One-Way Search Algorithm for Route Planning With Multiple Requests

Eric Hsueh-Chan Lu, *Member, IEEE*, and Sin-Sian Syu

***Abstract*— Multirequest route planning (MRRP) based on user demand is an active location-based service in point-of-interest (POI) networks. The goal of MRRP is to plan the shortest route that meets the requests specified by the user. However, because MRRP is an NP-hard problem, most existing methods involve the use of greedy search in the planning of approximate routes, leaving much room for improvement in route quality. Therefore, in this study, an anytime algorithm called one-way search (OWS) was proposed to efficiently solve MRRP queries. OWS integrates branch-and-bound and greedy search. It not only helped rapidly detect an accepted route but also allowed for more optimal routes to be iteratively generated until the most optimal route was identified. OWS has three pruning mechanisms named Filter, Potential, and Petrifaction and three operations named Wilting, Selection, and Reverse Update to avoid unnecessary searches and improve search efficiency. To the best of the authors' knowledge, this is the first study of MRRP queries that investigates both optimal route planning and searching efficiency. The experimental results obtained on real-world POI datasets indicated that OWS outperformed state-of-the-art anytime algorithms in terms of quality and efficiency.**

***Index Terms*— Multirequest route planning, one-way search, anytime algorithm, location-based service.**

I. Introduction

With the increasing popularity of mobile devices and the development of wireless communication technologies and global navigation satellite systems, the application of location-based services (LBSs) has gradually increased. In LBSs, a point-of-interest (POI) is a basic element that marks a map with points that people may be interested in, such as restaurants, gas stations, and hospitals. Each POI contains both latitude and longitude information and is usually associated with a name, description, and category, among other attributes. LBSs are commonly applied in electronic maps. For example, Google Maps, Bing Maps, and OpenStreetMap, provide POI query and route planning services. Given a starting point, destination, and several points *en route* to the destination, the service returns the shortest or fastest route(s). However, new queries for route planning, such as trip planning query (TPQ) [9] and optimal sequenced route (OSR) query [19], have recently been proposed; in these queries, POIs fall into one of many categories. For such queries, the user requests the type of POI, such as a restaurant or a hospital, and the route planning service selects an instance of the POI for each type and creates a route and returns it to the user. In real-world scenarios, a POI may have a composite function to address multiple user requests, and queries with this feature have been considered, named multirequest route planning (MRRP) [13]. However, although several variations of route planning exist, the user usually cares about the route quality and response efficiency of MRRP queries, i.e., the shorter the route length is and the faster the response time is, the more favorable the MRRP result is. MRRP is challenging for service providers because rich POI data and complete road network information are required and because several queries must be simultaneously addressed but computational costs cannot be excessively high.

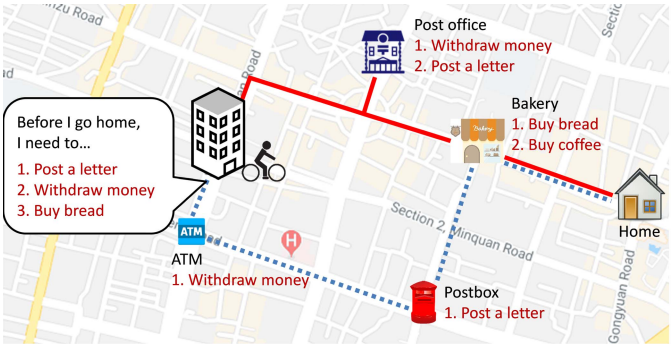

Fig. 1. Real-life example of an MRRP scenario.

Fig. 1 illustrates a real-life example of an MRRP scenario. In this example, a young man who has been working hard all day is ready to leave work and head home. However, before heading home, he needs to send a letter, withdraw some money, and buy some bread. This means that, in terms of POIs, the planning results may include an ATM, a postbox, and a bakery, as indicated by the blue dotted line in Fig. 1. However, given the actual needs of the young man, it is better to visit a post office that provides both letter-sending and money-withdrawal services (red line in Fig. 1) rather than a bank and postbox separately. The goal of MRRP is to plan a route that can address all the requests of the young man on his way home while minimizing the route length. Although the original MRRP problem involves short-distance route planning for urban areas, it may involve travel planning if it is a long-distance query. For example, a group of people wish to arrange a countrywide trip. They depart from the train station and rent a car nearby to go to their destination, which is 15 km away. On their way, they can and must do various things, such

Manuscript received 7 December 2021; revised 12 June 2022 and 5 October 2022; accepted 1 November 2022. Date of publication 11 November 2022; date of current version 8 February 2023. This research was supported by Ministry of Science and Technology, Taiwan, under grant no. MOST 107-2119-M-006-028 and MOST 109-2121-M-006-013-MY2. The Associate Editor for this article was M. Mesbah. *(Corresponding author: Eric Hsueh-Chan Lu.)*

The authors are with the Department of Geomatics, National Cheng Kung University, Tainan 701, Taiwan (e-mail: luhc@mail.ncku.edu.tw; syupi.ken@gmail.com).

Digital Object Identifier 10.1109/TITS.2022.3220429

as stopping to refuel, visiting a tourist attraction, or buying ingredients for dinner. Thus, they would benefit from a service that can help them make the most out of this trip.

This study proposed a method for handling MRRP queries. MRRP is a generalization of the traveling salesman problem (TSP) [14], in which all POIs provide only one service and each POI provides different services. Because TSP is an NP-hard problem, MRRP is also an NP-hard problem. This means that MRRP may be unable to determine the most optimal solution in polynomial time, namely, the shortest route. However, because it is expected by users, route planning methods must provide accurate solutions rapidly. However, current MRRP algorithms can provide only approximate solutions. In this study, an efficient algorithm called one-way search (OWS) was proposed to address MRRP queries. OWS integrates branch-and-bound and greedy mechanisms to rapidly return an accepted solution at any time and determine the most optimal solution. It involves three pruning mechanisms named Filter, Potential, and Petrifaction mechanisms and three operations named Wilting, Selection, and Reverse Update to avoid unnecessary searches and thus improve search efficiency. OWS was also applied to MRRP with request order constraints by adding the path-expanded constraint. The experimental results obtained on two real-world datasets indicated that OWS identified a near-optimal route in 1 s. This means that the most optimal route can be determined if OWS is left to perform further computations.

The following are the contributions of this study:

1) An algorithm called OWS was proposed to address MRRP queries. In experiments, OWS identified the most precise relative to its counterparts, which only offered approximate solutions. This is the main contribution of the OWS.
2) Because OWS integrates both greedy and branch-and-bound concepts, it is an anytime algorithm that can iteratively return suboptimal solutions before identifying the most optimal solution.
3) The experimental results obtained on two real-world datasets indicated the superiority of OWS to state-of-the-art anytime algorithms, such as anytime potential search/anytime nonparametric A* (APTS/ANA*) and iterative memory bounded A* (IMBA*), in terms of route quality and computational cost.
4) Because OWS successfully identified the near-optimal route in 1 s, it can be used to balance the accuracy and immediacy of MRRP queries.

The rest of this paper is organized as follows. Section 2 reviews existing work on route planning. Section 3 formally defines the study problem. Section 4 introduces the OWS algorithm. Section 5 discusses the experiments and evaluations performed, including the testing and tuning of OWS mechanisms, and a comparison with competitive algorithms. Finally, Section 6 concludes the paper and presents suggestions for future research.

II. Related Work

This section presents a review of studies on single-destination route planning and user-demand-oriented route planning.

A. Single-Destination Route Planning

Single-destination route planning is a classical problem in graph theory, where the goal is to determine the shortest route between two nodes. Dijkstra [2] and A* [5] are well-known algorithms for route planning. A* is a best-first-search approach in which a priority queue is used to expand nodes with the smallest cost on the basis of $f(n) = g(n) + h(n)$, where $g(n)$ is the actual distance from the starting point to node n and the heuristic function $h(n)$ is the estimated distance from n to the destination. To determine the shortest route, $h(n)$ should be admissible [6], meaning that the estimated distance cannot be larger than the actual distance. The closer $h(n)$ is to the actual distance, the more efficient A* is. This technique has undergone years of evolution. Iterative deepening A* [8] is an iterative form of A* where the limit is gradually relaxed to improve the search space. Simplified memory bounded A* [18] is a memory-restricted form of A*, in which the least optimal node is forgotten when the memory limit is exceeded. Anytime repairing A* (ARA*) [12] speeds up the search process by multiplying the heuristic function by a weight. It then gradually reduces the weight for each search and finally identifies the shortest path.

In contrast to standard A*, ARA* can continuously return approximate solutions before determining the optimal solution. However, the relevant parameters of the weight should be manually set. For APTS/ANA* [21], a new heuristic function was proposed by Stern et al. to solve this problem. They addressed the bounded-cost search problem [22] and proposed the potential search (PTS) method to solve the problem. Their goal was to identify a solution whose cost is not larger than the given budget by repeatedly using this method and reducing the cost until the last solution is identified. PTS can be modified to anytime PTS (APTS), which can help determine the optimal solution. Several studies have investigated the need for manually setting the parameters of ARA*. For example, van Den Berg et al. [23] proposed ANA* to derive an adaptive calculation function for weight values. Libralesso and Fontan [11] indicated that A* required the storage of a large number of nodes. Therefore, to reduce the memory requirements, they proposed an anytime algorithm called IMBA* that uses a heuristic variant of A* to cut low promising nodes. However, in general, compared with destination-oriented route planning, route planning based on user demand is considered to be more practical in real-life scenarios.

B. User-Demand-Oriented Route Planning

TSP [14] is a well-known problem where several POIs are selected depending on the user demand and arranges the visiting order to plan a low-cost route on a symmetric graph. Rodríguez and Ruiz [16] indicated that actual road networks should be asymmetric. They analyzed the effect of solving the TSP on an asymmetric road network and reported that applying symmetric TSP algorithms to asymmetric road networks may decrease the quality and efficiency of route planning. Therefore, to solve this problem, Roth [17] converted the TSP into a many-to-many path planning problem and proposed an algorithm called efficient many-to-many A* to

improve the planning efficiency by sharing the planning results between iterations. In TPQ, the POI type is generally considered to ensure that the shortest route is one that begins at the starting point, passes through at least one POI for each user-specified type, and ends at the destination [9]. Using a nearest neighbor search, Sharifzadeh et al. [19] formulated OSR query, which can identify the shortest route starting from a given source POI and passing through a specific order of POIs. In another study, Sharifzadeh and Shahabi [20] used the geometric properties of the solution space to prebuild a Voronoi diagram index, which helped improve the speed of OSR queries. In several real-world applications, route planning queries are subject to multiple constraints, such as the visiting order for partial categories. Chen et al. [1] proposed a multirule partial sequenced route query, which allows only a subset of the POI categories to be visited in order during a trip.

Given the aforementioned limitations, Li et al. [10] indicated that users may not have enough time to visit points in all categories. Therefore, a reasonable compromise is to identify a route that covers l categories from order constraints. Kanza et al. [7] defined the concept of interactive route search, which is computed as follows. After the user visits a given entity, they provide feedback specifying whether that entity was satisfactory for them, and the next entity on the route is then provided. Eisner et al. [3] assumed that common queries are local and proposed an iterative doubling method to limit the distance so as to rapidly obtain the results of route planning. Using an incremental Euclidean restriction framework, Ohsawa et al. [15] proposed Euclidean OSR to incrementally generate candidates and obtain the top k shortest routes. Lu et al. [13] indicated that each POI can address multiple user needs. They addressed the MRRP and proposed a planning module to plan a preliminary MRRP route based on pruning and caching strategies. They also used a refinement module to further improve the route quality by rearranging and replacing the route. However, these algorithms can provide only approximate solutions.

III. Problem Statement

In this section, we introduce the notation and terminology used throughout the paper and formally define the MRRP.

Definition 1 (Metric Space): Let $P = \{p_1, p_2, \ldots, p_{N_P}\}$ be a set of POIs. MRRP is considered in a metric space that is defined as the following conditions, where$d(p_i, p_j)$ denotes the travel distance between p_i and p_j, and $p_i, p_j, p_k \in P$.

1) $d(p_i, p_j) \geq 0$ (non-negativity)
2) $d(p_i, p_j) = 0 \Leftrightarrow p_i = p_j$ (identity of indiscernible)
3) $d(p_i, p_j) = d(p_j, p_i)$ (symmetry)
4) $d(p_i, p_j) \leq d(p_i, p_k) + d(p_k, p_j)$ (triangle inequality)

Definition 2 (Route): Route $T = \langle p_1, p_2, \ldots, p_m \rangle$ is an ordered sequence of one or several POIs, where $p_i \in P$ and $m = |T|$ is the number of POIs on route T.

Definition 3 (Route Length): Given a starting point s, a destination e, and a route T, $L(s, e, T)$ is defined as the length of the route from s through T and finally to e.

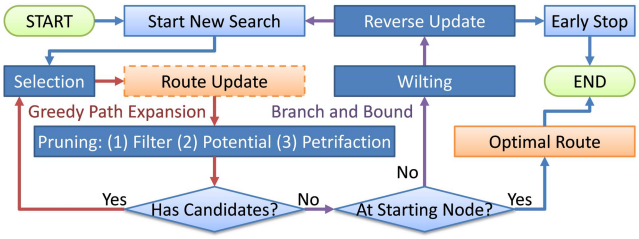

Fig. 2. Flowchart of the OWS algorithm.

The following is the formula for $L(s, e, T)$:

$$L(s, e, T) = d(s, p_1) + \sum_{i=1}^{m-1} d(p_i, p_{i+1}) + d(p_m, e) \tag{1}$$

Definition 4 (Request and Service): $R = \{r_1, r_2, \ldots, r_{N_R}\}$ is a set of requests (if $i \neq j$, then $r_i \neq r_j$) and $S = \{s_1, s_2, \ldots, s_{N_S}\}$ is the set of all services in the data set (if $i \neq j$, then $s_i \neq s_j$). Here, $R \subseteq S$ and $N_R \leq N_S$.

Definition 5 (PSR and RSP): Given a request $r \in R$ and a POI $p \in P$, $PSR(r)$ denotes the set of all POIs that provide service r, where $PSR(r) \subseteq P$. In addition, $RSP(p)$ denotes the set of all services provided by p, where $RSP(p) \subseteq S$.

Definition 6 (Valid Route): Given a set of user-specified requests R and a route T, T is called a valid route if and only if $R \subseteq \bigcup_{i=1}^{m} RSP(p_i)$, where $p_i \in T$. In other words, a route is valid if it solves all the requests specified by the user.

Definition 7 (MRRP): Given a set of POIs P, a starting point s, a destination e, and a set of user-specified requests R, the goal of MRRP is to identify a valid route T from s to e and minimize the route length $L(s, e, T)$.

Research Objective: According to the original definition of MRRP in Definition 7, the research objective of this study is to iteratively determine the most optimal route and reduce the search time as much as possible.

IV. Methodology

This section presents OWS for MRRP route search. The flowchart of OWS is presented in Fig. 2. OWS first searches a route on the basis of the POIs that provide at least one unsolved user request and pass three pruning mechanisms. One of them is greedily selected for path expansion and route update to rapidly identify a more favorable route. OWS then tightens the bounds along the branch nodes by wilting and reverse update to reduce the search space for new searches. OWS may also stop early on the basis of any consideration and return the last determined route. Finally, the optimal route is identified if no candidate exists at the starting node.

A. Fundamental

This section describes the graph structure used in OWS and the bound attributes of nodes in the graph. Table I summarizes the notations used in OWS.

1) Node Equivalence: This section introduces the graph structure G and the property of node equivalence. The graph structure G is described in Fig. 3, where node v represents one of the POIs and the symbol $\varphi_i|R$ in v represents

TABLE I
NOTATION USED IN THE OWS ALGORITHM

Notation	Meaning
G, v, s, e	Graph structure, node, start, destination
v_p, v_c, v_n	In-node, current node, out-node
$v.fw$	Minimal path length from s to v in G
$v.bw$	Minimal path length from v to e in G
$v.est$	Estimated length from v to e in G
V_{Petr}, V_{wilt}	Set of all petrifaction and wilting nodes in G
V_{cand}	Set of candidate nodes
$L_{OPT}(v_i, v_j)$	Shortest path length from v_i to v_j in G
$L_{PT}(v)$	Potential length from s through v to e in G
$prune_{FT}(v_c, v_n)$	Whether to exclude v_n as a candidate of v_c by the filter pruning mechanism
$prune_{PT}(v_c, v_n)$	Whether to exclude v_n as a candidate of v_c by the potential pruning mechanism
$prune_{PF}(v_n)$	Whether to exclude v_n as a candidate by the petrifaction pruning mechanism
$wilt(v_c)$	Wilting value of v_c

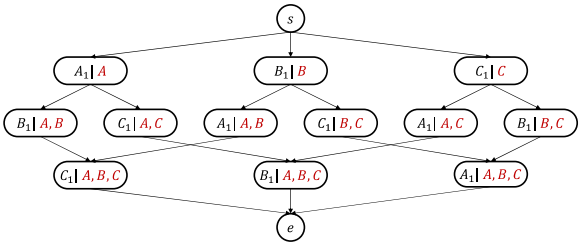

Fig. 3. Node equivalence.

the i-th POI that provides service φ and the set of requests R that have been solved so far, respectively. For example, A_1 is the first POI that provides service A. The search process starts from s to solve three requests A, B, and C. Because no order constraint exists for solving requests in MRRP, all POIs that provide unsolved services are placed on the first layer. The following layers are then explored per the following process. Depending on what the current node has not solved, all POIs that can provide services for that unsolved request are connected. Finally, the nodes that solve all requests are directly connected to e. As shown in Fig. 3, a POI may exist in several nodes on the graph; however, the requests that they solve are different. Thus, the following property for nodes is defined: *two nodes are equivalent if they represent the same POI and solve the same requests.* OWS guarantees that the nodes are not equivalent because the information of any two equivalent nodes is redundant and results in memory waste on the graph.

Although no order constraint of requests exists for MRRP, OWS can support order constraints by adding the path-expanded constraint for graph nodes. For example, if a user wishes to withdraw money before buying some bread, OWS can restrict the expansion of nodes providing bakery services until the user withdraws money. In addition, although MRRP is formulated on the travel distance, OWS can solve

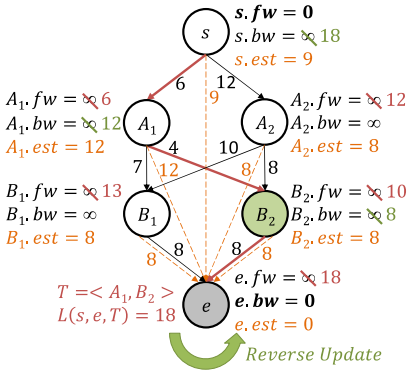

Fig. 4. Example of forward, backward, and estimate.

MRRP on the estimated traveling time, which is obtained by known traffic data if the four conditions in Definition 1 are satisfied. The first (non-negativity) and second (identity of indiscernible) conditions are intuitively held. For the third (symmetry) and fourth (triangle inequality) conditions, the traveling time is usually asymmetric and sometimes violates the triangle inequality. However, OWS can perform offline calculations and store all the asymmetric time costs of the shortest path between two POIs to solve these two problems.

2) Forward, Backward, and Estimate: Node v contains three bounds in OWS. The first two bounds, called *forward* ($v.fw$) and *backward* ($v.bw$), defined in (2) and (3), are the minimal path length determined thus far from s to v and from v to e, respectively. In (2), for each in-node v' of v, $v.fw$ is the minimal value of the sum of $v'.fw$ and the distance of v' and v, where $N^-(v)$ indicates the set of in-nodes of v. Here, $v.fw$ is greater than or equal to the shortest path length from s to v. In (3), $v.bw$ is similar to $v.fw$ with the exception being that v' is replaced by the out-node of v, where $N^+(v)$ indicates the set of out-nodes of v. Here, $v.bw$ is also greater than or equal to the shortest path length from v to e. The third bound is *estimate* ($v.est$), which is the estimated length from v to e. Here, $v.est$ is no longer than the shortest path length from v to e ($v.est$ is admissible). In (4), the Euclidean distance is one choice for measuring $v.est$, where $d_{Euc}(v, e)$ indicates the straight-line distance from v to e. Suppose that the cost between POIs is the traveling time, the definitions of $v.fw$ and $v.bw$ are the same, but $v.est$ can be measured from the straight-line distance divided by the maximum speed limitation in a city. The obtained $v.est$ can be guaranteed to be the minimal traveling time from v to e that is typically admissible if users obey the road speed limit.

$$v.fw = \min_{v' \in N^-(v)} \left(v'.fw + d\left(v', v\right)\right) \geq L_{OPT}(s, v) \quad (2)$$

$$v.bw = \min_{v' \in N^+(v)} \left(v'.bw + d\left(v', v\right)\right) \geq L_{OPT}(v, e) \quad (3)$$

$$v.est = d_{Euc}(v, e) \leq L_{OPT}(v, e) \quad (4)$$

Fig. 4 illustrates an example of the three bounds and the proposed mechanisms, in which the solid and dotted lines represent the actual and estimated distances, respectively. Initially, $s.fw$ and $e.bw$ are 0, and the rest bounds are unknown. OWS starts from s and updates *forward* in a top-down fashion. $A_1.fw$ and $A_2.fw$ become 6 and 12, respectively, after the

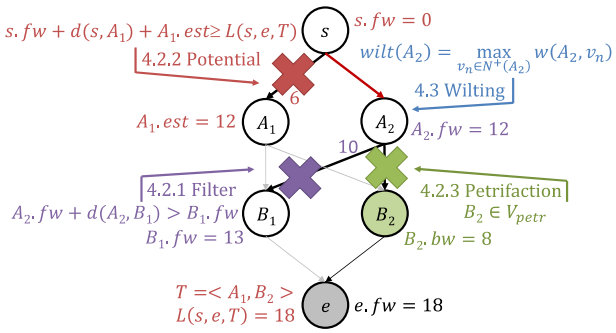

Fig. 5. Filter, potential, and petrifaction pruning.

first update. Next, $B_1.fw$ and $B_2.fw$ become 13 and 10, respectively, which are obtained from A_1, because only the minimal *forward* of all possible paths is recorded on each node. Finally, $e.fw$ becomes 18. OWS also reverse-updates *backward*, as it did to the *forward* update. Here, $B_2.bw$, $A_1.bw$, and $s.bw$ are 8, 12, and 18, respectively. For each node, *estimate* is not larger than *backward*.

B. Pruning Mechanism

The purpose of the pruning mechanism is to eliminate unreasonable candidates, which cannot be the optimal MRRP route solution, thereby reducing the search space. Fig. 5 presents all the mechanisms highlighted in Fig. 4.

1) Filter: The first mechanism is Filter pruning, $prune_{FT}(v_c, v_n)$, defined in (5), where v_n indicates one of the out-nodes of the current node v_c. Here, v_n is not considered a candidate node of v_c if the sum of $v_c.fw$ and $d(v_c, v_n)$ are larger than $v_n.fw$, because $v.fw$ represents the minimal path length determined thus far from s to v. This means that OWS searches for a shorter path from s to v_n. A longer partial path from s to v_n does not become a shorter MRRP route. As shown in Fig. 5, B_1 is not a candidate of A_2 because the sum of $A_2.fw$ and $d(A_2, B_1)$ is larger than $B_1.fw$ (i.e., $12 + 10 > 13$). However, when v_n is withered, which is described later in Section 4.3, to avoid repeated considerations of v_n as a candidate in the next search, V_{wilt} is used to store all the wilting nodes and control whether v_n with the same *forward* can be a candidate, where V_{wilt} is initially empty.

$$prune_{FT}(v_c, v_n) = \begin{cases} \text{TRUE}, & v_c.fw + d(v_c, v_n) > v_n.fw \\ \text{TRUE}, & v_c.fw + d(v_c, v_n) \geq v_n.fw \,\&\, v_n \in V_{wilt} \\ \text{FALSE}, & \text{otherwise} \end{cases} \tag{5}$$

2) Potential: Even if the out-node v_n of the current node v_c is not eliminated by Filter pruning, it does not necessarily have the potential to be the shortest path, because the distance between v_n and e is not yet considered. As indicated in Section 4.1.2, $v_n.est$ is an estimated length between v_n and e that is not greater than the shortest length from v_n to e. Here, v_n is not considered a candidate if the length from s through v_c to v_n exceeds $L(s, e, T)$ minus $v_n.est$, where $L(s, e, T)$ indicates the length of the shortest route T determined thus far. This means that the length from s through v_c and v_n to e should be longer than $L(s, e, T)$ because $v_n.est$ is a lower-bound length from v_n to e. Hence, Potential pruning $prune_{PT}(v_c, v_n)$ is defined as in (6), where v_n indicates one of the out-nodes of v_c. If the sum of the length from s to v_n through v_c and $v_n.est \geq L(s, e, T)$, then v_n cannot be a shorter path. Because a route T from s to e through A_1 and B_2 is found and $L(s, e, T)$ is 18 (Fig. 4), A_1 is not a candidate of s in the next search (Fig. 5), because the sum of $s.fw$, $d(s, A_1)$, and $A_1.est$ is larger than $L(s, e, T)$ (i.e., $0 + 6 + 12. \geq 18$).

$$prune_{PT}(v_c, v_n) = \begin{cases} \text{TRUE}, & v_c.fw + d(v_c, v_n) + v_n.est \geq L(s, e, T) \\ \text{FALSE}, & \text{otherwise} \end{cases} \tag{6}$$

3) Petrifaction: Petrifaction represents a node whose *backward* no longer changes. V_{petr} is used to store all petrifaction nodes, and node v is added to V_{petr} if and only if all out-nodes of v belong to V_{petr}. Fig. 4 shows the petrifaction process during a reverse update. Initially, only e is added to V_{petr}, and $e.bw$ is 0. After the path through nodes A_1 and B_2 arrives at e, OWS reverse-updates *backward* from e to s. Subsequently, $B_2.bw$ is updated to 8 according to (3), and B_2 is added to V_{petr} because B_2 has only a petrifaction out-node e. Similarly, $A_1.bw$ is updated to 12, but A_1 cannot be added to V_{petr} because A_1 has a nonpetrifaction out-node B_1. This means that $A_1.bw$ may change. Finally, $s.bw$ is updated to 18. In terms of pruning, any petrifaction node can be regarded as the destination. OWS is not required to consider any node from V_{petr} as a candidate for the next search, because their *backward* is the shortest length from here to e. Hence, Petrifaction pruning $prune_{PF}(v_n)$ is defined as in (7). If a node undergoes Petrifaction, even if the search continues, then no shorter route can be identified. B_2 is not a candidate of A_2 in Fig. 5 because B_2 is a petrifaction node.

$$prune_{PF}(v_n) = \begin{cases} \text{TRUE}, & v_n \in V_{petr} \\ \text{FALSE}, & otherwise \end{cases} \tag{7}$$

C. Wilting

When node v has no candidate based on the Filter, Potential, or Petrifaction pruning mechanism to expand, v engages in what is called wilting and is added to V_{wilt}. For a wilting node, *forward* must be reduced so that it is not considered a candidate based on other pruning mechanisms; this is done to reduce the number of candidates in the following searches. During the route search of OWS, the out-node v_n of node v_c is considered a candidate when none of the pruning conditions holds. Hence, the main inequalities (5) and (6) can be moved in Filter and Potential pruning, and $v_c.fw$ should be less than the right-hand sides of (8) and (9) to consider the out-node v_n as a candidate. Here, $v_c.fw$ is modified such that the pruning condition holds until v_c is not excluded by any pruning mechanism, and v_c can be selected as a candidate again.

$$v_c.fw < v_n.fw - d(v_c, v_n) \tag{8}$$

$$v_c.fw < L(s, e, T) - v_n.est - d(v_c, v_n) \tag{9}$$

Wilting can be analyzed from two cases. The first case is from a node to an out-node. Because a node can be considered a candidate if it is not excluded by any pruning mechanism, the minimal value of (8) and (9) is regarded as the wilting formula of v_c to v_n, as indicated by (10). The second case is from a node to all out-nodes. As indicated by (11), the maximum value based on (10) of all out-nodes of v_c is *wilt*(v_c), the wilting value of v_c. If $v_c.fw$ is smaller than *wilt*(v_c), then the out-node that allows for a larger wilting value is missed, and the shortest route may not be identified. In other words, if a node v_p considers a wilted node v_c (i.e., $v_c. \in V_{wilt}$) as a candidate, then the sum of $v_p.fw$ and $d(v_p, v_c)$ should be less than $v_c.fw$ according to (5). In Fig. 5, because A_2 has no candidate, it is wilted and added to V_{wilt}. $A_2.fw$ must be reduced to $wilt(A_2) = \max(w(A_2, B_1), w(A_2, B_2)) = 2$, where $w(A_2, B_1)$ and $w(A_2, B_2)$ are $\min(13, 18 - 8) - 10 = 0$ and $\min(10, 18 - 8) - 8 = 2$, respectively.

$$w(v_c, v_n) = \min(v_n.fw, L(s, e, T) - v_n.est) - d(v_c, v_n) \tag{10}$$

$$wilt(v_c) = \min_{v_n \in N^+(v_c)} w(v_c, v_n) \tag{11}$$

D. Selection

For v_c, all out-nodes v_n that are not excluded by any pruning mechanism are added to the candidate set V_{cand}. OWS selects one of the nodes in V_{cand} with the lowest *potential length* for path expansion, wherein the *potential length* can be measured using *forward*, *backward*, and *estimate*. For a node, because *estimate* is typically known, the sum of *forward* and *estimate* can be considered the *potential length*. However, this may easily yield an underestimation of the candidate and render the search process inefficient. Therefore, a better strategy is to consider *backward-first* instead of *estimate-only* if the node *backward* is known, because *backward* is closer to the actual length. However, *backward-first* may select candidate nodes that have not been selected to expand and lead to an inefficient search process. To solve this problem, a new formula is customized in (12) to dynamically balance *estimate* and *backward*. Here, *estimate* is more reliable when it is closer to *backward* and when the formula is closer to $v.fw + v.est$ (i.e., $v.est/v.bw \cong 1$). By contrast, the *potential length* should be closer to $v.fw + v.bw$ when *estimate* is much smaller than *backward* (i.e., $v.est/v.bw \cong 0$), because *estimate* is unreliable:

$$L_{PT}(v) = \begin{cases} v \cdot fw + v \cdot bw \\ -(v \cdot bw - v \cdot est) \times \frac{v \cdot est}{v \cdot bw}, & v \cdot bw \text{ is known} \\ v \cdot fw + v \cdot est, & \text{otherwise} \end{cases} \tag{12}$$

E. Reverse Update

In each search, the node bounds are updated and become tighter. For example, the *forward* of a next node is updated when Filter pruning is used. Although the *backward* and *estimate* of a node are updated similarly to *forward* in each search, the bound information is not immediately updated and is reflected to all nodes from the tail to the head. This strategy is called passive update. Compared with active update, in passive update, when a search is complete, the *backward* and *estimate* of all nodes are actively updated from the tail to the head of this search path. Active update is also called reverse update in OWS. Here, *backward* is reverse-updated along the path by (3), and *estimate* is reverse-updated by (13), where *v.est* is maintained as the minimal length obtained from all the out-nodes of v. For example, *s.est* is updated to $\min(A_1.est + d(s, A_1), A_2.est + d(s, A_2)) = 18$ after reverse update in Fig. 4.

$$v.est = \min_{v' \in N^+(v)} \left(v'.est + d\left(v, v'\right)\right) \leq L_{OPT}(v, e) \tag{13}$$

Algorithm 1. One-Way Search (OWS)

Input: s (starting point), e (destination), R (user requests)
Output: T (shortest MRRP route)

1 $G \leftarrow$ Graph structure of $R, V_{petr} \leftarrow \{e\}, V_{wilt} \leftarrow \{\}, T \leftarrow \langle\rangle$
2 $v.fw \leftarrow \infty,\ v.bw \leftarrow \infty, v.est \leftarrow d_{Euc}(v, e), \forall v \in G,$
3 $V_{cand} \leftarrow \{s\}$ //starts a new search
4 **repeat**
5 $v_c \leftarrow \arg\min_{v \in V_{cand}} L_{PT}(v), V_{cand} \leftarrow \{\}$, Add v_c to V_{petr}
6 **for each** v_n **in** $v_c.outNodes$ **do**
7 **if** NOT $prune_{FT}(v_c, v_n)$ **then**
8 $v_n.fw \leftarrow v_c.fw + d(v_c, v_n), v_n.prev \leftarrow v_c$
9 Remove v_n from V_{wilt}
10 **if** NOT $prune_{PT}(v_c, v_n)$ **and** NOT $prune_{PF}(v_n)$ **then**
11 Add v_n to V_{cand} //v_n is a candidate of v_c
12 **end if**
13 **end if**
14 **if** $v_n.bw + d(v_c, v_n) < v_c.bw$ **then**
15 $v_c.bw \leftarrow v_n.bw + d(v_c, v_n), v_c.next \leftarrow v_n$
16 $T \leftarrow traceRoute(v_c)$, **if** $v_c.fw + v_c.bw < L(s, e, T)$
17 **end if**
18 Remove v_c from V_{petr}, if $v_n \notin V_{petr}$
19 **end for each**
20 **until** V_{cand} is empty
21 **return** T, **if** $v_c = s$ //Optimal route is found
22 Add v_c to $V_{wilt}, v_c.fw \leftarrow wilt(v_c)$
23 reverseUpdate(v_c)
24 **goto** Line 3 for next search OR early stop OWS **return** T

Fig. 6. Pseudocode of the OWS algorithm.

F. Algorithm

The pseudocode of OWS is presented in Fig. 6. OWS restarts the search multiple times and terminates when no shorter route can be identified. The input includes s, e, and R, and the output is T. A graph G of R is constructed, in which only e is petrifaction and no wilting node exists. In addition, T is empty, and the corresponding length $L(s, e, T)$ is equal to infinity (Line 1). The node bounds in G are initialized (Line 2). For a search, only s is added to V_{cand} (Line 3). For each path expansion (Lines 4–20), the node v_c with the shortest *potential length* from V_{cand} is selected (v_c typically starts from s), V_{cand} is reset, and v_c is assumed as petrifaction (Line 5). For each out-node v_n of v_c (Lines 6–19), $v_n.fw$ is updated, the previous node of v_n ($v_n.prev$) is v_c, and v_n is

Function reverseUpdate

Input: v_c (current node)

25 **while** $v_c.prev \neq NULL$ **do**
26 $v_p \leftarrow v_c.prev$
27 **if** $v_c.bw + d(v_p, v_c) < v_p.bw$ **then**
28 $v_p.bw \leftarrow v_c.bw + d(v_p, v_c), v_p.next \leftarrow v_c$
29 **end if**
30 $v_p.est \leftarrow \min_{v_n \in N^+(v_p)}(v_n.est + d(v_p, v_n))$
31 $v_c \leftarrow v_p$
32 **end while**

Fig. 7. The reverseUpdate function in Line 23 of the OWS algorithm.

removed from V_{wilt} if v_n is not eliminated by Filter pruning. In addition, v_n is added to V_{cand} if v_n is not eliminated by Potential and Petrifaction pruning (Lines 7–13). If $v_n.bw$ is known and the length from v_c to e through v_n is less than $v_c.bw$, then $v_c.bw$ decreases and v_n becomes the next node of v_c ($v_c.next$). Simultaneously, T is updated if a shorter route is identified, wherein a *traceRoute* function is used to generate the route by recursively up-tracing along $v_c.prev$ to s and down-tracing along $v_c.next$ to e (Lines 14–17). In addition, v_c is removed from V_{petr} if any out-node of v_c does not represent petrifaction (Line 18). Path expansion is complete when no candidate remains in V_{cand}.

When a path expansion process is complete, the last determined shortest route T is regarded as the optimal route if v_c is the starting point s (Line 21). Otherwise, v_c should be wilted and added to V_{wilt}, and $v_c.fw$ is reduced according to (11) (Line 22). Before a new search is initialized or OWS is stopped early (Line 24), OWS reverse-updates the node bounds along this expanded path (Line 23). The *reverseUpdate* function is presented in Fig. 7. As mentioned in Section 4.5, the reverse update process includes *backward* and *estimate* updates. For the previous node v_p of v_c, the update of *backward* is simple because it is the shortest actual distance determined so far from v_p to e. The *backward* and the next node of v_p are updated to v_c if the sum of $v_c.bw$ and the distance between v_c and v_p is smaller than $v_p.bw$ (Lines 27–29). However, $v_p.est$ is the smallest estimated distance from v_p to e. During a reverse update process, OWS reverse-updates $v_p.est$ to the minimal distance from v_p to e through any out-node v_n of v_p (Line 30). OWS guarantees that the update of *estimate* remains the smallest value calculated for all out-nodes.

V. Experimental Evaluation

This section describes the POI dataset and experimental settings used in this study. The experiments were divided into two parts. The first part involved mechanism testing and tuning in OWS to obtain the most optimal configuration, and the second part involved comparing the performance of other algorithms. In all experiments, an average of 1,000 trials were performed, and the starting point and destination of each trial were randomly located in the experimental region. Evaluations were conducted using Java 8 on a computer running Linux Ubuntu 18.04 with an Intel Core i7 CPU at 3.40 GHz and 32 GB of RAM.

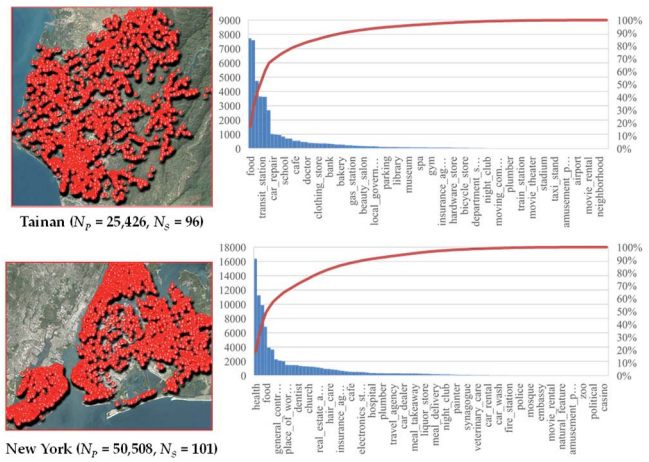

Fig. 8. Location distribution of POIs and the distribution of the number of POIs providing various services in Tainan and New York.

A. Experimental Data

Real-world POI datasets were collected using Google Places API [4] from Tainan City (Taiwan) and New York City (USA), which, respectively, span approximately 60 and 48 km^2, have 25,426 and 50,508 POIs (N_P), and have 96 and 101 distinct types of provided services. For each POI, the average number of services and the maximum number of distinct services were 1.8 and 8, respectively. Fig. 8 shows the distribution of POIs and the number of POIs providing this service in the two cities. Approximately 80% of the POIs provided the top 20% of services, whereas the maximum numbers of POIs providing any service were less than 7,740 and 16,388 in Tainan and New York, respectively. In all experiments, a service was randomly selected because the user-specified requests depended on the number of POIs providing this service. In addition, the distance between every two POIs was measured using the Euclidean distance according to their latitude and longitude, and the longest POI distance in both cities was approximately 60 km. According to the data statistics, the default number of requests and the distance between the starting point and the destination of each query were set as 6 and 15 km, respectively.

B. Mechanism Testing and Tuning

This section discusses the testing and tuning of the mechanisms of OWS, including the selection strategy and the effect of reverse update. To conduct a comprehensive experiment and verify the generalizability of OWS, two real-world datasets from Tainan and New York were used to assess the search efficiency and solution quality (versus optimal route) of the proposed mechanisms of OWS.

1) Effect of Various Selection Strategies: The candidate selection in OWS was described in Section 4.4.In this experiment, the search efficiency of the proposed customization strategy, denoted as *Cust*, was compared on the *potential length* in Eq. (12) with three other selection strategies: 1) *EstOnly*, where the sum of *forward* and *estimate* is typically used to measure the *potential length*; 2) *BwFirst*, where the sum of *forward* and *backward* is used to measure the *potential length* if *backward* is known; otherwise, the *potential length* is

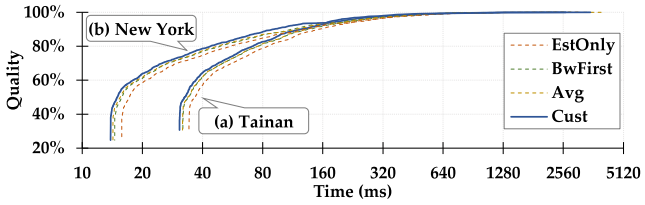

Fig. 9. Comparison of selection strategies.

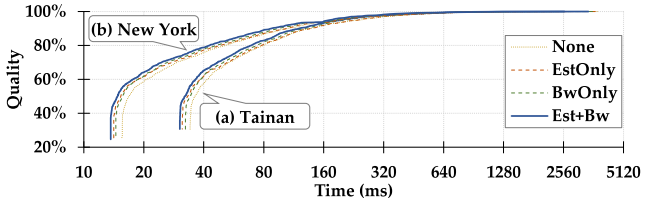

Fig. 10. Comparison of reverse update strategies.

measured using the sum of *forward* and *estimate*; and 3) *Avg*, which is similar to *Cust* except that the sum of *forward* and the average of *estimate* and *backward* are employed to measure the *potential length* if *backward* is known. Fig. 9 shows the search efficiency of OWS based on the four selection strategies. Although the performance of the four strategies was strongly similar, *Cust* performed the best for the two cities. Generally, *BwFirst* performed better than *EstOnly*. This is because *estimate* is a straight distance from a node to the destination. Although it increases during the search process, this bound is often much lower than the actual shortest length. By contrast, *backward* is the actual length and is more likely to be used in the calculation of the *potential length*. Both *Avg* and *Cust* performed slightly better than *BwFirst*. This means that measuring the *potential length* by simultaneously considering *estimate* and *backward* is useful. However, *Cust* dynamically balances *estimate* and *backward* depending on the reliability of *estimate*, whereas *Avg* only statically calculates their mean. Hence, the search efficiency of *Cust* remains higher than that of *Avg*. Overall, *Cust* performed well compared with other selection strategies in the two real-world datasets.

2) Effect of Various Reverse Update Strategies: As mentioned in Section 4.5, reverse update was conducted after the OWS search process was complete. This experiment was conducted to compare the search efficiency of the proposed reverse update strategy, denoted as *Est+Bw*, with those of three reverse update strategies: 1) *None*, where no reverse update is performed (also called passive update); 2) *EstOnly*, where only *estimate* is updated; and 3) *BwOnly*, where only *backward* is updated. As shown in Fig. 10, although all strategies exhibited highly similar performance, *Est+Bw* performed the best in the two cities. In addition, although the computational cost increased with the update of the node bounds, this benefited the path selection process in the next search. Hence, the strategies that involved reverse update are more efficient than *None*. Moreover, *BwOnly* was determined to be slightly more efficient than *EstOnly*. This, however, did not mean that *backward* was more critical than *estimate*. Rather, the update of *estimate* was more computationally expensive

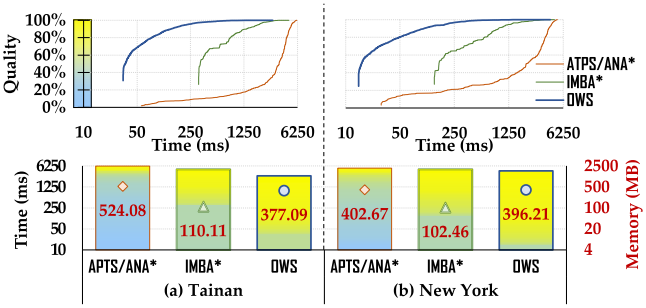

Fig. 11. Search efficiency and memory usage of various algorithms.

than that of *backward*. Finally, when *estimate* and *backward* were simultaneously updated, the improvement observed in search efficiency outweighed the increase in the computational cost.

C. Algorithm Comparison

OWS with a complete suite of mechanisms was compared with APTS/ANA* [21] and IMBA* [11] in terms of the search efficiency and solution quality (versus the optimal route) and under various conditions, including query scalability, city scalability, common/rare requests, and start-to-destination query distance. This experiment was conducted on the two real-world datasets.

1) Search Efficiency and Memory Usage: APTS/ANA*, IMBA*, and OWS are considered anytime algorithms, thus they can iteratively output improved MRRP routes. An experiment was conducted to compare the search efficiency of OWS with those of APTS/ANA* and IMBA* in the two cities with 6 user-specified requests. As shown in Fig. 11 (top), although the time taken to identify the most optimal solution was similar for all the algorithms, the search efficiency of OWS was considerably higher than those of APTS/ANA* and IMBA*. The average solution qualities of APTS/ANA* and IMBA* at 250 ms were 9% and 0% for Tainan, and 21% and 64% for New York, but that of OWS was over 95% for both cities. In addition, OWS identified a near-optimal route within 1 s. This indicated that the combination of a greedy algorithm and a branch-and-bound method was considerably more efficient than the best-first search and heuristic variant of A* to respond to MRRP queries.

To indicate the performance of OWS under various conditions, the search efficiency is represented by a stacked bar chart with a color gradient in Fig. 11 (bottom), in which the yellow and blue colors indicate high and low quality, respectively. This design highlights not only the execution time to identify the optimal route but also the route quality obtained with such an execution time for each algorithm. The symbols shown in the bar indicate the memory usage of each algorithm. According to the results, although the memory usage of OWS was higher than that of IMBA*, OWS output a high-quality solution considerably earlier than APTS-ANA* and IMBA* did.

2) Effect of Query Scalability: The effect of the number of user-specified requests (N_R) (from 4 to 8) was analyzed in the two cities. As shown in Fig. 12, the time and memory costs of all the algorithms considerably increased with N_R.

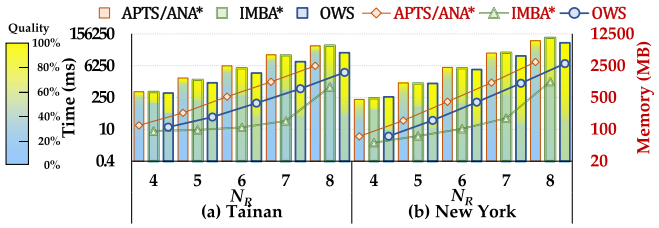

Fig. 12. Effect of query scalability.

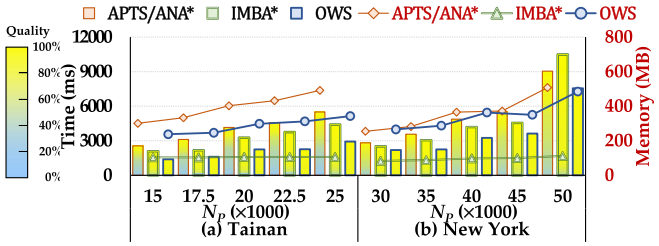

Fig. 13. Effect of city scalability.

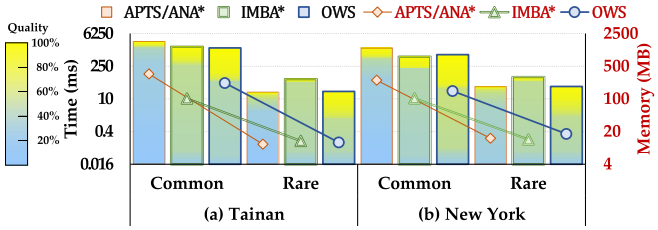

Fig. 14. Effect of common and rare requests.

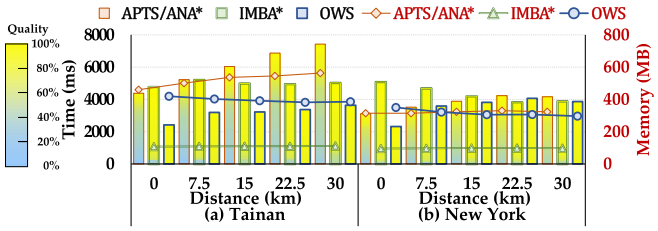

Fig. 15. Effect of start–destination distance.

This is because the search space increased, causing all the algorithms to spend considerably more time and memory searching for solutions. However, OWS identified high-quality solutions earlier than APTS-ANA* and IMBA* did, especially when the number of requests was large. This was presumably because APTS/ANA* and IMBA* strongly depended on the heuristic function. Once the function became unsatisfactory, numerous unnecessary nodes and paths were rapidly expanded. OWS integrated a greedy algorithm and a branch-and-bound method, which helped mitigate the problem of heavily relying on the heuristic function.

3) Effect of City Scalability: An experiment was conducted to analyze the effect of city scalability. To create various scalabilities, POIs were randomly deleted from the two datasets to control their number (N_P). N_P was varied from 15,000 to 25,000 for Tainan and from 30,000 to 50,000 for New York, whereas N_R was set as 6 by default. As shown in Fig. 13, when N_P increased, the increases observed in the execution time and memory usage were not as rapid as the increase in N_R highlighted in Section 5.3.2. This indicated that N_R had a larger effect on the search space than that of N_P for MRRP queries. Furthermore, because the MRRP queries did not consider the request order, the solution space increased with N_R factorial, but only with the N_P power. In conclusion, OWS is more efficient than APTS/ANA* and IMBA* in various city scalabilities.

4) Effect of Common and Rare Requests: An experiment was conducted to compare the effects of all algorithms when all user-specified requests are common or rare in the two cities. As shown in Fig. 8, common and rare requests are defined as the first 30% of services provided by most POIs and the last 30% of services provided by only a few POIs, respectively. The experimental results are presented in Fig. 14. Generally, common requests require more overheads than those required by rare requests, because common requests represent much more POIs that must be searched compared with rare requests. Although APTS/ANA* and IMBA* had their own strengths in the two types of requests, OWS still considerably outperformed APTS/ANA* and IMBA* in terms of the search efficiency in both types of requests.

5) Effect of Start–Destination Distance: A final experiment was conducted to investigate the effect of the query distance between the start and the destination from 0 to 30 km in the two cities. As shown in Fig. 15, the distance did not considerably affect OWS, but the time spent substantially increased for APTS/ANA* and slightly decreased for IMBA* with respect to distance. This is because APTS/ANA* relied on the effectiveness of the heuristic function. The evaluation value of the heuristic function was low and many POIs were excluded when the destination was considerably close to the start. However, many POIs were located within the allowed distance and required expansion when the destination distance increased. Nevertheless, IMBA* smoothly limited the queue size, hence overcoming the disadvantage of APTS/ANA*. OWS did not entirely rely on the effectiveness of the heuristic function, thus it did not include more candidates because the evaluation function allowed larger values. In conclusion, OWS is more efficient and stable than APTS/ANA* and IMBA* at various query distances.

VI. Conclusion and Future Work

MRRP is a query that involves identifying the shortest route that can solve all user requests in a POI network. However, most previous studies have focused on rapidly planning an approximate route based on a greedy search. In this study, MRRP queries were converted into a problem of determining the shortest path on a graph. An anytime algorithm called OWS, integrating both branch-and-bound and greedy methods, was proposed to efficiently identify high-quality MRRP routes and perform an iterative search before the most optimal route is determined. In OWS, three bounds, namely, *forward*, *backward*, and *estimate*, were recorded in each node, which were used to support three pruning mechanisms, namely,

Filter, Potential, and Petrifaction, to eliminate the nodes that could not be the optimal solution. OWS also involved three operations, namely, Wilting, Selection, and Reverse Update, to improve the search efficiency. This is the first study on MRRP queries that simultaneously considers optimal route planning and search efficiency.

Overall, the experiments involved testing and tuning the mechanisms of OWS in the two real-world datasets obtained from Tainan and New York. OWS was compared with two anytime algorithms, namely, APTS/ANA* and IMBA*, in terms of the search efficiency and memory usage under various conditions, namely, query scalability, city scalability, common and rare requests, and start–destination distance. The results indicated that although IMBA* had the lowest memory consumption, OWS had the highest search efficiency compared with APTS/ANA* and IMBA*, especially when the user queries were complicated. OWS also identified a near-optimal route within 1 s. By contrast, the efficiency of APTS/ANA* was only 15.5% for Tainan and 32.5% for New York, and the efficiency of IMBA* was 85.2% for Tainan and 91% for New York. These results indicated that OWS exhibited an excellent search efficiency for solving MRRP queries.

In future studies, MRRP should be extended according to actual considerations. First, users may have some requests that should be addressed before another request, such as withdrawing money before purchasing, is considered. Second, some intermediate POIs may be fixed by a user; for example, the user may want to visit a specific ATM, bookstore, or school before arriving at their destination. Although OWS can solve these queries by adding some search constraints, further systematic procedures are required. Furthermore, all the mechanisms of OWS should be further optimized and bidirectional search may be used to accelerate general searches. Actual road networks are asymmetric, and traffic situations change over time. One limitation of OWS is its potential inability to identify the optimal route if future traffic situations are considered in the MRRP. Therefore, future studies should consider planning routes with time-varying asymmetric road network architectures to provide more realistic applications. Finally, more various real-world data should be collected to evaluate the effectiveness of OWS.

REFERENCES

[1] H. Chen, W.-S. Ku, M.-T. Sun, and R. Zimmermann, "The multi-rule partial sequenced route query," in *Proc. 16th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst. (GIS)*, 2008, p. 10.

[2] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[3] J. Eisner and S. Funke, "Sequenced route queries: Getting things done on the way back home," in *Proc. 20th Int. Conf. Adv. Geographic Inf. Syst. (SIGSPATIAL)*, 2012, pp. 502–505.

[4] *Google Places API*. Accessed: Apr./Jul. 2019. [Online]. Available: https://cloud.google.com/maps-platform/places/

[5] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, Jul. 1968.

[6] P. E. Hart, N. J. Nilsson, and B. Raphael, "Correction to A formal basis for the heuristic determination of minimum cost paths," *ACM SIGART Bull.*, vol. 37, pp. 28–29, Dec. 1972.

[7] Y. Kanza, R. Levin, E. Safra, and Y. Sagiv, "Interactive route search in the presence of order constraints," *Proc. VLDB Endowment*, vol. 3, nos. 1–2, pp. 117–128, Sep. 2010.

[8] R. E. Korf, "Depth-first iterative-deepening: An optimal admissible tree search," *Artif. Intell.*, vol. 27, no. 1, pp. 97–109, Sep. 1985.

[9] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S. H. Teng, "On trip planning queries in spatial databases," in *Proc. Int. Symp. Spatial Temporal Databases*, 2005, pp. 273–290.

[10] J. Li, Y. D. Yang, and N. Mamoulis, "Optimal route queries with arbitrary order constraints," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 5, pp. 1097–1110, May 2013.

[11] L. Libralesso and F. Fontan, "An anytime tree search algorithm for the 2018 ROADEF/EURO challenge glass cutting problem," *Eur. J. Oper. Res.*, vol. 291, no. 3, pp. 883–893, Jun. 2021.

[12] M. Likhachev, G. J. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *Proc. Adv. Neural Inf. Process. Syst.*, 2004, pp. 767–774.

[13] E. H. Lu, H. Chen, and V. S. Tseng, "An efficient framework for multirequest route planning in urban environments," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 4, pp. 869–879, Apr. 2017.

[14] K. Menger, "Das botenproblem," *Ergebnisse Eines Mathematischen Kolloquiums*, vol. 2, pp. 11–12, 1932.

[15] Y. Ohsawa, H. Htoo, N. Sonehara, and M. Sakauchi, "Sequenced route query in road network distance based on incremental Euclidean restriction," in *Proc. Int. Conf. Database Expert Syst. Appl.*, 2012, pp. 484–491.

[16] A. Rodríguez and R. Ruiz, "The effect of the asymmetry of road transportation networks on the traveling salesman problem," *Comput. Operations Res.*, vol. 39, no. 7, pp. 1566–1576, Jul. 2012.

[17] J. Roth, "Efficient many-to-many path planning and the traveling salesman problem on road networks," *Int. J. Knowledge-based Intell. Eng. Syst.*, vol. 20, no. 3, pp. 135–148, Jul. 2016.

[18] S. Russell, "Efficient memory-bounded search methods," in *Proc. Eur. Conf. Artif. Intell.*, 1992, pp. 1–5.

[19] M. Sharifzadeh, M. R. Kolahdouzan, and C. Shahabi, "The optimal sequenced route query," *VLDB J.*, vol. 17, no. 4, pp. 765–787, 2008.

[20] M. Sharifzadeh and C. Shahabi, "Processing optimal sequenced route queries using Voronoi diagrams," *GeoInformatica*, vol. 12, no. 4, pp. 411–433, Dec. 2008.

[21] R. Stern, A. Felner, J. van den Berg, R. Puzis, R. Shah, and K. Goldberg, "Potential-based bounded-cost search and anytime non-parametric A," *Artif. Intell.*, vol. 214, pp. 1–25, Sep. 2014.

[22] R. T. Stern, R. Puzis, and A. Felner, "Potential search: A bounded-cost search algorithm," in *Proc. Int. Conf. Automated Planning Scheduling*, 2011, pp. 234–241.

[23] J. Van Den Berg, R. Shah, A. Huang, and K. Goldberg, "ANA*: Anytime nonparametric A," in *Proc. AAAI Conf. Artif. Intell.*, 2011, pp. 1–9.

Eric Hsueh-Chan Lu (Member, IEEE) received the Ph.D. degree in computer science and information engineering from National Cheng Kung University (NCKU), Taiwan, in 2010. He is currently an Associate Professor with the Department of Geomatics, NCKU. He has published more than 30 research papers in referred journals and international conferences. His research interests include data mining, location-based services, geographic information systems, mobile computing, and intelligent transportation systems. He has served as reviewers for a number of international conferences related to data mining and mobile application.

Sin-Sian Syu received the B.S. degree in computer science and information engineering from National Taitung University, Taiwan, in 2016, and the master's degree in geomatics from National Cheng Kung University, Taiwan, in 2019. His research interests include location-based service, mobile computing, and intelligent transportation systems.