Best Upgrade Plans for Large Road Networks

Yimin Lin and Kyriakos Mouratidis

School of Information Systems
Singapore Management University
80 Stamford Road, Singapore 178902
{yimin.lin.2007, kyriakos}@smu.edu.sg

Abstract. In this paper, we consider a new problem in the context of road network databases, named *Resource Constrained Best Upgrade Plan* computation (BUP, for short). 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. Given a source and a destination in G, and a budget (resource constraint) B, the BUP problem 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 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.

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 objective is to minimize the traveling time (shortest path distance) from a source to a destination 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 another key location (e.g., from the airport). 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 and destination is minimized and at the same time the summed upgrade cost does not exceed a specific budget (resource constraint).

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, a 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.

Although we focus on transportation networks, BUP finds application in other domains too. Consider for example 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 two nodes, BUP would indicate to the network operators which links to upgrade in order to minimize the network latency between the two nodes, subject to the available monetary budget.

Figure 1 shows an example of BUP query in a road network. The edges drawn as dashed lines are upgradable. Each upgradable edge is associated with a triplet of numbers (e.g., $\langle 9|10|16 \rangle$), 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).

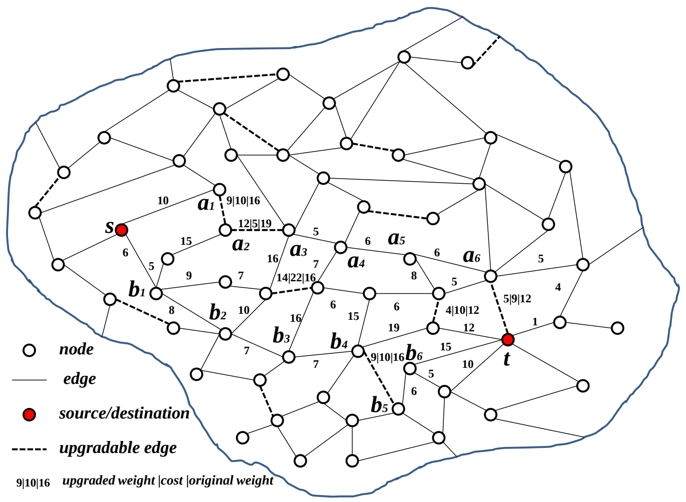


Fig. 1. Example of BUP query

The input of the query is a source node s and a destination node t 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 shortest path distance from s to t, while the sum of their upgrade costs does not exceed B. Assuming a resource constraint $B = 20$, the output of BUP in our example includes edge (b_4, b_5) and leads to a shortest path distance of 58 via path $\{s, b_1, b_2, b_3, b_4, b_5, b_6, t\}$. The resource consumption in this case is 10, i.e., smaller than B, and thus permissible. Note that if B were larger, an even shorter distance could be achieved (namely, 53) by upgrading edges (a_1, a_2), (a_2, a_3) and (a_6, a_t). 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 [1–3]. Currently, however, there is no work on BUP, while algorithms for related problems cannot be adapted to address it.

In this paper, we formalize 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 requirement for numerous shortest path computations. We develop techniques that limit both these factors with the aim of efficient processing. 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 areas 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) [4–6], the processing of queries that involve a notion of dominance based on proximity [7], continuous versions of proximity queries [8], etc. There have also been several studies on materialization with the purpose of accelerating shortest distance/path queries [9, 10]. 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 [11], 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 [12, 13], wired backbone networks [14–17] and service overlay networks [18–21]. 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 [22]. 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 [1, 23, 24]. 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 [2], the authors discuss the complexity of the problem with budget constraints. Budget constraints are also considered in [25], which proposes methods to minimize the diameter of a tree-structured network. [26] addresses the $q-$upgrading arc problem, where q edges are selected for upgrading to minimize the diameter of the graph. In [3] 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 [27–30]. *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 [31–33]. 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 for 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 source node $s \in V$, and a destination node $t \in V$, the BUP problem is to select a subset R of the edges in U for upgrade so that (i) the summed upgrade cost in R does not exceed budget B and (ii) the shortest path distance between s and t in the updated network is minimized. Formally, the output R of BUP is the result of the following optimization problem:

$$\begin{aligned} \underset{R \subseteq U}{\operatorname{argmin}} \quad & SP(s, t, R) \\ \text{subject to} \quad & \sum_{e \in R} e.c \leq B \end{aligned}$$

where $SP(s, t, R)$ is the (traveling time along the) new shortest path between s and t when edges in R are upgraded. If there are multiple subsets R that abide by the resource constraint and lead to the same smallest traveling time between s and t, BUP reports the one with the smallest total cost.

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 total traveling time along it. For ease of presentation, we use $SP(s, t, R)$ to refer both to the shortest path (between s and t 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 the shortest path $SP(s, t, R)$ 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 $SP(s, t, R)$.

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 a shortest path computation. 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 two fold 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

Symbol	Meaning
$G = \langle V, E \rangle$	Road network with node set V and edge set E
U	Set of upgradable edges ($U \subseteq E$)
$\lvert U \rvert$	The cardinality of U
B	Resource constraint (budget)
R	Edges chosen for upgrade ($R \subseteq U$)
$e.w$	Original weight of edge e
$e.c, e.w'$	Upgrade cost and upgraded weight of $e \in U$
s, t	Source and destination nodes
$C(R)$	Sum of upgrade costs (across all edges) in R
$SP(s, t, R)$	Shortest path (length) 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

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 Section 5. Graph reduction techniques assist towards both our design goals, while our 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.

4.1 Graph Shrinking via Edge Pruning

Our intuition is that any edge (upgradable or not) that lies too far from s and t 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 $SP(s, t, R)$ be the achieved shortest path in the updated network. R includes only upgradable edges along the path $SP(s, t, R)$.*

Proof. The lemma is based on our problem definition, and specifically on the fact that among permissible plans that lead to the same (minimal) distance between s and t, 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 $SP(s,t,R)$. If we apply upgrade set $R - \{e\}$, the shortest path between s and t will pass through exactly the same edges as with R, and we will have achieved the same 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 solution in G. Any subgraph of G that includes all the edges along $SP(s,t,R)$ is BUP-equivalent to G.*

Proof. Consider the subgraph G_{sp} of G that comprises *only* the (upgradable and non-upgradable) edges along path $SP(s,t,R)$. 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 $SP(s,t,R)$ (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 length of $SP(s,t,R)$, 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 s and t to belong to R, and are therefore safe to prune.

Lemma 3. *It is safe to prune every edge $e = (v_i, v_j)$ for which:*
(i) $SP(s, v_i, U) + w + SP(v_j, t, U) > T$ ***and***
(ii) $SP(s, v_j, U) + w + SP(v_i, t, U) > T$
where w equals $e.w'$ if e is upgradable, or simply $e.w$ if e is not upgradable.

Proof. The shortest possible path between s and t that passes through edge e is the one that corresponds to a fully upgraded G. The length of that path is either $SP(s, v_i, U) + w + SP(v_j, t, U)$ or $SP(s, v_j, U) + w + SP(v_i, t, U)$, whichever is smaller. If that minimum value is greater than T, edge e could not possibly belong to $SP(s,t,R)$ (where R is the final BUP result) and, therefore, e 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 $SP(s,t,R_{temp})$ instead of T, where R_{temp} is a permissible (suboptimal) plan that leads to a sufficiently short distance from s to t.

Effective R_{temp} selection: To derive R_{temp} quickly and effectively, we first compute the shortest path $SP(s,t,U)$ in the fully upgraded graph. Next, we form R_{temp} using only the upgradable edges included in $SP(s,t,U)$. If R_{temp} exceeds the resource constraint, we execute a knapsack algorithm [34] to derive the subset of R_{temp} that achieves the minimum sum of weights along path $SP(s,t,R_{temp})$ without violating B. Note that this is different from a BUP problem, because we essentially fix the path to $SP(s,t,R_{temp})$ and seek the permissible subset of its upgradable edges that minimizes the path length *along the specific path only*. The result of the knapsack algorithm is used as the R_{temp} for 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 a $SP(s,t,R_{temp})$ distance of 60, Lemma 3 prunes every edge out of the inner (green-border) closed curve.

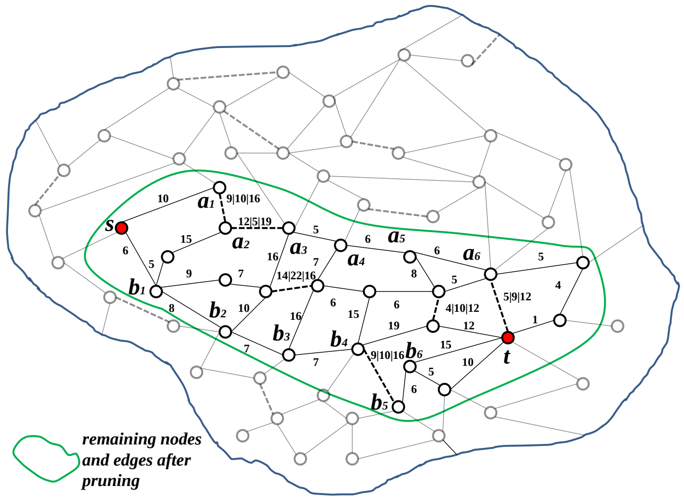


Fig. 2. Example of edge pruning

Implementation: To implement pruning, we perform two Dijkstra expansions [34] from s and t on the fully upgraded graph. Each expansion reaches up to distance $SP(s,t,R_{temp})$ from its start node (s and t, respectively). All edges that are not encountered or encountered by only one of the expansions, are pruned. Each of the remaining edges is 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 s,t,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 s, t, or the end-node of some 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 $SP(s, t, R)$ in order of appearance (from s to t). For every pair of consecutive key nodes v_i, v_j in this sequence, either v_i, v_j 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_i, v_j is also the shortest (by definition, every sub-path of a shortest path, is the shortest path between the intermediate nodes it connects). Thus, $SP(s, t, R)$ 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 $SP(s, t, R)$. 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 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_i we perform a Dijkstra search (with source at v_i). When another key node v_j is encountered (i.e., popped by the Dijkstra heap), we add a virtual edge between v_i and v_j to G_c. However, we do not expand v_j (i.e., we do not push into the heap the adjacent nodes of v_j) so as to ensure plain paths. Let M be the smallest of $SP(s, v_i, U)$ and $SP(v_i, t, U)$ (both these values are known since the pruning stage). The Dijkstra search can safely terminate if it has reached up to distance $SP(s, t, R_{temp}) - M$ from v_i (any virtual edge longer than that threshold is useless according to Lemma 3), where R_{temp} is the heuristic (suboptimal) BUP solution from Section 4.1.

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

5 BUP Processing Algorithms

In this section, we present algorithms to compute the BUP solution on G_c, i.e., the graph resulting after the application of the edge pruning and resource preserver techniques

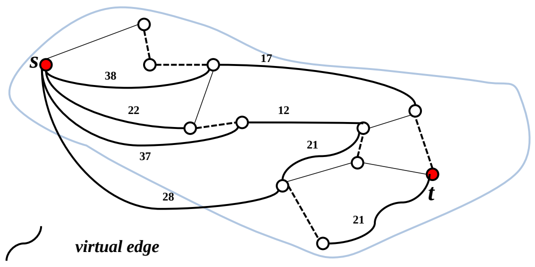


Fig. 3. The resulting graph G_c

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.

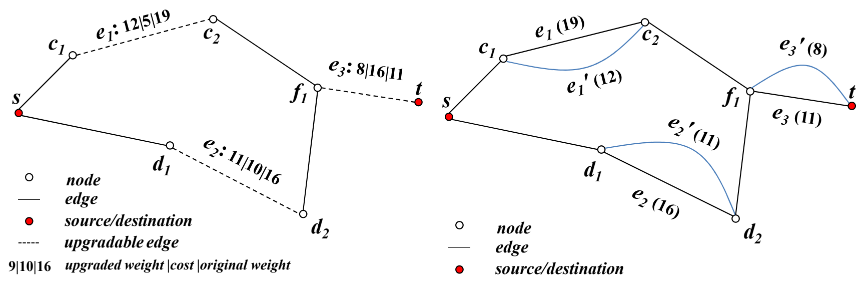
Our approach is to iteratively compute alternative paths (from s to t) in increasing order of length, and evaluate them in this order. We distinguish three variants of this general approach, depending on which version of G_c is used for the incremental path exploration; it could be the original G_c, 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 [35] 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).

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 [35] in $A(G_c)$, and iteratively examines paths in increasing length order. In our example, assume that the budget is $B = 20$. The



(a) Original G_c (b) Augmented G_c

Fig. 4. Augmented graph example

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 $\{e'_1, e'_2\}$.

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 $B = 20$ 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).

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

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 EXPERIMENTAL EVALUATION

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 Section 5).

As default network G we use the road network of Germany, which has 28,867 nodes, 30,429 edges, and a diameter (i.e., maximum distance between any pair of nodes) of 14,383. The network is available at: www.maproom.psu.edu/dcw/. We study the impact of three parameters: (original) *path length* between source and destination; *upgrade ratio*; and *resource ratio*. The upgrade ratio indicates the ratio of upgradable edges over their total number (i.e., $|U|/|E|$). Upgradable edges are selected randomly from E. 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 from s to t in the original network; let this sum be C. We set B to a fraction of this cost. The resource ratio equals B/C. Table 2 shows the parameter values tested and their default (in bold). In every experiment, we vary one parameter and set the other two 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
Upgrade ratio	0.04, 0.06, **0.08**, 0.1
Resource ratio	0.2, **0.4**, 0.6, 0.8

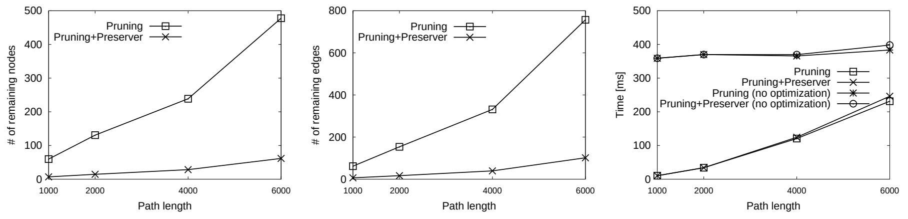
Table 2. Experiment parameters

6.1 Evaluation of Graph-size Reduction Methods

In this section, we leave aside BUP processing, and evaluate our graph-size reduction methods in three aspects: reduction of number of nodes, reduction of number of edges, and running time. 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 5, 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.

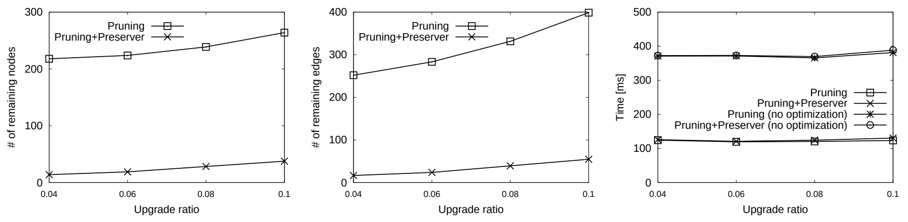
The original network has 28,867 nodes and 30,429 edges, out of which fewer than 500 nodes and 800 edges remain after pruning, achieving a vast reduction. The latter are further reduced by the preserver technique to fewer than 60 and 100, respectively, lowering down the problem size dramatically, even 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 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.



(a) No. of remaining nodes (b) No. of remaining edges (c) Processing time

Fig. 5. Effect of path length

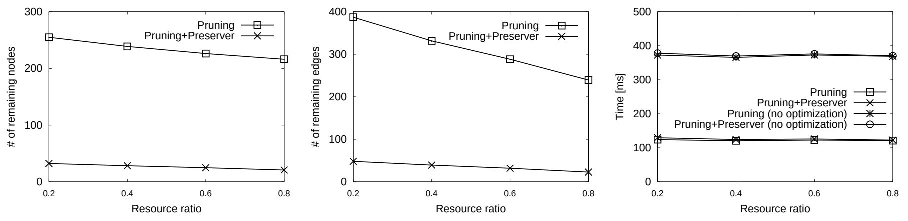
Effect of upgrade ratio. In Figure 6, we vary the upgrade ratio from 0.04 to 0.1, i.e., 4% to 10% of the network edges are upgradable. Lemma 3 is applied on the fully upgraded G, considering for each edge e its shortest possible distance from s and t, in order to guarantee correctness. Hence, a higher upgrade ratio implies looser pruning (equivalently, more remaining nodes and edges).



(a) No. of remaining nodes (b) No. of remaining edges (c) Processing time

Fig. 6. Effect of upgrade ratio

Effect of resource ratio. In Figure 7, 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.



(a) No. of remaining nodes (b) No. of remaining edges (c) Processing time

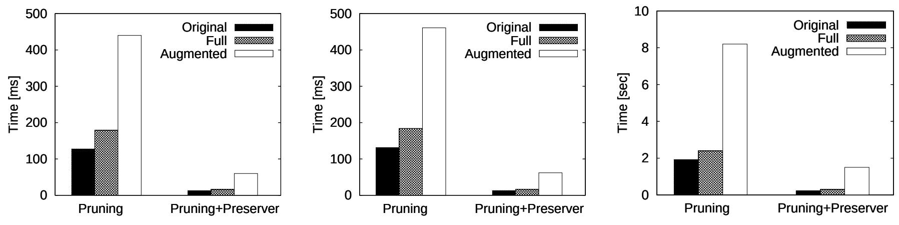
Fig. 7. Effect of resource ratio

6.2 Evaluation of BUP Processing Algorithms

Given a reduced graph G_c (produced either by "Pruning" or "Pruning+Preserver"), in this section we evaluate the three BUP algorithms from Section 5. For understandability, in the plots we label AG as "Augmented", FG as "Full", and OG as "Original".

In Figure 8, in addition to Germany, we use two other real road networks; one smaller (San Joaquin County, with 18,263 nodes and 23,874 edges, from www.cs.utah.edu/$\sim$lifeifei/SpatialDataset.htm) and the other larger (India, with 149,566 nodes and 155,483 edges, from www.maproom.psu.edu/dcw/). 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 2.

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 is longer. This is irrelevant to the size of the network. A path length of 4,000 (default) in India corresponds to paths with much more edges than paths of the same length in the other two networks[2]. To see this, after "Pruning" in Germany the remaining nodes/edges are 238 and 331, while for India the corresponding numbers are 711 and 1,087.

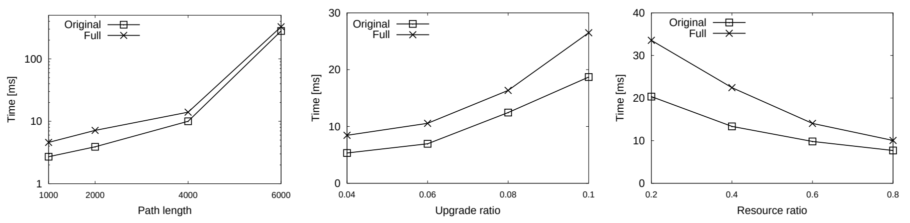


(a) San Joaquin County (b) Germany (c) India

Fig. 8. Running time of BUP algorithms on different networks

Having established the general superiority of OG, in Figure 9 we examine the effect of path length, upgrade ratio and resource ratio on its running time, plotting also measurements for the runner-up (FG) for the sake of comparison. The experiments use our default network (Germany) after reduction by "Pruning+Preserver". We observe a direct correlation between the running time of the BUP algorithms and the size of the reduced graph (investigated in Section 6.1) – for example, performance in Figure 9(a) follows the trends in Figures 5(a) and 5(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.

[2] We did not apply any normalization on edge weights across the three networks in order to retain their original distance semantics.



(a) Effect of path length (b) Effect of upgrade ratio (c) Effect of resource ratio

Fig. 9. Performance of OG and FG in Germany network

7 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 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). Another is BUP processing when the optimization goal involves multiple source-destination pairs instead of a just one.

References

1. Jain, R., Walrand, J.: An efficient nash-implementation mechanism for network resource allocation. Automatica **46**(8) (2010) 1276–1283
2. Zhang, L.: Upgrading arc problem with budget constraint. In: 43rd annual Southeast regional conference - Volume 1. (2005) 150–152
3. 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) (2009) 783–790
4. Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query processing in spatial network databases. In: VLDB. (2003) 802–813
5. Shahabi, C., Kolahdouzan, M.R., Sharifzadeh, M.: A road network embedding technique for k-nearest neighbor search in moving object databases. GeoInformatica **7**(3) (2003) 255–273
6. Jensen, C.S., Kolárvr, J., Pedersen, T.B., Timko, I.: Nearest neighbor queries in road networks. In: GIS. (2003) 1–8
7. Deng, K., Zhou, X., Shen, H.T.: Multi-source skyline query processing in road networks. In: ICDE. (2007) 796–805
8. 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) (2008) 77–100
9. Kriegel, H.P., Kröger, P., Renz, M., Schmidt, T.: Hierarchical graph embedding for efficient query processing in very large traffic networks. In: SSDBM. (2008) 150–167
10. Jung, S., Pramanik, S.: An efficient path computation model for hierarchically structured topographical road maps. IEEE Trans. Knowl. Data Eng. **14**(5) (2002)

11. Ding, B., Yu, J.X., Qin, L.: Finding time-dependent shortest paths over large graphs. In: EDBT. (2008) 205–216
12. Hills, A.: Mentor: an algorithm for mesh network topological optimization and routing. IEEE Transactions on Communications **39**(11) (2001) 98–107
13. Amaldi, E., Capone, A., Cesana, M., Malucelli, F.: Optimization models for the radio planning of wireless mesh networks. In: Networking. (2007) 287–298
14. Boorstyn, R.and Frank, H.: Large-scale network topological optimization. IEEE Transactions on Communications **25**(1) (1977) 29 – 47
15. Ratnasamy, S., Handley, M., Karp, R.M., Shenker, S.: Topologically-aware overlay construction and server selection. In: INFOCOM. (2002)
16. Kershenbaum, A., Kermani, P., Grover, G.A.: Mentor: an algorithm for mesh network topological optimization and routing. IEEE Transactions on Communications **39**(4) (1991) 503–513
17. Minoux, M.: Networks synthesis and optimum network design problems: Models, solution methods and applications. Networks **19** (1989) 313–360
18. Li, Z., Mohapatra, P.: On investigating overlay service topologies. Computer Networks **51**(1) (2007) 54–68
19. 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) (2008) 143–156
20. Fan, J., Ammar, M.H.: Dynamic topology configuration in service overlay networks: A study of reconfiguration policies. In: INFOCOM. (2006)
21. Roy, S., Pucha, H., Zhang, Z., Hu, Y.C., Qiu, L.: Overlay node placement: Analysis, algorithms and impact on applications. In: ICDCS. (2007) 53
22. Alumur, S.A., Kara, B.Y.: Network hub location problems: The state of the art. European Journal of Operational Research **190**(1) (2008) 1–21
23. Johari, R., Tsitsiklis, J.N.: Efficiency loss in a network resource allocation game. Math. Oper. Res. **29**(3) (2004) 407–435
24. Maillé, P., Tuffin, B.: Multi-bid auctions for bandwidth allocation in communication networks. In: INFOCOM. (2004)
25. Ben-Moshe, B., Omri, E., Elkin, M.: Optimizing budget allocation in graphs. In: CCCG. (2011)
26. Campbell, A.M., Lowe, T.J., Zhang, L.: Upgrading arcs to minimize the maximum travel time in a network. Networks **47**(2) (2006) 72–80
27. Handler, G.Y., Zang, I.: A dual algorithm for the constrained shortest path problem. Networks **10** (1980) 293–309
28. Beasley, J.E., Christofides, N.: An algorithm for the resource constrained shortest path problem. Networks **19** (1989) 379–394
29. Mehlhorn, K., Ziegelmann, M.: Resource constrained shortest paths. In: Algorithms - ESA 2000. Volume 1879. (2000) 326–337
30. Ribeiro, C.C., Minoux, M.: A heuristic approach to hard constrained shortest path problems. Discrete Applied Mathematics **10**(2) (1985) 125 – 137
31. Hassin, R.: Approximation schemes for the restricted shortest path problem. Math. Oper. Res. **17**(1) (1992) 36–42
32. Lorenz, D.H., Raz, D.: A simple efficient approximation scheme for the restricted shortest path problem. Operations Research Letters **28**(5) (2001) 213 – 219
33. Dumitrescu, I., Boland, N.: Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem. Networks **42** (2003) 135–153
34. 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)
35. de Queirós Vieira Martins, E., Pascoal, M.M.B.: A new implementation of yen's ranking loopless paths algorithm. 4OR **1**(2) (2003)