Expanding Reverse Nearest Neighbors

Wentao Li
The Hong Kong University of Science and Technology (Guangzhou)
wentaoli@hkust-gz.edu.cn

Maolin Cai
Chongqing University
caimaolin@cqu.edu.cn

Min Gao
Chongqing University
gaomin@cqu.edu.cn

Dong Wen
The University of New South Wales
dong.wen@unsw.edu.au

Lu Qin
AAII, FEIT, University of Technology Sydney
lu.qin@uts.edu.au

Wei Wang*
The Hong Kong University of Science and Technology (Guangzhou)
The Hong Kong University of Science and Technology
weiwcs@ust.hk

ABSTRACT

In a graph, the reverse nearest neighbors (RNN) of vertex f refer to the set of vertices that consider f as their nearest neighbor. When f represents a facility like a subway station, its RNN comprises potential users who prefer the nearest facility. In practice, there may be *underutilized* facilities with small RNN sizes, and relocating these facilities to expand their service can be costly or infeasible. A more cost-effective approach involves selectively upgrading some edges (e.g., reducing their weights) to expand the RNN sizes of underutilized facilities. This motivates our research on the **Expanding Reverse Nearest Neighbors** (ERNN) problem, which aims to maximize the RNN size of a target facility by upgrading a limited number of edges. Solving the ERNN problem allows underutilized facilities to serve more users and alleviate the burden on other facilities. Despite numerous potential applications, ERNN is hard to solve: It can be proven to be NP-hard and APX-hard, and it exhibits non-monotonic and non-submodular properties. To overcome these challenges, we propose novel greedy algorithms that improve efficiency by minimizing the number of edges that need to be processed and the cost of processing each edge. Experimental results demonstrate that the proposed algorithms achieve orders of magnitude speedup compared to the standard greedy algorithm while greatly expanding the RNN.

PVLDB Reference Format:
Wentao Li, Maolin Cai, Min Gao, Dong Wen, Lu Qin, and Wei Wang. Expanding Reverse Nearest Neighbors. PVLDB, 17(4): 630 - 642, 2023.
doi:10.14778/3636218.3636220

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/wentaoli-92/ERNN.

*Wei Wang is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 17, No. 4 ISSN 2150-8097.
doi:10.14778/3636218.3636220

1 INTRODUCTION

Graphs are widely used to represent entities and their relationships [9]. This paper focuses on road networks [27], which are a type of weighted graph. In a road network [20], entities located at specific positions are represented as vertices, while the roads between entities are represented as edges, with the travel time along the roads represented as edge weights. In applications such as decision support [1], the entities (i.e., the vertices) in the road network are usually divided into two categories: **users** C and **facilities** F.

For many applications, each user vertex $c \in C$ typically favors selecting the nearest facility vertex to access the desired services [24]. For a facility vertex $f \in F$, when we collect all user vertices that consider f as their nearest facility, we obtain facility f's **(bichromatic) reverse nearest neighbors** (RNN) [36]. In this context, the RNN for facility f includes all potential users who consider f as their preferred option for accessing services [11, 28]. The concept of RNN has many applications in different fields. For instance, in the business domain, analyzing the RNN of facilities like restaurants can help identify potential customers for targeted marketing campaigns [5]. Similarly, in transportation planning, carefully examining the RNN of transportation hubs can assist in determining suitable locations for establishing new stations [19].

Motivation. The size of facility f's RNN quantifies, to some extent, the degree to which it is utilized, as its RNN includes users that tend to select that facility. Indeed, the concept of RNN was initially used to quantify the scope of influence of a facility (i.e., its degree of usage) [11]. However, it is observed that not all facilities are fully utilized, often due to factors such as remote locations, resulting in small RNN sizes for these **underutilized facilities** [6, 19, 34]. This leads to resource wastage, as underutilized facilities fail to provide services to their full potential, resulting in additional burdens on other facilities. One way to improve the functionality of underutilized facilities is to relocate them to new locations. Yet, relocating facilities like fire stations is costly and may even be infeasible.

An alternative and cost-effective approach to enhance facility utilization involves selecting a limited number (i.e., a **budget**) of roads for upgrades to decrease their weights [25]. The road upgrade can be achieved by constructing express lanes or widening existing roads [3], which will effectively reduce the edge weights associated with these roads. While there has been extensive research on upgrading roads to minimize network diameter or delay [3, 22], no

studies have explored the specific goal of improving facility utilization. This motivates us to investigate the **Expanding Reverse Nearest Neighbors** (ERNN) problem in the road network: given a graph G, a target facility f, and a budget b, our objective is to maximize the size of the RNN of the target facility f by selecting (up to) b edges to reduce their weights for upgrading.

Some applications of the ERNN problem are listed below.

- *Public Transportation Planning.* Due to environmental and economic considerations, public transportation facilities such as the subway are increasingly used in daily life [18]. However, some subway stations have small RNN sizes, resulting in underutilization. Choosing new locations for these stations or constructing new ones would incur high costs and may even be infeasible [33]. To facilitate users' travel, a viable solution is to solve the ERNN problem and upgrade certain roads, thereby expanding the RNN of these stations. This method improves the utilization of the underutilized stations and relieves the burden on other stations.
- *Emergency Response Planning.* Emergency response facilities such as hospitals or fire departments are crucial for ensuring the safety of people's lives and properties [16]. However, there exist facilities whose RNN sizes are small, and they cannot be fully utilized during emergency events, resulting in wasted resources. To enhance the utilization of such facilities, one approach is to solve the ERNN problem and select certain edges (i.e., roads) for upgrades, thereby increasing the RNN size of the facility. This approach not only avoids the immense cost of relocating these underutilized facilities or constructing new facilities but also enables timely emergency response by reducing the travel time through the upgraded roads.

Challenges. Although the ERNN problem offers many attractive applications, solving it is challenging. Specifically, given a budget b, it is necessary to examine all edge combinations in the graph G that contain no more than b edges. For each edge combination, we need to determine the increase in the size of the target facility's RNN when upgrading the edges, thereby finding the edge combination that yields the maximum RNN size for the target facility vertex. For large-scale graphs, thoroughly checking all possible edge combinations is very time-consuming. In fact, we can prove that ERNN is NP-hard (see Theorem 3.5), indicating that finding an exact solution is intractable. One may consider using approximation algorithms to find solutions with an acceptable approximation ratio. However, we can prove that ERNN is APX-hard (see Theorem 3.6), which means seeking solutions with guaranteed approximation values in polynomial time is challenging. Furthermore, the ERNN problem lacks monotonicity and submodularity (see Theorem 3.7), undoubtedly posing great difficulties in solving the problem.

Our Solution. Given the challenges of the ERNN problem, we employ greedy algorithms to solve it practically. For a given budget b, the basic idea of the greedy algorithm is to select, in each round, the edge that brings the maximum RNN size for upgrading until the budget is exhausted (i.e., performing b rounds). By iteratively selecting the optimal edge in each round, the greedy algorithm eliminates the need to consider numerous edge combinations, thereby improving efficiency. However, directly applying a standard greedy algorithm to solve ERNN is still time-consuming due to the following reasons: (1) Each round requires considering a large number of edges to select the one that leads to the maximum RNN size. (2) For each considered edge, the updated size of the target facility's RNN needs to be calculated when the edge weight decreases.

To accelerate the solving of ERNN, we propose new techniques to fix the two shortcomings of the standard greedy algorithm.

- *Distance-Based Edge Inspection.* The standard greedy algorithm considers all edges in each round to find the edge that yields the maximum RNN size for the target facility. To improve the speed of the greedy algorithm, it becomes crucial to reduce the number of edges to be considered. To achieve this, we propose a distance-based edge inspection technique. This technique sorts the edges according to the shortest distance (from their endpoints) to the target facility and utilizes the distance to calculate an upper bound on the RNN size after upgrading the edge. We observe that as the distance increases, the upper bound gradually decreases. Therefore, when checking the edges in ascending order of distance, if at some point the upper bound of an edge is not greater than the already computed RNN size, subsequent edges (with the same or larger distances) do not need to be checked, thereby reducing the number of edges to be considered. Building upon this technique, we further propose two powerful pruning strategies to discard more invalid and dominated edges.
- *Incremental* RNN *Computation.* For each edge that needs to be examined, we need to calculate the updated RNN resulting from upgrading that edge. It is costly to recalculate the RNN of the target facility from scratch. Instead, we observe that the RNN of the target facility only undergoes a small change when the weight of an edge changes. Therefore, we propose an incremental RNN computation technique to avoid recomputing the RNN.

By integrating the two techniques mentioned above, we obtain our optimized greedy algorithms for solving the ERNN problem.

Contributions. The main contributions of the paper are as follows:

- *New Problem Formulation (Section 3).* We propose a new problem called ERNN, which involves selecting a limited number of edges from a graph for upgrading in order to maximize the RNN size of a target facility vertex. By analyzing the NP-hardness and APX-hardness of the ERNN problem, we uncover the challenges in solving it. Furthermore, we demonstrate that the problem lacks monotonicity and submodularity, which inspires us to use greedy algorithms for the practical solving of the problem.
- *New Algorithm Design (Section 4).* We introduce a standard greedy algorithm for solving the ERNN problem and analyze its limitations to identify opportunities for enhancing its performance. (1) To tackle the challenge of examining a large number of edges, we propose a distance-based edge inspection technique that utilizes upper bounds to enable early termination of edge checks. Additionally, we seamlessly incorporate two pruning strategies to further reduce the number of edges requiring inspection. (2) To mitigate the computational overhead associated with recalculating the RNN for the target facility when an edge weight decreases, we propose an incremental RNN computation technique that avoids computing the RNN from scratch.
- *Extensive Experimental Analysis (Section 5).* We conducted extensive experiments on real-world road networks. The experimental results show that the optimized greedy algorithms can be three

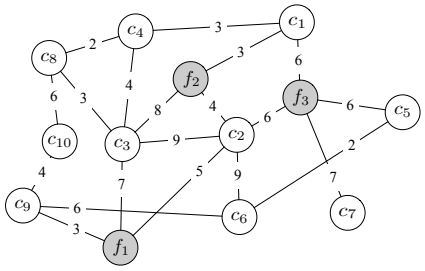


Figure 1: The Example Graph G

orders of magnitude faster than the standard greedy algorithm and can solve the ERNN problem on graphs with over 1 million edges in less than one minute, demonstrating the effectiveness of our algorithms. Additionally, we compared our algorithms with an exact algorithm and found that the updated RNN size caused by our algorithms is comparable, indicating that our approaches can effectively solve the ERNN problem. These results indicate that the proposed algorithms can efficiently and effectively solve the ERNN problem.

Due to space limitations, some proofs are omitted and can be found in the technical report [13].

2 PRELIMINARY

We introduce some notations in Section 2.1, and then we explain how to compute RNN in Section 2.2.

2.1 Notations

Given a road network $G(V, E, W)$, which consists of a vertex set V and an edge set E. For every vertex $v \in V$ of G, the **neighbors** of v, denoted $N_G(v)$, are the set of vertices adjacent to v, i.e., $N_G(v) = \{u \in V | (u, v) \in E\}$. For every edge $e = (u, v) \in E$, a weight function W assigns a **weight** $w(e)$ to e, and u and v are referred to as the **endpoints** of e. A path in the graph G from $s = v_0$ to $t = v_l$ is a sequence of edges, denoted as $p_G(s, t) = \{(v_0, v_1), (v_1, v_2), \cdots, (v_{l-1}, v_l)\}$, where $(v_i, v_{i+1}) \in E$ for $i \in [0, l-1]$. The length of a path $p_G(s, t)$ is defined as $len_G(p_G(s, t)) = \Sigma_{e \in p_G(s,t)} w(e)$. Given two vertices s and t in G, the path $p_G(s, t)$ with the minimum length between them is defined as the **shortest path**, and its length is defined as the **shortest distance** $dist_G(s, t) = len_G(p_G(s, t))$.

Example 2.1. Fig. 1 shows the graph G, which consists of 13 vertices and 20 edges. For the vertex $f_1 \in V$, its neighbors are $N_G(f_1) = \{c_2, c_3, c_9\}$. For the edge $e = (f_1, c_2) \in E$, its weight is $w(e) = 5$. For f_1 and f_3, a path between them is $p_G(f_1, f_3) = \{(f_1, c_2), (c_2, f_3)\}$, and the length of this path is 11. Among all paths between f_1 and f_3, the length of $p_G(f_1, f_3)$ is the shortest, so it is also the shortest path between f_1 and f_3, and the shortest distance between f_1 and f_3 is $dist_G(f_1, f_3) = 11$.

The vertices V of the road network G can be divided into two disjoint sets, namely users C and facilities F. For a user vertex $c \in C$, we define the **nearest neighbor** of c, denoted by $\mathsf{NN}_G(c)$, as the facility vertex in F that is closest to c, i.e., $\mathsf{NN}_G(c) = f$, when $dist_G(c, f) \leq dist_G(c, p)$, for $\forall p \in F$. Conversely, for a facility vertex $f \in F$, we define the **reverse nearest neighbors** of f, denoted by $\mathsf{RNN}_G(f)$, as the collection of user vertices in C that consider f as their nearest neighbor, i.e., $\mathsf{RNN}_G(f) = \{c \in C | \mathsf{NN}_G(c) = f\}$.

Example 2.2. Consider the graph G in Fig. 1, where the 13 vertices of G are divided into two categories: users $C = \{c_1, c_2, \ldots, c_{10}\}$ and facilities $F = \{f_1, f_2, f_3\}$. For the user $c_1 \in C$, the distances to all three facilities in F are as follows: $dist_G(c_1, f_1) = 14$, $dist_G(c_1, f_2) = 3$, $dist(c_1, f_3) = 6$, thus $\mathsf{NN}_G(c_1) = f_2$. Similarly, we can see that $\mathsf{NN}_G(c_9) = f_1$. By collecting all vertices in C that have f_1 as their nearest neighbor, we find that $\mathsf{RNN}_G(f_1) = \{c_3, c_9, c_{10}\}$. Similarly, we have $\mathsf{RNN}_G(f_2) = \{c_1, c_2, c_4, c_8\}$ and $\mathsf{RNN}_G(f_3) = \{c_5, c_6, c_7\}$.

Remark. We assume that the graph is an undirected connected graph, and other more general extensions will be explored in future work. For simplicity, we assume that the nearest neighbor of each $c \in C$ is unique. When the context is clear, we will omit the subscript notation G, so $N_G(v), P_G(s, t), dist_G(s, t), \mathsf{NN}_G(c)$, and $\mathsf{RNN}_G(f)$ will be simplified to $N(v), P(s, t), dist(s, t), \mathsf{NN}(c)$, and $\mathsf{RNN}(f)$.

2.2 RNN Computation

This paper aims to expand the RNN of the target facility vertex by selecting certain edges to upgrade. For this purpose, it is necessary to compute the RNN of the facility vertex before and after the edge upgrades. In this section, we will introduce the process of computing RNN. Then, we will formally define the problem under study in Section 3, and discuss how the computed RNN can be used to select edges to solve the proposed problem in Section 4.

To calculate the RNN of a facility vertex f, one approach is as follows: (1) For each user vertex $c \in C$ in the graph, execute Dijkstra's algorithm to visit the vertices in order of ascending distance. When encountering the first facility vertex, that facility becomes the nearest neighbor $\mathsf{NN}(c)$ of c; (2) Traverse all user vertices $c \in C$, and if $\mathsf{NN}(c) = f$, add c to the $\mathsf{RNN}(f)$ of f. Although this approach can correctly compute the RNN of f, it requires executing Dijkstra's algorithm for all user vertices C to obtain their nearest neighbors. Thus, the time cost is $O(|C|(|V| \log |V| + |E|))$. Given the large number of user vertices $|C|$ in G, this approach is too costly.

To accelerate the computation, we adopt the method proposed by Yiu et al. [36]. The basic idea of Yiu's algorithm is to have each vertex store its own nearest neighbor and propagate this information to its neighbors. This way, only one execution of Dijkstra's algorithm is needed to calculate the nearest neighbors of all vertices. The specific procedure of Yiu's algorithm is shown in Algorithm 1.

Algorithm. Given a graph $G(V = F \cup C, E, W)$ and a facility vertex $f \in F$, we aim to compute the RNN of f. We first initialize each node $v \in V$ (Line 1): (1) v is marked as unprocessed, (2) v's nearest neighbor is marked as unknown, and (3) the distance $dist(v)$ from v to its nearest neighbor is set to infinity. Next, we create a queue Q that is sorted based on the vertex's recorded distance to its (temporary) nearest neighbor (Line 2). Then, for each facility vertex $p \in F$, we assign its nearest neighbor $\mathsf{NN}(p)$ to be itself, and the distance to its nearest neighbor $dist(p)$ is set to 0. We push the triplet $\{p, dist(p), \mathsf{NN}(p)\}$ to Q (Line 3-4).

Next, we extract the triplet $\{u, dist(u), \mathsf{NN}(u)\}$ from Q (Line 6) until the queue Q becomes empty (Line 5). For vertex u, if it has already been processed, no further processing is performed (Line 7). Otherwise, we mark the vertex u as processed (Line 8). For each unprocessed neighbor $v \in N(u)$ of u, we check if $dist(u) + w(u, v)$ is less than the current distance $dist(v)$ (Line 9-10). If the condition is true, it means that the distance from node v to u's nearest neighbor

Algorithm 1: Yiu's Algorithm

Input: graph $G(V = F \cup C, E, W)$, facility $f \in F$
Output: $\mathsf{RNN}_G(f)$

1 $processed(v) \leftarrow 0, \mathsf{NN}(v) \leftarrow -1, dist(v) \leftarrow \infty$, for $\forall v \in V$;
2 $Q \leftarrow \emptyset$;
3 $\mathsf{NN}(p) = p, dist(p) \leftarrow 0$, for $\forall p \in F$;
4 push $\{p, dist(p), \mathsf{NN}(p)\}$ into Q, for $\forall p \in F$;
5 **while** Q *is not empty* **do**
6 pop $\{u, dist(u), \mathsf{NN}(u)\}$ from Q;
7 **if** $processed(u) = 1$ **then** continue;
8 $processed(u) \leftarrow 1$;
9 **for** *each* $v \in N(u)$ *and* $processed(v) \neq 1$ **do**
10 **if** $dist(u) + w(u, v) < dist(v)$ **then**
11 $\mathsf{NN}(v) \leftarrow \mathsf{NN}(u), dist(v) \leftarrow dist(u) + w(u, v)$;
12 push $\{v, dist(v), \mathsf{NN}(v)\}$ into Q;
13 **for** *each* $v \in C$ **do**
14 **if** $\mathsf{NN}(c) = f$ **then** add c into $\mathsf{RNN}(f)$;
15 **return** $\mathsf{RNN}(f)$;

$\mathsf{NN}(u)$ is smaller than the distance to its current nearest neighbor, so we update v's nearest neighbor $\mathsf{NN}(v)$ to $\mathsf{NN}(u)$ and set $dist(u) + w(u, v)$ as the new shortest distance to its nearest neighbor (Line 11). Then, we push the triplet $\{v, dist(v), \mathsf{NN}(v)\}$ to Q (Line 12). Once the nearest neighbors for all vertices are found, we iterate over user vertices $c \in C$. If $\mathsf{NN}(c) = f$, we add c to the RNN of f (Line 13-14).

Example 2.3. Consider graph G in Fig. 1. First, we set the nearest neighbors of $F = \{f_1, f_2, f_3\}$ to themselves and push these three vertices into the queue Q. Then, we dequeue $\{f_1, dist(f_2) = 0, \mathsf{NN}(f_1) = f_1\}$, mark f_1 as processed, and process the unprocessed neighbors of f_1 one by one, namely $\{c_2, c_3, c_3\}$. For c_2, its nearest neighbor is temporarily set as $\mathsf{NN}(c_2) = \mathsf{NN}(f_1) = f_1$, and its distance to nearest neighbor as $dist(c_2) = dist(f_1) + w(f_1, c_2) = 5$. We also push $\{c_2, dist(c_2) = 5, \mathsf{NN}(c_2) = f_1\}$ into Q. Similarly, we push $\{c_3, dist(c_3) = 7, \mathsf{NN}(c_3) = f_1\}$ and $\{c_9, dist(c_9) = 3, \mathsf{NN}(c_9) = f_1\}$ into the queue Q. Next, we dequeue $\{f_2, dist(f_2) = 0, \mathsf{NN}(f_2) = f_2\}$, mark f_2 as processed, and push the unprocessed neighbors $\{c_1, c_2, c_3\}$ into the queue. Note that although c_2 and c_3 were pushed into the queue by f_1, they have not been marked as processed yet, so $\{c_2, dist(c_2) = 4, \mathsf{NN}(c_2) = f_2\}$ and $\{c_3, dist(c_3) = 8, \mathsf{NN}(c_3) = f_2\}$ are also pushed into Q, creating different copies. Similarly, we dequeue $\{f_3, dist(f_3) = 0, \mathsf{NN}(f_3) = f_2\}$, mark f_3 as processed, and push the unprocessed neighbors $\{c_1, c_1, c_5, c_7\}$ into the queue. Next, the item with the minimum distance in Q, i.e., $\{c_9, dist(c_9) = 3, \mathsf{NN}(c_9) = f_1\}$, is dequeued, setting c_9 as processed. At this point, $\mathsf{NN}(c_9) = f_1$ becomes the true nearest neighbor of c_9. c_9 pushes the unprocessed neighbors c_6 and c_{10} into the queue Q. This process continues until Q becomes empty, and then it stops.

Remark. In the regular Dijkstra's algorithm [7], each vertex appears once in Q and decrease-key operations are needed to determine the correct position of a vertex in Q. In Algorithm 1, we instead allow a vertex to appear multiple times in Q (Line 12), thus avoiding decrease-key operations for efficiency.

Analysis. We show the correctness and time cost of Algorithm 1.

Lemma 2.4. *Algorithm 1 correctly computes the* RNN *for each facility vertex* $f \in F$.

Lemma 2.5. *The time cost of Algorithm 1 is* $O(|V| + |E| \log |E|)$.

3 THE ERNN PROBLEM

We first provide a formal definition of the studied problem in Section 3.1, followed by an analysis of its challenges in Section 3.2.

3.1 Problem Definition

The RNN of a facility $f \in F$ contains potential users seeking services from that facility. Therefore, the RNN size of vertex f reflects the degree of utilization of that facility to some extent [11]. Considering the existence of underutilized facilities with small RNN sizes, it is important to explore ways to expand the RNN of such facilities in order to improve their utilization. However, relocating an underutilized facility, such as a fire station or a subway station, to a new location can be costly or even infeasible in practice.

Alternatively, we focus on maximizing the size of the RNN for a **target facility** f by *upgrading* a limited number of roads (i.e., edges) to reduce their weights. Reducing the weights (such as travel time) of some **modifiable edges** M can be achieved by constructing expressways or widening road surfaces [25], making this approach feasible in practice. Formally, we present the following problem.

*Definition 3.1 (Expanding Reverse Nearest Neighbors (*ERNN*)).*

Input A graph $G(V = F \cup C, E, W)$, modifiable edges $M \subseteq E$, a budget b and a target facility $f \in F$.
Output A set of (up to) b edges $A \subseteq M$ whose weights can be reduced to some specific values, resulting in the modified graph G', such that the maximum size of $\mathsf{RNN}_{G'}(f)$ is reached[1].

In Definition 3.1, we allow that the weights of the modifiable edges M can be reduced to arbitrary values pre-determined by the user. To simplify the discussion, we adopt an approach similar to that of [23] and assume that the weights of all modifiable edges can only be reduced to zero. This leads to the following simplified version of the ERNN problem, *which we aim to solve in this paper.*

Definition 3.2 (Simplified Version of the ERNN *Problem).*

Input A graph $G(V = F \cup C, E, W)$, modifiable edges $M \subseteq E$, a budget b and a target facility $f \in F$.
Output A set of (up to) b edges $A \subseteq M$ whose weights can only be reduced to zero, resulting in the modified graph G', such that the maximum size of $\mathsf{RNN}_{G'}(f)$ is reached.

One might question if the generality of Definition 3.1 is compromised by the stipulation in Definition 3.2 that edge weights can only be reduced to zero. Yet, we show that this generality remains intact under Definition 3.2, as we show that any instance of the ERNN problem defined in Definition 3.1 can be transformed into the simplified form of the ERNN problem specified in Definition 3.2.

Lemma 3.3. *Any instance of* ERNN *defined in Definition 3.1 can be transformed into the form of* ERNN *defined in Definition 3.2.*

Example 3.4. Consider the graph G in Fig 1. For the ERNN problem, the inputs are (1) the graph G, (2) the modifiable edges $M = E$, (3) the budget $b = 2$, and (4) the target facility f_1. The output is a selection of 2 edges from M, denoted as $A = \{(f_1, c_3), (c_6, c_9)\}$. Specifically, for graph G, we have $\mathsf{RNN}_G(f_1) = \{c_3, c_9, c_{10}\}$. When the

[1]This objective is consistent with maximizing the increase in the size of RNN for f in G'. We do not differentiate between these two objectives.

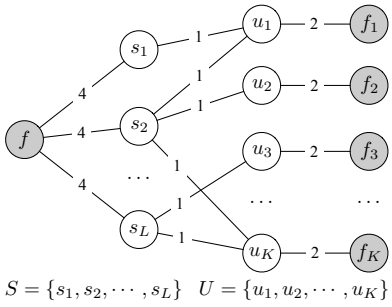


Figure 2: The Proof of NP-hardness

weights of the edges in A are reduced to 0, graph G transforms into a new graph G'. In this case, $\mathsf{RNN}_{G'}(f_1) = \{c_3, c_4, c_5, c_6, c_8, c_9, c_{10}\}$, whose size is the largest for f_1 after upgrading 2 edges.

Remark. Because of Lemma 3.3, we focus on how to design algorithms to handle the ERNN problem defined in Definition 3.2. To this end, we provide further discussion of Definition 3.2.

Modifiable edges. In practice, the weights of the edges are related to the traffic flow or the length of the road. While it is possible to reduce the weight of some roads (edges) by widening them or building highways, not all roads (edges) can be improved in this way. Therefore, in Definition 3.2, we restrict that only the weights of edges in the modifiable edges M can be reduced.

Weight estimation. In Definition 3.2, we assume that the weights of the edges in the graph G are known. Many existing works [29, 35] have been proposed to estimate the edge weights in G, e.g., estimating the weights based on trajectories. Thus, the problem of how to obtain the weights of edges is orthogonal to our work.

Reduced weight. In Definition 3.2, we specify that the weights of edges can only be reduced to zero. On the other hand, in practice, the choice of the reduced weight after an edge is upgraded can be decided by the user based on experience or budget [25]. For example, [25] and [6] consider reducing the weight of an edge to a fixed fraction of its original value, while [23] reduces the weight to zero. Yet, according to Lemma 3.3, reducing the weights to 0 can be used to handle cases where the user specifies that the weights be reduced to arbitrary values. Thus, the generality of Definition 3.2 allows it to be indifferent to how the user sets the reduced weights.

Cost of edge upgrade. In this paper, we assume that all edges have the same upgrade cost, so we focus on selecting b edges to upgrade. However, the cost of upgrading an edge may depend on the reduced weight and other factors. How to incorporate the cost into the ERNN problem definition is a direction we will explore further.

3.2 Problem Hardness

We prove the NP-hardness and APX-hardness of the ERNN problem in Theorems 3.5 and 3.6, respectively. Then, in Theorem 3.7, we demonstrate that the ERNN problem lacks monotonicity and submodularity. These analyses indicate the challenges involved in solving ERNN, thus motivating us to employ a greedy algorithm to tackle the problem practically.

Theorem 3.5. *The* ERNN *problem is NP-hard.*

Proof. In order to prove the NP-hardness of ERNN, we first introduce an NP-hard problem called Maximum Set Cover (MC) [10].

Input A universal set U containing K elements, denoted as $U = \{u_1, u_2, \cdots, u_K\}$, a set S containing L subsets, denoted as $S = \{s_1, s_2, \cdots, s_L\}$, where each $s_i \in S$ contains some elements from U, and a budget b.

Output (Up to) b elements from S, denoted as A, such that the maximum size of $\Sigma_{s_i \in A}$ is reached.

Next, we prove the equivalence between the ERNN problem and the MC problem, thereby demonstrating that ERNN is also NP-hard. To do this, we start by constructing a graph G (see Fig. 2), where each element s_i from S and u_j from U in MC are mapped to a vertex in G. If $u_j \subseteq s_i$, we connect the vertices u_j and s_i with an edge. Additionally, we add a facility vertex f and connect it to all vertices in S. We also introduce K facility vertices, denoted as f_j, and connect each f_j to the vertex u_j in U. The edge weights are assigned as shown in Fig. 2. Here, $F = \{f, f_1, f_2, \cdots, f_K\}$ and $C = U \cup S$. It can be observed that for $f \in F$, $\mathsf{RNN}_G(f) = \emptyset$. We consider a set of modifiable edges M, which includes the edges between f and vertices in S.

MC $\Rightarrow$ ERNN. Assuming MC can be solved, then selecting b elements from S can cover the maximum number of elements in U. For ERNN, when selecting edges between f and *any* b vertices S' in S, b vertices are added to the RNN of f in the updated graph (because the vertices in S' have a distance of 0 from f). Furthermore, since the vertex u_j connected to vertex $s_i \in S'$ have a distance of 1 from f, which is less than the distance 2 to the previous nearest neighbor f_j, u_j is also added to the RNN of f. Thus, based on the solution to MC, determining S' that covers the maximum number of elements in U allows for the maximum expansion of f's RNN.

ERNN $\Rightarrow$ MC. For ERNN, upgrading the edges between f to any b vertices $S' \subseteq S$ will result in the vertices in S' being added to f's RNN in the updated graph. Moreover, by solving ERNN, we can find the vertices S' that maximize the number of vertices in U connected to S' (since the adjacent vertices of S' in U are also added to f's RNN). This indicates that S' is the solution to MC. □

Theorem 3.6. ERNN *cannot be approximated in polynomial time within a ratio of* $(1 - \frac{1}{e} + \eta)$*, for* $\forall \eta > 0$*, unless P=NP.*

Theorem 3.7. ERNN *is non-submodular and non-monotonic.*

4 PROBLEM SOLUTION

In Section 4.1, we present a standard greedy algorithm for solving the ERNN problem, while in Sections 4.2 and 4.3, we introduce two key techniques to enhance its performance.

4.1 A Standard Greedy Algorithm

Considering the challenges of the ERNN problem, we employ a greedy algorithm to solve it. Let's recall the ERNN problem: Given a graph $G(V = F \cup C, E)$ and a target facility $f \in F$, our objective is to upgrade (up to) b edges A from the modifiable edges M in order to obtain a new graph G', such that the maximum size of the $\mathsf{RNN}_{G'}(f)$ for the target facility is achieved.

Algorithm. The idea of a standard greedy algorithm (denoted as Basic) is to select an edge that brings the maximum RNN size at each round, until the budget is exhausted. Specifically, in Algorithm 2, we provide the details of Basic. Basic is an iterative process that runs for a total of b rounds, where b is the budget for the ERNN

Algorithm 2: Basic

Input: graph $G(V = F \cup C, E, W)$, modifiable edges $M \subseteq E$, budget b, target facility $f \in F$
Output: edges $A \subseteq M$

1 **while** *budget* $b > 0$ **do**
2 $opt \leftarrow 0$;
  `// Distance-Based Edge Inspection (Sec. 4.2)`
3 **for** *each edge* $e \in M$ **do**
4 $G' \leftarrow$ upgrade e in G;
  `// Incremental RNN Computation (Sec. 4.3)`
5 compute $\mathsf{RNN}_{G'}(f)$ in G' using Algorithm 1;
6 **if** $opt > |\mathsf{RNN}_{G'}(f)|$ **then**
7 $opt \leftarrow |\mathsf{RNN}_{G'}(f)|$, $ans \leftarrow e$;
8 $M \leftarrow M \setminus ans$, $A \leftarrow A \cup ans$;
9 $G \leftarrow$ upgrade ans in G, $b \leftarrow b - 1$;
10 **return** *edges A;*

problem (Line 1, 8). In each round, we initialize a variable *opt* to record the maximum (updated) RNN size (Line 2). Then, we iterate over all edges $e \in M$ (Line 3). For each edge, we reduce its weight to 0 to obtain a new graph G'. We use Algorithm 1 to get the RNN of the target facility f in G' (Line 5) and update *ans* to record the edge e that yields the maximum RNN size up to now, and the corresponding size is recorded in *opt* (Line 6-7). After each round, we remove the edge *ans* from M and add it to the result set A (Line 8). Then, we set the weight of the edge *ans* to 0 to proceed with the next iteration (Line 9). After b rounds of iteration, the edges stored in the result set A form the solution to the ERNN problem.

Example 4.1. Consider graph G in Fig. 1, where $f_1 \in F$ is the target facility, the budget is 2, and all edges are modifiable edges M. In G, the RNN of target facility f_1 is $\{c_3, c_9, c_{10}\}$. For Basic, it runs in 2 rounds. In the first round, we examine all 20 edges in $M = E$. For the edge (f_1, c_2), after setting its weight to 0, the size of RNN for f_1 becomes 4. Similarly, when we inspect the edge (f_1, c_3) and set its weight to 0, the size of RNN for f_1 becomes 5. In this round, the edge (f_1, c_3) is chosen because it yields the maximum RNN size for f_1. Next, we set the weight of the edge (f_1, c_3) to 0 and remove it from M to proceed to the second iteration. We iterate through the remaining 19 edges in M and find that upgrading the edge (c_6, c_9) results in the maximum RNN size for f_1. At this point, the greedy algorithm outputs (f_1, c_3) and (c_6, c_9) as the solution for ERNN.

Opportunities. We will examine the time complexity of Algorithm 2 to uncover possibilities for enhancing its efficiency.

Lemma 4.2. *The time complexity of Algorithm 2 is* $O(b \times (|M| \times (|E| \log |E| + |V|)))$, *where* b *is the budget and* $|M|$ *is the number of modifiable edges.*

Opportunity 1. In Line 3, we need to check the modifiable edges M in each round, which introduces a factor of $|M|$ into the time complexity. To improve the practical performance of Algorithm 2, reducing the number of edges to be checked in each round becomes crucial. To achieve this, in Section 4.2, we propose a distance-based edge inspection technique, which avoids checking all modifiable edges. Based on this technique, we further introduce two powerful pruning rules to remove more edges.

Opportunity 2. In Line 5, for each edge e, we need to execute Algorithm 1 to compute RNN of f in the updated graph G'. However, it is not the optimal choice to re-invoke Algorithm 1 to compute the updated RNN of f when there is only one edge weight change. Therefore, we introduce an incremental RNN computation technique in Section 4.3 to improve the efficiency.

4.2 Distance-Based Edge Inspection

This section presents the distance-based edge inspection technique. First, we check the edges based on distance so as to obtain the upper bound for early termination of a greedy algorithm. Then, we propose two pruning strategies to remove more edges.

4.2.1 A Distance-Based Greedy Algorithm

In Algorithm 2, to solve the ERNN problem in G, we need to check each edge e in the modifiable edges M to find the optimal edge in each round. A natural question arises: can we avoid processing all the edges in M? To answer this question, we first introduce the concept of the shortest distance for an edge.

Definition 4.3. Given a graph $G(V = F \cup C, E)$ and a facility $f \in F$, assume that the shortest distance from vertex $v \in V$ to f is $dist_G(v, f)$. Then, the **shortest distance** of edge $e = (u, v)$ to f in G is defined as $dist_G((e), f) = \min(dist_G(u, f), dist_G(v, f))$.

Example 4.4. Consider graph G in Fig. 1, where the distance from c_3 to f_1 is 7 and the distance from c_2 to f_1 is 5. Therefore, the shortest distance from (c_3, c_2) to f_1 is $\min(7, 5) = 5$.

Then, we analyze whether upgrading a particular edge (u, v) can cause a vertex $c \notin \mathsf{RNN}_G(f)$, which is not in the RNN of f in graph G, to be included in the RNN of f in G' after the upgrading.

Lemma 4.5. *Given a vertex* c *whose nearest neighbor is* $\mathsf{NN}_G(c) \neq f$, *when an edge* e *decreases in weight to form a new graph* G', $\mathsf{NN}_{G'}(c) = f$ *only if* $dist_G(c, \mathsf{NN}_G(c)) > dist_G((e), f)$.

Example 4.6. For graph G in Fig. 1, the RNN of f_1 consists of $\{c_3, c_9, c_{10}\}$. For c_1, since $\mathsf{NN}_G(c_1) = f_2$ and $dist_G(c_1, f_2) = 3$. For the edge $e = (c_3, c_2)$, because $dist_G((e), f_1) = 5 > dist_G(c_1, f_2) = 3$, when upgrading e, in the new graph G', $\mathsf{NN}_{G'}(c) = f_2 \neq f_1$, which means c will not change its nearest neighbor to f_1.

Upper Bound Design. When upgrading edge e, Lemma 4.5 is a necessary condition for a vertex $c \notin \mathsf{RNN}_G(f)$ to be added to the RNN of f in the new graph G'. If we gather all vertices Δ_e that satisfy the condition in Lemma 4.5, where $\Delta_e = \{c \notin \mathsf{RNN}_G(f) | dist_G(c, \mathsf{NN}_G(c)) > dist_G((e), f)\}$, we can obtain a super set $\widetilde{\mathsf{RNN}}_{G'}(f) = \mathsf{RNN}_G(f) \cup \Delta(e)$ of f's RNN in G' since vertices in $\Delta(e)$ *may* be included in the RNN of f in G'.

Lemma 4.7. *Suppose upgrading edge* e *causes* G *to become* G', *then in* G', $\mathsf{RNN}_{G'}(f) \subseteq \widetilde{\mathsf{RNN}}_{G'}(f) = \mathsf{RNN}_G(f) \cup \Delta_e$, *where* $\Delta_e = \{c \notin \mathsf{RNN}_G(f) | dist_G(c, \mathsf{NN}_G(c)) > dist_G((e), f)\}$.

Example 4.8. Following the above example, when $e = (c_3, c_2)$ decreases in weight, we divide the vertices not in $\mathsf{RNN}_G(f_1)$ into two groups: (1) $\Delta_e = \{c_6, c_8\}$, since $dist_G(c_6, \mathsf{NN}_G(c_6) = f_3) = 8 > dist_G((e), f_1) = 7$ and $dist_G(c_8, \mathsf{NN}G(c_8) = f_2) = 8 > dist_G((e), f_1) = 7$, the vertices in Δ_e may be added to the RNN of f_1 in the new graph G'. (2) Other vertices not in Δ_e such as c_1, and upgarding e cannot cause $c_1 \notin \Delta_e$ to be included in $\mathsf{RNN}_{G'}(f)$.

Algorithm 3: DBEI

Input: graph $G(V = F \cup C, E, W)$, modifiable edges $M \subseteq E$, budget b, target facility $f \in F$
Output: edges $A \subseteq M$

1 **while** *budget* $b > 0$ **do**
2 $opt \leftarrow 0$;
3 $dist(v) \leftarrow -1, visited(v) \leftarrow 0$, for $\forall v \in V$;
4 push f into Q;
5 **while** Q *is not empty* **do**
6 pop u from Q;
7 $visited(u) \leftarrow 1$;
8 **for** $v \in N(u)$ *and* $visitd(v) = 0$ **do**
9 **if** $dist(u) + w(u, v) < dist(v)$ **then**
10 $dist(v) \leftarrow dist(u) + w(u, v)$;
11 push v into Q;
12 **if** $e = (u, v) \in M$ *and not used* **then**
13 $dist((e), f) \leftarrow dist(u)$;
14 $\Delta_e \leftarrow \{c \notin \mathsf{RNN}_G(f) | dist_G(c, \mathsf{NN}_G(c)) > dist((e), f)\}$;
15 $ub(e) \leftarrow |\Delta(e) \cup \mathsf{RNN}_G(f)|$;
16 **if** $ub(e) \leq opt$ **then** stop this round;
17 compute $\mathsf{RNN}_{G'}(f)$ in G' using Algorithm 1;
18 **if** $opt > |\mathsf{RNN}_{G'}(f)|$ **then**
19 $opt \leftarrow |\mathsf{RNN}_{G'}(f)|, ans \leftarrow e$;
20 $M \leftarrow M \setminus ans, A \leftarrow A \cup ans$;
21 $G \leftarrow$ upgrade ans in $G, b \leftarrow b - 1$;
22 **return** *edges A;*

The size of super set $\widetilde{\mathsf{RNN}}_{G'}(f)$ leads to the **upper bound** $|\widetilde{\mathsf{RNN}}_{G'}(f)|$ of the target facility f's updated RNN size after upgrading an edge. We observe a close relationship between the upper bound and the shortest distance of an edge.

Theorem 4.9. *For two edges e_1 and $e_2 \in M$, if $dist_G((e_1), f) \geq dist_G((e_2), f)$, then $|\widetilde{\mathsf{RNN}}G'_1(f)| \leq |\widetilde{\mathsf{RNN}}G'_2(f)|$, where G'_1 (resp. G'_2) is formed by upgrading e_1 (resp. e_2).*

Our Proposed Algorithm. Theorem 4.9 states that as the shortest distance from the edge to the target facility f increases, the upper bound (on the updated RNN size of f) can only decrease. This inspires us to examine the edges based on their distance to the target facility. When examining an edge at a certain distance, if the obtained upper bound is no larger than the maximum already computed RNN size achieved by upgrading previous edges, we can stop the process early: Subsequent edges do not need to be considered because their upper bound on the updated RNN size is not larger than the current edge's upper bound.

Algorithm. By implementing the above idea, we obtain a distance-based greedy algorithm (denoted as DBEI), as shown in Algorithm 3. DBEI is similar to Algorithm 2, with the only difference being that instead of sequentially scanning all edges in M, we use a modified Dijkstra's algorithm to access edges based on their distances to f (Lines 3-11). Specifically, we use a queue Q to visit vertices according to their distance to f, starting with target facility f (Line 4). We then pop vertex u from Q to access its neighbor v, thereby visiting edges $e = (u, v) \in M$ (Line 12). After computing the upper bound

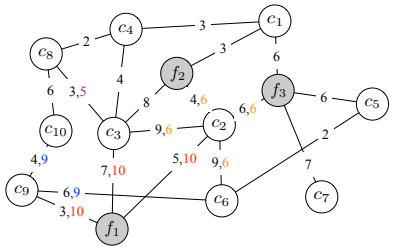


Figure 3: The Execution Process of Algorithm 3

$ub(e) = |\widetilde{\mathsf{RNN}}_{G'}(f)| = |\Delta_e \cup \mathsf{RNN}_G(f)|$ after upgrading edge e according to the definition[2] (Line 14-15), if this bound is not larger than the maximum value of the already computed RNN size opt, we can stop immediately without visiting other edges (Line 16).

Example 4.10. In G of Fig. 3, we select f_1 as the target facility, and $\mathsf{RNN}_G = \{c_3, c_9, c_{10}\}$. For clarity, we associate colored numbers with some edges of G to indicate the upper bounds of upgrading those edges. First, we push f_1 to the queue Q and then pop f_1 from Q. For the neighbor c_2 of f_1, we find that the edge $e = (f_1, c_9)$ has not been processed. The distance from $e = (f_1, c_9)$ to f_1 is $dist((e), f_1) = 0$, and we compute $\Delta_e = 7$ as there are 7 vertices outside of $\mathsf{RNN}_G(f_1)$ whose distance to their nearest neighbor is larger than $dist((e), f_1) = 0$. Then, the upper bound $ub(e)$ of e is $|\mathsf{RNN}_G \cup \Delta_e| = 10$. As the upper bound of 10 is greater than $opt = 0$, we execute Algorithm 1 to obtain the actual RNN size by upgrading e, which is 5. We set opt to 5 and then push c_9 into Q. Next, we visit the other neighbors (c_2 and c_3) of f_1. We find that the upper bounds for edges between f_1 and them are also 10, which is greater than $opt = 5$, so we execute Algorithm 1 to obtain the actual results for edges (f_1, c_2) and (f_1, c_3). This process continues until we remove c_3 from Q, and we visit its neighbor, c_8. The edge (c_3, c_8) has a distance of 7 to f_1, and we find that the upper bound is 5, which is the same as opt, so we stop without visiting other edges.

Lemma 4.11. *Algorithm 3 is correct.*

4.2.2 Pruning Strategies

Algorithm 3 utilizes an upper bound to avoid considering all edges. Based on Algorithm 3, we propose two pruning strategies to eliminate more edges and accelerate its execution.

Pruning Invalid Edges. We notice that many edges in M are invalid. An edge e is **invalid** for ERNN if it results in a non-positive increase in the RNN size of the target facility f after upgrading it, i.e., $|\mathsf{RNN}_{G'}(f)| \leq |\mathsf{RNN}_G(f)|$. We now explain the methods for determining if an edge is invalid. That is, if both endpoints of edge $e = (u, v)$ do not consider f as their nearest neighbor, e is invalid.

Lemma 4.12 (Strategy 1). *For edge (u, v) and target facility f in G, if $\mathsf{NN}_G(u) \neq f$ and $\mathsf{NN}_G(v) \neq f$, then (u, v) is invalid.*

Example 4.13. Consider graph G in Fig. 1, let f_1 be the target facility. For the edge (c_4, c_1), since $\mathsf{NN}_G(c_4) = f_2 \neq f_1$ and $\mathsf{NN}_G(c_1) = f_2 \neq f_1$, the edge (c_4, c_1) is invalid for f_1.

[2]To get Δ_e of edge e, we first sort the vertices in set $O = \{c \notin \mathsf{RNN}_G(f)\}$ in non-increasing order of their distance to the nearest neighbor. Starting from the vertex in O with the maximum distance, we stop at the first vertex s with a distance $\leq dist_G((e), f)$. All the vertices before that belong to Δ_e. As the subsequent edge e' satisfies $dist_G((e'), f) \geq dist_G((e), f)$, $\Delta_{e'} \subseteq \Delta_e$: we can backtrack from the vertex s (i.e., the first vertex in O stops e) to visit vertices with larger distances in O, thereby finding the *last* vertex with a distance $\leq dist_G((e'), f)$ for e' to find $\Delta_{e'}$.

Algorithm 4: Dynamic Computation

Input: graph $G(V, E, W)$, facility f, edge $e = (u, v)$ whose weight drops from $w(u, v)$ to $w'(u, v)$, $\mathsf{RNN}_G(f)$
Output: $\mathsf{RNN}_{G'}(f)$

1 $processed(v) \leftarrow 0$, for $\forall v \in V$;
2 $Q \leftarrow \emptyset$;
3 **if** $dist(u) + w'(u, v) < dist(v)$ **then**
4 $\quad \mathsf{NN}(v) \leftarrow \mathsf{NN}(u), dist(v) \leftarrow dist(u) + w'(u, v)$;
5 $\quad$ push $\{v, dist(v), \mathsf{NN}(v)\}$ into Q;
6 **if** $dist(v) + w'(u, v) < dist(u)$ **then**
7 $\quad \mathsf{NN}(u) \leftarrow \mathsf{NN}(v), dist(u) \leftarrow dist(v) + w'(u, v)$;
8 $\quad$ push $\{u, dist(u), \mathsf{NN}(u)\}$ into Q;
9 **while** *Q is not empty* **do**
10 $\quad$ Line 6-12 of Algorithm 1;
11 **for** *each* $v \in C$ **do**
12 $\quad$ **if** $\mathsf{NN}(c) = f$ **then** add c into $\mathsf{RNN}_{G'}(f)$;
13 **return** $\mathsf{RNN}_{G'}(f)$;

Pruning Dominated Edges. For two edges e_1 and e_2, we say that e_1 **dominates** e_2 (or e_2 is dominated by e_1) when upgrading e_1 results in a non-smaller RNN size for the target facility f than upgrading e_2. Recall that in Algorithm 3, we inspect the edges in a non-decreasing order of distances to the target facility. Based on this inspection order and the obtained distance from a vertex/edge to f, we can define a dominance relationship between edges, enabling us to prune the dominated edges during Algorithm 3.

Lemma 4.14 (Strategy 2). *Given e_1 on the shortest path[3] from target facility f to one endpoint u of e_2, if $dist_G((e_1), f) < dist(u, f) = dist_G((e_2), f)$ and $w(e_1) \geq w(e_2)$, then e_2 is dominated by e_1.*

Example 4.15. Consider graph G in Fig. 1. Since edge $e_1 = (f_1, c_3)$ lies on the shortest path from f to the endpoint c_3 of edge $e_2 = (c_3, c_4)$, $dist_G((e_1), f) = 0 < dist_G(c_3, f) = dist_G((e_2), f) = 7$, and $w(e_1) = 7 > w(e_2) = 4$, we conclude that e_1 dominates e_2.

4.3 Incremental RNN Computation

For each edge $e \in M$, to calculate the updated RNN size of the target facility f after upgrading e, one intuitive approach is to use Algorithm 1 to recalculate the RNN of f on the new graph G'. However, when there is only a single edge weight change, the change in the RNN of the target facility f is limited. To improve efficiency, an incremental algorithm is proposed for updating the RNN. Although updating the RNN is not a new research problem, to the best of our knowledge, how to update the RNN when edge weights change remains unresolved.

Algorithm. Suppose the weight of edge (u, v) in graph G decreases from $w(u, v)$ to $w'(u, v)$ to form graph G', Algorithm 4 incrementally updates the results of Algorithm 1. Algorithm 4 is similar to Algorithm 1, with the only difference being in the initialization stage. In Algorithm 1, all facility vertices F are pushed into the queue Q, while in Algorithm 4, the endpoints u and v of edge (u, v) are selectively pushed into the queue.

Specifically, if $dist(u)$ plus $w'(u, v)$ is less than $dist(v)$ (Line 2), we update the nearest neighbors $\mathsf{NN}(v)$ of vertex v in G' to $\mathsf{NN}(u)$. Furthermore, we adjust the distance $dist(v)$ from node v to $\mathsf{NN}(v)$ in G' to $dist(u) + w'(u, v)$. Additionally, we push node v to the queue Q (Line 3-5). Similar operations are performed on node u (Line 6-8). Then, we process the queue Q in a similar manner as in Algorithm 1 until the queue becomes empty, completing the update of nearest neighbors for all nodes (Line 9-10).

Example 4.16. Consider G in Fig. 1, where the weight of (f_1, c_3) decreases from 7 to 3. For endpoint f_1, since $dist(f_1, \mathsf{NN}(f_1)) = f_1) = 0$ and $dist(c_3, \mathsf{NN}(c_3)) = f_1 = 7$, we have $dist(c_3, \mathsf{NN}(c_3)) < dist(f_1, \mathsf{NN}(f_1)) + w'(f_1)$. Thus, we assign the nearest neighbor of f_1 to $\mathsf{NN}(c_3)$ and enqueue c_3. For endpoint c_3, it fails in Line 6. Next, we dequeue c_3 from Q. For its neighbor c_8, we have $dist(c_3, f_1) + w'(c_3, c_8) < dist(c_8, \mathsf{NN}(c_8)) = f_2$. Hence, we assign the nearest neighbor of c_3 to $\mathsf{NN}(c_8)$ and push c_8 into Q. For the other neighbors of c_3, namely c_4, f_2, f_1, none of them need to update their distances, so they are not pushed into Q. Then, we dequeue c_8 from Q. At this point, none of the neighbors of c_8 have their distances updated. Therefore, Q becomes empty, and the algorithm stops.

Theorem 4.17. *When the weight of edge (u, v) decreases, Algorithm 4 successfully updates the* RNN *of facility f.*

5 EXPERIMENTS

We first introduce the experimental setup, and then we will present the experimental results.

5.1 Settings

Datasets. We conducted experiments on eight real-world road networks to validate the effectiveness and efficiency of our proposed algorithms in solving the ERNN problem. These graphs were downloaded from the 9th DIMACS Implementation Challenge[4]. For the sake of convenience, we preprocess the dataset and extract the largest connected component from each graph. Table 1 provides detailed information about the datasets, including the number of nodes ($|V|$) and edges ($|E|$) for each graph.

Parameters. For each road network, we randomly selected 1000 vertices as facility vertices, while the remaining vertices were considered as user vertices. In Section 5.3, we test the impact of different numbers of facility vertices on the results. Also, we set the budget to a default value of 4 and in Section 5.3, we examine the effect of different budgets on the results. We randomly selected one vertex from the facility vertices as the target facility and conducted 50 independent experiments (each time selecting a random target facility). We report the results by averaging data from 50 experiments. For simplicity, we make all edges as modifiable edges M.

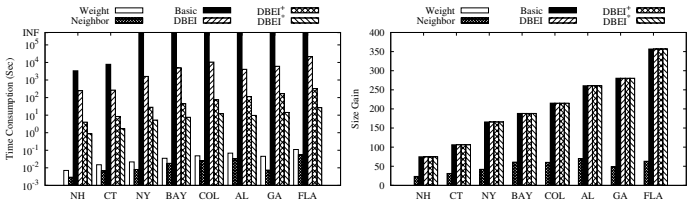
Metrics. We evaluated the performance of different methods to solve the ERNN problem in two aspects. (1) Effectiveness: For a given method, we measured the size of the RNN of the target facility f in the original graph G. Then, using the algorithm, we selected a set of edges for weight reduction, resulting in a new graph G'. We measured the size of the RNN of the target facility in the new graph G' and calculated the **gain** in RNN size, represented by $|\mathsf{RNN}_{G'}(f)| - |\mathsf{RNN}_G(f)|$, as a metric to evaluate the effectiveness. (2) Efficiency: For each method, we recorded its running time. If a method could not complete execution within six hours, we terminated its execution and marked its runtime as "INF".

[3]Whether e_1 is on shortest path can be achieved by recording shortest paths in Alg. 3.

[4]http://www.diag.uniroma1.it//challenge9/download.shtml

Table 1: Description of Datasets

Name	$\|V\|$	$\|E\|$
NH	116,920	133,415
CT	153,011	187,318
NY	264,346	366,923
BAY	321,270	400,086
COL	435,666	528,533
AL	566,843	661,487
GA	738,879	869,890
FLA	1,070,376	1,356,399



(a) The Comparison of the Running Time (b) The Comparison of the RNN Size Gain

Figure 4: The Comparison Among Various Methods

Algorithms. Since there has been no prior research on the ERNN problem, we cannot find existing methods for comparison. To make a fair comparison, we propose the following two non-trivial heuristic algorithms as baselines.

- Weight: Given a budget b, we select b edges with the highest weights from the graph for weight reduction.
- Neighbor: Given a budget b and a target facility f, we select b neighbors of f (if there are not enough neighbors of f, we choose neighbors of neighbors of f, and so on) and reduce the weights of the edges between vertex and the selected neighbors.

In Section 4.1, we presented the standard greedy algorithm, followed by improvements in Sections 4.2 and 4.3[5]. Specifically, we discuss the following four greedy algorithms.

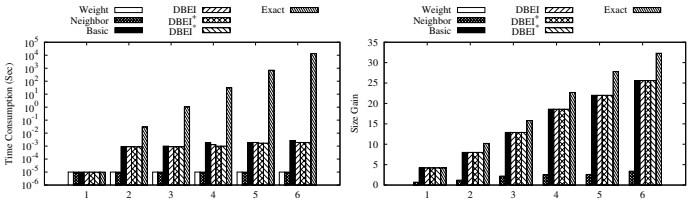
- Basic: This algorithm is described in Algorithm 2, which is a standard greedy algorithm.
- DBEI: This is our proposed algorithm, which is described in Algorithm 3. It uses a distance-based edge inspection technique to achieve early stopping using upper bounds.
- DBEI$^+$: DBEI combined with pruning strategy 1 (Lemma 4.12), which eliminates invalid edges.
- DBEI*: DBEI$^+$ combined with pruning strategy 2 (Lemma 4.14), which eliminates dominated edges.

All algorithms were implemented in C++ and compiled using GNU GCC 8.5.0 with optimizations at the -O3 level. The experiments were conducted on a machine with an Intel Xeon 2.50 GHz CPU and 512 GB of memory, running 64-bit Red Hat Linux 8.5.0.

5.2 Comparison of Different Methods

Exp-1: Efficiency Comparison. We tested two heuristic methods (Weight and Neighbor) and four greedy methods (Basic, DBEI, DBEI$^+$, and DBEI*) on all datasets, and the results are shown in Fig. 4(a). Based on the results, we draw the following conclusions.

[5]All greedy algorithms use the incremental RNN computation technique by default. We will explore the effectiveness of this technique in Section 5.3.



(a) The Comparison of the Running Time (b) The Comparison of the RNN Size Gain

Figure 5: The Comparison with the Exact Method

(1) The proposed distance-based inspection technique is effective. The standard greedy algorithm (i.e., Basic) requires inspecting all edges in M, making it unable to handle datasets other than NH and CT. On the other hand, using the distance-based edge inspection technique, our method DBEI can complete calculations on all datasets, and it is on average 21.54 times faster than Basic on graphs that Basic can handle. This demonstrates that the proposed framework achieves an order of magnitude improvement in efficiency compared to the standard greedy algorithm.

(2) The proposed pruning strategies are highly effective. The first pruning strategy, which analyzes whether each edge contains the target facility, can significantly remove invalid edges. This results in DBEI$^+$ being on average 67.33 times faster than DBEI. The second pruning strategy, based on dominance relationships, can further eliminate dominated edges. This leads to DBEI* being on average 7.85 times faster than DBEI$^+$. By combining both pruning strategies, DBEI* can complete calculations on all graphs, including the largest graph FLA (with 1.34 million edges), within 1 minute. These results demonstrate the efficiency of the proposed pruning strategies in solving the ERNN problem.

(3) Heuristic algorithms are fast. Heuristic algorithms can solve the ERNN problem relatively quickly. However, according to the following experiment (Exp-2), the RNN size gain achieved by these methods is not as good as that of greedy methods.

Exp-2: Effectiveness Comparison. We used a similar setup to Exp-1 to compare heuristic methods (Weight and Neighbor) with greedy methods (Basic, DBEI, DBEI$^+$, and DBEI*), and the results are shown in Fig. 4(b). The results reveal the following conclusions.

(1) Greedy algorithms have almost the same gain value. Although the greedy methods differ in efficiency, all four of these methods share the same spirit of problem-solving. Therefore, these greedy algorithms have almost the same gain value for the RNN size, with subtle differences due to the introduction of pruning rules.

(2) Greedy algorithms have a larger gain than heuristics. Heuristic algorithms (i.e., Weight and Neighbor) select edges based on some rules, but these rules do not necessarily guarantee good solutions to ERNN. For example, the method Weight upgrades the maximum weighted edges, but the gain is almost zero on all datasets, so it is ineffective in solving ERNN. Although Neighbor can achieve positive gain on all graphs, this method still relies on heuristic rules and cannot achieve results as good as greedy algorithms. For example, on GA, the RNN size gain caused by Neighbor is 5.73 times smaller than that of the greedy methods. Similar results can be observed in other datasets. This indicates the necessity of using greedy algorithms to solve ERNN.

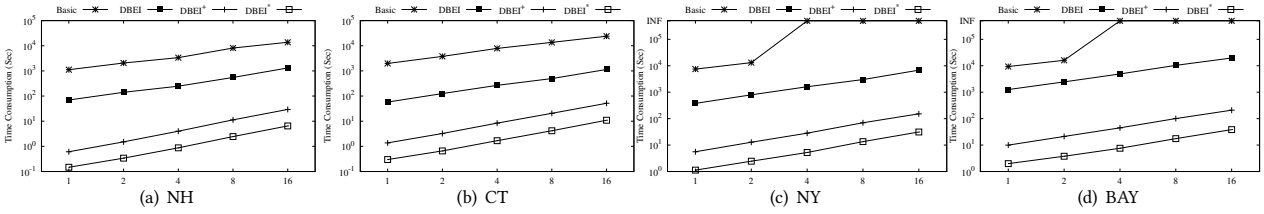


Figure 6: The Effect of the Budget on the Running Time

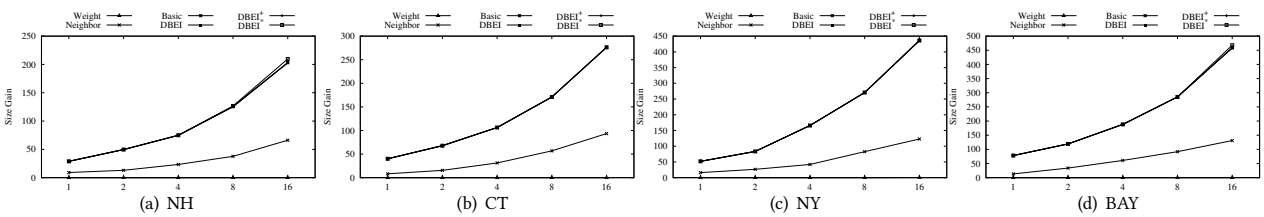


Figure 7: The Effect of the Budget on the RNN **Size Gain**

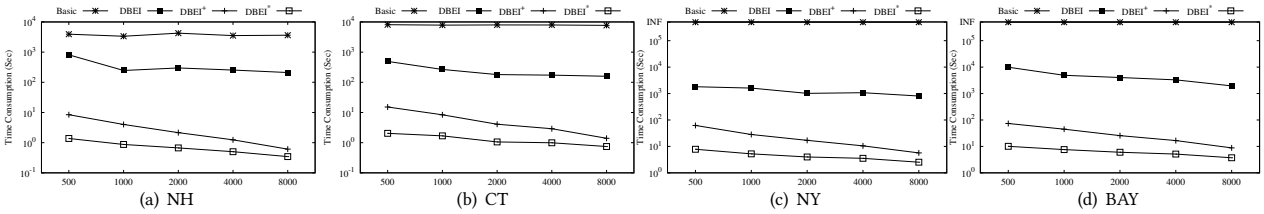


Figure 8: The Effect of the Facility Number on the Running Time

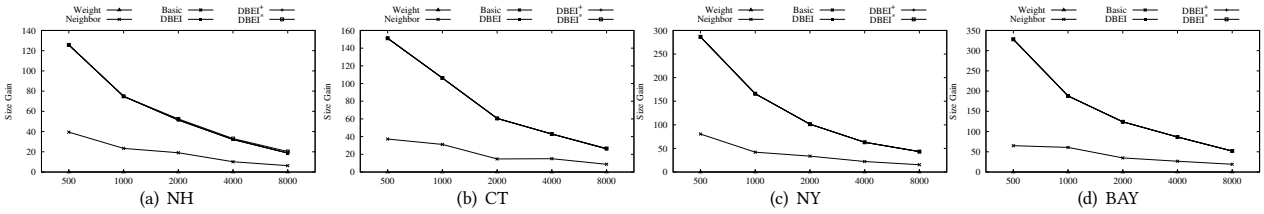


Figure 9: The Effect of the Facility Number on the RNN **Size Gain**

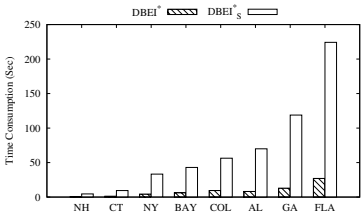


Figure 10: The Effect of the Increment RNN **Computation**

Exp-3: Comparison with Exact Method. Due to the NP-hardness and APX-hardness of the ERNN problem, we lack a standard to understand the performance of different algorithms. To address this, given a budget b, we design an exact algorithm (denoted as Exact) that enumerates all edge combinations that contain no more than b edges, and finds the edge combination that yields the maximum gain in RNN size. We report the running time and size gain of Exact as a benchmark for comparison with various methods.

It is worth noting that running Exact on the eight datasets we used is not feasible within a reasonable time. Therefore, we choose the smallest graph NH and sample 100 vertices from it. We compare various methods on the subgraph obtained after sampling NH and report the performance of different algorithms in Fig. 5 as the budget varies from 1 to 6. We draw the following conclusions.

(1) Exact *has an excessively long running time.* As the budget b increases, the running time of Exact becomes increasingly higher. For example, when b is 3, the running time of Exact is 35.43 times that when b is 2. When b is 4, the running time of Exact is $1,037.98$ times that when b is 2. Moreover, compared to our algorithm DBEI*, Exact is $7,067,526.32$ times slower when b is 6. This indicates that the Exact algorithm is impractical for solving ERNN.

(2) Greedy algorithms yield good size gain. In terms of expanding the RNN size of target facilities, the greedy algorithms (Basic, DBEI, DBEI+, and DBEI*) perform well compared to Exact. For example, when b is 1, the size gain produced by the greedy algorithms is the same as Exact. Even as b changes from 2 to 6, the greedy algorithms still have size gain that is only 1.28, 1.22, 1.22, 1.26, and 1.26 times smaller than Exact. These results show that the greedy algorithms

can achieve good size gain compared to Exact. On the other hand, the heuristic algorithm Weight does not have any effect on improving the RNN size, and as b varies, the size gain of Neighbor is on average 8.49 times smaller than Exact. This indicates that the heuristic algorithms are not effective in solving ERNN.

5.3 Further Analysis of the Greedy Methods

Exp-4: Effect of Budget on Runtime. In the previous experiments, we fixed the budget at 4. To analyze the effect of the budget on the greedy algorithms, we varied the budget from 1, 2, 4, 8 to 16. Since the results are similar on all graphs, we only report results on NH, CT, NY, and BAY in the following. The results are in Fig. 6, and we have the following conclusions.

(1) The runtime of all methods increases with the increase in budget. For example, in NH, our proposed DBEI takes 2.02 times longer when $b = 2$ compared to when $b = 1$, and it takes 3.54 times longer when $b = 4$ compared to when $b = 1$. The runtime of other methods shows a similar trend.

(2) Distance-based inspection is effective under different budgets. Similar to the results of Exp-1, the proposed distance-based edge inspection technique makes DBEI significantly reduce the time cost, when compared with Basic. For example, in NH, when $b = 8$, the runtime of Basic is 14.73 times longer than that of DBEI, and when $b = 16$, the runtime of Basic is 10.51 times longer than that of DBEI. This further demonstrates the effectiveness of the distance-based edge inspection technique under various budgets.

(3) The pruning strategies are effective under different budgets. The proposed pruning strategies show effectiveness under different budgets. For example, in NY, when $b = 16$, DBEI$^+$ with the first strategy is 44.86 times faster than DBEI, and DBEI* with both the first and second strategies is 218.13 times faster than DBEI.

Exp-5: Effect of Budget on Size Gain. We analyze the change in the RNN size gain of the different greedy methods when the budget varies from 1, 2, 4, 8 to 16. We have also included heuristic algorithms for better comparisons. The results are shown in Fig. 7. Consistent with the findings of Exp-2, the gains of all greedy algorithms are almost the same and outperform heuristics. We also find that the gains of the greedy algorithms have the following feature.

As the budget increases, the gain of greedy algorithms also increases. Take NH as an example. When the budget is 2, the gain of all greedy algorithms is 1.72 times that when the budget is 1. When the budget is 16, the greedy algorithm produces more than 7 times as much gain as when the budget is 1. This is intuitive because a larger budget means that we can choose more edges to upgrade, which leads to more vertices joining the RNN of the target facility.

Exp-6: Effect of Facility Number on Runtime. In the previous experiments, we fixed the number of facility vertices to be 1000. To test the effect of the numbers of facility vertices, we varied this number from 500, 1000, 2000, 4000 to 8000 and presented the results in Fig. 8. It can be observed that under different numbers of facilities, the proposed methods (DBEI, DBEI$^+$, and DBEI*) outperform the standard greedy algorithm (Basic). Additionally, the pruning strategies employed in DBEI$^+$ and DBEI* allow them to outperform the proposed DBEI under different facility numbers. Furthermore, we discovered another interesting phenomenon.

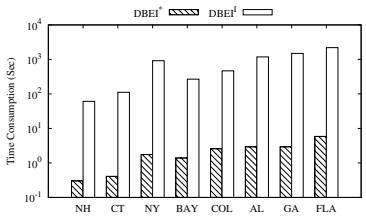


Figure 11: The Effect of the Indexing Technique

The Runtime of methods generally decreases as the number increases. For example, on BAY, when the number of facilities is 500, the running time of DBEI is 2.02 times that of 1000, 2.46 times for 2000, 3.01 times for 4000, and 5.08 times for 8000. Similar trends can be observed for other methods.

Exp-7: Effect of Facility Number on Size Gain. When the number of facilities varies from 500, 1000, 2000, 4000 to 8000, we analyze the change in size gain for different methods. The results are shown in Fig. 9. We observe that the greedy algorithms outperform the heuristic algorithms for different numbers of facilities. For example, on CT, when the number is 500, the gain of the greedy algorithms is 4.06 times that of Neighbor, while the gain of Weight is close to zero. Moreover, the RNN size gains of the greedy algorithms are characterized as follows.

As the number increases, the gain of greedy methods decreases. Taking CT as an example, when the facility number is 500, the gain of the greedy algorithms is 1.42 higher than when the number is 1000, 2.49 higher than when the number is 2000, 3.52 higher than when the number is 4000, and 5.76 higher than when the number is 8000. One reason for this is that as the number increases, it becomes harder for other vertices to join the RNN of the target facility for a fixed budget, which leads to a downward trend.

Exp-8: Effect of Increment RNN Computation. In Section 4.3, we presented Algorithm 4 for incremental RNN computation. To test the practical effect of this technique, we removed the incremental RNN computation technique and replaced it with the recompute-based RNN computation using Algorithm 1. As an example, when we replaced Algorithm 4 with Algorithm 1 in DBEI* to eliminate the effect of incremental RNN computation, we obtained the method DBEI$^*_S$. We show the difference in runtime between DBEI* and DBEI$^*_S$ in Fig. 10, and similar observations can be found in other methods. We draw the following conclusion.

The use of incremental computation greatly reduces the runtime. The incremental RNN computation brings 7.55 times speedup on average. For instance, on AL, DBEI* is 8.6 times faster than DBEI$^*_S$; on GA, DBEI* is 9.17 times faster than DBEI$^*_S$. These results validate the importance of the proposed incremental RNN computation technique.

Exp-9: Effect of Indexing Technique. In Section 2.2, we use an online search method (Algorithm 1) to compute the nearest neighbors of all vertices to determine the RNN of the facility vertices. Considering that it is a common practice to use indexes to speed up queries, we use the cutting-edge index-based method [24] to compute the nearest neighbors of each vertex and then obtain the RNN. We replace Algorithm 1 with this index-based method in our method DBEI* to obtain a new method DBEI^I for solving ERNN.

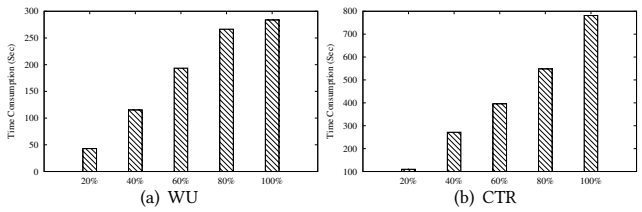


Figure 12: The Test of Scalability

Since DBEI^{I} and DBEI* have the same RNN size gain, we only compared the runtime of both (and we fixed the budget at 1), and the results are in Fig. 11. From Fig. 11, the following can be observed.

Indexing techniques are not well suited to solve the ERNN *problem.* On all graphs, the average running time of the index-based method DBEI^{I} is 333.1 times slower than our method DBEI*. Note that while indexing can speed up query processing, solving the ERNN problem requires recomputing RNN each time the edge weights in the graph change. Computing and maintaining the index during graph changes results in significant time overhead. In contrast, our DBEI* method avoids the reliance on expensive indexes by using online search and thus achieves faster computation.

Exp-10: Test of Scalability. To test the scalability of the proposed method, we use two large-scale graphs: WU (with 6, 262, 104 vertices and 7, 624, 073 edges) and CTR (with 14, 081, 816 vertices and 17, 146, 248 edges). For both graphs, we divide all vertices into five equal parts, thus obtaining five test graphs, where the i-th test graph includes the first i parts of the vertices, i.e., it includes $\frac{i}{5}$ of all vertices in the graph. We run the proposed method DBEI* in these 5 test graphs and report the running time in Fig. 12. From Fig. 12, we draw the following conclusion.

DBEI**'s running time increases steadily with the graph size.* For example, on CTR, when comparing the running time of the test graph with 20% vertices to the running time of the test graph with 40%, 60%, 80%, and 100% vertices, the running time increases by 2.47, 3.6, 4.98, and 7.11, respectively. These findings indicate that the runtime performance of the proposed method steadily improves as the graph size changes, thus ensuring its scalability. Similar results can be observed on WU. Moreover, the runtime of DBEI* stays within 15 minutes even on CTR with more than 14 million vertices, validating the efficiency of the proposed method.

6 RELATED WORK

RNN Computation. Previous research has mainly focused on computing the RNN of a vertex in metric spaces [11]. Yiu et al. [36] introduced the computation of RNN on graphs, including the bichromatic RNN problem (as adopted in this paper, where vertices are divided into users and facilities) and the monochromatic RNN problem (where the graph consists of vertices of a single type). Safer et al. [28] used the Voronoi diagram for RNN computation, while Efentakis et al. [8] employed hub labels to accelerate computation.

Existing research also addresses the issue of maintaining RNN. For example, Sun et al. [30] and Li et al. [12] proposed indexing road networks to track the updates of RNN for a query vertex. Cheema et al. [5] introduced a filter-verification framework to handle RNN updates. However, to the best of our knowledge, these studies mostly focus on how to monitor the RNN when facilities/users change their locations. Further research is needed on how to update the RNN of a query vertex when the weights in the network change.

Another related topic is the Maximizing Reverse Nearest Neighbor problem [6, 19, 34], which aims to determine an optimal location for a *new* facility to maximize its RNN size. In contrast, the ERNN problem we propose involves actively upgrading edges in the graph to maximize the size of a target facility's RNN. Therefore, our proposed problem differs from the existing problem and requires further investigation.

Network Upgrading. The ERNN problem studied in this paper belongs to the **network upgrading** problem, which involves upgrading network edges/vertices to optimize a specific objective. The research on network upgrading problems has a long history. For example, Paik et al. [25] began defining a series of network upgrading problems in 1995 with the aim of optimizing specific network metrics, such as shortest or longest distances. Zhang et al. [37] focused on reducing edge weights to decrease node-to-node distances. Similarly, Campbell et al. [3] studied how to minimize the longest travel time by reducing the weights of certain edges. Medya et al. [21–23] investigated methods to minimize network latency by reducing node weights to zero. Likewise, Lin et al. [17] studied how to minimize the total distance between sets of nodes by reducing the weights of certain edges. Differently, the objective of our ERNN problem is to maximize the RNN size of a target facility, requiring the development of new solutions.

Other related problems include adding edges to a graph (which can be seen as reducing the weights from infinity to zero) to minimize the graph diameter [26], maximize information propagation [4], maximize k-core [31] or k-truss [32], enhance centrality [2, 15], and optimize clustering [14]. Considering that adding edges (i.e., constructing new roads) in a road network may not always be feasible, we have not considered the case of adding edges. However, this can be an interesting avenue for future research.

7 CONCLUSION

This paper focuses on defining and solving the ERNN problem. The main goal of the ERNN problem is to maximize the RNN size of the target facility, thereby enhancing its utilization. To overcome the challenges associated with solving the ERNN problem, we devise a distance-based edge inspection technique and introduced two pruning rules. Furthermore, we develop a technique to dynamically maintain the RNN. By combining these techniques, we create optimized greedy algorithms that are empirically proven to be highly effective and efficient. Future research directions involve expanding the scope of the proposed ERNN problem, such as considering different costs associated with selecting various edges.

ACKNOWLEDGMENTS

Wentao Li is supported by NSFC 62302417. Min Gao is supported by NSFC 62176028. Dong Wen is supported by ARC DP230101445 and ARC DE240100668. Lu Qin is supported by ARC FT200100787 and ARC DP210101347. Wei Wang is supported by HKUST(GZ) Grant G0101000028, CCF-HuaweiDBC202302, Guangzhou Municipal Science and Technology Project (No. 2023A03J0003) and Guangzhou-HKUST(GZ) Joint Funding Program (No. 2023A03J0013).

REFERENCES

[1] Nasser Allheeib, Kiki Adhinugraha, David Taniar, and Md Saiful Islam. 2022. Computing reverse nearest neighbourhood on road maps. *World Wide Web* (2022), 1–32.

[2] Elisabetta Bergamini, Pierluigi Crescenzi, Gianlorenzo D'angelo, Henning Meyerhenke, Lorenzo Severini, and Yllka Velaj. 2018. Improving the betweenness centrality of a node by adding links. *Journal of Experimental Algorithmics (JEA)* 23 (2018), 1–32.

[3] Ann Melissa Campbell, Timothy J Lowe, and Li Zhang. 2006. Upgrading arcs to minimize the maximum travel time in a network. *Networks: An International Journal* 47, 2 (2006), 72–80.

[4] Vineet Chaoji, Sayan Ranu, Rajeev Rastogi, and Rushi Bhatt. 2012. Recommendations to boost content spread in social networks. In *Proceedings of the 21st international conference on World Wide Web*. 529–538.

[5] Muhammad Aamir Cheema, Wenjie Zhang, Xuemin Lin, Ying Zhang, and Xuefei Li. 2012. Continuous reverse k nearest neighbors queries in euclidean space and in spatial networks. *The VLDB Journal* 21 (2012), 69–95.

[6] Farhana M Choudhury, J Shane Culpepper, Timos Sellis, and Xin Cao. 2016. Maximizing bichromatic reverse spatial and textual k nearest neighbor queries. *Proceedings of the VLDB Endowment* 9, 6 (2016), 456–467.

[7] Edsger W. Dijkstra. 1959. A note on two problems in connexion with graphs. *Numer. Math.* 1 (1959), 269–271. https://doi.org/10.1007/BF01386390

[8] Alexandros Efentakis and Dieter Pfoser. 2016. ReHub: Extending hub labels for reverse k-nearest neighbor queries on large-scale networks. *Journal of Experimental Algorithmics (JEA)* 21 (2016), 1–35.

[9] Wenfei Fan. 2022. Big graphs: challenges and opportunities. *Proceedings of the VLDB Endowment* 15, 12 (2022), 3782–3797.

[10] Richard M Karp. 1972. Reducibility among combinatorial problems. In *Complexity of computer computations*. Springer, 85–103.

[11] Flip Korn and Suresh Muthukrishnan. 2000. Influence sets based on reverse nearest neighbor queries. *ACM Sigmod Record* 29, 2 (2000), 201–212.

[12] Guohui Li, Yanhong Li, Jianjun Li, LihChyun Shu, and Fumin Yang. 2010. Continuous reverse k nearest neighbor monitoring on moving objects in road networks. *Information Systems* 35, 8 (2010), 860–883.

[13] Wentao Li, Maolin Cai, Min Gao, Dong Wen, Lu Qin, and Wei Wang. 2023. *Technical Report*. https://www.dropbox.com/scl/fo/7jkd2bgwzvg5rv8bj2fmr/h?rlkey=673fyrek4rlvq10q4oyuva4ql&dl=0

[14] Wentao Li, Min Gao, Dong Wen, Hongwei Zhou, Cai Ke, and Lu Qin. 2022. Manipulating Structural Graph Clustering. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2749–2761.

[15] Wentao Li, Min Gao, Fan Wu, Wenge Rong, Junhao Wen, and Lu Qin. 2021. Manipulating black-box networks for centrality promotion. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 73–84.

[16] Xueping Li, Zhaoxia Zhao, Xiaoyan Zhu, and Tami Wyatt. 2011. Covering models and optimization techniques for emergency response facility location and planning: a review. *Mathematical Methods of Operations Research* 74 (2011), 281–310.

[17] Yimin Lin and Kyriakos Mouratidis. 2015. Best upgrade plans for single and multiple source-destination pairs. *GeoInformatica* 19 (2015), 365–404.

[18] Hao Liu, Ying Li, Yanjie Fu, Huaibo Mei, Jingbo Zhou, Xu Ma, and Hui Xiong. 2020. Polestar: An intelligent, efficient and national-wide public transportation routing engine. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2321–2329.

[19] Yubao Liu, Raymond Chi-Wing Wong, Ke Wang, Zhijie Li, Cheng Chen, and Zhitong Chen. 2013. A new approach for maximizing bichromatic reverse nearest neighbor search. *Knowledge and information systems* 36, 1 (2013), 23–58.

[20] Zihan Luo, Lei Li, Mengxuan Zhang, Wen Hua, Yehong Xu, and Xiaofang Zhou. 2022. Diversified top-k route planning in road network. *Proceedings of the VLDB Endowment* 15, 11 (2022), 3199–3212.

[21] Sourav Medya, Petko Bogdanov, and Ambuj Singh. 2016. Towards scalable network delay minimization. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 1083–1088.

[22] Sourav Medya, Petko Bogdanov, and Ambuj Singh. 2018. Making a small world smaller: Path optimization in networks. *IEEE Transactions on Knowledge and Data Engineering* 30, 8 (2018), 1533–1546.

[23] Sourav Medya, Sayan Ranu, Jithin Vachery, and Ambuj Singh. 2018. Noticeable network delay minimization via node upgrades. *Proceedings of the VLDB Endowment* 11, 9 (2018), 988–1001.

[24] Dian Ouyang, Dong Wen, Lu Qin, Lijun Chang, Ying Zhang, and Xuemin Lin. 2020. Progressive top-k nearest neighbors search in large road networks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1781–1795.

[25] Doowon Paik and Sartaj Sahni. 1995. Network upgrading problems. *Networks* 26, 1 (1995), 45–58.

[26] Manos Papagelis, Francesco Bonchi, and Aristides Gionis. 2011. Suggesting ghost edges for a smaller world. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. 2305–2308.

[27] Yu-Xuan Qiu, Dong Wen, Lu Qin, Wentao Li, Rong-Hua Li, and Ying Zhang. 2022. Efficient shortest path counting on large road networks. *Proceedings of the VLDB Endowment* 15, 10 (2022), 2098–2110.

[28] Maytham Safar, Dariush Ibrahimi, and David Taniar. 2009. Voronoi-based reverse nearest neighbor query processing on spatial networks. *Multimedia systems* 15 (2009), 295–308.

[29] Rade Stanojevic, Sofiane Abbar, and Mohamed Mokbel. 2018. W-edge: Weighing the edges of the road network. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 424–427.

[30] Huan-Liang Sun, Chao Jiang, Jun-Ling Liu, and Limei Sun. 2008. Continuous reverse nearest neighbor queries on moving objects in road networks. In *2008 The Ninth International Conference on Web-Age Information Management*. IEEE, 238–245.

[31] Xin Sun, Xin Huang, and Di Jin. 2022. Fast algorithms for core maximization on large graphs. *Proceedings of the VLDB Endowment* 15, 7 (2022), 1350–1362.

[32] Xin Sun, Xin Huang, Zitan Sun, and Di Jin. 2021. Budget-constrained Truss Maximization over Large Graphs: A Component-based Approach. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 1754–1763.

[33] Sheng Wang, Yuan Sun, Christopher Musco, and Zhifeng Bao. 2021. Public transport planning: When transit network connectivity meets commuting demand. In *Proceedings of the 2021 International Conference on Management of Data*. 1906–1919.

[34] Raymond Chi-Wing Wong, M Tamer Özsu, Philip S Yu, Ada Wai-Chee Fu, and Lian Liu. 2009. Efficient method for maximizing bichromatic reverse nearest neighbor. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1126–1137.

[35] Bin Yang, Manohar Kaul, and Christian S Jensen. 2013. Using incomplete information for complete weight annotation of road networks. *IEEE Transactions on Knowledge and Data Engineering* 26, 5 (2013), 1267–1279.

[36] Man Lung Yiu, Dimitris Papadias, Nikos Mamoulis, and Yufei Tao. 2006. Reverse nearest neighbors in large graphs. *IEEE Transactions on Knowledge and Data Engineering* 18, 4 (2006), 540–553.

[37] JZ Zhang, XG Yang, and MC Cai. 2004. A network improvement problem under different norms. *Computational Optimization and Applications* 27 (2004), 305–319.