

Singapore Management University Institutional Knowledge at Singapore Management University

Research Collection School Of Information Systems

School of Information Systems

4-2015

Best Upgrade Plans for Single and Multiple Source-Destination Pairs

Yimin Lin


Singapore Management University, yimin.lin.2007@smu.edu.sg

Kyriakos MOURATIDIS

Singapore Management University, kyriakos@smu.edu.sg

DOI: <https://doi.org/10.1007/s10707-014-0219-1>

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

 Part of the [Databases and Information Systems Commons](#), and the [Numerical Analysis and Scientific Computing Commons](#)

Citation

Lin, Yimin and MOURATIDIS, Kyriakos. Best Upgrade Plans for Single and Multiple Source-Destination Pairs. (2015). *GeoInformatica: Special Issue on the Best of SSTD 2013*. 19, (2), 365-404. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/2274

This Journal Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email libIR@smu.edu.sg.

Best Upgrade Plans for Single and Multiple Source-Destination Pairs

Yimin Lin · Kyriakos Mouratidis

Received: date / Accepted: date

Abstract In this paper, we study *Resource Constrained Best Upgrade Plan* (BUP) computation in road network databases. Consider a transportation network (weighted graph) G where a subset of the edges are *upgradable*, i.e., for each such edge there is a cost, which if spent, the weight of the edge can be reduced to a specific new value. In the single-pair version of BUP, the input includes a source and a destination in G , and a budget B (resource constraint). The goal is to identify which upgradable edges should be upgraded so that the shortest path distance between source and destination (in the updated network) is minimized, without exceeding the available budget for the upgrade. In the multiple-pair version of BUP, a set Q of source-destination pairs is given, and the problem is to choose for upgrade those edges that lead to the smallest sum of shortest path distances across all pairs in Q , subject to budget constraint B . In addition to transportation networks, the BUP query arises in other domains too, such as telecommunications. We propose a framework for BUP processing and evaluate it with experiments on large, real road networks.

Keywords Road network · Resource constraint · Network upgrade

Yimin Lin
80 Stamford Road
Singapore 178902
Tel.: +65-68280903
Fax: +65-68280919
E-mail: yimin.lin.2007@smu.edu.sg

Kyriakos Mouratidis
80 Stamford Road
Singapore 178902
Tel.: +65-68280649
Fax: +65-68280919
E-mail: kyriakos@smu.edu.sg

1 Introduction

Graph processing finds application in a multitude of domains. Problems in transportations, telecommunications, bioinformatics and social networks are often modeled by graphs. A large body of research considers queries related to reachability, shortest path computation, path matching, etc. One of the less studied topics, which however is of large practical significance, is the distribution of available resources in a graph in order to achieve certain objectives. Here we consider road networks in particular, and the goal is to minimize the total traveling time (shortest path distance) for a group of source-destination pairs by amending the weights of selected edges.

As an example, consider the transportation authority of a city, where a new hospital (or an important facility of another type) is opened, and the authority wishes to upgrade the road network to ease access to this facility from other key locations (e.g., from the airport or from CBD area). While several road segments (network edges) may be amenable to physical upgrade, this comes at a monetary cost. The *Resource Constrained Best Upgrade Plan* problem (BUP) is to select some among the upgradable edges so that the traveling time between source(s) and destination(s) is minimized and at the same time the summed upgrade cost does not exceed a specific budget B (resource constraint). In the given example, there are two source-destination pairs that are of interest, i.e., airport-hospital and CBD-hospital. These two pairs form set Q and the objective is to minimize the sum of shortest path distances for these pairs, subject to budget B . In the general case, there could be any number of pairs. Also, although in our example there is a common destination (the hospital), the source and destination of each pair in Q could be arbitrarily and independently chosen.

Another, more time-critical application example is an intelligent transportation system that monitors the traffic in the road network of a city, and schedules accordingly the traffic lights in road junctions in real-time. Assume that a major event is taking place in the city and heavy traffic is expected from a specific source (e.g., a sports stadium) to a specific destination (e.g., the marina). With appropriate traffic light reconfiguration, the driving time across some edges in the network can be reduced, at the cost of longer waits for walkers at affected pedestrian crosses. Assuming that along with each upgradable edge there is a cost associated to capture the burden imposed to pedestrians, BUP could indicate which road segments to favor in the traffic light schedule so that (i) the traveling time from the stadium to the marina is minimized and (ii) the summed cost against pedestrian priority does not exceed a certain value. Since there is only one source-destination pair that is of interest, we refer to this version of the problem as the single-pair BUP (as opposed to the multiple-pair case in our first example).

Although we focus on transportation networks, BUP finds application in other domains too. Consider for instance a communication network, where on-demand dynamic allocation of bandwidth and QoS parameters (e.g., latency) is possible for some links between nodes (routers). In the usual case of leased network infrastructure in the Internet Protocol (IP) layer, upgrading a link in terms of capacity or QoS access parameters would incur a monetary cost. When a large volume of time-sensitive traffic (e.g., VoIP) is expected between several pairs of nodes, (multiple-pair) BUP would indicate to the network operators which links to upgrade in order to minimize the total network latency between the pairs of nodes, subject to the available monetary budget.

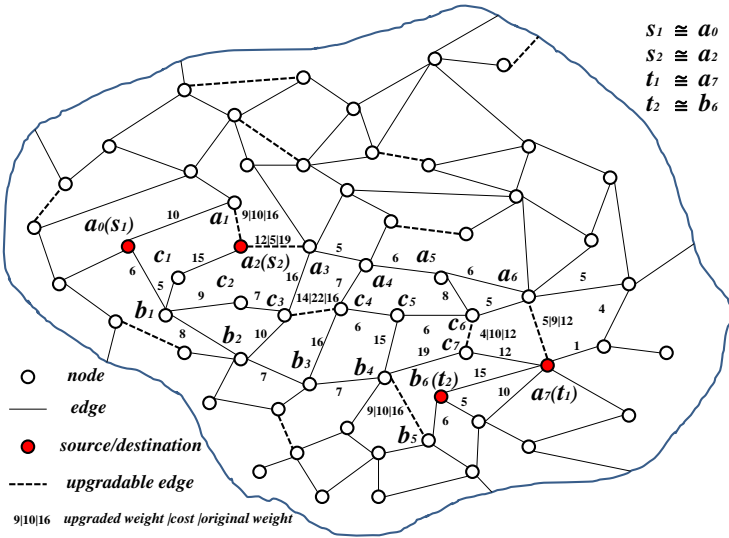


Fig. 1 Example of BUP query

Figure 1 shows an example of BUP (with $|Q| = 2$ query pairs) in a road network. The edges drawn as dashed lines are upgradable. Each upgradable edge is associated with a triplet of numbers (e.g., $(9|10|16)$), indicating respectively the new weight (if the edge is chosen for upgrade), the cost for the upgrade, and the original weight of the edge. For normal, non-upgradable edges, the number associated with them indicates their (unchangeable) weight; weights are only illustrated for edges that affect our example (all the rest are assumed to have a weight of 15).

The input of the query is a set Q of source-destination pairs in the network, plus a resource constraint B . Let U be the set of upgradable edges. The objective in BUP is to select a subset of edges from U which, if upgraded, will lead to the minimum possible sum of shortest path distances across all pairs $\langle s_i, t_i \rangle$ in Q , while the sum of their upgrade costs does not exceed B . Assuming a resource constraint $B = 20$ and $Q = \{ \langle s_1, t_1 \rangle, \langle s_2, t_2 \rangle \}$ in Figure 1, the output of BUP includes edges (a_2, a_3) , (a_6, a_7) and leads to a summed shortest path distance of 108 ($=59+49$) via paths $\{s_1, b_1, b_2, b_3, b_4, c_7, t_1\}$ and $\{s_2, a_3, a_4, a_5, a_6, a_7, t_2\}$. The resource consumption in this case is 14 ($=5+9$), i.e., smaller than B , and thus permissible. Note that if B were larger, an even shorter summed distance could be achieved (namely, 102 ($=53+49$)) by upgrading edges (a_1, a_2) , (a_2, a_3) and (a_6, a_7) . This however would incur a cost of 24 that exceeds our budget B .

There are several bodies of research that are related to BUP, such as methods to construct from scratch or modify the topology of a network to serve a specific objective [16, 36, 28]. However, the only work on BUP is [23], which considers only the single-pair version of the problem. The current paper subsumes [23] and addresses the general, multiple-pair BUP, which entails significant new challenges and calls for novel solutions.

In this paper, we formalize the general BUP and propose a suite of algorithms for its efficient processing. As will become clear in Section 3, the main performance challenge in BUP is the intractability of the search space and the need for numerous shortest path computations. We develop techniques that limit both these factors. To demonstrate the practicality of our framework, we conduct an extensive empirical evaluation using large, real road networks.

2 Related Work

There are several streams of work related to BUP query. In this section, we give a brief overview of these streams and indicate their differences from our problem.

2.1 Road Network Databases

There has been considerable research in the area of road network databases, including methods for network storage and querying (e.g., ranges and nearest neighbors) [29, 34, 17], the processing of queries that involve a notion of dominance based on proximity [9], continuous versions of proximity queries [35], etc. There have also been several studies on materialization with the purpose of accelerating shortest distance/path queries [21, 19]. All these techniques focus on data organization and querying mechanisms on a network that is used as-is, i.e., they do not consider the selective amendment of edge weights. The closest related piece in this area is [10], which considers shortest path computation over time-dependent networks, i.e., where the weight of each edge is given by a function of time. Again, there is no option to select edges for upgrade nor any control over edge weights.

2.2 Network Topology Modification

Another related body of work includes methods to modify the network topology in order to meet specific optimization objectives.

Algorithms on *network topology optimization* and *network design* compute/derive a topology for nodes and edges in a network (e.g., number of nodes, placement of nodes and edges, etc) to meet certain goals. Literature on this topic falls under wireless networks [15, 2], wired backbone networks [5, 31, 20, 27] and service overlay networks [22, 7, 12, 33]. These methods design the topology of the network, affecting its very structure. In BUP, instead, the topology is preserved and the question is which edges to upgrade within a specific budget (the budget not being a consideration among methods in this category).

In the *network hub allocation problem* the purpose is to locate hubs and allocate demand nodes to hubs in order to route the traffic between origin-destination pairs [1]. There are different lines of work: p -hub median techniques, p -hub center algorithms, hub covering, and hub allocation methods with fixed costs. These are essentially location-allocation problems, and far from the BUP setting.

2.3 Resource Allocation and Network Improvement

In this section, we review work which does not seek to modify the network topology, but is intended to allocate resources or select certain nodes/edges from the network to meet specific optimization objectives.

The *resource allocation problem in networks* is to efficiently distribute resources to users, such as bandwidth and energy, in order to achieve certain goals, like upholding QoS contracts. Most of the work in this field focuses on pricing and auction mechanisms [16, 18, 25]. Game theory is the main vehicle to address these problems, whose objectives differ from BUP.

Probably the closest related topic to BUP is the *network improvement problem*. The setting is similar to BUP, with an option to lower the weights of edges, but the objective is to reduce the diameter of the network, i.e., the maximum distance between any pair of nodes. In [36], the authors discuss the complexity of the problem with budget constraints. Budget constraints are also considered in [4], which proposes methods to minimize the diameter of a tree-structured network. [6] addresses the q -upgrading arc problem, where q edges are selected for upgrading to minimize the diameter of the graph. In [28] the problem is to identify the non-dominated paths in a space where each is represented by its upgrade cost and the overall improvement achieved in traveling time across a set of source-destination pairs if the path is upgraded. The problem definitions in this body of work are fundamentally different from BUP, and the proposed algorithms are inapplicable to it.

In a *resource constrained shortest path* problem, there are different types of resources required to cross each edge and the goal is to identify the shortest path that does not exceed the available budget in each resource type [13, 3, 26, 32]. *Restricted shortest path* is another example where each edge is associated with a cost and a delay. The objective is to identify the path which incurs the minimum cost while the delay along the path does not exceed a specific time limit. Both exact and approximate methods have been proposed [14, 24, 11]. In both aforementioned problems, the choice is whether to pass through a certain edge or not, as opposed to choosing whether to upgrade it.

3 Problem Formalization

We first formalize the problem and identify the key challenges in its processing.

Let $G = \langle V, E \rangle$ be a road network (weighted graph), where V is the set of nodes (vertices) and E is the set of edges (arcs). Every edge $e = (v_i, v_j)$ in E is associated with a weight $e.w$ that models the traveling time from v_i to v_j via e . For simplicity, we assume an undirected network (but our methods can be easily extended to directed ones too). A subset U of the network's edges are *upgradable*. That is, every $e \in U$ is also associated with an upgrade cost $e.c$ (where $e.c > 0$), and a new weight value $e.w'$ (where $e.w' < e.w$), indicating that if $e.c$ is spent, the weight of e can drop to $e.w'$. In this graph G , the BUP problem is defined as follows.

Given a budget (resource constraint) B , a set of source-destination node pairs Q (i.e., each pair in Q has the form $\langle s_i, t_i \rangle$ where $s_i, t_i \in V$), the BUP problem is to select a subset R of the edges in U for upgrade so that (i) the summed upgrade cost in

Symbol	Meaning
$G = \langle V, E \rangle$	Road network with node set V and edge set E
U	Set of upgradable edges ($U \subseteq E$)
B	Resource constraint (budget)
Q	Query set (of source-destination pairs)
R	Edges chosen for upgrade ($R \subseteq U$)
$ U $ and $ Q $	The cardinality of U and Q , respectively
$e.w$	Original weight of edge e
$e.c, e.w'$	Upgrade cost and upgraded weight of $e \in U$
$C(R)$	Sum of upgrade costs (across all edges) in R
$SP(v_i, v_j, R)$	Shortest path (length) from v_i to v_j after upgrades in R
G_c	Concise graph, BUP-equivalent to G (Section 4)
R_{temp}	A permissible, heuristic BUP solution (Section 4)
$A(G_c)$	Augmented version of G_c (Section 5.1)
U_c	Set of upgradable edges in G_c (Section 5.2)
$length(p, R)$	Length of path p after upgrade R (Section 5.2)
R_{max}	Maximum improvement set (Section 5.3)
$I(R_{max})$	Total weight improvement in R_{max} (Section 5.3)

Table 1 Notation

R does not exceed budget B and (ii) the summed shortest path distances across source-destination pairs in Q in the updated network is minimized. Formally, the output R of BUP is the result of the following optimization problem:

$$\begin{aligned} & \underset{R \subseteq U}{\operatorname{argmin}} && \sum_{i=1}^{|Q|} SP(s_i, t_i, R) \\ & \text{subject to} && \sum_{e \in R} e.c \leq B \end{aligned}$$

where $SP(s_i, t_i, R)$ is the (traveling time along the) new shortest path between s_i and t_i when edges in R are upgraded. If there are multiple subsets R that abide by the resource constraint and lead to the same summed traveling time for pairs in Q , BUP reports the one with the smallest total cost (i.e., the smallest $\sum_{e \in R} e.c \leq B$). Note that the above definition refers to the general (multiple-pair) BUP. The degenerate case where $|Q| = 1$ corresponds to the single-pair variant. We make the distinction because the single-pair case allows for specific processing techniques.

We now define some terms and establish conventions. Any subset $R \subseteq U$ is called a *plan*. The total cost of R is denoted by $C(R)$, i.e., $C(R) = \sum_{e \in R} e.c$. If $C(R)$ is no greater than B , we say that R is a *permissible* plan. For brevity, we call *length* of a path the traveling time along it. For ease of presentation, we use $SP(s_i, t_i, R)$ to refer both to the shortest path (between s_i and t_i in the updated network) and to its length, depending on the context. Table 1 summarizes frequently used notation.

To provide an idea about the difficulty of the problem, and to identify directions to tackle it, we consider a straightforward BUP processing method. A naïve approach is to consider all possible subsets of U . For each subset R , we check whether it is permissible, and if so, we *evaluate* it. That is, we compute all shortest paths $SP(s_i, t_i, R)$ (for $1 \leq i \leq |Q|$) when the edges in R (and only those) are upgraded. After considering all possible subsets, we report the permissible plan R that leads to the smallest summed shortest path distance.

The number of all possible subsets of U is $2^{|U|}$ (technically, the set of all subsets of U is called *power-set*). The problem of the naïve approach is that it needs to examine an excessive number of alternative plans, and for each (permissible) of them, to perform $|Q|$ shortest path computations. To give an example, if $|U| = 20$, the number of possible plans is 1,048,576, which are too many to even enumerate, let alone to evaluate.

In the following, we center our efforts on a twofold objective, i.e., we aim to reduce (i) the number of evaluated plans and (ii) the cost to evaluate each of them. We refer to item (i) as the *search space* of the problem. Item (ii) is bound to the cost of shortest path search and is directly dependent on the size of the graph (which we aim to reduce). Towards this dual goal, we propose graph reduction techniques in Section 4 and elaborate algorithms in Sections 5 and 6 (for single-pair and multiple-pair BUP, respectively). Graph reduction techniques assist towards both our design goals, while our BUP algorithms are targeted at search space reduction in particular (i.e., limiting the number of evaluated plans).

4 Graph-size Reduction Techniques

In this section, we propose two orthogonal methods to reduce G into a concise graph G_c , which is *BUP-equivalent* to G , i.e., the BUP solution R on (the much smaller) G_c is guaranteed to be the solution of BUP on the original network G . The first method is graph shrinking by edge pruning. The second is a resource constraint preserver technique that abstracts the remaining part of G (after pruning) into a concise graph which is BUP-equivalent to the original G . We present the general (multiple-pair) version of the methods, which apply directly to single-pair BUP.

4.1 Graph Shrinking via Edge Pruning

Our intuition is that any edge (upgradable or not) that lies too far from sources and destinations in Q cannot affect BUP processing and, thus, is safe to *prune*, i.e., to remove from G . We start with two important lemmas.

Lemma 1 *Let R be the BUP result and set $\{SP(s_i, t_i, R) | 1 \leq i \leq |Q|\}$ be the achieved shortest paths in the updated network. R includes only upgradable edges along the paths in $\{SP(s_i, t_i, R)\}$.*

Proof The lemma is based on our problem definition, and specifically on the fact that among permissible plans that lead to the same (minimal) summed distance for pairs in Q , BUP reports the lowest-cost one. We prove it by contradiction. Suppose that the BUP result R includes an upgradable edge e which is not along any $SP(s_i, t_i, R)$ for $1 \leq i \leq |Q|$. If we apply upgrade set $R - \{e\}$, the shortest paths for Q will pass through exactly the same edges as with R , and we will have achieved the same summed shortest path distance at a cost reduced by $e.c$, which contradicts the hypothesis that R is the BUP result.

Lemma 2 *Let R be the BUP result in G . Any subgraph of G that includes all the edges along any $SP(s_i, t_i, R)$ is BUP-equivalent to G .*

Proof Consider the subgraph G_{sp} of G that comprises *only* the union of (upgradable and non-upgradable) edges along all paths $SP(s_i, t_i, R)$ for $1 \leq i \leq |Q|$. A direct implication of Lemma 1 is that if we solved BUP on G_{sp} , we would derive the same result R . In turn, this means that any subgraph of G that is a supergraph of G_{sp} is BUP-equivalent to G .

Lemma 2 asserts that we can safely prune any edge, upgradable or not, that does not belong to any $SP(s_i, t_i, R)$ (where R is the BUP solution). We show how edges can be pruned safely, without knowing R in advance.

Consider the *fully upgraded* network G , i.e., where all edges in U are upgraded. $SP(v_i, v_j, U)$ denotes the distance between a pair of nodes v_i, v_j in this graph. By definition, $SP(v_i, v_j, U)$ is the lower bound of the distance between v_i and v_j after any possible upgrade plan $R \subseteq U$.

Let T be the summed lengths of $SP(s_i, t_i, R)$ for $1 \leq i \leq |Q|$, where R is the BUP solution, and assume that we *somehow* know T in advance. We will show that certain edges (be them upgradable or not) lie too far from sources and destinations in Q to belong to any of the final shortest paths $SP(s_i, t_i, R)$, and are therefore safe to prune.

Lemma 3 *It is safe to prune every edge $e = (v_x, v_y)$ if for each $\langle s_i, t_i \rangle$ in Q :*

(i) $SP(s_i, v_x, U) + w + SP(v_y, t_i, U) + S_i > T$ **and**

(ii) $SP(s_i, v_y, U) + w + SP(v_x, t_i, U) + S_i > T$

where w equals $e.w'$ if e is upgradable, or simply $e.w$ if e is not upgradable, and $S_i = \sum_{k=1, k \neq i}^{|Q|} SP(s_k, t_k, U)$.

Proof The shortest possible path between s_i and t_i that passes through edge e is the one that corresponds to a fully upgraded G . The length of that path is either $SP(s_i, v_x, U) + w + SP(v_y, t_i, U)$ or $SP(s_i, v_y, U) + w + SP(v_x, t_i, U)$, whichever is smaller; let the smaller of them be T_i . S_i , on the other hand, is the lower bound (under any upgrade plan) of summed lengths for all paths $\langle s_k, t_k \rangle \in Q$ for $k \neq i$. Thus, the sum of T_i and S_i is a lower bound of the total summed distance if $SP(s_i, t_i, R)$ were to pass through e (where R is the final BUP result). If the sum (i.e., $T_i + S_i$) is greater than T for every source-destination pair in Q , edge e cannot belong to any final shortest path $SP(s_i, t_i, R)$ and can be safely pruned.

Value T is not known in advance. However, Lemma 3 can be applied if T is replaced by any number that is greater than or equal to T . The closer that number to T , the more effective the lemma. We use $\sum_{i=1}^{|Q|} SP(s_i, t_i, R_{temp})$ instead of T , where R_{temp} is a permissible (suboptimal) plan that leads to sufficiently short distances for the source-destination pairs in Q .

Effective R_{temp} selection: To derive R_{temp} quickly and effectively, we first compute the shortest paths $SP(s_i, t_i, U)$ in the fully upgraded graph. Next, we form R_{temp} using only the upgradable edges included in those paths. If R_{temp} exceeds the resource constraint, we execute a knapsack algorithm [8] to derive the subset of R_{temp} that achieves the minimum sum of weights along paths $SP(s_i, t_i, R_{temp})$ without violating B . Note that this is different from a BUP problem, because we essentially fix the paths to $SP(s_i, t_i, R_{temp})$ for $1 \leq i \leq |Q|$ and consider upgradable edges *along the specific paths only*. The result of the knapsack algorithm is used as the R_{temp} for pruning.

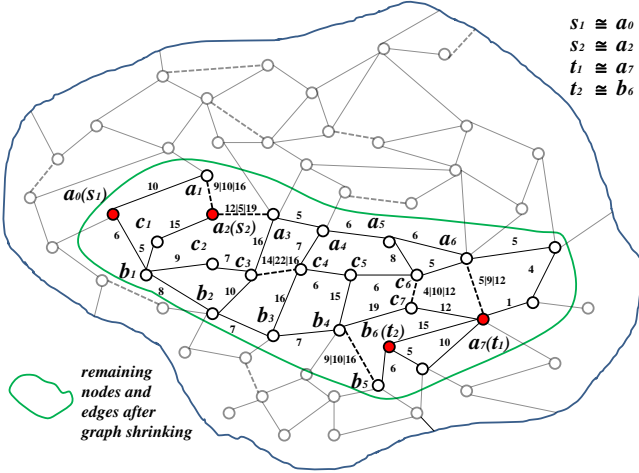


Fig. 2 Example of edge pruning

Figure 2 continues the running example of Figure 1. Assuming a weight of 15 units for every edge whose weight is not explicitly illustrated, and an R_{temp} that achieves a summed path length of 109, Lemma 3 prunes every edge out of the inner (green-border) closed curve.

Implementation: To implement pruning, for each query pair $\langle s_i, t_i \rangle$ we perform two Dijkstra expansions [8] from s_i and t_i on the original graph. Each expansion reaches up to distance $SP(s_i, t_i, \emptyset)$ from its start node (s_i and t_i , respectively). All edges that are encountered by both expansions for some pair $\langle s_i, t_i \rangle \in Q$, are kept in a temporary set and the rest discarded right away. Each of the edges in the temporary set is subsequently checked against Lemma 3 and pruned (or not) accordingly.

4.2 Resource Constraint Preserver

In this section, we propose the *resource constraint preserver* technique, which transforms the remaining part of G (after pruning) into a concise graph G_c that is BUP-equivalent to G , i.e., a much smaller graph whose BUP solution (for the same Q, B input) is guaranteed to be identical to the original road network. The concepts of *key nodes* and *plain paths* are central to this technique.

Definition 1 Key node. A node $v \in V$ is a key node iff it is a source or destination node in Q , or end-node of an upgradable edge.

Definition 2 Plain path. A path is plain if it does not include any key nodes (except for its very first and very last nodes).

We construct the network abstraction G_c as follows. First, we compute the *shortest plain path* for any pair of key nodes. The shortest plain path between key nodes v_i and v_j

is the shortest among the plain paths that connect them. Computing this path can be done using any standard shortest path algorithm, by treating key nodes other than v_i and v_j as non-existent (thus preventing the reported path from including any intermediate key node). The second step to produce G_c is to replace each shortest plain path by a *virtual edge*, whose weight is equal to the length of the path. The edge set of G_c comprises only virtual and upgradable edges; the non-upgradable edges of the original graph are discarded. The node set of G_c includes only key nodes.

Lemma 4 G_c is BUP-equivalent to the original network G .

Proof Let R be the BUP solution in G . Consider the sequence of key nodes in path $SP(s_i, t_i, R)$ for some source-destination pair in Q , in order of appearance (from s_i to t_i). For every pair of consecutive key nodes v_x, v_y in this sequence, either v_x, v_y are the end-nodes of the same upgradable edge, or they are connected by a plain path. In the latter case, that plain path between v_x, v_y is also the shortest (by definition, every sub-path of a shortest path, is the shortest path between the intermediate nodes it connects). Thus, $SP(s_i, t_i, R)$ is a sequence of upgradable edges and shortest plain paths. Since G_c preserves the upgradable edges and includes all shortest plain paths between key nodes (in the form of equivalent virtual edges), it contains all edges comprising $SP(s_i, t_i, R)$. Hence, by Lemma 2, G_c is BUP-equivalent to G .

Further shrinking: If the majority of edges are not upgradable, the preserver method will reduce the graph size. However, creating a fully connected graph among key nodes introduces many virtual edges, most of which are unnecessary. To cure the problem, we apply Lemma 3 to each virtual edge before inclusion into G_c and prune it if the lemma permits.

Implementation: To accelerate the construction of G_c , we incorporate Lemma 3 into the computation of shortest plain paths. Specifically, for each key node v_x we perform a Dijkstra search (with source at v_x). When another key node v_y is encountered (i.e., popped by the Dijkstra heap), we add a virtual edge between v_x and v_y to G_c . However, we do not expand v_y (i.e., we do not push into the heap the adjacent nodes of v_y) so as to ensure plain paths. For a specific pair $\langle s_i, t_i \rangle \in Q$, let M_i be the smallest of $SP(s_i, v_x, U)$ and $SP(v_x, t_i, U)$ (both these values are known since the pruning stage). For what $\langle s_i, t_i \rangle$ is concerned, the Dijkstra search can safely terminate if it has reached up to distance $SP(s_i, t_i, \emptyset) - M_i$ from v_x (any virtual edge longer than that threshold is useless according to Lemma 3). Therefore, taking into consideration all pairs in Q , the Dijkstra search only needs to reach up to $\max_{1 \leq i \leq |Q|} \{SP(s_i, t_i, \emptyset) - M_i\}$ units away from v_x . The single-pair BUP allows for an even tighter bound than directly applying the above technique (specifically, if $\langle s, t \rangle$ is the query pair, the search only needs to reach up to $SP(s, t, R_{temp}) - M$ units from v_x) – we refer the reader to [23] for the details.

Figure 3 shows the G_c abstraction derived in our running example by the resource constraint preserver technique.

5 BUP Processing for Single Query Pair

In this section, we consider the basic version of BUP problem where there is just one query pair (single-pair BUP); let it be pair $\langle s, t \rangle$. We present algorithms to compute

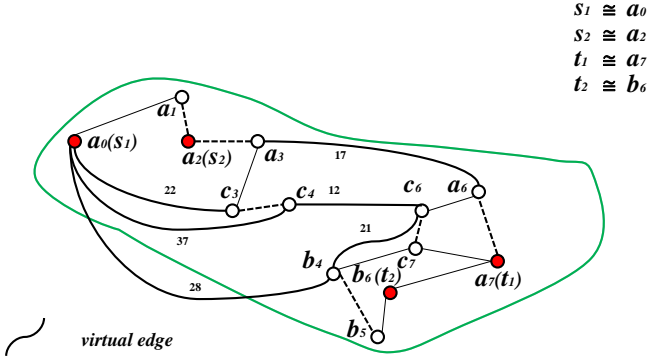


Fig. 3 The resulting graph G_c

the BUP solution on G_c , i.e., the graph resulting after the application of the edge pruning and resource preserver techniques from Section 4. G_c includes upgradable and virtual edges. For brevity, in the following we refer to virtual edges simply as edges. We denote by U_c the set of upgradable edges in G_c (since some edges in U have been pruned, $U_c \subseteq U$).

Even with a smaller set of candidate edges for upgrade, the approach of evaluating arbitrary subsets of U_c is not only impractical, but not very meaningful either. That is, Lemma 1 suggests that the BUP solution R includes only upgradable edges along the shortest path from s to t (in the updated network). If candidate plans (subsets of U_c) were arbitrarily chosen for evaluation, in the majority of cases, their upgradable edges would fall at random and irrelevant locations, rather than on the shortest path from s to t . This observation motivates our processing methodology, which is path-centered.

Our approach is to iteratively compute alternative paths (from s to t) in increasing order of length, and evaluate them in this order. We distinguish three variants of this general approach, depending on which version of G_c is used for the incremental path exploration; it could be the original G_c , the fully upgraded G_c , or another version of G_c that we call *augmented*. Regardless of the underlying graph, we use the path ranking method of [30] to incrementally produce paths from s to t in increasing length order.

5.1 Augmented Graph Algorithm

Our first technique relies on the *augmented* version of G_c , denoted as $A(G_c)$, to which it also owes its name, i.e., *Augmented Graph* algorithm (AG). The augmented graph has the same node set as G_c , but its edge set is a superset of G_c . Specifically, every edge e in G_c becomes an edge in $A(G_c)$, retaining its original weight $e.w$ (be it upgradable or not). Additionally, for every $e \in U_c$, the augmented graph also includes a second edge e' between the same end-nodes as e , but with weight equal to $e.w'$ (i.e., the new weight if e is upgraded).

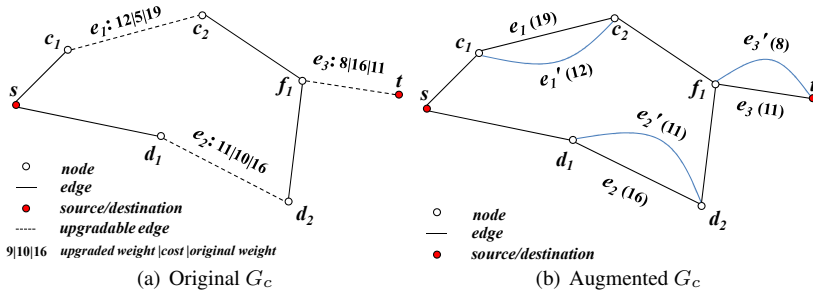


Fig. 4 Augmented graph example

Figure 4 gives an example, showing the original G_c on the left and its augmented version $A(G_c)$ on the right. For the sake of the example, assume that non-upgradable edges have a unit weight. All edges of G_c appear in $A(G_c)$ with their original weights. Since edges e_1, e_2, e_3 are upgradable in G_c , graph $A(G_c)$ additionally includes e'_1, e'_2, e'_3 with the respective upgraded weights (shown next to the edge labels).

AG calls the path ranking algorithm of [30] in $A(G_c)$, and iteratively examines paths in increasing length order. In our example, assume that the budget is $B = 20$. The shortest path in $A(G_c)$ is $p_1 = \{s, d_1, d_2, f_1, t\}$ via e'_2 and e'_3 (both are upgraded links). The length of p_1 is 21. It passes via upgraded edges e'_2 and e'_3 , thus requiring a total cost of $e_2.c + e_3.c = 26$. That cost exceeds B and the path is ignored. The path ranking algorithm is probed again to produce the next best path, that is $p_2 = \{s, c_1, c_2, f_1, t\}$ via e'_1 and e'_3 . Its length is 22, but its cost $e_1.c + e_3.c = 21$ exceeds B . Hence, this path is ignored too, and the path ranking algorithm is probed to produce the next best path, which is $p_3 = \{s, d_1, d_2, f_1, t\}$ via e'_2 and e_3 (upgraded and non-upgraded link, respectively). Its length is 24. The path passes via one upgraded edge, e'_2 , which means that the total path cost is $e_2.c = 10$. That is within our budget, and AG terminates here with result $R = \{e_2\}$, achieving $SP(s, t, R) = 24$.

Observe that every path p output by path ranking in $A(G_c)$ corresponds to a specific upgrade plan, namely, to the plan that includes all upgraded links e' that p passes from. Consider a pair of paths p and p' in $A(G_c)$ that are identical, except that p passes via edge e , while p' passes from that edge's upgraded counterpart e' . For what path ranking in $A(G_c)$ is concerned, these are two different paths (since e and e' are modeled as different edges) corresponding to different upgrade plans.

Correctness: Path ranking, if probed enough times, will output *all possible paths* between s and t in $A(G_c)$. Hence, $SP(s, t, R)$ is guaranteed to be among them (where R is the BUP result), as long as AG does not terminate prematurely. AG stops probing the path ranking process when the latter outputs the first path p that abides by resource constraint B . Since path ranking outputs paths in increasing length order, p is guaranteed to be the shortest permissible path, i.e., to coincide with $SP(s, t, R)$.

Note that, in the worst case, path ranking will need to output all possible paths between s and t in $A(G_c)$. This is still preferable to evaluating all subsets of U_c , because AG essentially considers only combinations of upgradable edges *along acyclic paths*

from s to t . For example, in Figure 4(b), path ranking would never output a path that includes both e'_1 and e'_2 , while a blind evaluation of U_c subsets would consider (and waste computations for the evaluation of) plan $\{e'_1, e'_2\}$.

Discussion: In Section 2.3 we described the restricted shortest path problem (RSP) where edges are associated with a cost and a delay, and the goal is to compute (for a single source-destination pair) the smallest-cost path that does not exceed a certain time limit. This means that, if we first apply our pruning/shrinking techniques from Section 4 (in order to ensure scalability) and use the augmented version of the remainder graph, i.e., $A(G_c)$, we may apply RSP methods on $A(G_c)$ by treating the upgrade cost of each upgradable edge as its “delay” and the budget as the “time limit”. That would be very similar to AG (since RSP methods also rely on path ranking), albeit the majority of RSP algorithms target at approximate solutions. A note is that this discussion applies only to single-pair BUP, because RSP approaches consider a single source-destination pair.

AG is conceptually simple. The drawback, however, is that the augmented graph $A(G_c)$ is larger than G_c , due to containing two versions of every edge in U_c . In turn, this implies larger processing cost for the path ranking component. This motivates our next two BUP methods, which do not increase the number of edges in G_c .

5.2 Fully Upgraded Graph Algorithm

In this section, we present the *Fully Upgraded Graph* algorithm (FG). FG runs path ranking on the fully upgraded G_c , where all edges $e \in U_c$ have their upgraded (reduced) weight $e.w'$. On this graph, each path p from s to t has the minimum possible length under any upgrade plan (permissible or not); we denote this length as $length(p, U_c)$ and use it as a lower bound for the length of p under any plan.

For every path p output by the path ranking algorithm, if the summed cost along its upgradable edges exceeds B , we perform a process similar to the computation of R_{temp} in Section 4.1. That is, we execute a knapsack algorithm among the upgradable edges *along the specific path*, and report their subset that minimizes the length of p under budget constraint B . Let R_p be the result of the knapsack algorithm, and $length(p, R_p)$ be the length of p under plan R_p . The knapsack process asserts that R_p is permissible, and therefore $length(p, R_p)$ is an achievable traveling time from s to t . In a nutshell, FG treats each path p output by path ranking as an umbrella construct representing all possible upgrade plans among the upgradable edges in p , and from these plans it keeps the best permissible plan, R_p .

While the path ranking algorithm iteratively reports new paths, we keep track of the one, say, p^* (and the respective R_{p^*} set) that achieves the smallest $length(p, R_p)$ among all paths considered so far.¹ Once path ranking reports a path p whose length (on the fully upgraded graph) is greater than $length(p^*, R_{p^*})$, FG terminates with R_{p^*} as the BUP result (achieving length $SP(s, t, R_{p^*}) = length(p^*, R_{p^*})$).

Referring to our example in Figure 4, the fully upgraded G_c would look like Figure 4(a) with weights 12, 11, and 8 for edges e_1, e_2 , and e_3 , respectively. Path ranking would first report path $p_1 = \{s, d_1, d_2, f_1, t\}$ with length 21. A knapsack algorithm on its set of upgradable edges (i.e., on set $\{e_2, e_3\}$) with resource constraint

¹ In case of tie between two alternative paths, we keep as p^* the one with the smallest $C(R_{p^*})$.

$B = 20$ reports $R_{p_1} = \{e_2\}$ and $length(p_1, R_{p_1}) = 24$. The next path output by path ranking is $p_2 = \{s, c_1, c_2, f_1, t\}$ with length 22. If that length were larger than $length(p_1, R_{p_1}) = 24$, FG would terminate. This is not the case, so a knapsack process on the upgradable edges along p_2 reports that $R_{p_2} = \{e_1\}$ with $length(p_2, R_{p_2}) = 25$. Path ranking is probed again, but reports NULL (i.e., all paths from s to t have been output) and FG terminates with result $R = R_{p_1} = \{e_2\}$.

Correctness: If probed enough times, path ranking in the fully upgraded G_c will report all possible paths from s to t on this graph. The paths are reported in increasing $length(p, U_c)$ order. Our termination condition guarantees that all paths not yet output by path ranking have $length(p, U_c)$ greater than $length(p^*, R_{p^*})$, and therefore could not lead to a shorter traveling time between s and t under any plan (permissible or not).

A note here is that every path output by path ranking in the augmented graph $A(G_c)$ (in Section 5.1) corresponds to an upgrade plan. In FG, instead, each path p output by path ranking in the fully upgraded G_c leads to the consideration of all possible upgrade plans along p (this is essentially what the knapsack-modeled derivation of R_p does).

5.3 Original Graph Algorithm

The *Original Graph* algorithm (OG) executes path ranking in the original G_c , i.e., assuming that no edge is upgraded. For every path p output by path ranking, it solves a knapsack problem to derive the subset R_p of the upgradable edges along p that achieves the minimum path length $length(p, R_p)$ without violating the resource constraint B . While new paths are being output by path ranking, OG maintains the path p^* (and the respective R_{p^*} set) that achieves the smallest $length(p, R_p)$ so far.

Regarding the termination condition of OG, we introduce the *maximum improvement set* R_{max} . Among all permissible subsets of U_c , R_{max} is the one that achieves the maximum total weight reduction (regardless of where the contained edges are located or whether they contribute to shorten the traveling time from s to t). We denote the total weight reduction achieved by R_{max} as $I(R_{max})$. The latter serves as an upper bound for the length reduction in *any* path under *any* permissible plan.

In the example of Figure 4(a), to derive R_{max} (and $I(R_{max})$), we solve a knapsack problem on $U_c = \{e_1, e_2, e_3\}$. Their individual weight reductions (i.e., values $e.w' - e.w$) are 7, 5, 3 and their costs ($e.c$) are 5, 10, 16. The knapsack problem uses limit $B = 20$ for the total cost. The result is $R_{max} = \{e_1, e_2\}$ with total reduction $I(R_{max}) = 12$.

Returning to OG execution, let p be the next best path output by path ranking in the original (un-updated) G_c . Under any permissible upgrade plan, the length of p can be reduced at maximum by $I(R_{max})$, i.e., under any upgrade plan the new length of p cannot be lower than $length(p, \emptyset) - I(R_{max})$. If the latter value is greater than $length(p^*, R_{p^*})$, OG can safely terminate with BUP result $R = R_{p^*}$.

In the original G_c in Figure 4(a), edges e_1, e_2 , and e_3 have weights 19, 16, and 11, respectively. Path ranking would first report path $p_1 = \{s, d_1, d_2, f_1, t\}$ with length 29. For p_1 we derive (via a knapsack execution on its upgradable edges) $R_{p_1} = \{e_2\}$ and $length(p_1, R_{p_1}) = 24$. The next path output by path ranking is $p_2 = \{s, c_1, c_2, f_1, t\}$ with length 32. Before we even solve a knapsack problem for p_2 , we know that under any permissible plan its length cannot drop below $length(p_2, \emptyset) - I(R_{max}) = 32 - 12 = 20$. If that last value were greater than $length(p_1, R_{p_1}) = 24$, OG would

terminate. This is not the case, so a knapsack process on p_2 reports $R_{p_2} = \{e_1\}$ with $length(p_2, R_{p_2}) = 25$. Path ranking is probed again, reports NULL (since all paths from s to t have been output), and OG terminates with BUP result $R = R_{p_1} = \{e_2\}$.

Correctness: The correctness of OG relies on similar principles to FG. First, path ranking, if probed enough times, will output all paths from s to t . For each of these paths p , OG computes the best permissible plan along its edges, i.e., R_p . Therefore, it will discover the optimal plan R at some point, unless terminated prematurely. Since path ranking in the original G_c outputs paths p in increasing $length(p, \emptyset)$ order, the termination condition of OG guarantees that all paths not yet output, even if improved to the maximum possible degree (i.e., $I(R_{max})$), cannot become shorter than p^* .

6 BUP Processing for Multiple Query Pairs

In this section, we present BUP algorithms for the multiple-pair version of the problem. The challenge and main difference from the single-pair case is that we have to evaluate *path combinations* instead of *paths*. We propose three algorithms, which, similar to the single-pair solutions, work on G_c , i.e., the graph resulting after the edge pruning and resource preserver techniques from Section 4.

6.1 Augmented Graph Algorithm

The first algorithm works on the augmented version of G_c (i.e., $A(G_c)$), constructed exactly like in Section 5.1. In the multiple-pair case, for each source-destination pair, paths are ranked as well, but instead of considering paths from a single query pair we need to consider path combinations formed by paths from multiple query pairs.

For better demonstration, we introduce an important structure, the *path ranking list*. Continuing our running example in Figure 3, suppose that for each query pair $\langle s_1, t_1 \rangle$ and $\langle s_2, t_2 \rangle$ we have already produced and ranked all possible paths from their source to their destination (on $A(G_c)$) as shown in Table 2. Paths for pair $\langle s_1, t_1 \rangle$ are denoted as p_{1-1}, p_{1-2}, \dots , ordered in increasing length. Similarly, paths for pair $\langle s_2, t_2 \rangle$ are denoted as p_{2-1}, p_{2-2}, \dots . For each path p , we maintain its length, the upgrade cost, and the corresponding upgraded edge set U_p .² We map the path information onto two lists, one for each pair. In Figure 5, the left list corresponds to query pair $\langle s_1, t_1 \rangle$ with 11 paths, illustrated as black dots with their lengths shown at the center, and ranked by length. The right list corresponds to query pair $\langle s_2, t_2 \rangle$ with 3 paths. For brevity, only the first few paths in each lists are illustrated. We call the two lists *path ranking lists*.

A key concept in multiple-pair BUP processing is the path combination.

Definition 3 Path combination. A *path combination* is a set of paths which includes $|Q|$ elements. Each element is a path (not necessarily the shortest) from s_i to t_i for some query pair. There is exactly one element (path) for each query pair $\langle s_i, t_i \rangle \in Q$.

² Note that in $A(G_c)$ it is unambiguous whether the path passes via an upgraded or un-upgraded link between two nodes – similarly, it is clear how to measure the upgrade cost.

No.	Dis.	Path	Cost	U_p
p_{1-1}	53	$\{s_1, a_1, a_2, a_3, a_6, t_1\}$	24	$(a_1, a_2), (a_2, a_3), (a_6, a_7)$
p_{1-2}	58	$\{s_1, b_4, b_5, b_6, t_1\}$	10	(b_4, b_5)
p_{1-3}	58	$\{s_1, c_3, c_4, c_6, a_6, t_1\}$	31	$(c_3, c_4), (a_6, a_7)$
p_{1-4}	59	$\{s_1, c_4, c_6, a_6, t_1\}$	9	(a_6, a_7)
p_{1-5}	59	$\{s_1, b_4, c_7, t_1\}$	0	\emptyset
p_{1-6}	59	$\{s_1, b_4, c_6, a_6, t_1\}$	9	(a_6, a_7)
p_{1-7}	60	$\{s_1, c_3, a_3, a_6, t_1\}$	9	(a_6, a_7)
p_{1-8}	60	$\{s_1, c_3, c_4, c_6, t_1\}$	9	(a_6, a_7)
p_{1-9}	60	$\{s_1, a_1, a_2, a_3, a_6, t_1\}$	14	$(a_2, a_3), (a_6, a_7)$
p_{1-10}	60	$\{s_1, a_1, a_2, a_3, a_6, t_1\}$	19	$(a_1, a_2), (a_6, a_7)$
p_{1-11}	60	$\{s_1, a_1, a_2, a_3, a_6, t_1\}$	15	$(a_1, a_2), (a_2, a_3)$
p_{2-1}	49	$\{s_2, a_3, a_6, a_7, t_2\}$	14	$(a_2, a_3), (a_6, a_7)$
p_{2-2}	56	$\{s_2, a_3, a_6, a_7, t_2\}$	5	(a_2, a_3)
p_{2-3}	56	$\{s_2, a_3, a_6, a_7, t_2\}$	9	(a_6, a_7)

Table 2 Ranked paths for the two source-destination pairs in running example

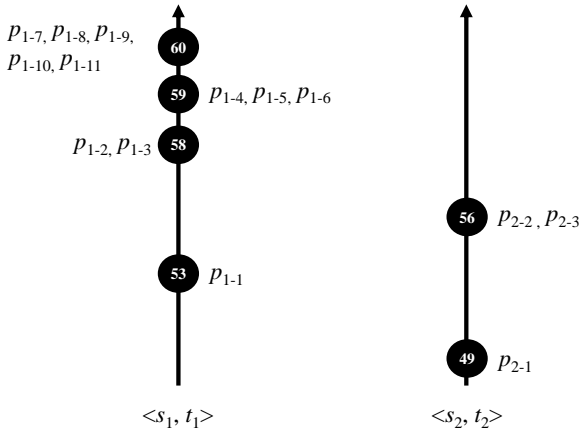


Fig. 5 Path ranking lists in running example

For example, path set $\{p_{1-1}, p_{2-2}\}$ is a path combination, but $\{p_{1-9}, p_{1-11}\}$ is not. Note that a path combination essentially indicates one position (path) in each of the $|Q|$ path ranking lists. Every path combination is associated with three pieces of information: summed path length, upgraded edge set and upgrade cost. The first is the sum of lengths across its included paths, the second is the union of their upgradable edges, and the third is the sum of costs in the combination's upgraded edge set.

The BUP result is a path combination. In particular, it is the path combination that has the smallest summed length with an upgrade cost that does not exceed budget B . The crux of our processing method is to explore the path ranking lists in such a way that will produce path combinations in increasing summed length order. The first of these combinations that has upgrade cost no larger than B is the BUP result.

In the hypothetical scenario where the entire path ranking lists are available, their exploration (i.e., iterative identification of path combinations in increasing summed length order) can be done by the *best neighbor method* (BN) in [23]. We treat this algorithm as a black box, but give the intuition behind it. The smallest-cost combination corresponds to the first element in every path ranking list. The second smallest-cost combination can be found among the “descendants” of that first combination – each descendant replaces exactly one path in the original combination by the immediately next path in the corresponding path ranking list. In our example, the smallest-cost combination is $\{p_{1-1}, p_{2-1}\}$. Its descendants are combinations $\{p_{1-2}, p_{2-1}\}$ and $\{p_{1-1}, p_{2-2}\}$. The second smallest-cost combination is one of the two descendants (which can be determined by comparing their upgrade costs). The third smallest-cost combination is among the descendants of the first and second smallest-cost combinations, and so on. For large $|Q|$, organizing and managing the descendants is non-trivial. The BN approach in [23] performs that effectively and efficiently.

We could use the method of [30] to generate the entire path ranking lists and feed them to BN, but it is simply impractical to populate each list to the last path. Instead, we determine for each list a *ranking height* H_i . That is the maximum path length in that list that could participate in the BUP result. Essentially, H_i is used to truncate the i -th path ranking list (i.e., the list corresponding to pair $\langle s_i, t_i \rangle$) by keeping only paths up to length H_i . The following lemma shows how to determine H_i .

Lemma 5 H_i is the smallest of the following two values:

- (i) The length of the first path in the i -th list where the upgraded edge set is \emptyset ;
- (ii) The summed path length under plan R_{temp} minus the sum of the first path lengths in all other ranking lists, i.e., $\sum_{k=1}^{|Q|} SP(s_k, t_k, R_{temp}) - \sum_{k=1, k \neq i}^{|Q|} SP(s_k, t_k, U_c)$.

To see the validity of the lemma, value (i) corresponds to the first path in the i -th list that does not utilize any upgrade, and therefore searching the list any more (i.e., considering any longer paths between s_i and t_i) is unnecessary. Regarding value (ii), the first sum of the expression is the achieved summed length under (suboptimal) plan R_{temp} . The second sum is the smallest possible summed length for all other source-destination pairs (except $\langle s_i, t_i \rangle$) under any plan – the smallest possible length for each pair $\langle s_k, t_k \rangle$ is $SP(s_k, t_k, U_c)$ that corresponds to the fully upgraded G_c or, equivalently, to the first element in the k -th path ranking list. Essentially, any path in the i -th list with length greater than value (ii) is guaranteed to lead to a path combination with summed length greater than that achieved by R_{temp} and can hence not belong to the BUP solution.

In the running example, the summed path length under plan R_{temp} is 109. The lengths of p_{1-1} and p_{2-1} are 53 and 49, respectively. Thus value (ii) for the first list is $109-49=60$ and for the second list it is $109-53=56$. Focusing on the first list, value (i) corresponds to p_{1-5} (the smallest-length path with an empty upgraded edge set) with length 59, which prunes every path with a greater length (actually, that value is smaller than value (ii) and therefore it determines H_1). A point we must stress here is that, although p_{1-4} and p_{1-6} have the same length as value (i) they are both pruned because they have non-empty upgraded edge sets. The convention we establish is that in case a path’s length is equal to value (i) and its upgraded edge set is non-empty, the path is pruned. Hence, in our example p_{1-5} is kept in the first list, while p_{1-4} and p_{1-6} are discarded. This leaves four paths in the first list ($p_{1-1}, p_{1-2}, p_{1-3}, p_{1-5}$). For the

second list, value (ii) is smaller than value (i) and determines $H_2 = 56$, keeping in it 3 paths $(p_{2-1}, p_{2-2}, p_{2-3})$.

To enhance performance, we may further prune the path ranking lists by eliminating paths whose upgrade cost already exceeds B . In our example, paths p_{1-1} and p_{1-3} can be discarded from the first list because their costs are larger than $B = 20$. The remaining 2 paths in the first list and the 3 paths in the second list are input to BN. The latter outputs $\{p_{1-5}, p_{2-1}\}$ as the smallest-length path combination that abides by $B = 20$ (with summed length 108 and upgrade cost 14). The BUP result is the upgraded edge set of that combination, i.e., $R = \{(a_2, a_3), (a_6, a_7)\}$.

6.2 Fully Upgraded Graph Algorithm

Our second algorithm for multiple-pair BUP executes on the fully upgraded graph G_c . Specifically, for each source-destination pair in Q it produces a path ranking list by invoking the path ranking method in [30]. Let P be a path combination in the produced $|Q|$ lists, and U_P be the set of its upgradable edges. The summed path length in P corresponds to a lower bound of the summed length achieved by any possible upgrade plan along its edges (because P is computed on the fully upgraded graph). In this aspect, path combination P serves as an umbrella construct that represents all possible upgrade plans along its paths, i.e., it represents all possible upgrade plans that are subsets of U_P .

Again, it is impractical to produce entire path ranking lists. Instead, we truncate them using Lemma 5 as is. The validity of the lemma can be shown following the same lines as in Section 6.1, by taking into account that in the context of a fully upgraded G_c the summed length of a path combination is the lower bound of what is achievable via subsets of its upgradable edges.

We invoke BN on the truncated path ranking lists, but (unlike Section 6.1) we do not wait until a permissible path combination is output. Instead, for each path combination P encountered by BN (i.e., popped from its search heap) we run a knapsack algorithm on its upgradable edge set U_P in order to derive the permissible subset $R_P \subseteq U_P$ that achieves the smallest summed path length. Among all different path combinations encountered by BN, we maintain the permissible subset R_{P^*} that leads to the smallest summed length. The process terminates when the next path combination encountered by BN has summed length (in the fully upgraded G_c) greater than that achieved by R_{P^*} . Set R_{P^*} is output as the BUP result. Note that BN encounters path combinations in increasing summed path length order.

The rationale behind the termination condition is that all non-encountered path combinations have a summed length (in the fully upgraded G_c) greater than the one achieved by R_{P^*} . Because the summed path length of these combinations is already a lower bound of the sum achievable under any subset of their upgradable edge set, the termination condition guarantees that no better permissible plan (than R_{P^*}) can be found.

Returning to our running example, Table 3 shows the first few paths (sorted in ascending length order) in the fully upgraded G_c for each source-destination pair in Q . Figure 6 illustrates the corresponding path ranking lists.

Similar to Section 6.1, from Lemma 5 we get ranking heights $H_1 = 59$ and $H_2 = 56$. Thus, the pruned lists include $p_{1-1}, p_{1-2}, p_{1-3}, p_{1-5}$ and p_{2-1} , respectively. BN first encounters path combination $P_1 = \{p_{1-1}, p_{2-1}\}$ with upgradable

No.	Dis.	Path	Cost	U_p
p_{1-1}	53	$\{s_1, a_1, a_2, a_3, a_6, t_1\}$	24	$(a_1, a_2), (a_2, a_3), (a_6, a_7)$
p_{1-2}	58	$\{s_1, b_4, b_5, b_6, t_1\}$	10	(b_4, b_5)
p_{1-3}	58	$\{s_1, c_3, c_4, c_6, a_6, t_1\}$	31	$(c_3, c_4), (a_6, a_7)$
p_{1-4}	59	$\{s_1, c_4, c_6, a_6, t_1\}$	9	(a_6, a_7)
p_{1-5}	59	$\{s_1, b_4, c_7, t_1\}$	0	\emptyset
p_{1-6}	59	$\{s_1, b_4, c_6, a_6, t_1\}$	9	(a_6, a_7)
p_{1-7}	60	$\{s_1, c_3, a_3, a_6, t_1\}$	9	(a_6, a_7)
p_{2-1}	49	$\{s_2, a_3, a_6, a_7, t_2\}$	14	$(a_2, a_3), (a_6, a_7)$
p_{2-2}	62	$\{s_2, a_1, a_0, b_4, b_5, t_2\}$	20	$(a_2, a_1), (b_4, b_5)$

Table 3 Ranked paths for the two source-destination pairs in running example

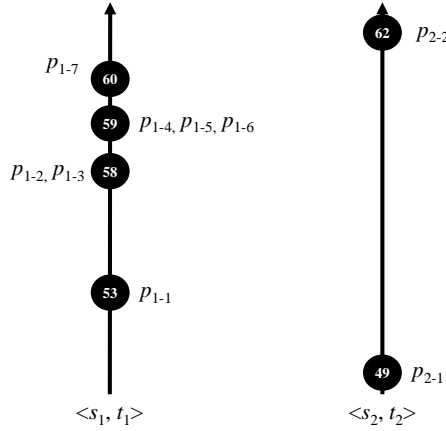


Fig. 6 Path ranking lists in running example

edge set $U_{P_1} = \{(a_1, a_2), (a_2, a_3), (a_6, a_7)\}$ and upgrade cost 24. A knapsack processing is performed to identify the best permissible subset of U_{P_1} , which is $R_{P_1} = \{(a_2, a_3), (a_6, a_7)\}$ with upgrade cost 14 and summed path length 109. The interim result R_{P^*} is initialized to R_{P_1} . The second path combination $P_2 = \{p_{1-2}, p_{2-1}\}$ and the third path combination $P_3 = \{p_{1-3}, p_{2-1}\}$ (after a knapsack execution for each) lead to permissible plans R_{P_2} and R_{P_3} that achieve no improvement of R_{P^*} . When the fourth path combination $P_4 = \{p_{1-5}, p_{2-1}\}$ is encountered by BN, its upgradable edge set $U_{P_4} = \{(a_2, a_3), (a_6, a_7)\}$ is already permissible (i.e., $R_{P_4} = U_{P_4}$) and, furthermore, it achieves a summed length of 108, which is smaller than the best so far (109). Thus, R_{P^*} is updated to R_{P_4} , i.e., to $\{(a_2, a_3), (a_6, a_7)\}$. At that point both path ranking lists are exhausted and BN terminates unable to produce more path combinations. The reported result is $R = R_{P^*} = \{(a_2, a_3), (a_6, a_7)\}$. Termination would also occur if BN encountered a path combination whose summed path length (in the fully upgraded G_c) was greater than that achieved by R_{P^*} .

6.3 Original Graph Algorithm

The third multiple-pair BUP method executes on the original graph G_c . Like in the other two methods, we form path ranking lists for each source-destination pair in Q , the difference being that the lists contain paths in the original graph. Let $I(R_{max})$ be the maximum possible reduction in summed path length achievable for any path combination in G_c (we describe how to compute an upper bound of $I(R_{max})$ later in this section). We use the following lemma to determine the ranking heights H_i (for $1 \leq i \leq |Q|$) and truncate the path ranking lists accordingly.

Lemma 6 *In the original G_c the ranking height H_i for the i -th path ranking list is equal to the summed path length under plan R_{temp} plus $I(R_{max})$ minus the sum of the first path lengths in all other ranking lists, i.e., $\sum_{k=1}^{|Q|} SP(s_k, t_k, R_{temp}) + I(R_{max}) - \sum_{k=1, k \neq i}^{|Q|} SP(s_k, t_k, \emptyset)$.*

Proof Let SP be (the length of) a path in the i -th path ranking list. The summed path length (in the original graph) of every path combination that includes SP is at least $SP + \sum_{k=1, k \neq i}^{|Q|} SP(s_k, t_k, \emptyset)$. Since the summed length of any path combination can be reduced by a maximum of $I(R_{max})$ units, the previous statement implies that the summed length achievable along any path combination that includes SP is lower bounded by $SP + \sum_{k=1, k \neq i}^{|Q|} SP(s_k, t_k, \emptyset) - I(R_{max})$. If the former quantity is greater than the summed path length under plan R_{temp} then SP cannot belong to the BUP result³ – that is, SP can be pruned if $SP + \sum_{k=1, k \neq i}^{|Q|} SP(s_k, t_k, \emptyset) - I(R_{max}) > \sum_{k=1}^{|Q|} SP(s_k, t_k, R_{temp})$ or equivalently if $SP > \sum_{k=1}^{|Q|} SP(s_k, t_k, R_{temp}) + I(R_{max}) - \sum_{k=1, k \neq i}^{|Q|} SP(s_k, t_k, \emptyset)$. The expression on the right side of the inequality is H_i .

We feed the truncated path ranking lists to BN. For every encountered path combination P , we compute (via a knapsack execution) the permissible subset R_P of its upgradable edges that leads to the smallest summed path length. As BN encounters new path combinations P and computes subset R_P for each of them, we maintain as R_{P^*} the best among these subsets. The algorithm terminates with R_{P^*} as the BUP result when BN encounters a path combination whose summed path length (in the original graph) minus $I(R_{max})$ is greater than the summed length achieved under R_{P^*} .

Recall that BN encounters path combinations in increasing summed path length order. The rationale behind the termination condition is that any non-encountered path combination will have summed length (in the original graph) even greater than the one encountered at the time of termination, and therefore even if the maximum improvement $I(R_{max})$ is possible, it can still not lead to a smaller summed length than R_{P^*} .

Effective $I(R_{max})$ computation: First, we clarify that we do not compute the exact $I(R_{max})$ but an upper bound of it. It can be easily seen that using any upper bound of $I(R_{max})$ instead of its exact value upholds the correctness of the above BUP algorithm. For simplicity, in the following we will still refer to the upper bound as $I(R_{max})$.

Unlike the single-pair case, we cannot directly solve a knapsack problem on U_c to derive $I(R_{max})$, because now an upgradable edge may be shared by multiple paths in a

³ Note that here SP is an umbrella concept covering any path that passes via the same nodes as SP , regardless of which edges are chosen for upgrade.

path combination. Unless we know how many paths in the combination are passing via an upgradable edge e , the reduction (in summed path length) lent if we upgrade the edge cannot be determined (i.e., it may be a multiple of $e.w - e.w'$).

The basic idea is to determine for every upgradable edge $e \in U_c$ the maximum number $e.max_no$ of query paths (in the updated network) that could potentially pass via e . In knapsack execution, we use the product of $e.max_no$ and $e.w - e.w'$ as the improvement e could lend us, at the expense of cost $e.c$. Let R_{max} be the result of knapsack processing on this “modified” input. $I(R_{max})$ is set to the summed path length achieved under R_{max} .

The last question regards the computation of $e.max_no$ for each upgradable edge $e \in U_c$. The following lemma shows how to derive it.

Lemma 7 Consider a source-destination pair $\langle s_i, t_i \rangle$ in Q . An upgradable edge $e = (v_x, v_y)$ could appear in the shortest path from s_i to t_i (in the updated network) only if:

(i) $SP(s_i, v_x, U_c) + e.w' + SP(v_y, t_i, U_c) + S_i \leq T$ or

(ii) $SP(s_i, v_y, U_c) + e.w' + SP(v_x, t_i, U_c) + S_i \leq T$

where $S_i = \sum_{k=1, k \neq i}^{|Q|} SP(s_k, t_k, U_c)$ and $T = \sum_{k=1}^{|Q|} SP(s_k, t_k, R_{temp})$.

The proof of Lemma 7 follows the same reasoning as Lemma 3 and is omitted⁴. Given an upgradable edge $e \in U_c$, we determine $e.max_no$ by applying Lemma 7 for every pair $\langle s_i, t_i \rangle$ in Q and counting how many of these pairs it could affect.

Table 4 and Figure 7 show the ranked paths and (truncated) path ranking lists, respectively, in the original G_c for our running example, assuming that $I(R_{max}) = 28$.

No.	Dis.	Path	Cost	U_p
p_{1-1}	59	$\{s_1, b_4, c_7, t_1\}$	0	\emptyset
p_{1-2}	65	$\{s_1, b_4, b_5, b_6, t_1\}$	10	(b_4, b_5)
p_{1-3}	66	$\{s_1, c_4, c_6, a_6, t_1\}$	9	(a_6, a_7)
p_{1-4}	66	$\{s_1, b_4, c_6, a_6, t_1\}$	9	(a_6, a_7)
p_{1-5}	67	$\{s_1, c_3, c_4, c_6, a_6, t_1\}$	31	$(c_3, c_4), (a_6, a_7)$
p_{1-6}	67	$\{s_1, c_3, a_3, a_6, t_1\}$	9	(a_6, a_7)
p_{1-7}	74	$\{s_1, a_1, a_2, a_3, a_6, t_1\}$	24	$(a_1, a_2), (a_2, a_3), (a_6, a_7)$
p_{2-1}	63	$\{s_2, a_3, a_6, a_7, t_2\}$	14	$(a_2, a_3), (a_6, a_7)$

Table 4 Ranked paths for the two source-destination pairs in running example

The first path combination encountered by BN is $P_1 = \{p_{1-1}, p_{2-1}\}$. A knapsack execution on its upgradable edges outputs $R_{P_1} = \{(a_2, a_3), (a_6, a_7)\}$ with summed path length 108. The interim result R_{P^*} is initialized to R_{P_1} . Subsequent path combinations P_2 up to P_6 do not improve R_{P^*} . When $P_7 = \{p_{1-7}, p_{2-1}\}$ is encountered, the algorithm terminates because the summed path length in P_7 (in the original graph) is $74+63 = 137$, which even if reduced by $I(R_{max}) = 28$, would lead to summed length 109 that is worse (greater) than that achieved by the current R_{P^*} (i.e., $\{(a_2, a_3), (a_6, a_7)\}$). The latter is reported as the BUP result.

⁴ Note that Lemma 7 uses U_c instead of U because it is applied on G_c (not on G).

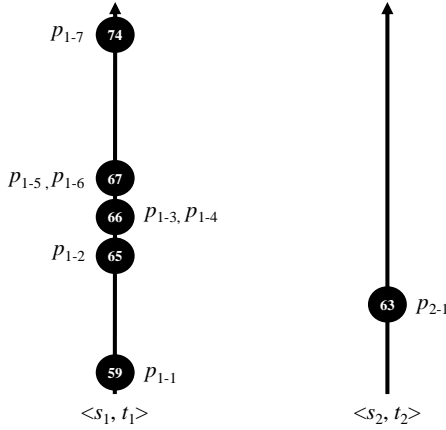


Fig. 7 Path ranking lists in running example

7 Experiments

In this section, we first experimentally evaluate the effectiveness of our graph-size reduction techniques (from Section 4). Then proceed to compare the efficiency of our processing algorithms (from Sections 5 and 6) – for each group of experiments, we first evaluate single-pair BUP processing and then multiple-pair BUP.

As default network G we use the road network of Germany (from www.maproom.psu.edu/dcw/), which has 28,867 nodes and 30,429 edges. We normalize the node coordinates into a $[0, 10, 000]^2$ space, and set the (original) weight of each edge to the Euclidean distance between its end-nodes. The diameter of the network (i.e., the maximum distance between any pair of nodes) is 14,383.

We study the impact of four parameters: (original) *path length* between source and destination; *upgrade ratio*; *resource ratio*; and *number of query pairs* (for multiple-pair BUP experiments). The upgrade ratio indicates the ratio of upgradable edges over their total number (i.e., $|U|/|E|$). By default, upgradable edges are selected randomly from E (although we also explore cases where edges along entire paths are chosen as upgradable). Their new weight is set to $e.w' = x \cdot e.w$, where x is a random number between 0.5 and 1. The upgrade cost is set to $e.c = y \cdot e.w$, where y is a random number from 0 to 1. The resource ratio indicates how strict the budget B is. Specifically, for each query we compute the sum of upgrade costs of all upgradable edges in the shortest path(s) from source(s) to destination(s) in the original network; let this sum be C . We set B to a fraction of this cost. The resource ratio equals B/C . In multiple-pair BUP, the $|Q|$ query pairs are chosen randomly among source-destination pairs of the desired path length. Table 5 shows the parameter values tested and their default (in bold). In every experiment, we vary one parameter and set the other three to their default. Each measurement is the average over 20 queries. We use an Intel Core 2 Duo CPU 2.40GHz with 2GB RAM and keep the networks in memory.

Parameter	Value Range
Path length	1000, 2000, 4000 , 6000, 8000
Upgrade ratio	0.04, 0.06, 0.08 , 0.1
Resource ratio	0.2, 0.4 , 0.6, 0.8
Number of query pairs	5, 10 , 15, 20, 25

Table 5 Experiment parameters

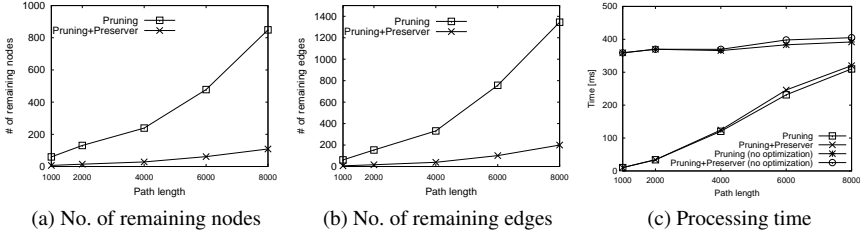


Fig. 8 Effect of path length (single pair)

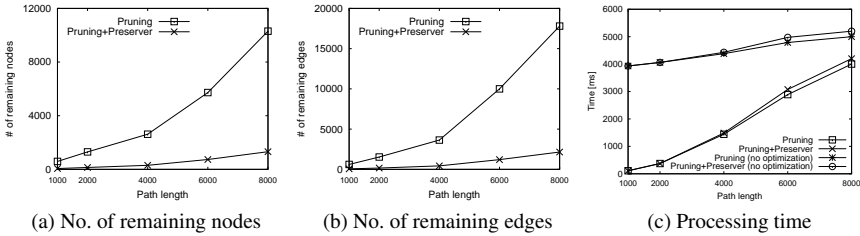


Fig. 9 Effect of path length (multiple pairs)

7.1 Evaluation of Graph-size Reduction Methods

In this section, we leave BUP processing aside, and evaluate our graph-size reduction methods in three aspects: number of remaining nodes, number of remaining edges, and running time (for graph-size reduction alone). We report results for pruning (from Section 4.1) when applied alone, and when applied in tandem with the preserver method (from Section 4.2).

Effect of path length: In Figure 8 (single pair), we vary the path length and plot the number of remaining nodes/edges with each approach. We also present their running times; for each method (“Pruning” and “Pruning+Preserver”) we include its full-fledged version (with all optimizations described in Section 4) and its version without the implementation optimization in the last paragraph of Section 4.1.

The original network has 28,867 nodes and 30,429 edges, out of which fewer than 500 nodes and 800 edges remain after pruning, achieving a vast reduction. The latter are further reduced by the preserver technique to fewer than 60 and 100, respectively, lowering down the problem size dramatically, even for the most distant source-destination pairs we tried. The number of remaining nodes/edges grows with the path length, because $SP(s, t, R_{temp})$ increases and, hence, Lemma 3 prunes fewer edges (recall that R_{temp} is a permissible heuristic BUP solution, and $SP(s, t, R_{temp})$ is the length of the

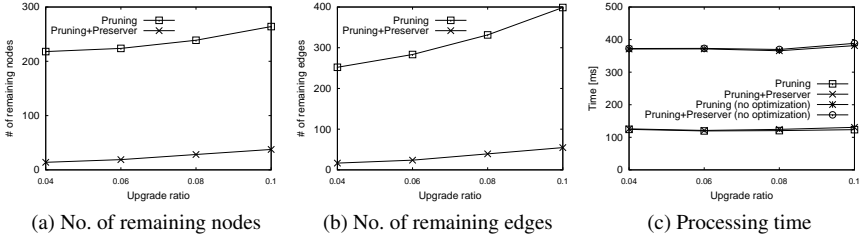


Fig. 10 Effect of upgrade ratio (single pair)

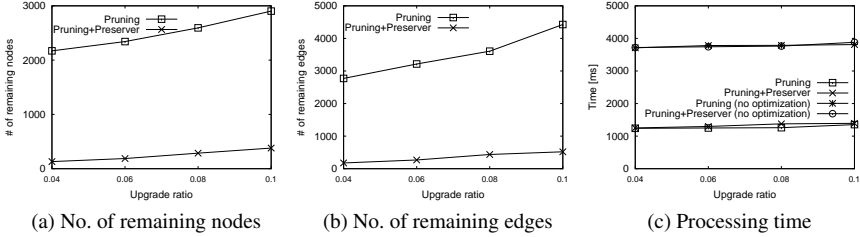


Fig. 11 Effect of upgrade ratio (multiple pairs)

shortest path from s to t under plan R_{temp}). In terms of running time, both approaches take longer for larger path lengths, because the reduced graph is larger. The optimized versions of the algorithms are very efficient, requiring fewer than 250msec in all cases.

In Figure 9 (multiple pairs), we observe similar trends as in single-pair BUP, except that the number of remaining nodes/edges and the processing time are about 11 to 12 times higher than those for single-pair BUP. Recall that in the default setting, there are $|Q| = 10$ source-destination pairs which explains why the relevant (i.e., remaining) part of G , and the cost to produce it, are proportionately larger.

Effect of upgrade ratio: In Figure 10 (single pair), we vary the upgrade ratio from 0.04 to 0.1, i.e., 4% to 10% of the network edges are upgradable. Lemma 3 is applied on the fully upgraded G , considering for each edge e its shortest possible distance from s and t , in order to guarantee correctness. Hence, a higher upgrade ratio implies looser pruning (equivalently, more remaining nodes and edges). In Figure 11 (multiple pairs), as in previous results in Figure 9, multiple pairs imply more remaining nodes/edges and a longer running time, but the measurements still follow a similar trend to the single-pair case.

Effect of resource ratio: In Figure 12 (single pair) and Figure 13 (multiple pairs), we vary the resource ratio from 0.2 to 0.8 – that is, B ranges from 20% to 80% of C (described in the beginning of the experiment section). A greater ratio implies a larger budget B and, therefore, a smaller $SP(s, t, R_{temp})$. In turn, this means more extensive pruning by Lemma 3.

Effect of number of query pairs: In Figure 14, we vary the number of query pairs (in multiple-pair BUP) and keep the remaining parameters at their defaults. With the increasing number of query pairs, the numbers of remaining nodes and edges naturally increase in a roughly linear fashion. As explained previously, a growing $|Q|$ implies that

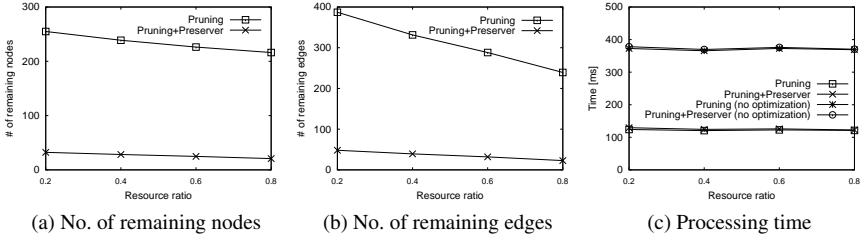


Fig. 12 Effect of resource ratio (single pair)

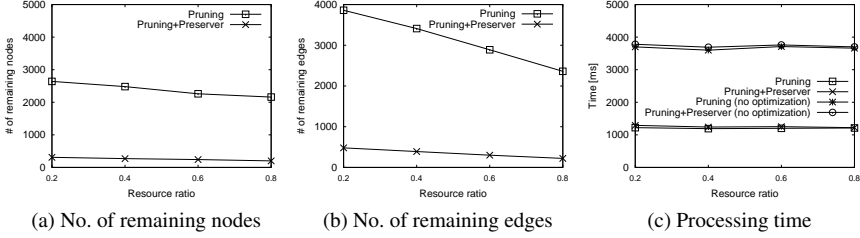


Fig. 13 Effect of resource ratio (multiple pairs)

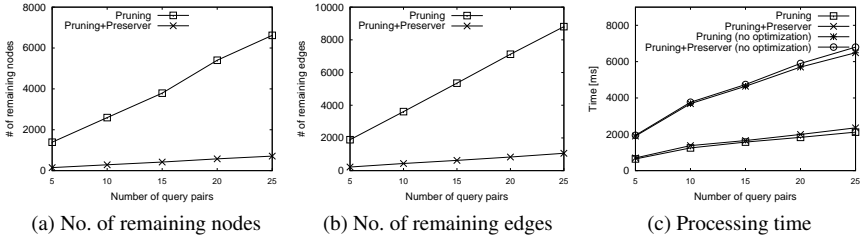


Fig. 14 Effect of number of query pairs

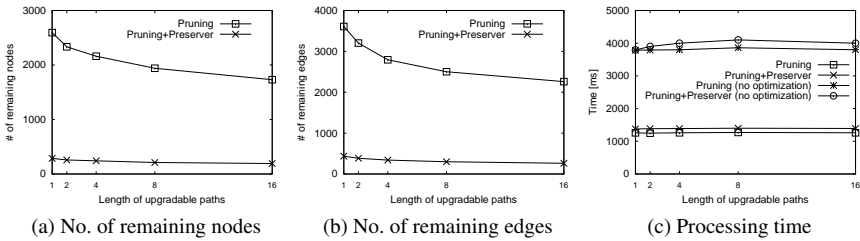


Fig. 15 Effect of length of upgradable path (multiple pairs)

a larger part of G becomes relevant to processing. Expanding more nodes and considering more edges also increases the processing time.

Effect of upgradable path length: Next, we experiment with upgradable edges that are clustered in the form of upgradable paths, resembling a situation where entire highway sections are amenable to improvement. For this, we use parameter l_{up} which

represents the number of edges per upgradable path. Each upgradable path is generated by randomly picking a start node, and performing a random walk around it that spans l_{up} edges. These edges are added to U , i.e., are considered upgradable. We generate $|U|/l_{up}$ upgradable paths to keep the total number of upgradable edges fixed. In Figure 15, we vary l_{up} from 1 to 16. Note that the case for $l_{up} = 1$ corresponds to our default setting of randomly distributed (and not necessarily connected) upgradable edges. For brevity, we present results for the multiple-pair BUP only. With the increase of l_{up} , the size of the reduced graph tends to be smaller. The reason is that the upgradeable clusters grow larger, thus making it more probable to prune a large bunch of edges if their containing cluster happens to lie far from all query pairs.

7.2 Evaluation of BUP Processing Algorithms

Given a reduced graph G_c (produced either by “Pruning” or “Pruning+Preserver”), in this section we evaluate BUP algorithms from Sections 5 and 6.⁵ In the plots we label AG as “Augmented”, FG as “Full”, and OG as “Original”.

In Figure 16 (single pair) and Figure 17 (multiple pairs), in addition to Germany, we use three other real road networks; one smaller (San Joaquin County, with 18,263 nodes and 23,874 edges, from www.cs.utah.edu/~lifeifei/SpatialDataset.htm), and two larger ones (India, with 149,566 nodes and 155,483 edges, from www.maproom.psu.edu/dcw/, and San Francisco Bay Area, with 321,170 nodes and 800,172 edges, from www.dis.uniroma1.it/challenge9/download.shtml). For each road network, we present the processing time of all three BUP algorithms, assuming reduction by “Pruning” or “Pruning+Preserver”, for the default parameter values in Table 5.

We observe that OG consistently outperforms alternatives, with FG being the runner-up. An interesting fact is that the running time of all algorithms in India/San Francisco Bay Area is longer. This is irrelevant to the size of the network. A path length of 4,000 (default) in India/San Francisco Bay Area corresponds to paths with much more edges than paths of the same length in the other two networks. To see this, in single-pair BUP for example, after “Pruning” in Germany the remaining nodes/edges are 238 and 331, while for India the corresponding numbers are 711 and 1,087.

In Figure 17 (for multiple-pair BUP), the performance trends of all three methods are similar to the single-pair case, but the running times are several times longer. The reason is that in multiple-pair BUP, we need to rank paths for multiple query pairs, but importantly, we also need to rank path combinations from $|Q|$ lists using the best neighbor method (BN) from [23]. The latter performs a significant number of heap operations that add up to the processing time.

Having established the general superiority of OG, in Figures 18 and 19 we examine the effect of path length, upgrade ratio and resource ratio on the running time of OG, plotting also measurements for the runner-up (FG) for the sake of comparison. In the multiple-pair case (Figure 19) we also study the effect of $|Q|$ and l_{up} (described in the context of Figure 15). The experiments use our default network (Germany) after reduction by “Pruning+Preserver”. We observe a direct correlation between the running

⁵ For completeness, we mention that a brute force evaluation of subsets of U_c (after “Pruning+Preserver” reduction) in our default setting for single-pair BUP takes longer than 20sec, which is orders of magnitude slower than our methods, evaluated shortly.

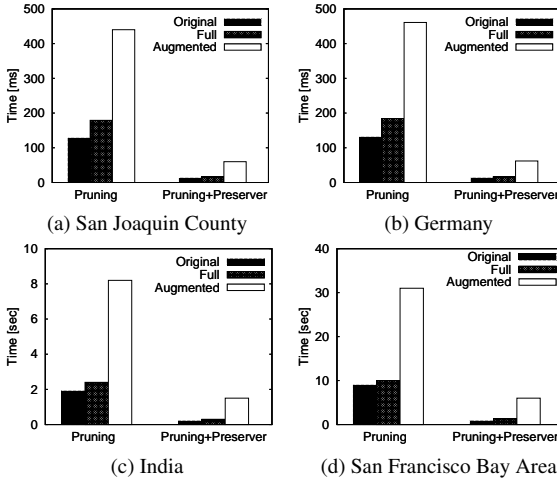


Fig. 16 Running time of BUP algorithms on different networks (single pair)

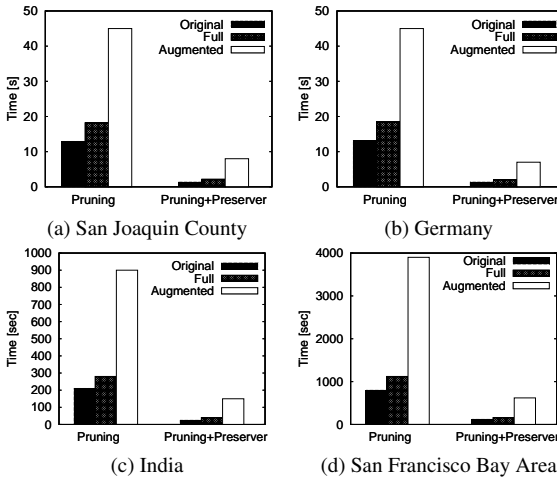


Fig. 17 Running time of BUP algorithms on different networks (multiple pairs)

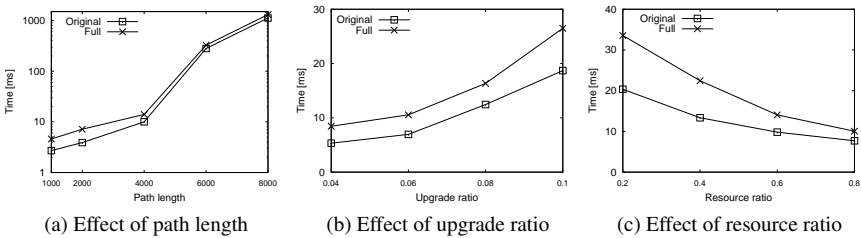


Fig. 18 Performance of OG and FG in Germany network (single pair)

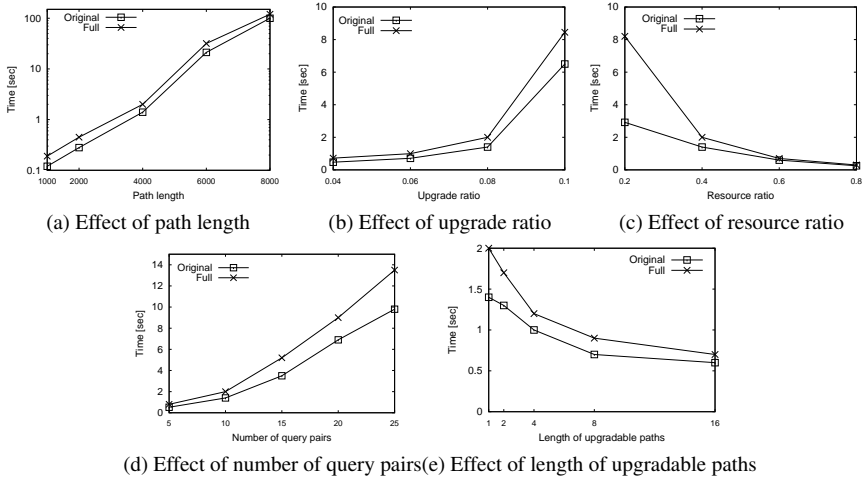


Fig. 19 Performance of OG and FG in Germany network (multiple pairs)

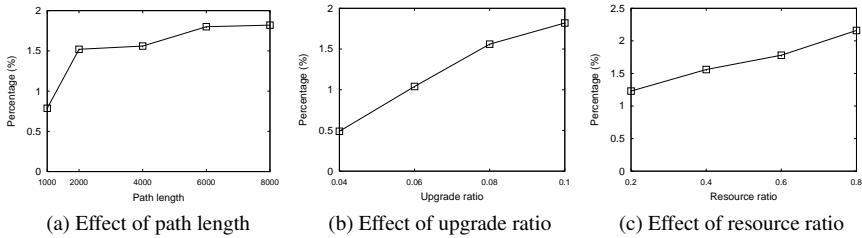


Fig. 20 Percentage of path length improvement (multiple pairs)

time of the BUP algorithms and the size of the reduced graph (investigated in Section 7.1) – for example, performance in Figure 18(a) follows the trends in Figures 8(a) and 8(b). This verifies that indeed the size of graph G_c is a major performance determinant and justifies our design effort in Section 4 to reduce it.

In Figure 20 we plot the (percentage of) path length reduction when varying the (original) path length, the upgrade ratio, and the resource ratio, in multiple-pair BUP. The reduction increases with all three parameters. In the first case, that is because as the path length increases, more upgradable edges remain in G_c (see Figure 9) and, therefore, more options become available to shorten the paths. Similarly, an increased upgrade ratio implies more permissible plans and, thus, allows for a larger reduction in path lengths. Finally, a higher resource ratio implies that more edges can be chosen for upgrade, and leads to shorter paths (in the updated network).

8 Conclusion

In this paper, we study the *Resource Constrained Best Upgrade Plan* query (BUP). In a road network where a fraction of the edges are upgradable at some cost, the BUP

query computes the subset of these edges to be upgraded so that the shortest path distance for a source-destination pair (or the sum of shortest path distances for a set of source-destination pairs) is minimized and the total upgrade cost does not exceed a user-specified budget. Our methodology centers on two axes: the effective reduction of the network size and the efficient BUP processing in the resulting graph. Experiments on real road networks verify the effectiveness of our techniques and the efficiency of our framework overall. A direction for future work is the consideration of multiple concurrent constraints (on different resource types).

References

1. Alumur, S.A., Kara, B.Y.: Network hub location problems: The state of the art. *European Journal of Operational Research* **190**(1), 1–21 (2008)
2. Amaldi, E., Capone, A., Cesana, M., Malucelli, F.: Optimization models for the radio planning of wireless mesh networks. In: *Networking*, pp. 287–298 (2007)
3. Beasley, J.E., Christofides, N.: An algorithm for the resource constrained shortest path problem. *Networks* **19**, 379–394 (1989)
4. Ben-Moshe, B., Omri, E., Elkin, M.: Optimizing budget allocation in graphs. In: *CCCG* (2011)
5. Boorstyn R. and Frank, H.: Large-scale network topological optimization. *IEEE Transactions on Communications* **25**(1), 29 – 47 (1977)
6. Campbell, A.M., Lowe, T.J., Zhang, L.: Upgrading arcs to minimize the maximum travel time in a network. *Networks* **47**(2), 72–80 (2006)
7. Capone, A., Elias, J., Martignon, F.: Models and algorithms for the design of service overlay networks. *IEEE Transactions on Network and Service Management* **5**(3), 143–156 (2008)
8. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: *Introduction to Algorithms*, Second Edition. The MIT Press and McGraw-Hill Book Company (2001)
9. Deng, K., Zhou, X., Shen, H.T.: Multi-source skyline query processing in road networks. In: *ICDE*, pp. 796–805 (2007)
10. Ding, B., Yu, J.X., Qin, L.: Finding time-dependent shortest paths over large graphs. In: *EDBT*, pp. 205–216 (2008)
11. Dumitrescu, I., Boland, N.: Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem. *Networks* **42**, 135–153 (2003)
12. Fan, J., Ammar, M.H.: Dynamic topology configuration in service overlay networks: A study of reconfiguration policies. In: *INFOCOM* (2006)
13. Handler, G.Y., Zang, I.: A dual algorithm for the constrained shortest path problem. *Networks* **10**, 293–309 (1980)
14. Hassin, R.: Approximation schemes for the restricted shortest path problem. *Math. Oper. Res.* **17**(1), 36–42 (1992)
15. Hills, A.: Mentor: an algorithm for mesh network topological optimization and routing. *IEEE Transactions on Communications* **39**(11), 98–107 (2001)
16. Jain, R., Walrand, J.: An efficient nash-implementation mechanism for network resource allocation. *Automatica* **46**(8), 1276–1283 (2010)
17. Jensen, C.S., Kolársv, J., Pedersen, T.B., Timko, I.: Nearest neighbor queries in road networks. In: *GIS*, pp. 1–8 (2003)
18. Johari, R., Tsitsiklis, J.N.: Efficiency loss in a network resource allocation game. *Math. Oper. Res.* **29**(3), 407–435 (2004)
19. Jung, S., Pramanik, S.: An efficient path computation model for hierarchically structured topographical road maps. *IEEE Trans. Knowl. Data Eng.* **14**(5) (2002)
20. Kershenbaum, A., Kermani, P., Grover, G.A.: Mentor: an algorithm for mesh network topological optimization and routing. *IEEE Transactions on Communications* **39**(4), 503–513 (1991)
21. Kriegel, H.P., Kröger, P., Renz, M., Schmidt, T.: Hierarchical graph embedding for efficient query processing in very large traffic networks. In: *SSDBM*, pp. 150–167 (2008)
22. Li, Z., Mohapatra, P.: On investigating overlay service topologies. *Computer Networks* **51**(1), 54–68 (2007)
23. Lin, Y., Mouratidis, K.: Best upgrade plans for large road networks. In: *SSTD*, pp. 223–240 (2013)

24. Lorenz, D.H., Raz, D.: A simple efficient approximation scheme for the restricted shortest path problem. *Operations Research Letters* **28**(5), 213 – 219 (2001)
25. Maillé, P., Tuffin, B.: Multi-bid auctions for bandwidth allocation in communication networks. In: *INFOCOM* (2004)
26. Mehlhorn, K., Ziegelmann, M.: Resource constrained shortest paths. In: *Algorithms - ESA 2000*, vol. 1879, pp. 326–337 (2000)
27. Minoux, M.: Networks synthesis and optimum network design problems: Models, solution methods and applications. *Networks* **19**, 313–360 (1989)
28. Nepal, K.P., Park, D., Choi, C.H.: Upgrading arc median shortest path problem for an urban transportation network. *Journal of Transportation Engineering* **135**(10), 783–790 (2009)
29. Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query processing in spatial network databases. In: *VLDB*, pp. 802–813 (2003)
30. de Queirós Vieira Martins, E., Pascoal, M.M.B.: A new implementation of yen’s ranking loopless paths algorithm. *4OR* **1**(2) (2003)
31. Ratnasamy, S., Handley, M., Karp, R.M., Shenker, S.: Topologically-aware overlay construction and server selection. In: *INFOCOM* (2002)
32. Ribeiro, C.C., Minoux, M.: A heuristic approach to hard constrained shortest path problems. *Discrete Applied Mathematics* **10**(2), 125 – 137 (1985)
33. Roy, S., Pucha, H., Zhang, Z., Hu, Y.C., Qiu, L.: Overlay node placement: Analysis, algorithms and impact on applications. In: *ICDCS*, p. 53 (2007)
34. Shahabi, C., Kolahdouzan, M.R., Sharifzadeh, M.: A road network embedding technique for k-nearest neighbor search in moving object databases. *GeoInformatica* **7**(3), 255–273 (2003)
35. Stojanovic, D., Papadopoulos, A.N., Predic, B., Djordjevic-Kajan, S., Nanopoulos, A.: Continuous range monitoring of mobile objects in road networks. *Data Knowl. Eng.* **64**(1), 77–100 (2008)
36. Zhang, L.: Upgrading arc problem with budget constraint. In: *43rd annual Southeast regional conference - Volume 1*, pp. 150–152 (2005)