An arc orienteering algorithm to find the most scenic path on a large-scale road network

Ying Lu Cyrus Shahabi
Integrated Media Systems Center, University of Southern California, Los Angeles, CA
{ylu720,shahabi}@usc.edu

ABSTRACT

Traditional route planning problems mainly focus on finding the shortest path considering the travel distance or time. In this paper, we aim to find the most scenic path that offers the most beautiful sceneries on the arcs of a path while the total travel cost (distance or time) is within a user-specified budget. This is a challenging problem as the optimization objective is to maximize the value of the path (i.e., its scenic value) instead of minimizing its cost (distance or time). The problem can be formulated as a variant of the Arc Orienteering Problem (AOP), which is a well-known NP-hard combinatorial optimization problem. Due to the fast response-time requirements of interactive mobile and online applications (e.g., within 300 milliseconds) and the large scale of real-world road networks, existing heuristic algorithms for AOP fail to solve the most scenic road problem. Therefore, unlike the existing approaches for AOP where they treat the road network as a traditional graph in which all-pair distances are pre-computed a priori, in this work, we treat the road network as a spatial network, utilizing the techniques from the field of spatial database: ellipse pruning and spatial indexing. Experiments on two real-world datasets demonstrate the efficiency and accuracy of our proposed algorithms, which can achieve over 95% accuracy within 300 milliseconds on large-scale datasets (over 100K network nodes).

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Application—*Spatial databases and GIS*

General Terms

Algorithms, Experimentation, Performance

Keywords

Scenic path, Trip routing, Arc orienteering problem, Geo-tagging

1. INTRODUCTION

Due to the availability of large transportation (e.g., high quality road network data) and transportation-related (e.g., pollution, safety, scenic) data as well as the ubiquity and popularity of online, mobile and in-car navigation systems, many innovative path-planning applications have been proposed. Examples include the fastest path [6], the most scenic/attractive path [18], the safest path [1], the least polluted path [24], etc. The algorithmic techniques in support of these applications mostly include the fast computation of the minimum-cost path where either the cost is travel-time or pollution or some inverse function of a *utility* such as safety or beautifulness. However, in practice, while we may prefer a safer or more attractive path, there is usually a limit on how much farther we are willing to travel for this purpose. For example, consider a mobile tourism app where we can enter the amount of time we have, a source and a destination and then ask for the most scenic route. Here, we are not looking for the shortest path between our source to destination but the one that is most scenic as long as the total travel time stays within our time *budget*.

While this problem seems very similar to the traditional shortest-path problem, due to its *budget* restriction and its *maximization* objective (instead of minimization), it becomes a different problem with its own unique challenges. In this paper, without loss of generality, we focus on the running example of our mobile tourism app, where the objective is to find the most scenic path: to find a path that offers the most beautiful sceneries on the arcs of the path while the total travel cost is within a defined budget. This problem can be formulated as a variant of the Arc Orienteering Problem (AOP) [23], which is a well-known NP-hard combinatorial optimization problem [9]. Exact algorithms for AOP are based on integer programming [19] or branch-and-cut [27] and hence are only capable of solving for small graphs, thus approximate algorithms are mainly used in practice.

The AOP problem has been studied very recently by different fields: as recent as 2014 and 2015 by theoreticians [5, 9] providing approximate solutions with proven bounds, as well as by the fields of operation research [23] and transportation research [27] proposing heuristic approaches. However, none of these approaches focus on the practicality of the solution where milliseconds response time is required (given, say, the interactivity requirements of a mobile app) to find the most scenic route on a large real road network.

The main reason for their impracticality is that they all assume the shortest path between every pair of the graph's vertices are pre-computed and stored a priori. This is of course impractical for real world road networks for two reasons: scale and dynamism. Hence, invocation of some sort of an online shortest-path computation is unavoidable. Note that this online computation can be a state-of-the-art shortest-path algorithm from the field of spatial-databases such as [13, 20, 28] that relies on its own pre-computation or even perhaps a time-dependent algorithm finding the least travel-time such as [6]. Nevertheless, once such an algorithm, no matter how fast, is invoked repeatedly, it dominates the cost of the proposed approaches in [23] and [27].

SIGSPATIAL'15, November 03 - 06 2015, Bellevue, WA, USA
2015 ACM ISBN 978-1-4503-3967-4/15/11$15.00
DOI: http://dx.doi.org/10.1145/2820783.2820835.

Interestingly, the solution to this dilemma is by relaxing another simplifying assumption made by these approaches: the input road network is a generic directed graph. That is, none of these approaches take advantage of the geographical properties associated with these networks. Instead, we utilize the techniques from the field of spatial databases, in particular using ellipse filtering and spatial indexing to filter out non-interesting arcs of the network, geographically, and hence significantly reduce the number of invocations of the shortest-path computations.

Finally, in the process of studying these two recent metaheuristic algorithms for AOP: Greedy Randomized Adaptive Search Procedure-based algorithm (GRASP) [23] and Iterated Local Search (ILS) [27], we identified and fixed a shortcoming in their approach. In particular, the value improvement of the path at each iteration is small since they do not consider both cost (e.g., distance) and value (e.g., attractiveness). For example, at each iteration, ILS performs depth-first search (DFS) to examine which arcs need to be selected to update the solution without comparing arcs based on their costs and values. Hence, in one of our experiments (see Sec. 6), we show that even for a small directed graph, our approach, even without the spatial database techniques, still outperforms GRASP and ILS. Moreover, to show the effectiveness of our spatial-database techniques, we added them to GRASP and ILS and our results show significant improvement over the traditional versions.

We would like to mention that there are several studies [18, 21, 29] about finding the scenic path. However, these studies mainly focus on the attractiveness value calculation modeling, instead of the routing problem itself. In this paper, we focus on the routing algorithm assuming all the attractiveness values are available. There are also a number of studies on routing problems that consider both travel costs and attractiveness values. However, their problem settings are different from AOP. For example, in the orienteering problem (OP) [25, 26], the attractiveness values are associated with the nodes instead of the arcs in the road network.

In sum, the main contribution of this paper is a series of algorithms to solve the AOP problem with milliseconds response time for large road networks, by utilizing the techniques from the field of spatial databases. In particular:

- Our first algorithm, ILS*(C), increases the value improvement of ILS at each iteration based on two criteria "ImprovePotential" and "QualityRatio" to decide which arcs should be visited first to update the solution, considering both costs and values of the arcs. This modification by itself improves upon GRASP and ILS for generic directed graphs.
- To reduce the number of iterations, without loss of value improvement, our second algorithm, ILS*(CE), takes advantage of the ellipse property to efficiently prune irrelevant arcs to avoid unnecessary shortest path computations.
- To further speed up each iteration, our third algorithm, ILS*(CEI), utilizes spatial indexes (e.g., Grid) to reduce the search space significantly.
- Our extensive experiments using two real-world datasets, one of which is in large scale with more than 100 thousands of nodes, demonstrate that our proposed algorithms significantly outperform the state of-the-art solutions for AOP in terms of efficiency and accuracy. Finally, ILS*(CEI) can achieve over 95% accuracy within 300 milliseconds on the large-scale dataset.

The remainder of this paper is organized as follows. In Sec. 2, we formally define the AOP problem. We review the related work in Sec. 3 and present two baselines in Sec. 4. In Sec. 5, we propose our search algorithms. Sec. 6 reports our experimental results. Finally, Sec. 7 concludes the paper and discusses the future work.

2. PROBLEM DEFINITION

Let $G(V, A)$ be a directed graph representing a road network, where V is a set of nodes and A is a set of arcs. Each arc a in the arc set A is associated with a traveling cost $a.cost$ ($a.cost > 0$) and a non-negative value $a.value$ ($a.value \geq 0$) denoting the "attractiveness value". We denote the arc a in the road network with a positive attractiveness value, i.e., $a.value > 0$ as an "attractive arc". In our work, a path from a node v_1 to a node v_2, denoted by $P(v_1 \overset{sp}{\rightsquigarrow} a_1 \overset{sp}{\rightsquigarrow} a_2, ..., a_n \overset{sp}{\rightsquigarrow} v_2)$, is represented as a sequence of attractive arcs $< a_1, a_2, ..., a_n >$ starting from the node v_1 to the node v_2. In which, n is the *length* of the path, i.e., the number of attractive arcs in the path. The path segment $l_i = a_i \overset{sp}{\rightsquigarrow} a_{i+1}$ between two adjacent attractive arcs a_i and a_{i+1} in the path, named *blank path segment*, is the shortest path from the ending vertex of the attractive arc a_i to the starting vertex of the attractive arc a_{i+1}. Let $l_i.startvex$ and $l_i.endvex$ represent the starting vertex (i.e., the ending node of a_i) and ending vertex (the starting node of a_{i+1}) of the blank path segment l_i, respectively. Further, let $a_i.pre = a_{i-1}$ and $a_i.post = a_{i+1}$ represent the previous and the following attractive arcs of the attractive arc a_i in the path P, respectively. The cost of the path is the total cost of the arcs (including arcs in the blank path segments) in the path. The value of the path is the total value collected from the attractive arcs in the path. The arc orienteering problem (AOP) [23] is defined as follows:

DEFINITION 1 (AOP). *Given a graph $G(V, A)$ and a source node s and a destination node d, further given a total travel cost budget B, the **arc orienteering problem (AOP)** is finding a path $\mathcal{S}(s \overset{sp}{\rightsquigarrow} a_1 \overset{sp}{\rightsquigarrow} a_2, ..., a_n \overset{sp}{\rightsquigarrow} d)$ from s to d such that the total travel cost is no more than the cost budget B and the total collected value is maximized, i.e.,*

$$\textit{Maximize} \quad \sum_{\forall a \in \mathcal{S}} a.value \tag{1}$$

$$\textit{Subject to:} \quad \sum_{\forall a \in \mathcal{S}} a.cost \leq B \tag{2}$$

In this paper, we allow multiple visits of a vertex in the graph. However, an arc is not allowed to appear twice in the solution.

Fig. 1 illustrates an example of AOP. In the graph G in Fig. 1, the vertex s is the source, the vertex d is the destination, and arcs in bold are the attractive arcs $\{a_1, a_2, a_3\}$ where the numbers on the arcs are the attractiveness values of the corresponding arcs (e.g., $a_1.value = 18$) while the values of other arcs are zero. Assuming the costs of the all the arcs in the graph are 1, then given the cost budget $B = 5$, the solution of AOP (s, d, G, B) is the path($s \overset{sp}{\rightsquigarrow} a_3 \overset{sp}{\rightsquigarrow} a_4 \overset{sp}{\rightsquigarrow} a_2 \overset{sp}{\rightsquigarrow} d$), i.e., the path in red in Fig. 1.

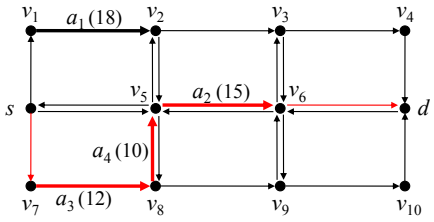


Figure 1: An example of AOP

3. RELATED WORK

In this section, we describe the related work in two categories.

3.1 Scenic trip planning

The problem of finding a scenic path is very useful in practice, hence many existing work studied different variations of the problem. Quercia et al. [18] proposed a probability model to calculate

the "emotional" score on the route based on the crowdsourced people's perceptions. They studied the problem of finding a path from a source s to a destination t with two constraints: the path should be short and the total value is maximized. To this end, they developed a two-phase algorithm: (1) find the top-k shortest paths from s to t; (2) among those k paths, find the one with maximum value. However, for a sightseeing application for example, the user may be willing to take a longer scenic path as long as the travel time of the path is within his/her time restrictions (i.e., budget). Hence, in our problem setting, we have the cost constraint B, and hence their solutions cannot be applied to our problem. Zheng et al. also studied a similar problem [29]. Their goal was to find a shortest path with maximum scenic landscapes. To solve the problem, they designed an algorithm that transform the existing road network G to a "scenic" network G' where all nodes on the graph are scenic landscapes. Then the problem can be easily solved by finding the shortest path in G'. Their problem is also different from us, because they also ignored the cost constraint B.

In the literature, Skoumas et al. targeted a similar problem but without knowing the explicit information (e.g., edge value) of the road network [21, 22]. Thus, their focus was to derive that information by mining geo-textual objects. However, unlike this study that is mainly focusing on the "scenic" value calculation modeling, in this work, we focus on the routing algorithm assuming all the attractiveness values are available.

The AOP problem is also related to the classical orienteering problem (OP), which aims to find a path such that the total value is maximized while the travel cost is within a pre-defined cost budget. However, the difference between AOP and OP is that, the values are associated with *arcs* in AOP whereas the values are associated with vertices in OP. There are a large number of studies (e.g., [4, 12, 14, 25, 26]) on OP. Besides that, there are also a few studies considering arc routing variants of the OP problem where the value is associated with each arc instead of each vertex. For example, the mixed Orienteering problem [19], where all edges and vertices have values. The profitable rural postman problem [2,10,17] finds a path that goes through all the arcs in a required arc set such that the total collected values is maximized. Another arc routing variant is the capacitated arc routing problem [3] which finds a set of routes satisfying the constraint on the resource threshold and the constraint on the capacity of each vehicle, as well as maximizing the total value needed. The profitable arc tour problem [3, 7] finds a path to maximize the difference between value and travel cost. All the problems are different from AOP defined in Sec. 2.

3.2 AOP algorithms

Next we describe some algorithms that were specifically designed to solve the AOP problem. We classify them into three categories: (1) Exact algorithms [19, 27]. Deitch proposed an algorithm using integer linear programming to solve the problem [19]. Verbeeck et al. designed the branch-and-cut algorithm [27]. However, both of them are too slow in large-scale graphs because AOP is an NP-hard problem [5, 9]. (2) Approximate algorithms [5, 9]. The focus of these studies is to develop approximate algorithms with proven theoretical bounds. However, the proposed algorithms are still very slow for large-scale graphs. (3) Heuristic algorithms [23, 27]. To our best knowledge, there are only two recent studies in the literature which are most relevant to our work: GRASP [23] and ILS [27], which will be described in Sec. 4.

4. BASELINE METHODS

There are two metaheuristic algorithms for the AOP problem in the literature: the Greedy Randomized Adaptive Search Procedure-based algorithm (GRASP) [23] and Iterated Local Search (ILS) [27].

4.1 GRASP

The first metaheuristic algorithm is GRASP [23]. It first selects the cheapest arc a from the network G that can be *feasibly* (see Definition 2) inserted between the source s and the destination d, and then initialize the solution $\mathcal{S}$ as the path P_0. Subsequently, at each iteration, GRASP generates a set $\mathcal{C}$ of candidate solutions, and calculates the average value avg of the solutions in $\mathcal{C}$. Then among the solutions in $\mathcal{C}$ whose values are no less than the average value avg, GRASP randomly selects a solution as the new solution. The candidate solution set $\mathcal{C}$ is generated exhaustively: for every pair of adjacent arcs a_i, a_{i+1} (i.e., blank path segment) in the current solution $\mathcal{S}$, it chooses the cheapest arc a from G that can be *feasibly* inserted between a_i and a_{i+1}. GRASP performs this procedure iteratively until the execution time reaches the time threshold (e.g., 300 milliseconds).

DEFINITION 2. *We say an arc $a \notin \mathcal{S}$ can be **feasibly inserted** between two vertices v_1, v_2 in the solution $\mathcal{S}(s,\ldots,v_1,v_2,\ldots,d)$ iff the total cost of the new solution $\mathcal{S}'$ is still no more than B, i.e., $\mathcal{S}'(s,\ldots, v_1 \overset{sp}{\rightsquigarrow} a \overset{sp}{\rightsquigarrow} v_2,\ldots,d).cost \leq B$. We say an arc $a' \notin \mathcal{S}$ can **feasibly update** an arc $a \in \mathcal{S}(s,\ldots, a.pre \overset{sp}{\rightsquigarrow} a \overset{sp}{\rightsquigarrow} a.post,\ldots,d)$ iff $\mathcal{S}'(s,\ldots, a.pre \overset{sp}{\rightsquigarrow} a' \overset{sp}{\rightsquigarrow} a.post,\ldots,d).cost \leq B$. Additionally, we call the procedure of checking whether an arc can be feasibly inserted into (or can feasibly update an arc in $\mathcal{S}$) $\mathcal{S}$ as the **feasibility checking** for the arc.* □

Example 1: Consider the road network G in Fig. 1 and assume the budget is 5. The trace of the GRASP algorithm is shown in Table 1. Iteration 0 (initialization): the algorithm first selects the cheapest arc a_2 among all the arcs in the graph since the cost (=3) of the path $(s \overset{sp}{\rightsquigarrow} a_2 \overset{sp}{\rightsquigarrow} d)$ after the insertion of a_2 is lower than the costs (=5) of the paths after the insertion of any other arcs $\{a_1, a_3, a_4\}$, then it initializes the solution $\mathcal{S}$ with the path $(s \overset{sp}{\rightsquigarrow} a_2 \overset{sp}{\rightsquigarrow} d)$. Iteration 1: there are two blank path segments $l_1(s \overset{sp}{\rightsquigarrow} a_2)$ and $l_2(a_2 \overset{sp}{\rightsquigarrow} d)$ in the current solution $\mathcal{S}$. For l_1, arcs a_1, a_3 and a_4 can be feasibly inserted into l_1, and their corresponding updated solutions, $(s \overset{sp}{\rightsquigarrow} a_1 \overset{sp}{\rightsquigarrow} a_2 \overset{sp}{\rightsquigarrow} d)$ and $(s \overset{sp}{\rightsquigarrow} a_3, a_4, a_2 \overset{sp}{\rightsquigarrow} d)$, are referred as candidate solutions as their costs are the same (=5). Note that a_3 (resp., a_4) appears in the shortest path between s and a_2 through a_4 (resp., a_3). For the blank path segment l_2, no arcs can be feasibly inserted into l_2. After generating the candidate solution set, the algorithm selects the solution $(s \overset{sp}{\rightsquigarrow} a_3, a_4, a_2 \overset{sp}{\rightsquigarrow} d)$ which has the value (=37, the optimal solution) higher than the average value (=(37+33)/2=35) of the candidate solutions. □

Table 1: Trace of GRAPS algorithm in Example 1

Iteration#	candidates computation	candidate solution set	solution $\mathcal{S}$
0	$argmin_{cost}\{$ $(s \overset{sp}{\rightsquigarrow} a_1 \overset{sp}{\rightsquigarrow} d)$, $(s \overset{sp}{\rightsquigarrow} a_2 \overset{sp}{\rightsquigarrow} d)$, $(s \overset{sp}{\rightsquigarrow} a_3 \overset{sp}{\rightsquigarrow} d)$, $(s \overset{sp}{\rightsquigarrow} a_4 \overset{sp}{\rightsquigarrow} d)\}$	$argmax_{value}\{$ $(s \overset{sp}{\rightsquigarrow} a_2 \overset{sp}{\rightsquigarrow} d)\}$	$(s \overset{sp}{\rightsquigarrow} a_2 \overset{sp}{\rightsquigarrow} d)$
1	$argmin_{cost}\{$ $\{(s \overset{sp}{\rightsquigarrow} a_1 \overset{sp}{\rightsquigarrow} a_2 \overset{sp}{\rightsquigarrow} d)$, $(s \overset{sp}{\rightsquigarrow} a_3, a_4, a_2 \overset{sp}{\rightsquigarrow} d)\}$ $argmin_{cost}\{\emptyset\}$	$argmax_{value}\{$ $(s \overset{sp}{\rightsquigarrow} a_1 \overset{sp}{\rightsquigarrow} a_2 \overset{sp}{\rightsquigarrow} d)$, $(s \overset{sp}{\rightsquigarrow} a_3, a_4, a_2 \overset{sp}{\rightsquigarrow} d)$, $\emptyset\}$	$(s \overset{sp}{\rightsquigarrow} a_3, a_4, a_2 \overset{sp}{\rightsquigarrow} d)$

GRASP has the following drawbacks:

1. Slow iteration. GRASP generates a set of candidate solutions at each iteration. To compute each candidate, it searches the entire graph to find the cheapest arc that can be feasibly used for insertion where the dominant component is

the *feasibility checking* of arcs. GRASP requires $\mathcal{O}(L|A|)$ number of feasibility checking at each iteration, where L is the solution length, $|A|$ is the total arc number in the road network. Let $\mathcal{O}(FC)$ be the complexity of each feasibility checking, then the complexity of each GRASP iteration is $\mathcal{O}(L|A|) * \mathcal{O}(FC)$.

2. Based on all-pair shortest path pre-computation. From Definition 2, we know that the dominant operation for each feasibility checking is the shortest path calculation (each feasibility checking needs two shortest path calculations). GRASP precomputes the shortest path between all the pairs of vertices in the graph, so the complexity $\mathcal{O}(FC)$ of each feasibility checking is $\mathcal{O}(1)$. However, the storage requirement of pre-computing the all-pair shortest paths is $\mathcal{O}(|V|^3)$ [20], which is impractical for a large-scale road network. Thus an online shortest path computation is needed for the feasibility checking. To our knowledge, the state-of-the-art shortest path algorithm is *Contraction Hierarchies* (CH) [28] whose space complexity is $\mathcal{O}(|V|log|V|logD)$ and the time complexity is $\mathcal{O}(L + log^2|V|log^2D)$, where $D = l_{max}/l_{min}$, l_{max} (resp., l_{min}) is the largest (resp., smallest) cost of the arcs in the road network [30].
3. Small value improvement at each iteration. It only considers the cost for candidate solution generation.

4.2 ILS

To speed up the iteration, the ILS [27] approach is proposed by using depth-first search (DFS) strategy at each iteration.

ILS follows the iterated local search framework. It initializes the solution and finds a new solution at each iteration by invoking the subprocedure DFS. This procedure applies DFS to find a reachable path $path$ to close the "gap" between a vertex $start$ and a vertex end. To reduce the DFS search space, ILS also has the feasibility checking mechanism for the arcs to be visited. Beside that, it uses a parameter ($maxDepth$) to restrict the depth of the search. If a reachable path $path$ can be found that can feasibly close the "gap" between $start$ and end (i.e., $path.cost \leq budget$) and improves the value (i.e., $path.value > minVal$), then ILS closes the "gap" between $start$ and end with $path$. ILS perturbs the solution by removing a path segment from the solution. Specifically, ILS removes R consecutive arcs from the current solution $\mathcal{S}$ starting from the A-th arc in $\mathcal{S}$. As shown in Algorithm 1, Lines 2-4 and 7-8 show the way to set the two variables A and R. Basically, A and R are initialized as 1 (Line 2), and after each iteration both A and R are increased by one if there is no improvement (Line 8), otherwise both A and R are reset to be one (Line 7).

Example 2: We use the running example in Fig. 1 to illustrate ILS [27]. Table 2 gives the trace of the algorithm. For a node, assume DFS visits its edges in clockwise order starting from the edge directing to the north. Iteration 0: initially, ILS applies the DFS to find the feasible path $< s, v_1, v_2, v_3, v_4, d >$ (i.e., $s \overset{sp}{\rightsquigarrow} a_1 \overset{sp}{\rightsquigarrow} d$ with value 18). Iteration 1–2: it tries to remove the path segments $< s, v_1 >$ and $< v_1, v_2, v_3 >$, respectively, however, no arcs can be found to feasibly close the "gap". Iteration 3: by removing the path segment $< v_2, v_3, v_4, d >$ from the current solution, the procedure DFS closes the gap with a new path segment $< v_2, v_5, v_6, d >$ and improves the solution value to 33. The algorithm continues to follow the same way to remove path segments from the solution and results in no improvement at Iterations 4–6. At Iteration 7, the solution value is improved to 37 by replacing the path segment $< s, v_1, v_2, v_5, v_6 >$ with the new path segment $< s, v_7, v_8, v_5, v_6 >$, and the algorithm reaches the optimal solution $< s, v_7, v_8, v_5, v_6, d >$ (i.e., $s \overset{sp}{\rightsquigarrow} a_3, a_4, a_2 \overset{sp}{\rightsquigarrow} d$). □

Algorithm 1 ILS [27] (G: road network, s: source, d: destination, B: maximum travel distance, `TimeThreshold`)

1: $\mathcal{S} \leftarrow$ DFS (s, d, B, 0) *//Initialization*
2: A = 1, R = 1
3: **if** ($R > \mathcal{S}.length$) **then** $R = 1$
4: **if** ($A + R > \mathcal{S}.length - 1$) **then** $R = \mathcal{S}.length - 1 - A$
5: remove R arcs from $\mathcal{S}$ starting from the A-th arc *//Perturbation*
6: $path \leftarrow$ DFS (the A-th arc, the $A + R$-th arc, $B - \mathcal{S}.cost$, total score of the removed arcs)
7: **if** ($\mathcal{S}$ has value improvement) **then** A = 1, R = 1
8: **else** $A = A + 1$; $R = R + 1$
9: **repeat** Steps 3 – 8 until the execution time reaches `TimeThreshold`

Subprocedure DFS ($start$, end, $budget$, $minVal$)

10: Apply the DFS strategy to find a reachable path $path$ from $start$ to end. *//maxDepth and feasibility arc checking*
11: **if**($path.cost \leq budge$ and $path.value > minVal$) **then** close the gap of $\mathcal{S}$ with $path$

Table 2: Trace of ILS algorithm [27] in Example 2

Iteration#	remove path segment	solution $\mathcal{S}$	$\mathcal{S}.value$
0	$\emptyset$	$< s, v_1, v_2, v_3, v_4, d >$	18
1	$< s, v_1 >$	$< s, v_1, v_2, v_3, v_4, d >$	18
2	$< v_1, v_2, v_3 >$	$< s, v_1, v_2, v_3, v_4, d >$	18
3	$< v_2, v_3, v_4, d >$	$< s, v_1, v_2, v_5, v_6, d >$	33
4	$< s, v_1 >$	$< s, v_1, v_2, v_5, v_6, d >$	33
5	$< v_1, v_2, v_5 >$	$< s, v_1, v_2, v_5, v_6, d >$	33
6	$< v_2, v_5, v_6, d >$	$< s, v_1, v_2, v_5, v_6, d >$	33
7	$< s, v_1, v_2, v_5, v_6 >$	$< s, v_7, v_8, v_5, v_6, d >$	37

Unlike GRASP that repeats random local search trials, ILS builds a sequence of local search solutions, i.e., generates one solution at each iteration, and continuously improves the solution. ILS's iterations are faster than those of GRASP because ILS does not compare arcs based on their cost and value but just performs DFS when considering which arcs need to be selected to update the solution. Due to the shorter iteration, for a fixed running time, ILS can perform more iterations than GRASP. However, ILS has the following drawbacks:

1. Small value improvement at each iteration. As shown in Example 2, ILS performs a number of *idle* iterations with no improvement. This is because ILS just performs DFS and examines whether the solution after inserting the arc is within the budget and whether its value has improved, instead of comparing arcs based on their costs and values. Additionally, it does not consider the value nor the cost of the path segment to be perturbed. ILS accelerates each iteration by sacrificing value improvement at each iteration.
2. Still slow iteration for large-scale road network. ILS performs DFS at each iteration, in which the number of feasibility checking is $\mathcal{O}(degree^{maxDepth})$, where $degree$ is the average outward degree of the nodes in the road network and $maxDepth$ is a parameter used to restrict the depth of the DFS search. For a large-scale road network, the parameter $maxDepth$ is also supposed to be large, and thus it is slow for a large-scale road network.
3. Based on all-pair shortest path pre-computation. ILS precomputes the shortest paths between all pair vertices; therefore, similarly it is impractical for a large-scale graph.

5. PROPOSED ALGORITHMS

To overcome the drawbacks of the two baseline methods to support large-scale road networks, we propose a series of algorithms

that increase the value improvement rate based on two new criteria and reduce the iteration time using two spatial techniques: ellipse pruning and spatial indexing.

5.1 ILS*(C): ILS* algorithm with criteria

Our algorithms follow the ILS iterated local search framework. The motivation to choose this framework rather than GRASP is because the ILS framework has lower complexity at each iteration, as discussed in Sec. 4.

Algorithm 2 illustrates the general iterated local search framework. As shown in Algorithm 2, at each iteration, iterated local search algorithms remove (perturb) a path segment from the current solution and choose a set of new arcs to update the solution. To solve the AOP problem based on the iterated local search framework effectively, we need to solve the following two important problems at each iteration: 1) which path segments in the solution should be removed (perturbed) and in what order such that the solution's value improves? and 2) which arcs should be selected to update (insert into) the solution and in what order such that the solution's value improves?

Algorithm 2 ILS General Search Framework

1: $s \leftarrow$ Generate an initial solution
2: $s' \leftarrow$ Perturbation(s)
3: $s \leftarrow$ LocalSearch(s')
4: Repeat Steps 2 – 3 until the execution time reaches `TimeThreshold`

To address these problems, we propose a new iterated local search algorithm with two new criteria, denoted by ILS*(C). We next introduce the two criteria and then present the search algorithm ILS*(C) based on them.

The two criteria are applied on a set of particular arcs. Specifically, each arc a in the current solution $\mathcal{S}$ is associated with a set of candidate arcs which is defined bellow.

DEFINITION 3 (CANDIDATE ARC SET (CAS)). *The candidate arc set of an arc a in the current solution $\mathcal{S}$, denoted by $a.CAS$, is defined as: a set of candidate arcs whose values are positive and can feasibly update a, i.e., $\forall a_c \in a.CAS, a_c.value > 0, (a.pre \overset{sp}{\rightsquigarrow} a_c \overset{sp}{\rightsquigarrow} a.post).cost < B - \mathcal{S}.cost + (a.pre \overset{sp}{\rightsquigarrow} a \overset{sp}{\rightsquigarrow} a.post).cost$.*

Candidate arc sets have the following "inherit" property.

LEMMA 1 (CAS INHERIT). *Given the candidate arc set $a.CAS$ of an arc a, $\forall a_c \in a.CAS$, $a_c.CAS \subseteq a.CAS$.*

PROOF. As shown in Fig. 2, without of loss of generality, let a be any an arc in the current solution $\mathcal{S}$, and let arc a_c be in $a.CAS$. Suppose Lemma 1 is not true, then there exists an arc $a' \in a_c.CAS$, s.t., $a' \notin a.CAS$. Since $a' \in a_c.CAS$, we have $(a.pre \overset{sp}{\rightsquigarrow} a' \overset{sp}{\rightsquigarrow} a.post).cost \leq B - \mathcal{S}'.cost + (a.pre \overset{sp}{\rightsquigarrow} a_c \overset{sp}{\rightsquigarrow} a.post).cost$. Further, $\mathcal{S}'.cost$ = $\mathcal{S}.cost$ - $(a.pre \overset{sp}{\rightsquigarrow} a \overset{sp}{\rightsquigarrow} a.post).cost + (a.pre \overset{sp}{\rightsquigarrow} a_c \overset{sp}{\rightsquigarrow} a.post).cost$. Therefore, $(a.pre \overset{sp}{\rightsquigarrow} a' \overset{sp}{\rightsquigarrow} a.post).cost \leq B - \mathcal{S}.cost + (a.pre \overset{sp}{\rightsquigarrow} a \overset{sp}{\rightsquigarrow} a.post).cost$, i.e., $a' \in a.CAS$ which contradicts the assumption that $a' \notin a.CAS$, and thus Lemma 1 is true. □

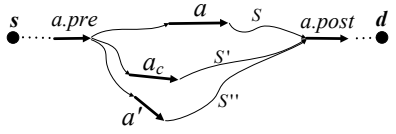


Figure 2: Illustration of the proof of CAS Inherit

To address Problem 1), we propose a criterion, named "ImprovePotential", to give higher priority to update the arcs in the solution that have more potential to improve the solution. The intuition is that arcs in the solution with lower values and more nearby valuable arcs have higher potential to improve the solution. Based on the intuition, we define the "ImprovePotential" score of arcs in the solution bellow:

$$ImprovePotential_N(a) = \frac{\sum_{e \in a.CAS} e.value - a.value}{MaxD_N(a.pre, a.CAS, a.post) - (a.pre \overset{sp}{\rightsquigarrow} a \overset{sp}{\rightsquigarrow} a.post).cost} \quad (3)$$

$$MaxD_N(a.pre, a.CAS, a.post) = \max_{e \in a.CAS} \big((a.pre \overset{sp}{\rightsquigarrow} e \overset{sp}{\rightsquigarrow} a.post).cost\big).$$

For Problem 2), we define a criteria, named "QualityRatio", to give higher priority to the candidate arcs that are more likely to be used to improve the solution. Intuitively, arcs with higher quality ratio, i.e., with higher value and lower cost, have higher chance to improve the solution. Based on the intuition, we define the "QualityRatio" score for the arc a_c in the candidate arc set $a.CAS$ of a solution arc a.

$$QualityRatio_N(a.pre, a.post, a_c) = \frac{(a.pre \overset{sp}{\rightsquigarrow} a_c \overset{sp}{\rightsquigarrow} a.post).value}{(a.pre \overset{sp}{\rightsquigarrow} a_c \overset{sp}{\rightsquigarrow} a.post).cost} \quad (4)$$

We next present the search algorithm ILS*(C) (see Algorithm 3) based on the two criteria. At high level, ILS*(C) follows the iterated local search framework to update the solution iteratively. ILS*(C) initializes the solution by continuously inserting candidate arcs until the entire budget is consumed. Subsequently, ILS*(C) iteratively perturbs the solution by removing an arc from the solution and then replace the selected arc in the solution with its candidate arcs. The choice of the arcs in the solution to be removed is based on their "ImprovePotential" values, and the choice of the candidate arcs to insert to the solution is based on their "QualityRatio" values.

Algorithm ILS*(C) first examines whether the shortest path ($s \overset{sp}{\rightsquigarrow} d$) from the source s to the destination d is within the budget B. If yes, then it returns no solution (Line 1–2). Otherwise we perform the iterated local search (Lines 4–18). We initialize the solution $\mathcal{S}$ with a fake arc $a(s, d, B, 0)$ starting from the source s and ending at the destination d, and $a.cost = B$, $a.value = 0$, and $a.CAS$ is the candidate arc set that can be feasibly inserted between s and d (Line 4). At each iteration, we randomly select an arc a from the candidate arc set $\mathcal{A}_{ip}$ in which the arcs have higher "ImprovePotential" than the average "ImprovePotential" of the arcs in $\mathcal{S}$ (Lines 6–7). To replace the path segment ($a.pre \overset{sp}{\rightsquigarrow} a \overset{sp}{\rightsquigarrow} a.post$) (i.e., to replace the arc a) with a new path segment, we invoke the procedure genPath to generate a path $\mathcal{P}$ from the ending vertex of $a.pre$ to the starting vertex of $a.post$ such that 1) $\mathcal{P}$ is *feasible* and 2) $\mathcal{P}$ results in value improvement, by continuously inserting candidate arcs from $a.CAS$ (Lines 8–9). The details of the genPath procedure is discussed later. If such a path segment $\mathcal{P}$ exists, then we update the solution by replacing the arc a with the new path $\mathcal{P}$ (Lines 11–12). Next, we compute the candidate arc sets for the new arcs in the updated solution (Line 13–15), and update the candidate arc sets for the old arcs in the solution since the cost of the solution changes (Lines 16–18).

We next describe the genPath procedure. We first obtain an arc set $\mathcal{A}_{qr}$ in which arcs $a_c \in \mathcal{A}_{qr}$ are from the candidate arc set $\mathcal{A}$ and the "QualityRatio" scores of the arcs a are no less than the average "QualityRatio" score of the arcs in $\mathcal{A}$ (Lines 20). Then we continously insert the arcs a_c from $\mathcal{A}_{qr}$ into the path to generate a feasible path $\mathcal{P}$ with value improvement until all the arcs in $\mathcal{A}_{qr}$ are used or it reaches the budget limit (Lines 21–27).

Note that to reduce the search space for computing the candidate arc set $a_c.CAS$ of an arc a_c in A, we use the "inherit" technique of initializing $a_c.CAS$ with the arcs inherited from $a.CAS$, where $a_c \in a.CAS$ (see Line 9). According to the inherit property of candidate arc set in Lemma 1, we can safely reduce the search space for computing $a_c.CAS$ from the entire road network to $a.CAS$.

Algorithm 3 ILS*(C) (G: road network, s: source, d: destination, B: maximum travel distance, `TimeThreshold`)

Output: solution $\mathcal{S}$

1: **if** $(s \overset{sp}{\rightsquigarrow} d).cost < B$ **then**
2: $\mathcal{S} \leftarrow \emptyset$
3: **else**
4: $\mathcal{S} \leftarrow \{a(s,d,B,0), a.CAS$=compCAS($G$,$s$,$d$,$B$),$a.ip$=$+\infty\}$
5: **while** the running time is less than `TimeThreshold` **do**
6: let $\mathcal{A}_{ip}$ the arcs a in $\mathcal{S}$, s.t., $a.ip \geq$ avg$(e.ip), \forall e \in \mathcal{S}$
7: randomly pop out an arc a from $\mathcal{A}_{ip}$
8: $b \leftarrow$ the remaining cost budget after removing a from $\mathcal{S}$
9: $\mathcal{P} \leftarrow$genPath($a.pre$,$a.post$,b,$a.value$,$a.CAS$)//`inherit`
10: **if** $\mathcal{P} \neq \emptyset$ **then**
11: Remove a from $\mathcal{S}$
12: $\mathcal{S}.insert(a.pre, a.post, \mathcal{P})$
13: **for** each $a_c \in \mathcal{P} \cup \{a.pre, a.post\}$ **do**
14: $b \leftarrow$ the remaining cost budget after removing a_c from $\mathcal{S}$
15: $a_c.CAS$ = compCAS($a.CAS$, $a_c.pre$, $a_c.post$, b)
16: **for** each $a_c \in \mathcal{S} \setminus \{\mathcal{P} \cup \{a.pre, a.post\}\}$ **do**
17: $b \leftarrow$ the remaining cost budget after removing a_c from $\mathcal{S}$
18: $a_c.CAS$ = updateCAS(a, $a_c.pre$, $a_c.post$, b)

Subprocedure genPath ($start$, end, $budget$, $minVal$, A)

19: $\mathcal{P} \leftarrow \{a_c(start, end, 0, 0)\}$; //$a_c.value = 0$
20: let $\mathcal{A}_{qr}$ the arcs a_c in A, s.t., $a_c.qr \geq$ avg$(e.qr), \forall e \in A$
21: **while** $\mathcal{A}_{qr} \neq \emptyset$ and $\mathcal{P}.cost \leq budget$ **do**
22: randomly pop out an arc a_c from $\mathcal{A}_{qr}$
23: $l \leftarrow \underset{l_i \in \mathcal{P}, l_i.value=0}{argmin}$ Distance(a_c, l_i)
24: $path \leftarrow (l.startvex \overset{sp}{\rightsquigarrow} a \overset{sp}{\rightsquigarrow} l.endvex)$
25: **if** $path.cost \leq budget - \mathcal{P}.cost + l.cost$ **then**
26: $\mathcal{P}$.insert(l, a_c) //$\mathcal{P}$ `is feasible`
27: **if** $\mathcal{P}.value > minVal$ **then**
28: **return** $\mathcal{P}$ //`with value improvement`
29: **else return** $\emptyset$

Subprocedure compCAS(A, v_1, v_2, $budget$)

30: $CAS \leftarrow \forall a \in A$, s.t., $a.value > 0$ and $(v_1 \overset{sp}{\rightsquigarrow} a \overset{sp}{\rightsquigarrow} v_2).cost \leq bgt$.
31: $\forall a \in CAS$, $a.qr = QualityRatio_N(v_1, v_2, a)$
32: **Return** CAS

Subprocedure updateCAS(a, v_1, v_2, $newBudget$)

33: let $oldBgt$ be the old remaining budget after removing a from $\mathcal{S}$
34: **if** $newBgt < oldBgt$ **then**
35: $a.CAS \leftarrow a.CAS \setminus \{a | a \in a.CAS, (v_1 \overset{sp}{\rightsquigarrow} a \overset{sp}{\rightsquigarrow} v_2).cost > newBgt\}$
36: **else if** $newBgt > oldBgt$ **then**
37: $a.CAS \leftarrow a.CAS \cup \{a | a \in a.CAS$ and $a.value > 0$ and $(v_1 \overset{sp}{\rightsquigarrow} a \overset{sp}{\rightsquigarrow} v_2).cost \leq newBgt\}$
38: Return $a.CAS$

Example 3: Table 3 depicts the trace of the ILS*(C) algorithm for the running example in Fig. 1. Iteration 0: we calculate the candidate arc set $a.CAS$ for the fake arc $a(s, d, B, 0)$, obtaining $a.CAS$ = $<a_2$:5, a_1:3.6, a_3:2.4, a_4:2$>$. Next, we insert the candidate arcs a_2, a_1 from $a.CAS$ between s and d sequentially since their "QualityRatio" scores are larger than the average score (=3.25) of $a.CAS$. We obtain the solution $(s \overset{sp}{\rightsquigarrow} a_1 \overset{sp}{\rightsquigarrow} a_2 \overset{sp}{\rightsquigarrow} d)$ with value 33. Iteration 1: we remove arc a_1 from the solution since its "ImprovePotential" score (=∞) is larger than the score of the arc a_2. Finally, we insert the arcs a_3 and a_4 into the solution to close the "gap" obtaining the final solution $(s \overset{sp}{\rightsquigarrow} a_3, a_4, a_2 \overset{sp}{\rightsquigarrow} d)$ with value 37. □

Table 3: Trace of ILS*(C) algorithm in Example 3

Iteration#	Removed arc	Q	Solution S	$S.value$
0	$\emptyset$	$\{a(\infty, <a_2$:5, a_1:3.6, a_3:2.4, a_4:2 $>)\}$	$(s \overset{sp}{\rightarrow} a_1 \overset{sp}{\rightarrow} a_2 \overset{sp}{\rightarrow} d)$	33
1	a_1	$\{a_1(\infty, <a_3$:7.3, a_4:7.3 $>)$, $a_2(0, \emptyset)\}$	$(s \overset{sp}{\rightarrow} a_3, a_4, a_2 \overset{sp}{\rightarrow} d)$	37

However, one question in the genPath procedure is unanswered: where in the solution we should insert each of the candidate arcs? To answer this question, we propose a greedy solution (Line 23): for each candidate arc, we select the closest blank path segment in the solution. As shown in Fig. 3, a_1, a_2 and a_3 are three candidate arcs, where a_1 and a_2 are already inserted in the solution, and l_1, l_2 and l_3 are three blank path segments in the solution. To insert the candidate arc a_3, we choose the closest blank path segment (l_2).

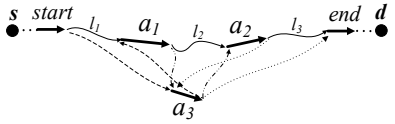


Figure 3: Illustration on the blank path segment selection where candidate arcs can be inserted. The dashed path ($start \overset{sp}{\rightsquigarrow} a_3 \overset{sp}{\rightsquigarrow} a_1$) is the path of inserting a_3 into l_1, the dotted path ($a_2 \overset{sp}{\rightsquigarrow} a_3 \overset{sp}{\rightsquigarrow} end$) is the path of inserting a_3 into l_3, and the dash-dot-hybrid path ($a_1 \overset{sp}{\rightsquigarrow} a_3 \overset{sp}{\rightsquigarrow} a_2$) is the path of inserting a_3 into l_2.

ILS*(C) can effectively solve Problems 1) and 2) to improve the quality of the solution at each iteration, however, the iteration is still slow in computing and maintaining the candidate arc set of each arc in the solution where a large number of feasibility checkings are required. The time complexity of ILS*(C) at each iteration is $\mathcal{O}(L|A|) * \mathcal{O}(FC)$. "Inherit" technique can effectively reduce the search space. To further accelerate the iteration without loss of value improvement, we propose two spatial techniques in Sections 5.2 and 5.3, respectively.

5.2 ILS*(CE): ILS*(C) algorithm with ellipse

As we discussed in Sec. 4, due to the large size of the real world road networks and the dynamism of their edges' travel times, we cannot assume a pre-computed all-pair shortest path matrix. Hence, we need to include an online shortest-path computation algorithms such as the CH [28] algorithm[1]. Therefore, we focus on reducing the number of times we should invoke the shortest-path computation algorithm during the feasibility checking phase of the ILS algorithm. In particular, instead of applying the shortest path calculation for the feasibility checking of all the arcs in the road network, we take advantage of the ellipse property to efficiently perform the feasibility checking (with complexity of $\mathcal{O}(1)$) for all the network arcs first, and then use the online shortest path computation to check the remaining arcs for refinement.

According to Lemma 2, we can apply the ellipse pruning strategy in Lemma 3 during the calculation of the candidate arc set which

[1]Note that once we replace pre-computation with invoking an online shortest-path algorithm, we can use a time-dependent shortest path algorithm (e.g., [6]) to consider travel-time instead of travel-distance.

can avoid unnecessary feasibility checks and thus reduce the number of times we invoke the shortest path computation.

LEMMA 2. *Given two vertices v_1, v_2 in a road network $G(V, E)$, a cost budget b, and given a path path from v_1 to v_2 with road network cost $d_N(v_1, v_2)$, if there exists a point in the path that is outside of the ellipse $Elip(v_1, v_2, b)$ whose foci are (v_1, v_2) and whose major axis is b, then $path.cost > b$.*

PROOF. As shown in Fig. 4, $\forall o \in path$,

$$\begin{aligned} path.cost = d_N(v_1, v_2) &= d_N(v_1, o) + d_N(o, v_2) \\ &\geq d_E(v_1, o) + d_E(o, v_2), \end{aligned}$$

where $d_E(v_1, o)$ (resp., $d_E(o, v_2)$) denotes the Euclidean distance between v_1 (resp., v_2) and o. According to the definition of ellipse, $\forall o \in path$, o is outside of the ellipse, then $d_E(v_1, o) + d_E(o, v_2) > b$, i.e., $path.cost > b$, and thus Lemma 2 holds. □

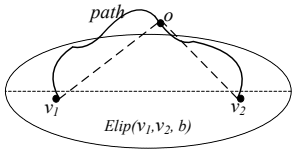


Figure 4: Illustration on proof of ellipse pruning

DEFINITION 4. *Given an arc a and a ellipse $Elip$, we say a is **inside** of $Elip$ iff $\forall p \in a$, $p \in Elip$; a is **outside** of $Elip$ iff $\forall p \in a$, $p \notin Elip$; a **intersects** $Elip$ iff $\exists p \in a$, $p \in Elip$ and $\exists p' \in a$, $p' \notin Elip$.*

LEMMA 3 (ELLIPSE PRUNING). *Given an arc a in the solution, and let b be the budget after removing a from the solution, any arc e that is outside of or intersects with the ellipse $Elip(a_{pre}, a_{post}, b)$ is definitely not a candidate to update a, i.e., $e \notin a.CAS$.*

PROOF. If the arc e is outside of or intersects with the ellipse $Elip(a_{pre}, a_{post}, b)$, then $\exists p \in e$, $p \notin Elip$. According to Lemma 2, we can show that e is not a feasible arc, and thus $e \notin a.CAS$. So Lemma 3 holds. □

LEMMA 4 (ELLIPSE INHERIT). *Given the ellipse $Elip(a.pre, a.post, b)$ of an arc a in the current solution $\mathcal{S}$, $\forall\, a_c \in Elip(a.pre, a.post, b)$, $Elip_c(a_c.pre, a_c.post, b) \subseteq Elip(a.pre, a.post, b)$*

PROOF. As shown in Fig. 2 again, suppose Lemma 4 is not true, then there exists an arc $a' \in Elip_c$, s.t., $a' \notin Elip$. Since $a' \in Elip_c$, we can obtain that $d_E(a_c.pre, a') + a'.cost + d_E(a', a_c.post) \leq B - \mathcal{S}'.cost + (a.pre \overset{sp}{\rightsquigarrow} a_c \overset{sp}{\rightsquigarrow} a.post).cost$ which is no larger than $B - \mathcal{S}.cost + (a.pre \overset{sp}{\rightsquigarrow} a \overset{sp}{\rightsquigarrow} a.post).cost$ according to Lemma 1. Thus $a' \in Ellipse$ which contradicts the assumption that $a' \notin Ellipse$. Hence Lemma 4 is true. □

Additionally, we use Euclidean distance to approximate the road network distance for the ImprovePotential and QualityRatio computation to avoid the shortest path calculation. The ImprovePotential and QualityRatio computation formulas based on Euclidean distance are given in Eqn(5) and Eqn(6).

$$\begin{aligned} &ImprovePotential_E(a) \\ = &\frac{\sum_{e \in a.CAS} e.value - a.value}{MaxD_E(a.pre, a.CAS, a.post) - (a.pre \overset{sp}{\rightsquigarrow} a \overset{sp}{\rightsquigarrow} a.post).cost} \end{aligned} \tag{5}$$

$MaxD_E(a.pre, a.CAS, a.post) = \max_{e \in a.CAS} \big(d_E(a.pre, e) + e.cost + d_E(e, a.post)\big)$.

$$\begin{aligned} &QualityRatio_E(a_{start}, a_{end}, a_{qr}) \\ = &\frac{a_{qr}.value}{d_E(a_{start}, a_{qr}) + a_{qr}.cost + d_E(a_{qr}, a_{end})} \end{aligned} \tag{6}$$

To implement the ILS*(CE) algorithm with ellipse pruning, the only change is to replace the procedures compCAS and updateCAS with the following pseudocodes. Specifically, we add ellipse filtering during the CAS calucation (Line 2 in compCAS and Lines 7,9 in updateCAS), and change the QualityRatio calculation that is based on the Euclidean distance (Line 3 in compCAS and Line 10 in updateCAS). Note that similar to the "CAS inherit" property, the ellipses of arcs also satisfy the "inherit" property according to Lemma 4, and thus we can still use the "inherit" technique to reduce the search space when ellipse technique is applied.

Algorithm 4 Modified pseudocode for ILS*(CE)

1: **Replace compCAS(A, v_1, v_2, bgt) in Algorithm 3**
2: $CAS \leftarrow \{a \mid a \in A$ and $a.value > 0$ and $a \in Elip(v_1, v_2, bgt)$ and $(v_1 \overset{sp}{\rightsquigarrow} a \overset{sp}{\rightsquigarrow} v_2).cost \leq bgt\}$
3: $\forall a \in CAS$, $a.qr = QualityRatio_E(v_1, v_2, a)$
4: **Return** CAS

Replace updateCAS(a, v_1, v_2, $newBgt$) in Algorithm 3
5: let $oldBgt$ be the old remaining budget after removing a_c from $\mathcal{S}$
6: **if** $newBgt < oldBgt$ **then**
7: $a.CAS \leftarrow a.CAS \setminus \{a | a \notin Elip(v_1, v_2, newBgt)$ or $(v_1 \overset{sp}{\rightsquigarrow} a \overset{sp}{\rightsquigarrow} v_2).cost > newBgt\}$
8: **else if** $newBgt > oldBgt$ **then**
9: $a.CAS \leftarrow a.CAS \cup \{a | a \in Elip(v_1, v_2, newBgt)$ and $a.value > 0$ and $(v_1 \overset{sp}{\rightsquigarrow} a \overset{sp}{\rightsquigarrow} v_2).cost \leq newBgt\}$
10: $\forall a \in CAS$, $a.qr = QualityRatio_E(v_1, v_2, a)$
11: Return $a.CAS$

Using ellipse pruning technique, we can efficiently prune irrelevant arcs to speed up the feasibility checking procedure. However, the number of feasibility checking is still large ($\mathcal{O}(L|A|)$) since ILS*(CE) needs to check the arcs in the road network one by one to identify whether they are contained in the ellipse or not. To reduce the feasibility checking number, we propose an index-based search algorithm to further speed up each iteration.

5.3 ILS*(CEI): ILS*(C) algorithm with index

To compute the candidate arc sets efficiently, we use the grid [16] to index the road network. We can also use other spatial indexes such as R-trees [11] or Quadtrees [8]. In particular, each grid cell C is associated with a set of arcs, denoted by $C.olpArcs$, that overlap with C. For an arc a, we denote the grid cells that overlap with a as $a.olpCells$. As shown the example in Fig. 5(a), $C(1,7).olpArcs = \{a_1, a_2, a_3, a_4\}$, and $a_{10}.olpCells = \{C(5,5), C(5,6), C(5,7)\}$.

Given an arc a in the current solution $\mathcal{S}$, and let b be the remaining cost budget after removing a from $\mathcal{S}$, we denote $a.Elip$ as the ellipse $Ellipse(a.pre, a.post, b)$ whose foci are $(a.pre, a.post)$ and whose major axis is b. In the following, before presenting ILS*(CEI) algorithm, we first present two newly defined strategies: 1) pruning strategy and 2) hit strategy, to identify whether arcs in a grid cell can be included in the candidate arc set $a.CAS$ or not without performing the *feasibility checking*. Subsequently, we discuss how to apply the strategies to compute and update $a.CAS$.

DEFINITION 5. *Given a grid cell C and a ellipse $Elip$, we say C is **inside** of $Elip$ iff $\forall\, p \in C$, $p \in Elip$; C is **outside** of $Elip$ iff $\forall p \in C$, $p \notin Elip$.*

LEMMA 5. *Given an ellipse $Elip$ and a grid cell C that is outside of $Elip$, then $\forall a \in C.olpArcs$, $a \notin Elip$.*

PROOF. Since C is outside of $Elip$, all the points in C are outside of $Elip$. For any an arc $a \in C.olpArcs$, there exists a point $p \in a$, $p \in C$, and thus $a \notin Elip$. Hence Lemma 5 holds. □

Based on Lemmas 5 and 6, we can develop the pruning strategy to examine whether a grid cell can be pruned or not.

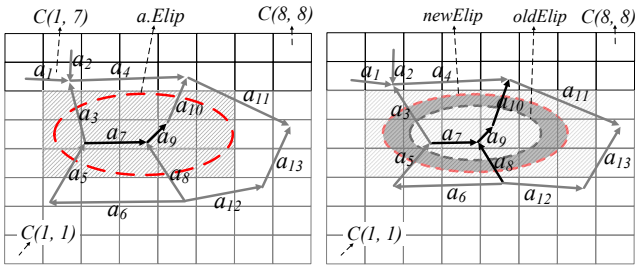
LEMMA 6 (INDEX PRUNING). *Given an arc a in the current solution $\mathcal{S}$, and given a grid cell C that is outside of the ellipse $a.Elip$, then $\forall e \in C.olpArcs$, $e \notin a.CAS$.*

PROOF. According to Lemma 5, we can get $e \notin a.Elip$. Further according to Lemma 3, we can infer that $e \notin a.CAS$, and thus Lemma 6 holds. □

Lemma 6 implies that if there exists a cell $C \in e.olpCells$ is outside of $a.Elip$, then the arc e will not be included in $a.CAS$. For the example in Fig. 5(a), arcs $\{a_1, a_2, ..., a_6, a_8, a_{10}, ..., a_{13}\}$ in gray color can be pruned since for each arc e of them, there exists a cell $C \in e.olpCells$ outside of $a.Elip$. For instance, a_{10} can be pruned since $C(5,7) \in a_{10}.olpCells$ is outside of $a.Elip$.

We are ready to discuss the computation for the candidate arc set $a.CAS$ of a. At the high level, we apply the index pruning first to obtain a set E of arcs, then apply the ellipse pruning for each arc in E individually, and finally process the feasibility checking for the rest of arcs in E for refinement. Specifically, as shown in Algorithm 5, we first compute the set $\mathcal{C}$ of grid cells that overlap with $a.Elip$. Let E be $\{C.olpArcs|\forall C \in \mathcal{C}\}$. We then apply the ellipse pruning strategy to filter out the arcs that are not inside of $a.Elip$ (according to Lemma 3). Finally, we do the feasibility checking through shortest path calculation to examine whether the remaining arcs in E can be included in $a.CAS$. For the example in Fig. 5(a), after the index pruning we obtain the arc set $E = \{a_3, a_5, a_7, a_8, a_9, a_{10}, a_{11}\}$ (other 6 arcs are pruned), of which, $\{a_3, a_5, a_8, a_{10}, a_{11}\}$ can be further filtered out with the ellipse pruning strategy. Finally, we just need to do the feasibility checking for two arcs: a_7 and a_9.

We next discuss the update for the candidate arc set $a.CAS$ of an arc a in the solution when the solution is changed (see Algorithm 5). Let $oldBgt$ (resp., $newBgt$) be the remaining cost budget after removing a from the old (resp., new) solution. We denote $oldElip$ (resp., $newElip$) as the ellipse $Elip(a.pre, a.post, oldBgt)$ (resp., $Elip(a.pre, a.post, newBgt)$) whose foci are ($a.pre$, $a.post$) and whose major axis is $oldBgt$ (resp., $newBgt$). To update $a.CAS$, we first compute grid cells $cells$ that overlap with $newElip$. Let E be the arcs associated with the cells in $cells$. If the budget increases, then we add $a.CAS$ with the arcs in E (Lemma 6) with positive attractiveness values, that are inside of $newElip$ (Lemma 3) and the shortest paths though a are within $newBgt$ (feasibility checking); otherwise, then we remove arcs from $a.CAS$ that are not inside of $newElip$ (Lemma 3) or are not feasible.



(a) CAS computation (b) CAS update

Figure 5: CAS computation and update with grid index

As shown in Fig. 5(b), the budget increases. Assume the old candidate arc set $a.CAS_{old} = \{a_9\}$. To update $a.CAS$, after index pruning, the arcs associated in the cells overlapping with $newElip$ are $\{a_3, a_5, a_7, a_8, a_{10}, a_{11}\}$ (the index pruning strategy filters out other 6 arcs). Among which, we only need to perform the feasibility checking for the arc a_7 to examine whether it can be inserted into $a.CAS$, while other arcs in E can be filtered out though ellipse pruning.

Lemma 6 suggests that we only need to check the arcs whose overlapping cells are not outside of the ellipse which can significantly reduce the number of feasibility checks.

Algorithm 5 Modified pseudocode for ILS*(CEI)

1: **Replace compCAS**(A, v_1, v_2, bgt, idx) **in Algorithm 3**
2: $cells \leftarrow$ overlapCells(idx, $Elip(v_1, v_2, bgt)$)
3: Return arcs a in $A \cap c.olpArcs, \forall c \in cells$, s.t., $a.value > 0$ && $a \in Elip(v_1, v_2, bgt)$ && $(v_1 \overset{sp}{\rightsquigarrow} a \overset{sp}{\rightsquigarrow} v_2).cost \leq bgt$.

Replace updateCAS(a, v_1, v_2, $newBgt$, idx) **in Algo. 3**
4: let $oldBgt$ be the old remaining budget after removing a from $\mathcal{S}$
5: let $oldElip$ (resp., $newElip$) be the ellipse $Elip(v_1, v_2, oldBgt)$ (resp., $Elip(v_1, v_2, newBgt)$)
6: $cells \leftarrow$ overlapCells(idx, $newElip$)
7: **if** $newBgt > oldBgt$ **then**
8: Insert $a.CAS$ with arcs a_c in $\{c.olpArcs \backslash a.CAS\}, \forall c \in cells$, s.t., $a_c.value > 0$, $a_c \in newElip$, and $(v_1 \overset{sp}{\rightsquigarrow} a_c \overset{sp}{\rightsquigarrow} v_2).cost \leq newBgt$.
9: **else if** $newBgt < oldBgt$ **then**
10: Remove arcs a_c from $a.CAS$, s.t., $a_c \notin newElip$ or $(v_1 \overset{sp}{\rightsquigarrow} a_c \overset{sp}{\rightsquigarrow} v_2).cost > newBgt$

6. EXPERIMENTAL STUDIES

We conducted several experiments to evaluate the efficiency and accuracy of our proposed algorithms.

6.1 Experimental methodology

We compared our proposed algorithms: ILS*(C), ILS*(CE) and ILS*(CEI) with the two baseline algorithms: GRASP [23] and ILS [27]. We used two real datasets: $LAFlickr$ and $FlandersCycle$. In $LAFlickr$, we use the Los Angeles road network, which contains 111,532 nodes and 183,945 arcs. Each node in $LAFlickr$ is associated with the corresponding geo-information (latitude and longitude). The costs associated with the arcs in $LAFlickr$ are the real network distance, and the values associated with the arcs are calculated based on the geo-tagged Flickr photos in Los Angeles with 217,391 photos [15]. For each arc a in $LAFlickr$, $a.value = |RNN|$, where $|RNN|$ is the number of photos pho that have a as their nearest neighbor (the closest arc) and the distance $Dist(a, pho)$ between a and pho is no larger than 1 kilometer, $Dist(a, pho) = min\{d_E(a.startvex, pho), d_E(a.endvex, pho)\}$. Since there is no benchmark instance based on $LAFlickr$ for AOP, we created the query instances as follows: we first randomly choosing 10 starting nodes from the road network. From each starting node s, among the nodes v in the graph G whose Euclidean distances from s $d_E(s, v)$ are no larger than the budget B and the costs of the shortest path from s to v are no less than B, i.e., $\{v \in G | d_E(s, v) \leq B, (s \overset{sp}{\rightsquigarrow} v).cost \geq B\}$, we randomly select one node as the corresponding destination node d. We set the budget B to be 10, 20, 30, 40 and 50 km, resulting in a total of 50 [2] instances. We also evaluate our algorithms on the benchmark instances (intance# = 50) used in the previous studies [23,27], which is based on a real-life cycle network of East-Flanders, named $FlandersCycle$, consisting of 989 vertices and 2,961 arcs. The cost of each arc in $FlandersCycle$ is the actual travel cost, and

[2]It will take too long to obtain the accurate results if the number of instances is too large.

the attractiveness value of each arc is obtained from users' personal preferences involving length, travel difficulty, sights, etc. Note that no geo-information is attached to the nodes in $FlandersCycle$.

As the metric for our evaluation, we report the average accuracy of 10 query instances over the running time. To evaluate the accuracy of all of these metaheuristic algorithms, following the studies [23, 27], we obtain the optimal solution using the commercial solver CPLEX 12.5 (64-bit), which is an open source mixed integer optimization solver, on the MS Azure server with an AMD Opteron(tm) Processor 4171 HE 2.1GHZ and 56GB of memory. Recall AOP is an NP-hard problem, hence it takes very long time to obtain the accurate results for a large-scale road network. To reduce the computation time, we perform the feasibility checking with our proposed ellipse pruning technique to filter out the irrelevant arcs to reduce the size of the graph before inputting the graph into the CPLEX solver. Nevertheless, since the computation time is still orders of magnitudes slower (over 2 hours) than the worst heuristics (less than 1 second), we do not include the running time of the optimal in the graphs. Note that we still do find the optimal path as a benchmark to compute the accuracy of the heuristics. To this end, we measure the performance of an algorithm $\mathcal{S}$ with the accuracy percentage, i.e., $accuracy = \frac{\mathcal{S}.value}{Opt.value} \times 100\%$.

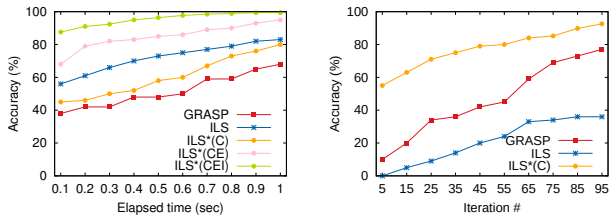
We implemented all the metaheuristic algorithms on a machine with Intel Core^{TM}2 Duo CPU E8500@3.16GHz, 4GB of RAM. By default, we set the budget B to be 30 km for all the algorithms. For ILS*(CEI), unless specified, the grid size is assumed to be 30×30 (i.e., 30 cells at X-axis and 30 cells at Y-axis). Besides, the grid index in ILS*(CEI) is resident in memory. Recall that an online shortest path computation is needed for the feasibility checking on a large-scale road network. For $LAFlickr$, we used the state-of-the-art shortest path algorithm CH [28]. For $FlandersCycle$ which is relative smaller, we pre-compute the all-pair shortest paths.

6.2 Experimental results

Results on LAFlickr We first present the experimental results on the $LAFlickr$ dataset.

Fig. 6(a) plots the accuracy when varying the elapsed time. Note that the elapsed time is the threshold of the running time (i.e., the parameter `TimeThreshold`) in the search algorithms (e.g., ILS Algorithm 1, ILS*(C) Algorithm 3). When the running time of a search algorithm reaches the elapsed time (e.g., at 0.1 second) we stop the algorithm and report the accuracy of the algorithm. Obviously, the longer the elapsed time, the higher the achieved accuracy. The results show two of our algorithms — ILS*(CE) and ILS*(CEI) — perform significantly better than the baseline algorithms (GRASP and ILS), while the other algorithm ILS*(C) performs worse than ILS but better than GRASP. The reasons are as follows: (1) For ILS*(CE) and ILS*(CEI), they use two criteria "ImprovePotential" and "QualityRatio" to effectively select the promising arcs to maximize the value of the solution during each iteration. Further, they apply spatial techniques, ellipse and grid index, to prune irrelevant arcs to speed up the iteration time without loss of value improvement. Note that our best algorithm, ILS*(CEI), can achieve over 95% accuracy within 300 milliseconds on the $LAFlickr$ dataset. (2) The reason why ILS*(C) performs worse than ILS is due to its slow iteration to maintain the candidate arc sets (CAS). However, ILS*(C) is approaching to ILS as the elapsed time increases because of the *CAS Inherit* technique, i.e., we do not need to compute the candidate arc sets from scratch and the CAS size is becoming smaller and smaller. (3) ILS*(C) performs better than GRASP is because it uses the two proposed criteria to decide which arcs should be visited first to update the solution, considering both costs and values of the arcs; however, GRASP considers cost only for candidate solution generation.

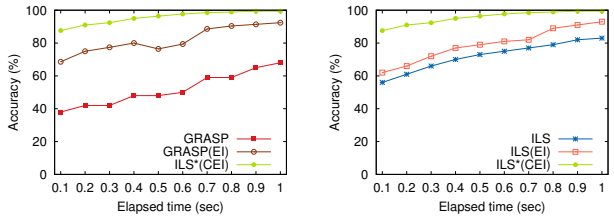
To further illustrate the reason why our algorithms outperform the baselines, we plot Fig. 6(b) to show the accuracy when varying the number of iterations. Our spatial pruning techniques can only be used to speed up the iteration but cannot be utilized to improve the accuracy at each iterations. Consequently, all of our proposed algorithms have the same performance, and we plot ILS*(C) only in this experiment. It clearly shows that for the same the number of iterations, ILS*(C) has much higher accuracy than ILS and GRASP. That is mainly because our algorithms use value and cost comparisons at the end of each iteration to select the most promising arcs, however, ILS just performs DFS which may lead to a large number of *idle* iterations with no value improvement.



(a) Elapsed time (b) The number of iterations

Figure 6: Evaluation on various algorithms on LAFlickr

Since our proposed new spatial techniques can also improve the baseline algorithms, we next evaluate the performances of GRASP and ILS with our spatial techniques. To this end, we apply both of our spatial techniques, ellipse pruning and grid index, to the feasibility checking of arcs in GRASP and ILS, respectively, and denote the corresponding two algorithms as GRASP(EI) and ILS(EI). As shown in Fig. 7(a), we can see that the performance of GRASP can be improved significantly with our spatial techniques and can be further improved marginally with the new criteria. This demonstrates that GRASP's poor performance is mainly due to the slow iteration. For ILS, as shown in Fig. 7(b), it can improve slightly with the spatial techniques but can improve with the new criteria since ILS performs DFS with poor value improvement at each iteration.



(a) GRASP (b) ILS

Figure 7: Evaluation on various algorithms on LAFlickr

We also evaluate the effect of the budget in AOP. In Fig. 8, we vary B from 10 km to 50 km by fixing the elapsed time to be 300 milliseconds. We can see that the baselines ILS and GRASP are very sensitive to B whereas our algorithm can nicely adapt to the increase in the budget B. This is because that the search spaces of both ILS and GRASP increase dramatically as the budget B increases. ILS is even worse than GRASP when B is larger (>40 km) since it just performs DFS and thus it may select arc with poor "quality" (large cost and low value) at the first iterations. However, in our algorithms, we apply the spatial techniques as well as the "inherit" techniques to reduce the search space significantly.

Finally, we evaluate the effect of the grid size on ILS*(CEI) in Fig. 9. The elapsed time is also set to be 300 milliseconds in this experiment. As shown in Fig. 9, the accuracy of ILS*(CEI) is not

very sensitive to the grid size, and it achieves the best performance when the grid size is 30×30. When the grid size decreases, the pruning power of the index in ILS*(CEI) diminishes. When the grid size becomes to 1×1, ILS*(CEI) actually becomes ILS*(EI). On the contrary, if the grid size is too large, i.e., the cell size is too small, then ILS*(CEI) needs to check a large number of grid cells to examine whether they overlap with the ellipse and thus the performance degrades.

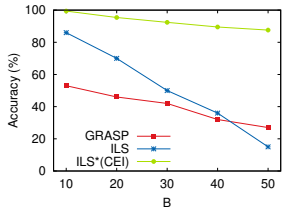


Figure 8: Varying the budget B on LAFlickr

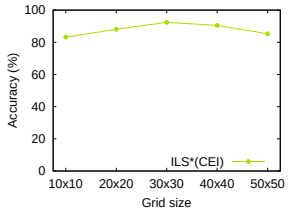


Figure 9: Varying grid size for ILS*(EI) on LAFlickr

Results on FlandersCycle We omit the results of ILS*(CE) and ILS*(CEI) because there are no longitude and latitude information on this dataset, i.e., this is a graph and not a road network. In this experiment, all the algorithms can achieve the optimal solution since the $FlandersCycle$ dataset is rather small and all the shortest-path between the pairs of vertices in the graph are precomputed. Still, Fig. 10 clearly confirms the superiority of ILS*(C) over the baseline solutions because of the new criteria and the "inherit" technique used in ILS*(C).

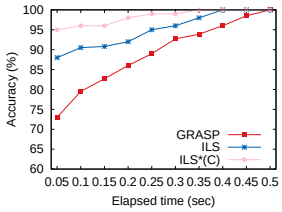


Figure 10: Evaluation on the $FlandersCycle$ dataset

7. CONCLUSION AND FUTURE WORK

This paper studied the problem of finding the most scenic path while keeping the total travel cost within a budget. The problem can be formulated as a variant of AOP which is an NP-hard combinatorial optimization problem. We proposed a series of metaheuristic algorithms to solve the problem to support fast response time (milliseconds) on large-scale road networks, by utilizing the techniques from the field of spatial databases: ellipse pruning and spatial indexing. While indexing the road network we only considered the costs of arcs. However, incorporating the attractiveness values of the network arcs into the index would further improve the performance. To this end, in the future, we plan to study the road network hybrid indexing by combining both the travel costs and the attractiveness values of arcs to solve the AOP problem.

8. ACKNOWLEDGEMENTS

This research has been funded in part by NSF grants IIS-1115153, IIS-1320149, and CNS-1461963, the USC Integrated Media Systems Center (IMSC), and unrestricted cash gifts from Google, Northrop Grumman, Microsoft, and Oracle. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any of the sponsors such as the National Science Foundation.

9. REFERENCES

[1] S. Aljubayrin, J. Qi, C. Jensen, R. Zhang, Z. He, and Z. Wen. The safest path via safe zones. In *Proc. of ICDE*, pages 531 – 542.

[2] J. Arĺćoz, E. Fernĺćndez, and O. Meza. Solving the prize-collecting rural postman problem. *Eur. J. Oper. Res.*, 196(3):886 – 896, 2009.

[3] A. H. C. Archetti, D. Feillet and M. G. Speranza. The undirected capacitated arc routing problem with profits. *Comput. Oper. Res.*, 37(11), 2010.

[4] K. N. P. M. Chekuri, C. Improved algorithms for orienteering and related problems. In *symposium on Discrete Algorithms, SODA*, pages 661–670, 2008.

[5] K. M. G. P. Damianos Gavalas, Charalampos Konstantopoulos and N. Vathis. Approximation algorithms for arc orienteering problems. *Ecompass*, 2014.

[6] U. Demiryurek, F. B. Kashani, and C. Shahabi. Online computation of fastest path in time-dependent spatial networks. In *12th International Symposium Spatial and Temporal Databases, SSTD*, pages 92–111, 2011.

[7] D. Feillet, P. Dejax, and M. Gendreau. The profitable arc tour problem: Solution with a branch-and-price algorithm. *Transportation Science*, 39(4):539–552, 2005.

[8] R. Finkel and J. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.

[9] D. Gavalas, C. Konstantopoulos, K. Mastakas, G. Pantziou, and N. Vathis. Approximation algorithms for the arc orienteering problem. *Information Processing Letters*, 115(2):313–315, 2015.

[10] P. Greistorfer. A tabu scatter search metaheuristic for the arc routing problem. *Computers & Industrial Engineering*, 44(2):249–266, 2003.

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

[12] E. A. W. I-Ming Chao, Bruce L. Golden. A fast and effective heuristic for the orienteering problem. *Eur. J. Oper. Res.*, 88(3):475–489, 1996.

[13] M. R. Kolahdouzan and C. Shahabi. Voronoi-based K nearest neighbor search for spatial network databases. In *Proc. of the 30th International Conference on VLDB*, pages 840–851, 2004.

[14] J. Maervoet, P. Brackman, K. Verbeeck, P. D. Causmaecker, and G. V. Berghe. Tour suggestion for outdoor activities. *W2GIS, Lecture Notes in Computer Science*, 7820:54–63, 2013.

[15] H. Mousselly-Sergieh, D. Watzinger, B. Huber, M. Döller, E. Egyed-Zsigmond, and H. Kosch. World-wide scale geotagged image dataset for automatic image annotation and reverse geotagging. In *ACM Multimedia Systems*, pages 47–52, 2014.

[16] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Trans. Database Syst.*, 9(1):38–71, Mar. 1984.

[17] G. Palma. A tabu search heuristic for the prize-collecting rural postman problem. *ENTCS*, 281:85–100, 2011.

[18] D. Quercia, R. Schifanella, and L. M. Aiello. The shortest path to happiness: Recommending beautiful, quiet, and happy routes in the city. In *ACM Hypertext and Social Media*, pages 116–125, 2014.

[19] S. P. L. Ray Deitch. The one-period bus touring problem: Solved by an effective heuristic for the orienteering tour problem and improvement algorithm. *Eur. J. Oper. Res.*, 127:69–77, 2000.

[20] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *SIGMOD*, pages 43–54, 2008.

[21] G. Skoumas, K. A. Schmid, A. Z. Gregor Jossé, M. A. Nascimento, M. Renz, and D. Pfoser. Towards knowledge-enriched path computation. In *ACM SIGSPATIAL GIS*, pages 661–670, 2014.

[22] G. Skoumas, K. A. Schmid, G. Jossé, M. Schubert, M. A. Nascimento, A. Züfle, M. Renz, and D. Pfoser. Knowledge-enriched route computation. In *14th International Symposium Spatial and Temporal Databases, SSTD*, pages 157–176, 2015.

[23] W. Souffriau, P. Vansteenwegen, G. V. Berghe, and D. V. Oudheusden. The planning of cycle trips in the province of east flanders. *Omega*, 39(2):209–213, 2011.

[24] J. G. Su, M. Winters, M. Nunes, and M. Brauer. Designing a route planner to facilitate and promote cycling in metro vancouver, canada. *Trans. Res. Part A: Policy and Practice*, 44(7):495 – 505, 2010.

[25] T. Tsiligirides. Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, 35:797–809, 1984.

[26] P. Vansteenwegen, W. Souffriau, and D. Oudheusden. The orienteering problem: a survey. *Eur. J. Oper. Res.*, 209(1):1–10, 2011.

[27] C. Verbeeck, P. Vansteenwegen, and E.-H. Aghezzaf. An extension of the arc orienteering problem and its application to cycle trip planning. *Transportation research*, 68:64–78, 2014.

[28] L. Wu, X. Xiao, D. Deng, G. Cong, A. D. Zhu, and S. Zhou. Shortest path and distance queries on road networks: An experimental evaluation. *Proc. VLDB Endow.*, 5(5):406–417, Jan. 2012.

[29] Y.-T. Zheng, S. Yan, Z.-J. Zha, Y. Li, X. Zhou, T.-S. Chua, and R. Jain. Gpsview: A scenic driving route planner. *ACM TOMCCAP*, 9(3), 2013.

[30] A. D. Zhu, H. Ma, X. Xiao, S. Luo, Y. Tang, and S. Zhou. Shortest path and distance queries on road networks: Towards bridging theory and practice. In *Proc. of ACM SIGMOD*, pages 857–868, 2013.