Efficient Public Transport Planning on Roads

Libin Wang, Raymond Chi-Wing Wong
The Hong Kong University of Science and Technology, Hong Kong SAR, China
{lwangct, raywong}@cse.ust.hk

***Abstract*—Public transport contributes significantly to addressing some city issues such as air pollution and traffic congestion. As the public transport demand changes in urban development, we need to plan new routes to match the demand. Existing methods of planning new bus routes either are inefficient in using the path's cost or use other inaccurate cost measurements. This paper focuses on finding a new bus route efficiently on road networks. Specifically, we first propose the Bus Routing on Roads (BRR) problem which combines two common goals of minimizing the walking costs of passengers and maximizing the connectivity of the new route to the existing transit network. They are consistent with matching the demand and facilitating the transfer. We first show the NP-hardness of the BRR and design an approximation algorithm called Efficient Bus Routing on Roads (EBRR). We theoretically analyzed its approximation ratio and time complexity. Extensive evaluations with state-of-the-art solutions on three real-world datasets validate the effectiveness and efficiency of EBRR. It could recommend a new bus route with high quality in around 10 seconds, 60x faster than the baselines.**

I. Introduction

Public transport, as an alternative to traveling by private vehicles, brings many societal and environmental benefits through ride-sharing. These benefits may include improving road congestion, providing economic opportunities, or reducing air pollution. To promote the use of public transport, a transit network (connected by all bus routes in a city) should suit as much demand as possible. However, as cities are constantly expanding or reshaping themselves [1], the demand pattern, which mainly depends on the spatial distribution of passengers' origins and destinations, never ceases to change [2], [3]. Since the current transit network could lag behind such changes in some local areas, we need to plan new routes for better use of public transport.

The studies of the route planning problem, including the traditional methods of using surveys and recent research efforts [4]–[10], all first focus on fulfilling passengers' demands, which are reflected in the spatial distribution of origins and destinations of passengers' trips. Specifically, an ideal bus route should pass through some bus stops near the demand since it would reduce passengers' travel time. Take the city of Orlando as an example. We visualize the demand extracted from [11] near Lake Nona in Figure 1 since there could be a promising neighborhood in that area [12]. The blue bus icons represent the existing bus stops, and the red areas show the demand. An ideal bus route, shown in cyan blue, would set some new bus stops (shown by green icons) near the red areas and pass through them. Moreover, it would be convenient for

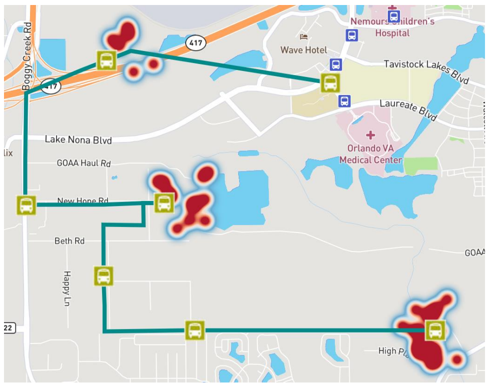


Fig. 1: A toy example in Orlando

passengers if the route passes through some existing bus stops which provide more transfer choices.

It is nontrivial to determine the bus stops for a new route. Manual selection through surveys has issues such as subjective decisions, limited samples, and high implementation cost. With the availability of spatial data, algorithms give reliable advice by quantitative comparison. However, it remains to define the *utility score* for different potential bus stops, *e.g.*, bus stops of high utility are close to the demand and offer more transfer choices. Furthermore, since the utility values for some bus stops may decrease after we select the others (*e.g.*, the demand pattern changes since much demand near the selected ones would be satisfied), we need to carefully consider the consequences of selecting different bus stops. Apart from the utility concern, the selected bus stops should be finally linked by a valid route satisfying some *regularity* constraints such as the restriction on the maximum number of bus stops. These routing constraints should be carefully treated in the selection.

Despite many existing solutions, they overlook two practical issues. First, none of them is explicitly concerned about the road network. Since the passengers have to walk some paths along network edges to the bus stops to take buses, the cost measurement should be the path's cost which is more consistent with the real scenario. However, some solutions use arbitrary and inaccurate cost measurements, such as the Manhattan distance [5] and the Euclidean distance [13]. They would fail to identify the real demand centers. Others that use the path's cost have the efficiency issue. They either assume that the cost between all pairs of nodes are given beforehand [9], [14] or compute them repeatedly on the fly [15], which may result in redundant searches and run slowly on large road networks. Furthermore, [10] performs matrix operations to measure the utility. Its preprocessing procedure for new data takes many hours long and involves complicated matrix transformations. Since practitioners may fine-tune some parameters or adjust the input (*e.g.*, the demand of different targeted areas) frequently, more efficient solutions which optimize the search on networks would be preferred.

Motivated by the above limitations, in this paper, we formulate a problem called *Bus Routing on Roads* (BRR) which redefines the two goals under the context of road networks. It aims to find a new bus route which 1) minimizes the walking cost of passengers by setting *new* nearby bus stops and 2) maximizes the connectivity to the existing transit network by passing through *existing* stops with more transfer choices. To make the new bus route practical, we propose 1) *the maximum bus stop number constraint* and 2) *the maximum adjacent bus stop cost constraint*. We prove its NP-hardness and propose an algorithm named *Efficient Bus Routing on Roads* (EBRR) with theoretical guarantees. Since computing path's cost is time-consuming, EBRR is carefully designed to avoid redundant computations. We accelerate EBRR by designing the *filtered priority queue* and pruning bounds.

We summarize our contributions as follows.

- We consider the efficiency issue of planning a bus route on road networks. We formally propose the Bus Routing on Roads (BRR) problem which plans a new route satisfying passengers' demands and connecting to the existing transit network. We also prove its NP-hardness.
- We design the Efficient Bus Routing on Roads (EBRR) algorithm which generates a bus route of *high utility* and efficiently deals with path's cost. We also show its approximation ratio and analyze its complexity.
- We propose several speed-up techniques, such as the filtered priority queue and pruning bounds.
- We empirically evaluated EBRR on three real-world datasets. It could run 60x faster than state-of-the-art solutions and generate a new route superior in many aspects. We also conducted a case study in Chicago which showed that our route covers more previously "uncovered" demand than all routes found by baselines.

II. Related Work

Traditional methods of planning a new bus route involved surveying the demand of local areas and following some intuitive principles [16]. However, it could be costly to obtain comprehensive surveys and the results were easily biased due to the subjective decision-making. Recent methodologies could be basically divided into two groups: learning from human mobility patterns (*e.g.*, origins, destinations, or trajectories) [5], [6], [8], [10], [17]–[21] and mathematical programming on detailed elements [4], [9], [13]–[15], [22].

The first group usually adopts a two-phase framework which first obtains the demand flow patterns within different pickup and dropoff areas by clustering the trajectories and then identifies a bus route based on the patterns by some heuristic ideas. Though they essentially aimed to fulfill the demand of passengers, specific goals could be improving existing routes by incorporating multiple features such as the bus fare [17], maximizing the total profits [19], matching the supply and demand in the temporal dimension [8], or matching as many trajectories as possible [5], [10], [21]. Some considered different types of bus routes: the night routes (where the demand at different time windows is distinguished) [6] and the shared buses with dynamic routes [5], [19], [20]. Most of them (except for [10], [18]) ignore the routes' connectivity with the existing transit network, but we consider it. [18] tried to rebuild the network configuration which might be impractical since it changes many existing routes. The state-of-the-art solution focused on the demand and the connectivity [10]. It maximized the new route's similarity to existing trajectories plus the gain on the network connectivity produced by the new route. However, the demand that the new route serves will overlap some existing stops based on the transit demand from the trajectories, and the evaluation of the connectivity is often time-consuming.

The second group usually defines a mathematical program to optimize a function with many objectives and constraints and uses some solvers or the heuristic algorithms to obtain the solution. The objectives could be maximizing the passengers [15] and the total profits [14], [23]–[25], and minimizing the travel cost [4], [9], [13], [22], the walking cost [14], and the number of transfers [22]. The recent solution combined k-means clustering and the genetic heuristic algorithm to solve the mathematical program [13]. The formulations can be comprehensive since it characterizes many elements, but it is hard to implement them in practice for several reasons such as the inefficiency, hard choices on parameters, and no theoretical guarantee on the final solution. Besides, they did not consider the connectivity and using the path's cost.

III. Problem Statement

A. Problem Definitions

Definition 1 (Road Network): A road network $G = (V, E)$ is a connected undirected graph with the node set V and the edge set E. Each edge e is associated with its cost $c_e \in \mathbb{R}$, which could be the travel time or the distance by user preferences.

Definition 2 (Path): A path π is a finite sequence of nodes where $\pi = (v_1, v_2, \ldots, v_{|\pi|})$ such that each $(v_i, v_{i+1}) \in E$ for $i = 1, 2, \ldots, |\pi| - 1$. Its cost is $\sum_{i=1}^{|\pi|-1} c(v_i, v_{i+1})$.

Definition 3 (Existing Bus Stop and Bus Route): The set of existing bus stops and the set of existing routes are denoted by $S_{\textit{existing}}$ and $R_{\textit{existing}}$, respectively. Each existing bus stop is a node $v \in S_{\textit{existing}} \subset V$. Each existing bus route $r \in R_{\textit{existing}}$ is represented by a pair (B_r, π_r) where $B_r \subseteq S_{\textit{existing}}$ is a set of existing stops that the route r passes through and π_r is the path connecting all the stops in B_r.

Example 1: Figure 2 shows a road network with $S_{\textit{existing}} = \{v_1, v_2\}$, where we omit the other bus stops for the ease of illustration. There are four bus routes in $R_{\textit{existing}}$ (represented by red, blue, gray, and purple lines). Route 1-4 pass through $\{v_1\}$, $\{v_1\}$, $\{v_1, v_2\}$, and $\{v_2\}$, respectively.

Note that several bus routes could pass through the same stop. The set $R_{\textit{existing}}$ of routes is also called a *transit network*.

Users would then issue the *transit routing queries* (defined below) and submit their pick-up and drop-off locations to the platforms such as Google Maps [26] or Uber [27].

Definition 4 (Transit Routing Query): Each transit routing query $q = (v_s, v_t)$ specifies an origin node $v_s \in V$ and a destination node $v_t \in V$.

The new bus route should fulfill as much demand as possible. It can be achieved by building a certain number of **new bus stops** from a set S_{new} of candidate locations ($S_{existing} \cap S_{new} = \emptyset$). Building a new bus stop with its pole and flag has a cost starting from 300 dollars [28]. It is common in cities' new bus plans [1], [29]. These new bus stops should be close to the origins and destinations of transit routing queries so that passengers can easily access them. If the set S_{new} of candidate locations is not specified, it suffices to consider the midpoints of all edges E since the edges, representing small road segments, are dense enough to cover all roads. Note that we only consider city roads and ignore highways.

To measure the travel cost from the origins/destinations to the bus stops, we define the walking cost outside the transit network for each query. Let $dist(v_i, v_j)$ be the path with the smallest cost between $v_i, v_j \in V \cup S_{new}$.

Definition 5 (Walking Cost): The walking cost of a query $q = (v_s, v_t)$ walking to a set S of bus stops, denoted by $f(q, S)$, is defined as the minimum walking cost from its origin to its nearest bus stop in S plus the minimum walking cost from its destination to its nearest bus stop in S, i.e., $f(q, S) = \min_{v \in S} dist(v_s, v) + \min_{v \in S} dist(v_t, v)$.

Example 2: Back to Example 1, suppose that there is a transit routing query $q = (v_6, v_1)$. Its walking cost to $S_{existing}$ is $f(q, S_{existing}) = dist(v_6, v_2) + dist(v_1, v_1) = 7$ since the nearest existing stops of v_6 and v_1 are v_2 and v_1, respectively.

Definition 6 (Sum of Walking Costs): Given a set T_{query} of all transit routing queries, we use Q to denote the multiset of all origins and destinations, i.e., $Q = \{v_s, v_t | q = (v_s, v_t) \in T_{\text{query}}\}$. The sum of walking costs to a set S of bus stops can be written as: $Walk(S) = \sum_{v \in Q} \min_{v' \in S} dist(v, v')$.

By the symmetry of the origin and destination, we could regard them as one type of nodes without differentiating them. We will use the "query" node to represent either of them. Also note that any query data collected by map applications would be representative enough since they could be viewed as uniform samples from the demand distribution.

Example 3: Suppose that there are three transit routing queries $q_1 = (v_6, v_1)$, $q_2 = (v_1, v_7)$, and $q_3 = (v_8, v_1)$. Then, $Q = \{v_1, v_1, v_1, v_6, v_7, v_8\}$. The walking cost $Walk(S_{existing}) = 3 \cdot dist(v_1, v_1) + dist(v_6, v_2) + dist(v_7, v_2) + dist(v_8, v_2) = 7 + 11 + 8 = 26$ since the nearest existing stops for the last three query nodes are all v_2. Consider $Walk(\{v_1, v_2, v_3, v_4\})$. Since the nearest stops for v_6, v_7, v_8 are v_3, v_4, v_3, respectively, $Walk(\{v_1, v_2, v_3, v_4\}) = 3 \cdot dist(v_1, v_1) + dist(v_6, v_3) + dist(v_7, v_4) + dist(v_8, v_3) = 3 + 3 + 4 = 10$.

We want to find a set B of new stops where $B \subseteq S_{new}$ to *minimize* $Walk(S_{existing} \cup B)$ after incorporating B into the transit network. This would help passengers reduce their trips' travel costs and promote the use of buses. It is equivalent to

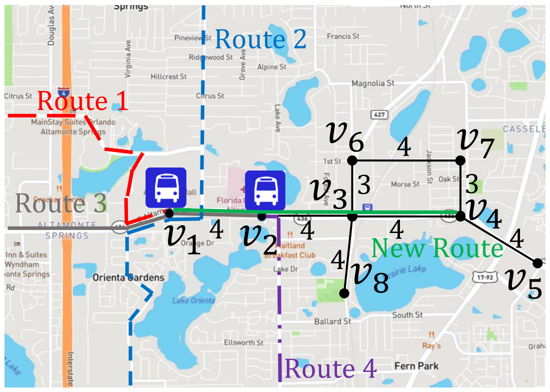


Fig. 2: A toy example of the road network

maximizing the difference $Walk(S_{existing}) - Walk(S_{existing} \cup B)$ since its first term is a constant and it could be viewed as the decrease on the walking cost affected by the set B.

Apart from decreasing the walking cost, a new bus route should also be connected to the existing transit network to provide the queries with more transfer choices, which can be done by choosing some **existing bus stops** from $S_{existing}$.

Definition 7 (Connectivity): Consider a bus route $r = (B_r, \pi_r)$. For each bus stop $v \in S_{existing}$, we use $routes(v)$ to denote the set of routes that pass through the stop v, i.e., $routes(v) = \{r | v \in B_r, r \in R_{existing}\}$. Given a set B of stops where $B \subseteq S_{existing} \cup S_{new}$, we define its connectivity to the existing transit network $R_{existing}$ as the number of distinct existing routes passing through the existing stops in B, i.e., $Connect(B) = Connect(B \backslash S_{new}) = | \cup_{v \in B \backslash S_{new}} routes(v)|$.

Example 4: Back to Example 1, if $B = \{v_1\}$, $Connect(B) = 3$ because stop v_1 serves three routes, i.e., Route 1, 2, 3 and $routes(v_1) = \{\text{Route } 1, \text{Route } 2, \text{Route } 3\}$. If $B = \{v_1, v_2\}$, $Connect(B) = 4$.

Since the operating budget of a practical bus route is limited, we propose two constraints to regularize the new bus route.

Definition 8 (New Bus Route): For a new bus route $r^* = (B_{r^*}, \pi_{r^*})$, where $B_{r^*} \subseteq S_{new} \cup S_{existing}$, the maximum bus stop number constraint requires that its number of stops $|B_{r^*}|$ should be at most the maximum number, denoted by K. Given an ordered sequence of stops $(v_1, \dots v_{|B_{r^*}|})$, the maximum adjacent bus stop cost constraint requires that the cost between any two adjacent stops $dist(v_i, v_{i+1})$ is at most the maximum cost, denoted by C, for $i = 1, \dots |B_{r^*}| - 1$.

Note that the values of K and C could be manually set or follow some statistics from data, such as the maximum number of bus stops for each route and the maximum cost for each pair of adjacent bus stops of an existing route. Also note that the cost of the new bus route is implicitly bounded by $(K - 1)C$.

To combine the two objectives about the walking cost and the connectivity, we define the utility function as follows.

Definition 9 (Utility Function): Given a new bus route with its $B_{r^*} \subseteq S_{new} \cup S_{existing}$, the utility function, denoted by U, is defined as the decrease on the sum of the walking costs plus

TABLE I: Summary of notations

Notations	Descriptions
V, E	The node and edge sets of the network
$R_{\textit{existing}}, r$	The set of all existing bus routes and a route
$S_{\textit{existing}}, S_{\textit{new}}$	The set of existing stops and locations for new ones
B_r, π_r	Route r's set of bus stops and path
T_{query}, q	The set of transit routing queries and a query
Q	The multiset of queries' origins and destinations
$routes(v)$	The set of existing routes that traverse the stop v
$\textit{Walk}(B)$,	The walking cost of a bus stop set B from Q
$\textit{Connect}(B)$	The connectivity of a bus stop set B
K	The maximum number of bus stops in the new route
C	The maximum distance between two adjacent bus stops
$U(B_{r^*}), U(v)$	The utility value of the new bus route r^* and a stop v
$v^{(i)}, B^{(i)}$	The selected stop and set of stops in the i-th iteration

its connectivity balanced by a parameter $\alpha > 0$:

$$U(B_{r^*}) = \textit{Walk}(S_{\textit{existing}}) - \textit{Walk}(S_{\textit{existing}} \cup B_{r^*}) + \alpha \cdot \textit{Connect}(B_{r^*}). \quad (1)$$

Note that the parameter α used to capture the influence of two factors is common in bus routing [10], [14]. It can be subjectively refined multiple times or set according to the corresponding values of some sample bus routes in a city.

Abusing notations slightly, we define $U(v) = U(\{v\})$.

Example 5: We still use Example 1 to illustrate the utility $U(\cdot)$. Suppose that $S_{\textit{new}} = \{v_3, v_4, v_5\}$ and we take $B_{r^*} = \{v_1, v_2, v_3, v_4\}$ from $S_{\textit{new}} \cup S_{\textit{existing}}$ shown by the green line (where we take two existing stops v_1 and v_2 and two new ones v_3 and v_4) and $\alpha = 1$. The previous Example 3 and Example 4 show that $\textit{Walk}(S_{\textit{existing}}) = 26$ and $\textit{Walk}(S_{\textit{existing}} \cup B_{r^*}) = 10$ since $S_{\textit{existing}} \cup B_{r^*} = \{v_1, v_2, v_3, v_4\}$, and $\textit{Connect}(B_{r^*}) = 4$. Hence, the utility $U(B_{r^*}) = 26 - 10 + 1 \times 4 = 20$.

We finally propose our Bus Routing on Roads problem.

Definition 10 (Bus Routing on Roads (BRR)): Given the road network G, a set $R_{\textit{existing}}$ of existing routes, and a multiset Q of query nodes, the problem aims to find a new bus route $r^* = (B_{r^*}, \pi_{r^*})$, where $B_{r^*} \subseteq S_{\textit{new}} \cup S_{\textit{existing}}$, to maximize the utility such that $|B_{r^*}| \le K$ and the cost between any two adjacent stops in the path π_{r^*} is at most C.

Table I lists the main notations used throughout the paper.

B. Hardness

We first show that the utility function (1) is a monotone *submodular* function, and then derive the NP-hardness of BRR as the *constrained submodular maximization*.

Let $V' = S_{\textit{new}} \cup S_{\textit{existing}}$. A function $U : 2^{V'} \to \mathbb{R}_{\ge 0}$ is said to be *submodular* if for any $B, B' \subseteq V'$ and $v \in V'/(B' \cup B)$, $U(B \cup \{v\}) - U(B) \ge U(B \cup B' \cup \{v\}) - U(B \cup B')$. Let $\Delta U_B(v) = U(B \cup \{v\}) - U(B)$. The inequality can be rewritten as $\Delta U_B(v) \ge \Delta U_{B \cup B'}(v)$.

Theorem 1: The function (1) is monotone submodular.

Proof: The monotonicity can be easily observed since either the decrease on the walking cost or the connectivity should not decrease as we include more nodes in the set.

To prove $\Delta U_B(v) \ge \Delta U_{B \cup B'}(v)$, we could first ignore the constant $\textit{Walk}(S_{\textit{existing}})$ in the $U(B)$ since the difference on both sides of the inequality would all be 0. If $v \in S_{\textit{new}}$, $\textit{Connect}(B \cup \{v\}) = \textit{Connect}(B)$ for any B, which means that the differences about the connectivity are all 0 on both sides of the inequality. We only need to show that $\textit{Walk}(S_{\textit{existing}} \cup B) - \textit{Walk}(S_{\textit{existing}} \cup B \cup \{v\}) \ge \textit{Walk}(S_{\textit{existing}} \cup B \cup B') - \textit{Walk}(S_{\textit{existing}} \cup B \cup B' \cup \{v\})$. It suffices to prove it for any $v_q \in Q$. Let v' be the nearest node of v_q in $S_{\textit{existing}} \cup B \cup B'$. If $v' \in S_{\textit{existing}} \cup B$, we can remove B' in the RHS and the two sides are equal. If $v' \in B'$, let v'' be the nearest node of v_q in $S_{\textit{existing}} \cup B$ and $dist(v_q, v') < dist(v_q, v'')$. Since the two sides can be rewritten as $\max(dist(v_q, v'') - dist(v_q, v), 0)$ and $\max(dist(v_q, v') - dist(v_q, v), 0)$, the inequality still holds.

If $v \in S_{\textit{existing}}$, the differences about the walking cost are all 0. We next show that $\textit{Connect}(B \cup \{v\}) - \textit{Connect}(B) \ge \textit{Connect}(B \cup B' \cup \{v\}) - \textit{Connect}(B \cup B')$. It holds because $\textit{Connect}(B) = |\cup_{v \in B}\ routes(v)|$ is a type of coverage function and the difference is the number of routes that are different from the obtained distinct routes and pass through v, which is smaller with more selected nodes. ■

Theorem 2: The BRR problem is NP-hard.

Proof: We will use the reduction from the submodular maximization problem with cardinality constraints [30]. The decision version of the submodular maximization problem tries to answer whether the function value of a submodular function on a set of items could be at least some value $L > 0$ under the constraint of the maximum number of items, denoted by k. We construct our decision problem by setting $K = k$, C as the maximum cost between any two nodes, and using the same submodular function. Since C makes no restriction, we could choose any stop given any current solution set. The submodular maximization problem has a yes answer if and only if ours has a yes solution. ■

IV. Methodology

A. Overview

Recall that the utility of a new stop in $S_{\textit{new}}$ is its decrease on the walking cost and the utility of an existing stop in $S_{\textit{existing}}$ is its increase on the number of distinct transfer choices. We want to include some stops with high utility values in the solution. Intuitively, a route including multiple stops that significantly decrease the walking cost is supposed to be scattered to suit more demand queries. However, if they are far from each other, we may break the maximum bus stop number constraint of K and the maximum adjacent bus stop cost constraint of C. Therefore, we propose to select stops one by one while caring about the constraints and terminate until we estimate that the constraints should be tight if the selected stops were linked by a path. Two essential parts, the iterative selection of stops and the estimation of the constraints, are all based on our proposed *price function* for a stop w.r.t. a set of selected stops. The price of a stop can be seen as an approximation of its effect on the increase on the number of stops in the final bus route with the constraint of C satisfied. Then, we can terminate the algorithm when the sum of the prices indicates that the maximum bus stop number K would be violated. The iterative selection finds

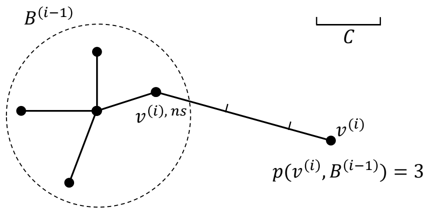


Fig. 3: The price $p(v^{(i)}, B^{(i-1)})$ of the stop $v^{(i)}$

the most "profitable" stop that maximizes the increase on the utility divided by its price.

We define the price function by considering the two constraints of K and C. Let $B^{(i-1)}$ denote the currently selected stops at the end of the $(i-1)$-th iteration and also the beginning of the i-th iteration. Let $v^{(i)}$ denote the stop that we select in the i-th iteration. Let $v^{(0)}$ be an arbitrarily selected stop and $B^{(0)} = \{v^{(0)}\}$. Now we are about to select the stop $v^{(i)}$. If a stop has high utility values and a cost at most C from one of the selected stops in $B^{(i-1)}$, we can include it in the solution without worrying about C. However, if it has a cost larger than C from any stop in $B^{(i-1)}$ and we still want to select it (due to its high utility), we need to additionally select some necessary *intermediate* stops in $S_{new} \cup S_{existing}$ so that there is a path between $v^{(i)}$ and its nearest stop in $B^{(i-1)}$, denoted by $v^{(i),ns} = \min_{v \in B^{(i-1)}} dist(v^{(i)}, v)$, with the adjacent cost between intermediate stops and $v^{(i)}$ and $v^{(i),ns}$ at most C. The minimum number of intermediate stops plus one (corresponding to $v^{(i)}$) is the price. Take Figure 3 as an example. The price of the stop $v^{(i)}$ is 3 since we need at least 2 intermediate stops from $v^{(i)}$ to the node $v^{(i),ns}$ in $B^{(i-1)}$. Formally, the price is defined as follows.

Definition 11: The price of a stop $v \in S_{new} \cup S_{existing} \backslash B^{(i-1)}$, denoted by $p(v, B^{(i-1)})$, is the minimum number of intermediate stops that we have to use in any path between v and its nearest stop $v^{ns} \in B^{(i-1)}$ without violating the constraint of C plus one (*i.e.*, v).

Example 6: Suppose that $B^{(i-1)} = \{v_1\}$ and $C = 4$. $p(v_3, B^{(i-1)}) = 2$ because $dist(v_3, v_1) = 8 > C$ and we need to add at least one intermediate stop (e.g., v_2) to link v_3 to $v_1 \in B^{(i-1)}$. $p(v_2, B^{(i-1)}) = 1$ since the number of necessary stops between v_2 and $v_1 \in B^{(i-1)}$ is zero.

It is useful to consider $(v^{(i)}, v^{(i),ns})$ as a virtual edge below.

Definition 12: There exists a virtual edge (v_i, v_j) between any two stops v_i and v_j. The price of this virtual edge is defined to be the minimum number of intermediate stops that we have to use in any path between v_i and v_j without violating the constraint about C plus one.

It follows directly that the price of $(v^{(i)}, v^{(i),ns})$ is equal to $v^{(i)}$'s price $p(v^{(i)}, B^{(i-1)})$. Moreover, the i virtual edges $(v^{(1)}, v^{(1),ns}), (v^{(2)}, v^{(2),ns}) \dots, (v^{(i)}, v^{(i),ns})$ form a spanning tree which connects $v^{(0)}, v^{(1)}, \dots, v^{(i)}$ because they form a connected graph with i virtual edges and $i+1$ nodes. The following corollary can also be derived.

Corollary 1: The sum of prices of all virtual edges in a tree is equal to the number of stops to connect all the endpoints of virtual edges minus one.

Algorithm 1: Efficient Bus Routing on Roads (EBRR)

```
   input : The network $G$, a set $R_{existing}$ of existing
           routes, a multiset $Q$ of query nodes, the
           constraints of $K$ and $C$
   output: A new bus route $r^* = (B_{r^*}, \pi_{r^*})$
1  preprocess to get initial utility values $U(\{v\})$ (Alg. 2)
2  $v^{(0)} \leftarrow$ an arbitrarily selected stop in $S_{new} \cup S_{existing}$
3  $B^{(0)} \leftarrow \{v^{(0)}\}$
4  repeat
5      $v^{(i)} \leftarrow \arg\max_{v \in S_{new} \cup S_{existing} \backslash B^{(i-1)}} \frac{\Delta U_{B^{(i-1)}}(v)}{p(v, B^{(i-1)})}$
       (Alg. 3)
6      $B^{(i)} \leftarrow B^{(i-1)} \cup \{v^{(i)}\}$
7  until $\sum_{j=1}^{i} p(v^{(j)}, B^{(j-1)}) \geq \frac{2K}{3}$;
8  use the Christofides' algorithm to get an order on $B^{(i)}$
9  use the order to generate the final bus route $r^*$ (Alg. 5)
10 return $r^* = (B_{r^*}, \pi_{r^*})$
```

In other words, the sum of prices of the virtual edges in the spanning tree is equal to the number of stops (*i.e.*, the intermediate and profitable ones) to connect all profitable ones (*i.e.*, $v^{(0)}, \dots, v^{(i)}$) minus one (*i.e.*, $v^{(0)}$).

We select the profitable stops repeatedly once the price $\sum_{j=1}^{i} p(v^{(j)}, B^{(j-1)})$ is no less than $\frac{2K}{3}$. To finally create a path (instead of a tree) visiting each profitable stop once, we can use the Christofides' algorithm [31] to first create a *virtual path*, defined to be any permutation order of all profitable ones. The reason why we use $\frac{2K}{3}$ in the stopping condition is that the Christofides' algorithm guarantees that the resulting virtual path has its sum of prices of all consecutive virtual edges less than $\frac{3}{2}$ times the sum of prices of the virtual edges in the spanning tree on the Euclidean plane. The virtual path then has basically less than K stop.

The whole procedure is shown in Algorithm 1. We preprocess the utility values in line 1 and start with an arbitrary stop in lines 2-3. If the increase on the utility is denoted by $\Delta U_{B^{(i-1)}}(v) = U(B^{(i-1)} \cup \{v\}) - U(B^{(i-1)})$, we select the most profitable stop in each iteration which has the maximum ratio of the increase on the utility to the price (*i.e.*, $\frac{\Delta U_{B^{(i-1)}}(v)}{p(v, B^{(i-1)})}$) until the number of stops is greater than $\frac{2K}{3}$ in lines 4-7. We get a visiting order by the Chritofides' algorithm in line 8 and generate the final path in line 9.

B. Query Preprocessing

The purpose of preprocessing the utility values $U(\{v\})$ for each stop $v \in S_{new} \cup S_{existing}$ is to obtain an initial order for the stops and to efficiently select the profitable ones later. We use $U(v) = U(\{v\})$ for the ease of notations. For each $v \in S_{existing}$, $U(v)$ can be easily computed as $\alpha \cdot |routes(v)|$ by Definition 7. However, for each $v \in S_{new}$, computing its initial $U(v)$ is hard. A straightforward but costly method is to separately consider each $v \in S_{new}$. It requires us to identify the query nodes whose walking costs would decrease if the stop v were selected. Since we need to independently find the set of query nodes which are the *reverse nearest neighbors* of each

$v \in S_{new}$ (where the reverse nearest neighbor of v means that its nearest neighbor is v), denoted by $RNN(v)$, this method could search the whole road network redundantly. Instead, we start the search from each query node until a stop $v \in S_{existing}$, which stops quickly in a small area of the network.

For each $v \in S_{new}$, its utility value $U(v)$ can be viewed as the difference between the two walking costs before and after the stop v is included. Specifically, there is only $S_{existing}$ first and each query node v_q has to walk to its nearest existing stop, denoted by $nn(v_q) \in S_{existing}$. We could gain a decrease on its walking cost if a stop $v \in S_{new}$ closer than $nn(v_q)$ were selected. We will iterate through all the query nodes to find such stops $v \in S_{new}$ where the decreases could happen and finally sum over the corresponding decreases on each $v \in S_{new}$.

Since the Dijkstra search has the property that the cost from the starting stop in each iteration monotonically increases, we could use a single search from each query node v_q to find those close $v \in S_{new}$ until its nearest existing stop $nn(v_q)$ is visited. Specifically, we start the search from each query node v_q. If a stop $v \in S_{existing}$ is visited for the first time, it must be the nearest existing stop $nn(v_q)$. We stop the search since the remaining stops must have costs no less than $dist(v_q, nn(v_q))$ by the Dijkstra property. If we visit a stop $v \in S_{new}$ before the search stops, we add the query node v_q in v's set of reverse nearest neighbors since $dist(v_q, v) \leq dist(v_q, nn(q))$ and v would be the nearest stop of v_q if v were selected.

Algorithm 2 summarizes the procedure. We do the Dijkstra search for each query node in lines 1-10 and compute the $U(v)$ for $v \in S_{new}$ in lines 11-14 and for $v \in S_{existing}$ in lines 15-16. As in a classical Dijkstra procedure, we initialize the priority queue in line 2 and relax edges in lines 10. In lines 5-7, when visiting a stop $v \in S_{existing}$, we set it as the $nn(v_q)$, store its cost $dist(v_q, nn(v_q))$, and stop the search. In lines 8-9, for $v \in S_{new}$ before the search stops, we include v_q in $RNN(v)$. We then sum over $RNN(v)$ to compute $U(v)$ in lines 11-14.

Example 7: Using the query nodes in Example 3, we start the search from v_6. As shown in Figure 4, the search will first visit the stop v_3 with $dist(v_6, v_3) = 3$. We will update $RNN(v_3) = \{(v_6, 3)\}$ since the nearest stop of v_6 would be v_3 if v_3 were selected. Then, the search visits v_7 and we do nothing since it is a query node. It does not matter whether we visit v_2 or v_4 next since $dist(v_6, v_2) = dist(v_6, v_4)$ indicates that v_4 offers no decrease on the walking cost. Suppose that we process v_2 first. We save $nn(v_6) = v_2$ and its cost $dist(v_6, v_2) = 7$, and stop the search.

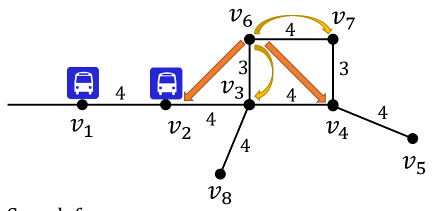
Next, to compute the utility values for each $v \in S_{new}$, we have to go through each $RNN(v)$. For v_3, $U(v_3) = dist(v_6, nn(v_6)) - 3 + dist(v_7, nn(v_7)) - 7 + dist(v_8, nn(v_8)) - 4 = 4+4+4 = 12$. Similarly, $U(v_4) = dist(v_7, nn(v_7)) - 3 = 8$ and $U(v_5) = dist(v_7, nn(v_7)) - 7 = 4$. For $v \in S_{existing}$, the utility values $U(v_1) = 3$ and $U(v_2) = 2$. Finally, the priority queue stores v_3, v_4, v_5, v_1, v_2 with their utility values.

C. Stop Selection

To find the most profitable stop in each iteration, which maximizes the ratio $\frac{\Delta U_{B^{(i-1)}}(v)}{p(v, B^{(i-1)})}$, a straightforward idea is to

Algorithm 2: Query Preprocessing

```
input : The road network $G$, a set $R_{existing}$ of existing
        bus routes, a multiset $Q$ of query nodes
output: The initial utility values $U(v)$ for each
        $v \in S_{new} \cup S_{existing}$ in a priority queue
1  foreach $v_q \in Q$ do
2      initialize a priority queue $Queue$ for costs
3      while $Queue$ is not empty do
4          $v \leftarrow$ the stop with the smallest priority in
            $Queue$
5          if $v \in S_{existing}$ then
6              set $v$ as $nn(v_q)$ and store $dist(v_q, nn(v_q))$
7              break
8          if $v \in S_{new}$ then
9              $RNN(v) \leftarrow RNN(v) \cup \{(v_q, dist(v_q, v))\}$
10         relax edges incident to $v$
11 foreach $v \in S_{new}$ do
12     $U(v) \leftarrow 0$
13     foreach $(v_q, dist(v_q, v)) \in RNN(v)$ do
14         $U(v) \leftarrow U(v) + dist(v_q, nn(v_q)) - dist(v_q, v)$
15 foreach $v \in S_{existing}$ do
16     $U(v) \leftarrow \alpha \cdot |\{r \in R : v \in B_r\}|$
17 return a queue in the decreasing order of $U(v)$
```



Search from v_6:
$v_3 \in S_{new}$: $RNN(v_3) = \{(v_6, 3)\}$
$v_2 \in S_{existing}$: $nn(v_6) = v_2, dist(v_6, nn(v_6)) = 7$

Fig. 4: Search from v_6 in the query preprocessing

do function evaluations to get the true utility $\Delta U_{B^{(i-1)}}(v)$ and price $p(v, B^{(i-1)})$ for each stop v and compare their ratios. However, this requires us to perform a large number of function evaluations. A single evaluation is time-consuming since it may search the whole network. The lazy forward selection is well-known for performing far fewer evaluations, but it only handles the case of fixed price [32]. It is infeasible here since our price is different under different $B^{(i-1)}$ and the rankings of the ratios could change dramatically. We propose the filtered queue to improve efficiency.

The filtered queue is built by first pruning some stops from the preprocessing result, then adding upper bound values (as an approximation of the true ratios) into the filtered queue, and finally using the lazy selection technique to find the maximum ratio. For the upper bound of the true ratio, it uses a technique of the lower bound price which can be updated efficiently.

Algorithm 3: Stop Selection

```
input : The initial utility $U(v)$ in the decreasing order
output: The most profitable stop $v^{(i)}$
1  $thresh \leftarrow \frac{\Delta U_{B^{(i-1)}}(v)}{p(v,B^{(i-1)})}$ where $v = \arg\max_v U(v)$
2  initialize an empty priority queue $RQueue$
3  foreach $v$ in the decreasing order of $U(v)$ do
4      if $U(v) < thresh$ then
5          break
6      insert $v$ with its priority $\frac{U(v)}{lbp(v)}$ into $RQueue$ where
       $lbp(v)$ is given by Algorithm 4
7  while $|RQueue| \geq 0$ do
8      pop the top $v$ and evaluate $\Delta U_{B^{(i-1)}}(v)$ and
       $p(v, B^{(i-1)})$
9      if $|RQueue|$ is empty or $\frac{\Delta U_{B^{(i-1)}}(v)}{p(v,B^{(i-1)})}$ is no smaller
       than the next largest priority then
10         return $v^{(i)} \leftarrow v$
11     else
12         insert $v$ with its priority $\frac{\Delta U_{B^{(i-1)}}(v)}{p(v,B^{(i-1)})}$ to
           $RQueue$ again
```

1) Pruning Stops: We first show that we could filter some stops in each iteration from the priority queue. Initially, from the preprocessing result, we have $U(v_1) \geq U(v_2) \geq \ldots$. Now suppose that we get two true values $\Delta U_{B^{(i-1)}}(v)$ and $p(v, B^{(i-1)})$ for one stop v by function evaluations. We claim that we could ignore any stop v_j with $U(v_j) \leq \frac{\Delta U_{B^{(i-1)}}(v)}{p(v,B^{(i-1)})}$.

Claim 1: In the i-th iteration, given any v with its true ratio $\frac{\Delta U_{B^{(i-1)}}(v)}{p(v,B^{(i-1)})}$, any v_j with $U(v_j) \leq \frac{\Delta U_{B^{(i-1)}}(v)}{p(v,B^{(i-1)})}$ can be pruned.

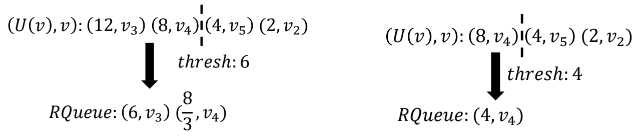
Proof: It holds since $\frac{\Delta U_{B^{(i-1)}}(v)}{p(v,B^{(i-1)})} \geq U(v_j) \geq \Delta U_{B^{(i-1)}}(v_j) \geq \frac{\Delta U_{B^{(i-1)}}(v_j)}{p(v_j,B^{(i-1)})}$, where the second inequality is due to the submodular property that $U(v_j) = \Delta U_{\emptyset}(v_j) \geq \Delta U_{B^{(i-1)}}(v_j)$ and the third one is because each price is no smaller than 1 (Definition 11). ∎

The ratio of any stop v could be a threshold for filtering stops in the decreasing order of $U(v)$. A large threshold can surely filter more stops, but computing one requires us to perform two costly function evaluations. In practice, peeking at the first one in the queue or a few stops suffices to provide a threshold without costing too much time.

2) Lazy Selection: The goal is to efficiently find the stop with the maximum ratio. To perform few function evaluations, we propose to use an upper bound $\frac{U(v)}{lbp(v)}$ as an approximation, where $lbp(v)$ is the *lower bound price* defined as $lbp(v) = \min_{v' \in B^{(i-1)}} \frac{dist_E(v,v')}{C}$ and $dist_E(v, v')$ is the cost under the Euclidean metric. $\frac{U(v)}{lbp(v)} \geq \frac{\Delta U_{B^{(i-1)}}(v)}{p(v,B^{(i-1)})}$ because $U(v) \geq \Delta U_{B^{(i-1)}}(v)$ as in Claim 1 and $lbp(v) \leq p(v, B^{(i-1)})$.

The upper bound is then applied by the following claim.

Claim 2: The stop v achieves the maximum ratio $\frac{\Delta U_{B^{(i-1)}}(v)}{p(v,B^{(i-1)})}$ if $\frac{\Delta U_{B^{(i-1)}}(v)}{p(v,B^{(i-1)})} \geq \frac{U(v')}{lbp(v')}$ holds for any other v'.


$(U(v), v)$: $(12, v_3)$ $(8, v_4)$ $(4, v_5)$ $(2, v_2)$
$thresh$: 6
$RQueue$: $(6, v_3)$ $(\frac{8}{3}, v_4)$
$(U(v), v)$: $(8, v_4)$ $(4, v_5)$ $(2, v_2)$
$thresh$: 4
$RQueue$: $(4, v_4)$

(a) 1st iteration of Algorithm 1 (b) 2nd iteration of Algorithm 1

Fig. 5: The running example of the stop selection

Proof: We have $\frac{\Delta U_{B^{(i-1)}}(v)}{p(v,B^{(i-1)})} \geq \frac{U(v')}{lbp(v')} \geq \frac{\Delta U_{B^{(i-1)}}(\{v'\})}{p(v',B^{(i-1)})}$. ∎

We can then construct a priority queue $\frac{U(v_1)}{lbp(v_1)} \geq \frac{U(v_2)}{lbp(v_2)} \geq$ ..., called $RQueue$. Each time we pop the top element we evaluate its true ratio. If the true ratio is no smaller than the second largest element, we find the most profitable stop by Claim 2. Otherwise, we push it to the queue again and inspect the next one. The advantage of upper bounds is that they can be computed efficiently and we avoid many function evaluations.

The procedures of pruning stops and lazy selection are illustrated in Algorithm 3. We compute a threshold in line 1 and use it to filter stops in lines 3-6. We insert stops with the upper bounds $\frac{U(v)}{lbp(v)}$ in line 6 and terminate it when the rest of $U(v)$ is less than the threshold. In line 8, we take the current largest element in the $RQueue$ and perform function evaluations. If it is no smaller than the second largest element, we find the profitable stop in lines 9-10. Otherwise, we push it to the queue again in lines 11-12.

Example 8: Suppose that in Algorithm 1, $K = 4$ and we select an arbitrary stop in $B^{(0)} = \{v_1\}$. In the first iteration, as shown in Figure 5a, we have a descending sequence of $U(v)$. To get a threshold (Claim 1), we evaluate v_3's utility and price, i.e., $\Delta U_{B^{(0)}}(v_3) = 12$ and $p(v_3, B^{(0)}) = 2$ (Example 6). The $thresh = 6$ and the $RQueue$ is built by considering only the first two elements. The ratio of v_3 directly uses its true value. For v_4, its ratio $\frac{U(v_4)}{lbp(v_4)} = \frac{8}{3}$ because $lbp(v_4) = \frac{dist(v_1,v_4)}{4} = \frac{12}{4} = 3$. In $RQueue$, since the first value is larger than the second one, we select v_3 as the most profitable stop in the first iteration by Claim 2 and $B^{(1)} = \{v_1, v_3\}$. In the second iteration, since $\Delta U_{B^{(1)}}(v_4) = 4$ and $p(v_4, B^{(1)}) = 1$, the threshold is 4, shown in Figure 5b. Similarly, we could obtain an $RQueue$ and the profitable stop v_4. Algorithm 1 stops here because $p(v_1, B^{(0)}) + p(v_2, B^{(1)}) = 2 + 1 \geq \frac{2K}{3} = \frac{8}{3}$.

For any stop $v \in S_{existing}$, we will use binary values for different routes, which is a common technique to speed up the set intersection. Different routes are mapped into different array elements of binary indicators. Then, we maintain such arrays for $B^{(i-1)}$ and $v \in S_{existing}$. The value $\Delta U_{B^{(i-1)}}(v)$ can be simply obtained by counting the number of common array elements. Note that it works because the number of existing routes $|R_{existing}|$ is usually bounded.

3) Updating Lower Bounds of Costs: If we evaluate the lower bound price $lbp(v) = \min_{v' \in B^{(i-1)}} \frac{dist_E(v,v')}{C}$ by iterating through each $v' \in B^{(i-1)}$ directly, there are redundant computations since the costs to some stops in $B^{(i-1)}$ could have been computed in previous iterations. We will maintain an index $lbIndex(v)$ for each stop v to record the part of the

Algorithm 4: Lower bound Price Update

```
input : The index $lbIndex(v)$ and a stop $v$
output: The lower bound price $lbp(v)$
1 if $v$'s price $p(v, B^{(i-1)})$ is valid then
2     return its true price $p(v, B^{(i-1)})$
3 else
4     for $i \leftarrow lbIndex(v), \ldots, |B^{(i-1)}|$ do
5         $lbp(v) \leftarrow \min(lbp(v), \frac{dist_E(v,v^{(i)})}{C})$
6     $lbIndex(v) \leftarrow |B^{(i-1)}|$
7 return $\max(1, lbp(v))$
```

stops that have been computed and ignore them in fetching the lower bound price. It is only used when we evaluate $lbp(v)$.

The procedure is shown in Algorithm 4. For each stop v, we return its true price if its price is valid in lines 1-2. Otherwise, we start with the $lbIndex$ until the last stop in $B^{(i-1)}$ to update the lower bound price $lbp(v)$ in lines 4-5. We update the $lbIndex$ in line 6 and return in line 7 $\max(1, lbp(v))$.

Example 9: We will show how to obtain $lbp(v_4)$ in the first iteration of Algorithm 1. Since the price $p(v_4, B^{(0)})$ is invalid and $lbIndex(v_4) = 0$ initially, we go through the stops in $B^{(0)}$ and get $lbp(v_4) = \frac{dist(v_4,v_1)}{C} = 3$. Finally, we update $lbIndex(v_4) = 1$ which means that we could ignore the stop in $B^{(0)}$ next time when we fetch the lower bound value.

D. Path Refinement

The final path refinement essentially finds a path visiting our selected profitable stops exactly once and makes it satisfy all the regularity constraints (Definition 8). Before the path refinement, we use the Christofides' algorithm to generate a visiting order on the final $B^{(i)}$ (*i.e.*, line 8 of Algorithm 1). Its visiting order starts from some stop, visits each stop exactly once, and goes back to the origin stop. Since we do not have to go back to the origin stop, we discard the longest part which uses the maximum number of intermediate stops in its visiting order. It guarantees that the visiting order uses few intermediate stops. We finally add/delete some terminal stops to match the maximum number K. This final step usually adds stops because we use a strict bound of $2K/3$ in the stopping condition of Algorithm 1 and we discard the longest part.

Algorithm 5 states the basic procedure. In lines 1-4, we use the order to visit all the stops in $B^{(i)}$. We still need to add necessary intermediate stops if the cost between two adjacent stops v_i and v_{i+1} in the order is larger than C (lines 3-4). In lines 5, we make the bus stop number satisfy the constraint.

Example 10: For the final set $B^{(2)} = \{v_1, v_3, v_4\}$, the Christofides' algorithm may give an order (v_1, v_3, v_4). We add one necessary stop v_2 since the adjacent cost between v_1 and v_3 is larger than $C = 4$. Finally, we get $\pi_{r^*} = (v_1, v_2, v_3, v_4)$.

V. Theoretical Analysis

Let $v^{(i)}$ be the most profitable stop in the i-th iteration and $B^{(i)}$ be the set of these stops until $v^{(i)}$, *i.e.*, $\{v^{(j)}|j = 1, \ldots, i\}$. We first prove the correctness of the algorithm.

Algorithm 5: Path Refinement

```
input : An order generated by the Christofides'
        algorithm
output: The new bus route $r^* = (B_{r^*}, \pi_{r^*})$
1 for each two adjacent stops $v_i$ and $v_{i+1}$ in the input
    visiting order do
2     $\pi_i \leftarrow$ the path between $v_i$ and $v_{i+1}$ with the
        minimum number of intermediate stops
3     for each stop in $S_{new} \cup S_{existing}$ along $\pi_i$ do
4         Add it into $B_{r^*}$ and $\pi_{r^*}$ if it is the farest one
            from its previous stop with its cost at most $C$
5 Add or delete terminal stops until $|B_{r^*}| = K$
6 return $r^* = (B_{r^*}, \pi_{r^*})$
```

Theorem 3: Suppose that in the last iteration i, $\sum_{j=1}^{i} p(v^{(j)}, B^{(j-1)}) = \frac{2K}{3}$. The path has at most K nodes.

Proof: Given a set $B^{(i)}$ of points in a plane, the Christofides' algorithm [31] produces a tour with its price less than $\frac{3p(MST(B^{(i)}))}{2}$, where $p(MST(B^{(i)}))$ is the sum of prices of virtual edges in the minimum spanning tree on $B^{(i)}$. Since our tree has its price $\frac{2K}{3} \geq p(MST(B^{(i)}))$, the final path has $\frac{3p(MST(B^{(i)}))}{2} \leq K$ nodes. ■

Note that the resulting path usually has fewer than K nodes since the theoretical guarantee of Christofides' algorithm gives the worst case ratio and we also discard the longest part. We next show the approximation ratio of the EBRR by first proving some inequalities. A monotone submodular function has the following inequality for any B', B:

Lemma 1 ([33]): $U(B') \leq U(B) + \sum_{v \in B' \setminus B} \Delta U_B(v)$.

Let B_{OPT} be the optimal solution set of the stops.

Lemma 2: $U(B^{(i)}) - U(B_{OPT}) \geq (U(B^{(i-1)}) - U(B_{OPT}))(1 - \frac{p(v^{(i)}, B^{(i-1)})}{\sum_{v \in B_{OPT}} p(v, B^{(i-1)})})$.

Proof: By Lemma 1, we have $U(B_{OPT}) \leq U(B^{(i-1)}) + \sum_{v \in B_{OPT} \setminus B^{(i-1)}} \Delta U_{B^{(i-1)}}(v)$. The right hand side $RHS = U(B^{(i-1)}) + \sum_{v \in B_{OPT} \setminus B^{(i-1)}} \frac{\Delta U_{B^{(i-1)}}(v)}{p(v, B^{(i-1)})} p(v^{(i)}, B^{(i-1)})$. Since we choose the profitable node $v^{(i)} = \arg\max_v \frac{\Delta U_{B^{(i-1)}}(v)}{p(v, B^{(i-1)})}$, $U(B_{OPT}) \leq U(B^{(i-1)}) + \frac{\Delta U_{B^{(i-1)}}(v^{(i)})}{p(v^{(i)}, B^{(i-1)})} \sum_{v \in B_{OPT} \setminus B^{(i-1)}} p(v, B^{(i-1)})$. Taking the super set of $B_{OPT} \setminus B^{(i-1)}$, we know that $RHS \leq U(B^{(i-1)}) + \frac{\Delta U_{B^{(i-1)}}(v^{(i)})}{p(v^{(i)}, B^{(i-1)})} \sum_{v \in B_{OPT}} p(v, B^{(i-1)})$. After rearranging the inequality, we prove this lemma. ■

Theorem 4: Algorithm 1 is a $1 - \exp(-\frac{2C}{3 \max_{i,j} dist(v_i, v_j)})$-approximation algorithm. An upper bound of the approximation ratio is $1 - \exp(-\frac{2}{3}) = 0.49$.

Proof: Using Lemma 2 recursively, we get $U(B^{(i)}) \geq (1 - \prod_{j=1}^{i}(1 - \frac{p(v^{(j)}, B^{(j-1)})}{\sum_{v \in B_{OPT}} p(v, B^{(j-1)})}))U(B_{OPT})$. By the inequality $1 - x \leq e^{-x}$ for $x \geq 0$, we have $\frac{U(B^{(i)})}{U(B_{OPT})} \geq 1 - \exp(-\sum_{j=1}^{i} \frac{p(v^{(j)}, B^{(j-1)})}{\sum_{v \in B_{OPT}} p(v, B^{(j-1)})})$. We have $\sum_{v \in B_{OPT}} p(v, B^{(j-1)}) \leq \sum_{v \in B_{OPT}} p(v, B^{(0)}) \leq$

$\frac{K \max_{i,j} dist(v_i,v_j)}{C}$, where the first inequality is because the price will decrease as $B^{(i)}$ is larger and the second inequality is from the definition of the price function. Since our algorithm ensures that the numerator $\sum_{j=1}^{i} p(v^{(j)}, B^{(j-1)}) \geq \frac{2K}{3}$, after replacing the two terms, we get the approximation ratio of $1 - \exp(-\frac{2C}{3 \max_{i,j} dist(v_i,v_j)})$. An upper bound of the approximation ratio is $1 - \exp(-\frac{2}{3}) = 0.49$. We can obtain it by using K as the lower bound of $\sum_{v \in B_{OPT}} p(v, B^{(j-1)})$ and replacing the numerator with $\frac{2K}{3}$. ■

Since $C = 2$ and $\max_{i,j} dist(v_i, v_j) = 80$ in the default settings of the experiments, an lower bound of the approximation ratio is $1 - \exp(-\frac{1}{60}) \approx 0.02$ after we replace them.

We next analyze the time complexity. Its space complexity is the same as the input since no extra structures are involved.

Theorem 5: The EBRR has its time complexity of $\mathcal{O}(|Q|T_1(S_{existing}) + |V|\ln|V| + KT_2(S_{existing})|RQueue|)$, where $T_1(S_{existing})$ and $T_2(S_{existing})$ denote the maximum time of searching a network from each query node until an existing stop and that with an upper bound cost.

Proof: We will analyze the time complexity of each algorithm. For Algorithm 2, the Dijkstra search is fast since it stops at an existing stop soon. The time cost of the search dominates the later computation of the utility values because both the set $RNN(v)$ and the stops in $S_{existing}$ are also processed by the search. Hence, the time complexity of Algorithm 2 is $\mathcal{O}(|Q|T_1(S_{existing}))$.

For Algorithm 3, using a priority queue to store an ordered sequence of the initial utility values $U(v)$ needs $\mathcal{O}(|V|\ln|V|)$. In each iteration of stop selection, the lazy selection evaluates functions for all the nodes in the $RQueue$ in the worst case. Each evaluation searches the network with the upper bound cost $\max_{v_q} dist(v_q, nn(v_q))$ and its time cost $T_2(S_{existing})$ (determined by the maximum cost from a stop to its existing stops). Algorithm 4 uses at most $\mathcal{O}(|RQueue|)$ to compute the lower bound price for each node. Note that though there is a for-loop in Algorithm 4, the amortized time over each profitable stop $v^{(i)}$ is a constant. Since there could be at most $\frac{2K}{3}$ iterations, Algorithm 3 runs in $\mathcal{O}(|V|\ln|V| + KT_2(S_{existing})|RQueue|)$.

The time complexities of the Christofides' algorithm and Algorithm 5 of finding paths between adjacent nodes are $\mathcal{O}(K^2 \log K)$ and $\mathcal{O}(K \max(T_1(S_{existing}), T_2(S_{existing})))$, respectively. They could be ignored since K is a small constant. Overall, the EBRR runs in $\mathcal{O}(|Q|T_1(S_{existing}) + |V|\ln|V| + KT_2(S_{existing})|RQueue|)$. ■

VI. Experimental Study

A. Experiment Setup

1) Implementation: The proposed algorithms were implemented in C++ with the compiler gcc 9.4.0 and were performed in a machine with 2.66GHz CPU and 48GB RAM under the CentOS 7 Linux distribution. We used Python 3 to clean the data and Mapbox API [34] for visualization.

2) Datasets: The problem requires the network, transit, and query data in a city. We collected them for three cities: Chicago, New York City (NYC), and Orlando. The detailed statistics of the datasets are shown in Table II. All the network data of the three cities are from DIMACS [35]. The transit data include the information of existing bus stops. The Chicago, NYC, and Orlando transit data are from Chicago Transit Authority [36], Metropolitan Transportation Authority [37], and Lynx [38], respectively. The past workloads of transit routing queries for Chicago and NYC use the same sources as in [10], and those for Orlando are extracted from Uber Movement [27]. We collected their pick-up and drop-off locations to make the multiset Q. An overview of the road network and transit stops is shown in Figure 6. Red lines are the edges representing road segments and blue icons are the existing stops.

TABLE II: Real datasets for three cities.

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert S_{new}\rvert$	$\lvert S_{existing}\rvert$	$\lvert Q\rvert$
Chicago	58,337	178,102	89,051	10,517	1,076,324
NYC	134,551	397,956	198,978	9,225	793,496
Orlando	95,678	238,674	119,337	3,949	136,813

3) Compared Algorithms: Following previous work [10], we compared two state-of-the-art baselines below. We could directly use their results since they finally generate new bus routes with K stops. Note that their routes could violate the constraint of C because their problems do not require it.

(1) ETA-Pre [10]. It aims to find the bus route that maximizes a linear objective function. It mainly wants to match as many trajectories (corresponding to the demand) as possible and to maximize the natural connectivity [39] of the whole transit network. It first generates a set of candidate paths and compares their objective values by using the matrix method as an estimation.

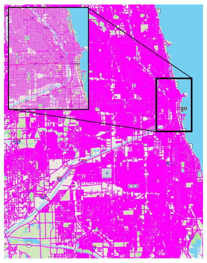
(2) vk-TSP [21]. It defines a distance measurement between two paths and tries to find the route which minimizes such distances from all trajectories. It uses the greedy idea to append new edges shown in many trajectories into the route.

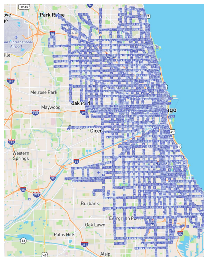
Note that since their preprocessing steps could run in at least 4 hours long, we directly use their preprocessing results.

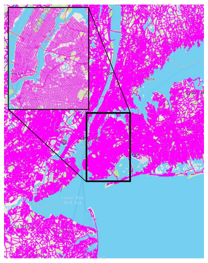
We assess their performance in terms of the walking cost, the connectivity, and the execution time. We vary the maximum number K of stops, the maximum cost between two adjacent stops C, and the multisets Q of query nodes to test the performance. We also conduct real case studies and a survey.

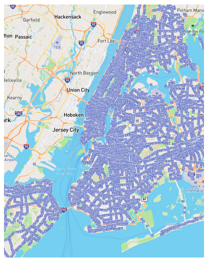
B. Experiment Results

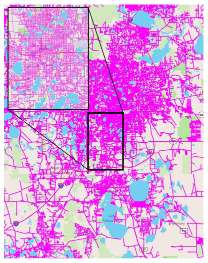
1) Effectiveness: We first tested the effectiveness by the quantitative study on the walking costs (after including the new stops) and the connectivity, *i.e.*, $Walk(S_{existing} \cup B_{r^*})$ and $Connect(B_{r^*})$. Since the other two baselines do not impose the constraint of C, we only vary the maximum number K from 10 to 50 and the set Q of query nodes in the experiments about effectiveness. All the costs represent the distances in kilometers, and our EBRR uses $C = 2$. They can be transformed to the travel time or other costs according to user preferences. For Chicago, we divide the whole query nodes into four parts of similar size along the vertical direction. For NYC, we use four sets of query nodes from four boroughs: Brooklyn, Manhattan, Queens, and Bronx.


(a) Chicago Roads


(b) Chicago Stops


(c) NYC Roads


(d) NYC Stops


(e) Orlando Roads

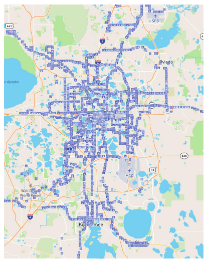
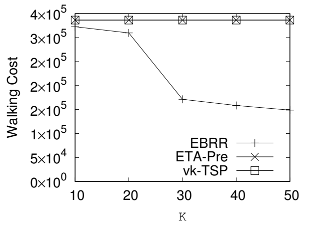
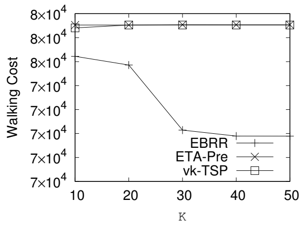

(f) Orlando Stops

Fig. 6: An overview of road networks and existing bus stops

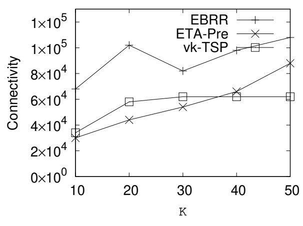


(a) Results on Chicago datasets

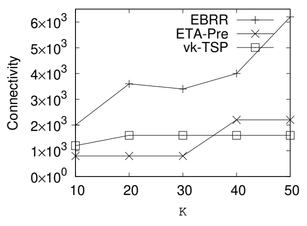


(b) Results on NYC datasets

Fig. 7: Walking cost of varying K



(a) Results on Chicago datasets



(b) Results on NYC datasets

Fig. 8: Connectivity of varying K

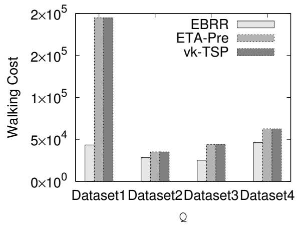
Effect of K. The results of varying the maximum number K of stops are shown in Figure 7 and Figure 8. The multiset Q uses all the query nodes spanning over the entire city as in [10]. Note that we choose the parameter α in the utility function (Eq. 1) as 2000 and 200 in the Chicago and NYC datasets, respectively, and fine-tuning it with other values may give even higher utility values.

For the walking cost, in both two sub-figures of Figure 7, our *EBRR* achieves smaller costs than *ETA-Pre* and *vk-TSP* and will help passengers access the transit network easily. Since we adopt the path's cost, our new route could place stops in the real demand centers. Our *EBRR* decreases the cost as K is larger because there are more stops in the solution. *ETA-Pre* and *vk-TSP* barely optimize the walking cost, which makes their values nearly flat. They pass through many demand centers where there are already many existing stops. The walking costs in Figure 7a are longer than those in Figure 7b because there are more queries in the Chicago dataset than those in the NYC dataset.

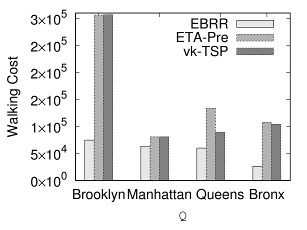
For the connectivity, in Figure 8, our *EBRR* gives higher values than those of *ETA-Pre* and *vk-TSP*, which will facilitate passengers' transfer since a higher value means more transfer choices. It could be observed there is a drop when K is equal to 30. This is because *EBRR* finds more profits on some distant nodes which decrease the walking cost, which is consistent with the drop when K is 30 in Figure 7. These nodes could be pruned when K is small but are reconsidered in the node selection when K is larger.

Effect of Q. Figure 9 and Figure 10 present the results of varying Q. The maximum number K of stops is 30, and α is 2000 and 10000 in Chicago and NYC data, respectively.

In Figure 9, it can be observed that *EBRR* achieves the minimum costs among all algorithms for all datasets. In some



(a) Results on Chicago datasets



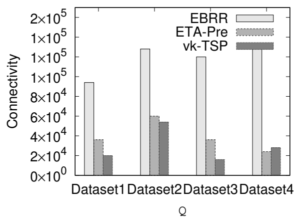
(b) Results on NYC datasets

Fig. 9: Walking cost of varying Q

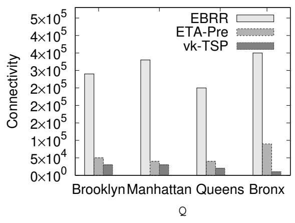
cases, such as Dataset1 in Chicago and the Brooklyn one, the walking cost of *EBRR* is four times shorter than the other two. For the other cases where *EBRR* reduces less cost, it could have higher connectivity values, as shown in Figure 10. The other two baselines optimize the walking cost less.

For the connectivity, shown in Figure 10, we can still find that *EBRR* performs better than the other baselines since it has the highest connectivity values on all datasets. For the dataset in Queens, NYC, the connectivity value of *EBRR* could be 6 times higher than those of the other two algorithms.

Approximation Ratio. Since our EBR problem is NP-hard, we could only compare it with the optimal solution (OPT for short) on a small dataset. From the NYC data, we extract a small graph with 110 nodes and 324 edges, 132 query nodes, 7 new and 7 existing stops. OPT is derived through an exhaustive search on all combinations of the stops. In Figure 11a, we directly report the utility values of the two solutions when varying K. It can be observed that *EBRR* achieves a smaller utility value than OPT for each K. However, the approximation ratio of *EBRR* over OPT is close to 1, which also indicates a more competitive performance than the theory suggests in practice.

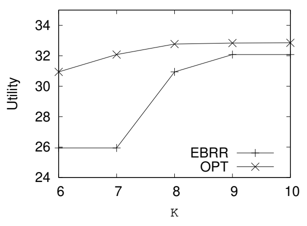


(a) Results on Chicago datasets

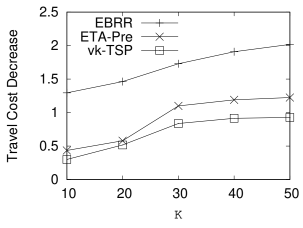


(b) Results on NYC datasets

Fig. 10: Connectivity of varying Q



(a) Comparison with OPT



(b) Travel cost decrease

Fig. 11: Other effectiveness metrics

Travel Cost. We also tested the travel cost (i.e., the cost of the entire trip) on Chicago datasets. It includes the walking cost, the cost of using the public transit, and the transfer cost between bus routes, in terms of minutes, shown in Figure 11b. By considering the travel cost of each query, we reported the average difference between the two travel costs before and after the new bus route is incorporated into the transit system. When K is getting larger, it can be found that all algorithms reduce the travel cost more since more bus stops should fulfill more demand queries. However, when K is 40 or 50, the travel cost decrease reaches a plateau, which is because 40 stops are sufficient to cover most areas of high demands. Note that a bus route of 40 stops could be 80 km long since $C = 2$. We can also observe that *EBRR* always achieves the largest decrease of travel cost among the three algorithms, which suggests that *EBRR* could satisfy more demand queries.

Case Study. We conducted two case studies, one in Orlando shown in Figure 1 and the other in Chicago shown in Figure 12. For Orlando, we generated the query multiset Q by using the ridership data from the Lynx Bus Service in Orlando [11]. The ideal route in cyan blue in Figure 1 can be obtained by running *EBRR* with parameters $\alpha = 100$, $K = 7$, and $C = 2$. Since we use the accurate network cost, the route passes through all demand centers shown by red areas and some existing stops as transfer choices. People would prefer using public transit more since they can easily access it.

In Figure 12, we considered the query nodes across the whole city of Chicago. Similarly, the route in light green is obtained by *EBRR* with $\alpha = 2000$, $K = 30$, and $C = 2$. *ETA-Pre* and *vk-TSP* use purple and orrange lines. Though the other two algorithms pass through some red areas along the coastline, there are so many existing bus stops (shown by small blue dots) along the routes, which makes them satisfy less demand. However, EBRR not only tries to match the uncovered demand in new areas around the airport (in the top-

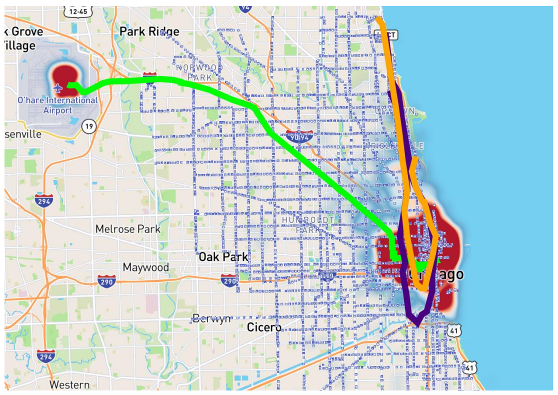
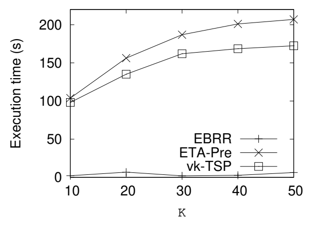
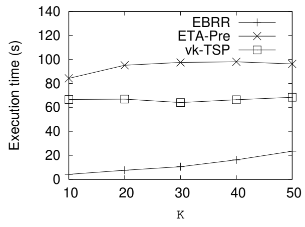


Fig. 12: Case study in Chicago



(a) Results on Chicago datasets



(b) Results on NYC datasets

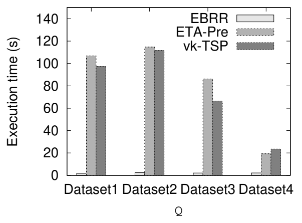
Fig. 13: Execution time of varying K

left region) but also spans over the existing network. Besides, we conducted a survey by showing the routes in Figure 12 and questioning real users from Prolific [40] and collected 96 responses. The results showed that the number of users voting *EBRR* as the best route is the greatest.

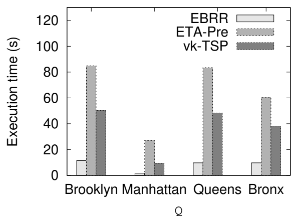
2) Efficiency: We evaluated our algorithm's efficiency by varying all the parameters. The results of varying the maximum number K of stops and the multiset Q of query nodes were obtained from the same experiments as in Section VI-B1. Since the other two baselines do not require the maximum cost between two adjacent stops C and the balancing parameter α, we only reported the time cost of *EBRR* by varying $C = [1, \underline{2}, 3, 4, 5]$ and $\alpha = [1000, 2000, \underline{3000}, 4000, 5000]$. The range of C from 1km to 5km is because the costs between two adjacent stops in the datasets fall into it. The range of α from 1k to 5k corresponds to a change from focusing on the walking costs to connectivity. The default settings are marked in bold and use $K = 30$ and the whole query nodes.

Effect of K. The results of varying the maximum number K of stops are shown in Figure 13. Our *EBRR* takes around 10 seconds to plan a new bus route, which is faster than the other two baselines. The execution time is basically larger as K increases since we consider more nodes. Besides, the time cost on the final path refinement is greater when there are more nodes, but it could be ignored since a bus route in a real scenario usually has few stops.

Effect of Q. Figure 14 plots the results of varying the multiset Q of query nodes. The superiority of *EBRR* can be easily seen from both the Chicago and NYC datasets. The time cost of *EBRR* is negligible, but the other two algorithms take minutes long. The execution time could be even longer when



(a) Results on Chicago datasets

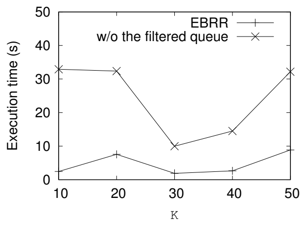


(b) Results on NYC datasets

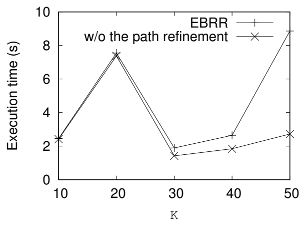
Fig. 14: Execution time of varying Q

TABLE III: Execution time (s) of *EBRR* of varying C (km)

Datasets	$C = 1$	$C = 2$	$C = 3$	$C = 4$	$C = 5$
Chicago	4.51	2.10	3.54	3.59	3.64
NYC	4.73	7.43	7.06	9.62	12.4
Orlando	1.50	1.67	2.15	2.43	4.48

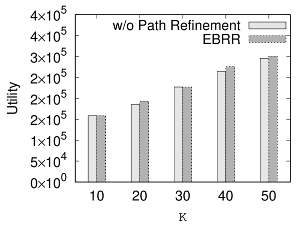


(a) Execution time of varying K

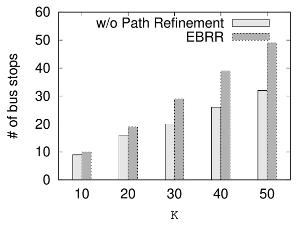


(b) Execution time of varying K

Fig. 15: Ablation study on Chicago datasets



(a) Utility of varying K



(b) # of bus stops of varying K

Fig. 16: Ablation study about the path refinement

they preprocess the other data by some matrix operations.

Effect of C. Table III presents the time cost of varying the maximum cost between two adjacent stops C. Basically, the execution time increases as C is larger on all datasets because more potential stops satisfy the constraints and are taken into consideration. It could be observed that the time cost for NYC is higher than that of the other two cities, which is consistent with the largest size of NYC datasets. For all settings, it just takes at most 20 seconds for *EBRR* to plan a bus route, which is efficient in practice.

Effect of α. Table IV shows the execution time of varying α. When α is larger, the results include more existing stops with more transfer choices. It can be seen that the time cost is insensitive to α. *EBRR* still runs fast in 10 seconds.

Ablation Study. To learn the performance of our proposed speed-up techniques, we did an ablation study. Specifically, we considered five variants: the vanilla one which finds the node with the maximum utility by enumerating all nodes, *EBRR* using the real cost instead of the lower bound price

TABLE IV: Execution time (s) of *EBRR* of varying α

Datasets	$\alpha = 1k$	$\alpha = 2k$	$\alpha = 3k$	$\alpha = 4k$	$\alpha = 5k$
Chicago	1.62	1.87	2.10	3.65	5.48
NYC	7.39	7.35	7.43	7.37	7.40
Orlando	2.95	1.68	1.67	1.78	1.75

(Algorithm 4), *EBRR* without the filtered queue, *EBRR* without the path refinement, and *EBRR*. Since the first two variants need at least one hour long, we only reported the results of the last three algorithms by varying the maximum number K in Figure 16. In Figure 15a, it can be seen that *EBRR* runs faster than its variant without the filtered queue for each K, which further indicates that *EBRR* does benefit from using the filtered queue. Since path refinement is the final step of *EBRR*, it can be observed in Figure 15b that *EBRR* runs a bit slower than its variant without the path refinement. However, path refinement is important for *EBRR* because it increases the utility values of *EBRR* (shown in Figure 16a) and the number of bus stops (shown in Figure 16b).

C. Summary

We summarize our findings as follows.

(i) Our proposed *EBRR* outperforms the other two baselines by achieving up to a 50% reduction on the walking cost and offering transfer choices twice as many as the others.

(ii) EBRR is efficient for the large network and query data of a city. It runs in 20 seconds without preprocessing data, whereas the baselines need more than one minute with their preprocessing times of at least 4 hours.

(iii) The proposed speed-up techniques could save the execution time by at least one hour long.

VII. Conclusion

This paper studies the problem of planning a bus route on road networks. Specifically, we propose the Bus Routing on Roads problem which takes into consideration two common goals of matching the demand and connecting the transit network. Noticing the efficiency issue of previous work, we designed the algorithm named Efficient Bus Routing on Roads (EBRR). It could plan a new bus route for large road networks in around 10 seconds while using the accurate cost measurement. We also propose speed-up techniques such as the filtered queue and the lower bound price. We show its approximation ratio and time complexity and empirically compare it with two state-of-the-art solutions. Experimental results demonstrate the superiority of our EBRR. Our EBRR could run 60x faster than the baselines while achieving higher utility values. It also does not preprocess data, whereas other baselines spend at least 4 hours. For future work, since other factors may be considered in reality, one may study the post-processing solutions when considering our results as the first-stage output.

Acknowledgements

We are grateful to the anonymous reviewers for their constructive comments on this paper. The research of Libin Wang and Raymond Chi-Wing Wong is supported by PRP/026/21FX.

References

[1] "New bus routes in nyc," 2022, https://patch.com/new-york/bayside/new-routes-faster-buses-could-come-bayside-mta-new-bus-plan.

[2] X. Liu, J. Wu, J. Huang, J. Zhang, B. Y. Chen, and A. Chen, "Spatial-interaction network analysis of built environmental influence on daily public transport demand," *Journal of Transport Geography*, vol. 92, p. 102991, 2021.

[3] D. Toro-González, V. Cantillo, and V. Cantillo-García, "Factors influencing demand for public transport in colombia," *Research in Transportation Business & Management*, vol. 36, p. 100514, 2020.

[4] O. Y. Xian, M. A. Chitre, and D. Rus, "An approximate bus route planning algorithm," in *CIVTS*, 2013.

[5] F. Bastani, X. Xie, Y. Huang, and J. W. Powell, "A greener transportation mode: flexible routes discovery from GPS trajectory data," in *SIGSPATIAL*, 2011.

[6] C. Chen, D. Zhang, N. Li, and Z. Zhou, "B-planner: Planning bidirectional night bus routes using large-scale taxi GPS traces," *IEEE Trans. Intell. Transp. Syst.*, vol. 15, no. 4, pp. 1451–1465, 2014.

[7] S. Wang, W. Lin, Y. Yang, X. Xiao, and S. Zhou, "Efficient route planning on public transportation networks: A labelling approach," in *SIGMOD*, 2015.

[8] N. S. Hadjidimitriou, M. Lippi, and M. Mamei, "A data driven approach to match demand and supply for public transport planning," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 10, pp. 6384–6394, 2021.

[9] C. Ma, C. Wang, and X. Xu, "A multi-objective robust optimization model for customized bus routes," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 4, pp. 2359–2370, 2021.

[10] S. Wang, Y. Sun, C. Musco, and Z. Bao, "Public transport planning: When transit network connectivity meets commuting demand," in *SIGMOD*, 2021.

[11] "Lynx bus service," 2022, https://lynx.maps.arcgis.com/home/item.html?id=63d81441eb804d72be4fe761ea550847.

[12] "Lake nona in orlando," 2021, https://www.getbellhops.com/blog/best-up-and-coming-neighborhoods-in-orlando/.

[13] C. Sung, X. Yang, C. Liao, and W. Liu, "Introute: An integer programming based approach for best bus route discovery," in *DASFAA*, 2021.

[14] Y. Lyu, C.-Y. Chow, V. C. Lee, J. K. Ng, Y. Li, and J. Zeng, "Cb-planner: A bus line planning framework for customized bus systems," *Transportation Research Part C: Emerging Technologies*, vol. 101, pp. 233–253, 2019.

[15] Z. Yang, B. Yu, and C. Cheng, "A parallel ant colony algorithm for bus network optimization," *Comput. Aided Civ. Infrastructure Eng.*, vol. 22, no. 1, pp. 44–55, 2007.

[16] V. Guihaire and J.-K. Hao, "Transit network design and scheduling: A global review," *Transportation Research Part A: Policy and Practice*, vol. 42, no. 10, pp. 1251–1273, 2008.

[17] Y. Liu, C. Liu, N. J. Yuan, L. Duan, Y. Fu, H. Xiong, S. Xu, and J. Wu, "Exploiting heterogeneous human mobility patterns for intelligent bus routing," in *ICDM*, 2014.

[18] F. Pinelli, R. Nair, F. Calabrese, M. Berlingerio, G. D. Lorenzo, and M. L. Sbodio, "Data-driven transit network design from mobile phone trajectories," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 6, pp. 1724–1733, 2016.

[19] Y. Lyu, C. Chow, V. C. S. Lee, Y. Li, and J. Zeng, "T2CBS: mining taxi trajectories for customized bus systems," in *INFOCOM Workshops*, 2016.

[20] X. Kong, M. Li, T. Tang, K. Tian, L. Moreira-Matias, and F. Xia, "Shared subway shuttle bus route planning based on transport data analytics," *IEEE Trans Autom. Sci. Eng.*, vol. 15, no. 4, pp. 1507–1520, 2018.

[21] S. Wang, Z. Bao, J. S. Culpepper, T. Sellis, and X. Qin, "Fast large-scale trajectory clustering," *Proc. VLDB Endow.*, vol. 13, no. 1, pp. 29–42, 2019.

[22] W. Y. Szeto and Y. Wu, "A simultaneous bus route design and frequency setting problem for tin shui wai, hong kong," *Eur. J. Oper. Res.*, vol. 209, no. 2, pp. 141–155, 2011.

[23] D. V. Oudheusden, S. Ranjithan, and K. Singh, "The design of bus route systems—an interactive location-allocation approach," *Transportation*, vol. 14, no. 3, pp. 253–270, 1987.

[24] K. N. Singh, "The uncapacitated facility location problem: some applications in scheduling and routing," *Int. J. Oper. Res.*, vol. 5, no. 1, pp. 36–43, 2008.

[25] X. Chen, X. Cao, Y. Zeng, Y. Fang, S. Wang, X. Lin, and L. Feng, "Constrained path search with submodular function maximization," in *ICDE*, 2022.

[26] "Google maps," 2022, https://www.google.com/maps/dir/.

[27] "Uber movement," 2022, https://movement.uber.com/?lang=en-US.

[28] "The cost of building a bus stop," 2018, https://essexlibdems.org.uk/en/article/2018/1286942/why-do-essex-cc-bus-shelters-cost-13-000.

[29] "Miles of new bus lanes in england," 2021, https://www.bbc.com/news/business-56395526.

[30] N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz, "Submodular maximization with cardinality constraints," in *SODA*, 2014.

[31] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, Tech. Rep., 1976.

[32] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. M. VanBriesen, and N. S. Glance, "Cost-effective outbreak detection in networks," in *SIGKDD*, 2007.

[33] M. Sviridenko, "A note on maximizing a submodular set function subject to a knapsack constraint," *Oper. Res. Lett.*, vol. 32, no. 1, pp. 41–43, 2004.

[34] "Mapbox," 2022, https://www.mapbox.com/.

[35] "Dimacs," 2022, http://www.dis.uniroma1.it/challenge9/download.shtml.

[36] "Chicago transit authority," 2022, https://catalog.data.gov/dataset/cta-bus-stops.

[37] "Metropolitan transportation authority," 2022, https://github.com/miranda-adams/NYC-bus-stops-by-route.

[38] "Lynx," 2022, http://www.golynx.com/maps-schedules/data-download.stml.

[39] C. Chen, R. Peng, L. Ying, and H. Tong, "Network connectivity optimization: Fundamental limits and effective algorithms," in *SIGKDD*, 2018.

[40] "Prolific academia," 2022, https://www.prolific.co/.