3397536.3422276

1 INTRODUCTION

A personal shopper is a person that serves customers by fulfilling their shopping lists. We assume the main concern of the personal shopper to be how to determine which sequence of stores to visit, and which products to acquire in those stores, in order to fulfill the customer’s shopping list, while, at the same time, minimizing both the shopping time and the shopping list’s cost. Unfortunately, from a practical perspective, it is seldom possible to find a single set of stores that satisfies both criteria. We call this new and practical problem as the Personal Shopper’s Dilemma (PSD) query.

At this point one could be tempted to argue that the two considered criteria could be linearly combined into a single one, thus casting PSD into a single cost function minimization problem. For that, appropriate weights would need to be predetermined by the shopper a priori and potentially more interesting solution for slightly different combinations of weights could be missed. We overcome this limitation by computing the set of all optimal sequence of stores for any linear combination of shopping cost and shopping time, which is also known as linear skyline set [2].

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
SIGSPATIAL ’20, November 3–6, 2020, Seattle, WA, USA
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8019-5/20/11.
https://doi.org/10.1145/3397536.3422276

Figure 1: Sample network.

In order to illustrate the PSD query, let us use the instance depicted in Figure 1. Suppose that a customer wishes to buy products A, B, C and D and issues the corresponding shopping list to the shopper. For the sake of presentation assume that only one unit of each product is required. Stores are denoted by si, while the customer’s delivery location and the shopper’s current location are denoted by lc and ls, respectively. Edges represent the fastest paths connecting vertices, with labels denoting the associated travel time; the list of products, together with their respective unit prices, available at each store is shown at the figure’s right-hand side.

From the above scenario we see that there are multiple solutions to the PSD query. For instance, route R1 = ⟨ls, s1, s2, lc⟩ represents the best solution in terms of shopping time (28), yet its shopping cost ($33) is the largest, while R5 = ⟨ls, s5, s6, lc⟩ offers the least expensive shopping ($21), but it imposes an expensive route (48). Between these two there are several solutions that may interest the shopper, depending on the shopper’s particular preferences or needs at query time.

When dealing with multiple cost criteria and the problem of determining a set of results that are optimal under any arbitrary combination thereof, a well-known and extensively used tool is the skyline query [3]. Computing skyline queries is a computationally expensive and typically returns many similar solutions, thus possibly making the choice of a specific solution rather difficult for users. To tackle these issues, Shekelyan et al. [2] introduced the notion of linear skylines. A linear skyline is the subset of the objects defining the convex hull of the (conventional) skyline. Objects belonging to a linear skyline are required to be optimal under any linear combination of the competing cost criteria. Considering the above example, the linear skyline is represented by the set of routes LS = {R1, R2, R5}, depicted in Figure 2.

Now we can easily observe the shopper’s dilemma. None of the routes R1, R2 or R5 is strictly better than the others, in that each solution creates a different trade-off between shopping time and shopping cost, and that each can be interesting under different circumstances. Therefore we target the problem of computing a set of interesting, meaningful, and sufficiently diverse solutions, i.e., the linear skyline of all shopping routes.
We assume as underlying framework for the PSD query a city’s road network modeled by an undirected graph \( G(V,E,W) \), where the weight of an edge \( e \in E \) connecting two vertices \( v_i \) and \( v_j \) in \( V \), and denoted by \( w(v_i,v_j) \in W \), is given by the shortest time needed to traverse the associated road network segment.

There are four main entities within the PSD’s model: stores, a single customer and his/her shopping list, and a personal shopper.

- A store \( t_j \in T \) is located at a vertex \( v_e \). We denote by \( c(i, t_j) \) the cost of a product \( i \) at a store \( t_j \). For simplicity we assume that stores have an arbitrarily large inventory of products they sell. A customer \( \sigma \) wants to buy a set of products, i.e., a shopping list (described next). We assume the products need to be delivered at the customer’s location, which is a vertex in \( V \) denoted by \( v_o \).
- A customer \( \sigma \) denotes the shopping list issued by \( \sigma \) and we represent it as a set of pairs \((i, q)\) where each \( i \) is a product identifier and \( q \) represents the number of required units of such product. Finally, the personal shopper \( \omega \) is in charge of satisfying a customer’s request and we denote the shopper’s location by \( v_o \in V \).

The answer to a PSD query is a set of “shopping routes”, each representing a sequence of stores to be visited and yielding a shopping cost as well as shopping time, each defined next.

- A shopping route \( \theta_i \) over \( G \) is a sequence of stores \( \langle \tau^i_1, \ldots, \tau^i_N \rangle \). Furthermore, given a shopping list \( \lambda \) we say that \( \theta_i \) is feasible w.r.t. \( \lambda \) if all products in \( \lambda \) are sold in at least one store in \( \theta_i \).
- Let \( \theta_i \) represent a feasible shopping route w.r.t. a shopping list \( \lambda \). We define the shopping time associated with \( \theta_i \), denoted as \( ST(\theta_i) \), as the time needed by the shopper to traverse the path that departs from \( v_o(s) \), visits the stores according to the order defined by \( \theta_i \), and finally ends at \( v_o(r) \).
- Let \( \lambda \) be the shopping list issued by a customer \( \sigma \) and let \( \theta_i \) be a feasible shopping route w.r.t. \( \lambda \). Then, we define the shopping cost of \( \theta_i \) as
  \[
  SC(\theta_i) = \sum_{(i,q) \in \lambda} (c(i,ss(\theta_i,j)) \times q)
  \]
  where \( ss(\theta_i,j) \) is a function that returns the store in \( \theta_i \) from which the product \( j \) in \( \lambda \) is to be bought (e.g., the cheapest one in \( \theta_i \)).

Note that if one allows the concept of a warehouse, the approach presented in this paper can be trivially extended to address the multiple customer PSD query – please refer to \([1]\) for further details.

Given a set of cost criteria and a pair of objects \( o_1 \) and \( o_2 \), let us say that \( o_1 \) dominates \( o_2 \) if (i) for each cost criterion the cost of \( o_1 \) is smaller or equal than that of \( o_2 \) and (ii) there is at least one criterion for which the cost of \( o_1 \) is strictly smaller than that of \( o_2 \). As a consequence, the set of objects that are non-dominated by any other object defines the notion of skyline, and represents the desired solution \([3]\). In the case of PSD, if \( \theta_i \) and \( \theta_j \) are two shopping routes and we denote that \( \theta_i \) dominates \( \theta_j \) by \( \theta_i < \theta_j \). From that it follows that if \( \Theta \) is a set of shopping routes, the conventional skyline of \( \Theta \) is the set of shopping routes that are not dominated, i.e., the set \( \{ \theta_i \in \Theta \mid \not\exists \theta_j \in \Theta : \theta_i < \theta_j \} \).

A linear skyline consists of the subset of a conventional skyline that is optimal under all linear combinations of the competing cost criteria \([2]\). Hence, in the scenario considered in this work a linear skyline is composed of combination of stores that minimize the linear combination \( (\delta_1 SC(\theta_i) + \delta_2 ST(\theta_i)) \), \( \forall \delta = (\delta_1, \delta_2) \in \mathbb{R}^2_\geq \).

More formally, let \( \Theta \) be a set of shopping routes. Let also \( \Theta' = \{ \theta_1, \ldots, \theta_K \} \subseteq \Theta \). Then, we say that \( \Theta' \) linearly dominates a shopping route \( \theta \in \Theta \) if and only if \( (30' \in \Theta' \colon 30' < \theta) \cup \forall \delta \in \mathbb{R}^2_\geq\), \( 30' \in \Theta' \colon 30' < \delta \cdot CV(\theta) \). Finally, the maximal set of linearly non-dominated combination of stores is referred to as linear skyline and it can be seen as an ordered set w.r.t. the first cost criterion.

3 APPROXIMATED LINEAR SKYLINES

Given some shopping list, computing its optimal linear skyline requires to evaluate in strict increasing order of one of the cost criteria shopping routes that fulfill the list, and orchestrate the construction of the skyline accordingly. The major drawback behind such approach lies in its necessity to possibly evaluate a number of shopping routes that is factorial in the number of stores \([1]\), thus limiting its scalability and applicability to real-world scenarios. To overcome to we propose APX-PSD, an approximated approach that trades the optimality of linear skylines for a greatly reduced number of candidates to evaluate. The key idea behind APX-PSD is to consider stores at a coarser granularity, i.e., partitions of stores rather than individual stores, to generate – and thus evaluate – shopping routes from those partitions, rather than the whole set of stores, that look the most “promising” in terms of shopping time and shopping cost w.r.t. to a given PSD query.
To partition the stores of a road network APX-PSD superimposes a point-region quad-tree \( Q \) over the minimum bounding rectangle enclosing the stores. This corresponds to the root quadrant of \( Q \). The quad-tree is then constructed by recursively splitting each quadrant having a number of elements (stores) larger than a given capacity threshold in four sub-quadrants. Given that it relies only on the spatial location of the stores, APX-PSD can pre-compute \( Q \) over the stores of a road-network, along with information concerning travel time between quadrants and statistics on the products each quadrant holds. Such statistics are subsequently used to drive the generation and evaluation of candidate routes, i.e., to decide which quadrants are the most promising w.r.t. shopping time and shopping cost.

The travel time between a pair of quadrants and travel time between a vertex and a quadrant, as they are key to the evaluation strategy employed by APX-PSD. We define the former to be the travel time yielded by the pair of stores minimizing such cost. In order to improve efficiency, and since it is a one-time computation, this is pre-computed once \( Q \) is materialized. Analogously, we define the latter to be the time yielded by the store in the considered partition that minimizes travel time w.r.t. the vertex. Both definitions allow APX-PSD to enforce upper bounds on shopping time when generating shopping routes from different partitions.

With \( Q \) in place, APX-PSD can then proceed to process PSD queries. Its evaluation strategy relies on a depth-first search (DFS) of the quad-tree driven by a scoring function we introduce below. Given the set of shopping routes under construction – we denote such set by \( PR \) (initially empty) – the scoring function estimates how “good” each quad-tree quadrant, be it an intermediate node or a leaf, is in terms of shopping time and shopping cost w.r.t. the characteristics of a PSD query and the shopping routes in \( PR \) generated so far, thus driving the generation and evaluation of shopping routes towards quadrants that are deemed the “most promising”. Specifically, the scoring function takes into account the optimistic travel time from some quadrant (or the shopper’s location, initially) to the one under evaluation, as well as the average cost of products in the shopping list available in the stores of the considered quadrant. Both are normalized and given equal weight. The quadrant with lowest score is then selected to be explored next.

At this point we are ready to introduce APX-PSD. We assume that APX-PSD performs several preliminary operations. First, it removes from \( T \) the stores that do not offer any of the products in \( \lambda \), and updates \( T \) and \( Q \) accordingly. Next, it performs two single-source shortest path searches to compute the fastest paths between the shopper’s and customer’s delivery locations and the stores in \( T \). Finally it computes \( ST^U \), which is later used by the scoring function for normalization purposes. The algorithm then initiates a depth-first visit of the quad-tree, constructing shopping routes from the tree’s leaves that look the “most promising” and update the linear skyline accordingly. Such operations are shown in Algorithm 1.

APX-PSD first determines the set of products that cannot be bought from the routes currently stored in \( PR \) and stores such set in \( \lambda' \) (line 1, function GetMissingProducts). Note that if \( \lambda' = \emptyset \) then APX-PSD can terminate, as this implies that the depth first search conducted within \( Q \) already found out feasible shopping routes w.r.t. \( \lambda \) and inserted them into \( LS \). APX-PSD then verifies if the currently considered quadrant \( P \) (initially \( Q \)’s root) is a leaf or not (line 3). If \( P \) is a leaf, then the function ComputePartitionRoutes is executed (line 4). Such function first counts the number of products in \( \lambda' \) that can be bought from \( P \)’s stores – let us suppose \( m \) products. Subsequently, the function generates the set of (partial) shopping routes from \( P \)’s stores, where each such route allows to buy exactly those \( m \) products (stores that do not offer any of the products in \( \lambda' \) are ignored). ComputePartitionRoutes then goes on to compute the Cartesian product between the set of routes currently stored in \( PR \) and that computed from \( P \). The result then becomes the new \( PR \)’s content. Each route in \( PR \) is subsequently checked to verify if its shopping time is above \( ST^U \) – in such case the route is removed from \( PR \). Finally, ComputePartitionRoutes verifies if the routes that are still in \( PR \) buy all the products in \( \lambda \) (i.e., when \( \lambda' = \emptyset \)) and, if so, attempts to insert them into \( LS \).

If \( P \) is an intermediate node of \( Q \) then APX-PSD first proceeds to determine in which partition the routes currently stored in \( PR \) terminate and set \( SRC \) accordingly (line 6, function GetStart). Then, APX-PSD scores \( P \)’s four sub-quadrants (line 5, function Score) and finally recursively invokes itself by considering such sub-quadrants in ascending order of score (line 10). Note that the function ReScore (line 12) takes advantage of any update in \( PR \) to update the scores of partitions still within \( Z \), and thus direct the evaluation towards promising candidate routes.

4 EXPERIMENTAL EVALUATION

Due to limited space next we present a brief summary of our experimental results – a thorough analysis of all parameters, including datasets of other cities, can be found in [1]. In order to evaluate the APX-PSD approach we used datasets containing the road network and realistic pseudo-stores locations for Oslo\(^1\), which has 305,175 vertices, 330,633 edges, and a total of 207 stores located all over the city. The parameters considered for the experimental evaluation are: (1) store cardinality (i.e., number of stores in a network), (2) distribution of the products’ costs in a store, (3) spatial distribution

---

\(^1\)https://sites.google.com/ualberta.ca/nascimento/datasets/
of differently sized stores, and (4) size of shopping lists. Store cardinality varies between 10 and 50, with 25 being the default value. The default price distribution follows a $U(5,15)$ distribution, but we also looked into the case where prices decrease or increase with the distance of the store w.r.t. the city’s centre (assumed to be the centroid of the network’s vertices). The store size is modelled by the percentage it sells of all possible products. The default case is a random variable, though we also considered the cases where the farther (closer) to the city’s center, the smaller the store size. The size of the shopping list varied between 5 and 15, with the default being 10. Finally, experiments discussed in [1] suggest that setting the capacity of leaf nodes of the underlying PR quad-tree to 8 is a good compromise between processing time and solution quality.

In order to compare APX-PSD’s performance we implemented an optimal approach, BSL-PSD [1], that, due to the complexity of the problem discussed earlier, is only feasible for small problems. Comparing an optimal linear skyline (opt-LS) to an approximated one (apx-LS) requires comparing two aspects of skylines: optimality and coverage. The optimality gap measures the normalized fraction of the non-dominated area created by apx-LS ($A_{apx}$) that does not overlap with the non-dominated area created by opt-LS ($A_{opt}$). The coverage gap measures the non-dominated area of opt-LS that does not overlap with $A_{apx}$ and it is normalized w.r.t the total non-dominated area of opt-LS. The lower both are, the better.

**Effect of store cardinality.** Table 1(a) shows that the optimality gap increases when increasing the store cardinality, while the coverage gap decreases. We explain such behaviour by observing how quad-trees and the scoring function react to such changes. When store cardinality increases the number of stores in each quad-tree leaf increases accordingly, which smooths the average cost of each product in different partitions reducing the impact of product costs in APX-PSD’s scoring function. Therefore, the linear skyline will include routes with higher costs decreasing the coverage gap. Consequently, with larger store cardinality BSL-PSD generates shopping routes with lower shopping time that APX-PSD fails to find, thus increasing the optimality gap. BSL-PSD’s processing time increases when store cardinality increases, mainly due to an increased number of stores to consider. Conversely, APX-PSD’s processing time exhibits small changes as increasing the store cardinality only increases the quad-tree’s depth with fixed leaf capacity.

**Effect of product cost distribution.** In Table 1(b) we can see that the “Declining” and “Rising” cases have comparable coverage and optimality gaps to the “Uniform” one. Similarly to the previous case, we argue that these results can be explained by observing the characteristics of APX-PSD’s scoring function, which makes it insensitive to different distributions. BSL-PSD’s processing time is larger when dealing with the “Decreasing” and “Increasing” cases. Since the locations of larger stores are concentrated in certain areas, BSL-PSD takes longer to generate routes with minimum cost, an effect that is further compounded by the city’s network size. Finally, note that APX-PSD’s processing time remains unaffected.

**Effect of shopping list size.** The optimality (coverage) gap increases (decreases) with the shopping list size as evidenced in Table 1(d). Larger shopping lists require more traversals of the network. As shopping routes are appended to existing partial routes, the previous not-so-good choices remain and their effects are further compounded by new potentially not-so-good choices as the algorithm’s execution evolves, thus worsening the approximation. As a result both the shopping time and cost increase which increases the optimality gap and decreases the coverage gap. BSL-PSD’s processing time increases when increasing the shopping list size. This can be explained by observing that, on average, large shopping lists require more stores per route to be satisfied, and thus likely require to evaluate more candidate routes. Conversely, APX-PSD is mildly affected by such increase due to noticeably fewer candidates it generates and evaluates by design.

### Table 1: APX-PSD’s performance analysis.

| Store Cardinality | Optimality Gap | Coverage Gap | Time (sec.) |
|-------------------|----------------|--------------|-------------|
| 10                | 0.31           | 0.081        | 0.71        |
| 25 (default)      | 0.40           | 0.064        | 0.78        |
| 50                | 0.47           | 0.054        | 0.96        |
|                   |                |              |             |
| Product Cost Distribution | Optimality Gap | Coverage Gap | Time (sec.) |
| $U(5, 15)$ (default) | 0.40           | 0.064        | 0.78        |
| Declining         | 0.36           | 0.067        | 0.79        |
| Rising            | 0.37           | 0.071        | 0.81        |
|                   |                |              |             |
| Store Size Spatial Distribution | Optimality Gap | Coverage Gap | Time (sec.) |
| Random (default)  | 0.40           | 0.064        | 0.78        |
| Decreasing        | 0.33           | 0.052        | 0.92        |
| Increasing        | 0.41           | 0.069        | 0.85        |
|                   |                |              |             |
| Shopping List Size | Optimality Gap | Coverage Gap | Time (sec.) |
| 5                 | 0.26           | 0.085        | 0.76        |
| 10 (default)      | 0.40           | 0.064        | 0.78        |
| 15                | 0.46           | 0.035        | 0.84        |

### REFERENCES

[1] S. Anwar, F. Lettich, and M.A. Nascimento. Towards a personal shopper’s dilemma: Time vs cost. arXiv:2008.11705 [cs.DB], 2020.

[2] M. Shekelyan et al. Linear path skyline computation in bicriteria networks. In DASFAA, pages 173–187, 2014.

[3] S. Borzsony, D. Kossmann, and K. Stocker. The skyline operator. In IEEE ICDE, pages 421–430, 2001.

[4] M Sharifzadeh, M R. Kolahdouzan, and C. Shahabi. The optimal sequenced route path planning approach. In IEEE ICDE, pages 261–272, 2010.

[5] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S. Teng. On trip planning queries in spatial databases. In SSTD, pages 273–290, 2005.

[6] H. Kriegel, M. Renz, and M. Schubert. Route skyline queries: A multi-preference path planning approach. In IEEE ICDE, pages 41:1–41:10, 2017.

[7] E. Ahmadi, C.F. Costa, and M.A. Nascimento. Best-compromise in-route nearest neighbor queries. In ACM SIGSPATIAL, pages 41:1–41:10, 2017.