Università di Pisa
Dipartimento di Informatica

Technical Report: TR-01-11

A new algorithm for reoptimizing shortest paths when the arc costs change

Stefano Pallottino
Dipartimento di Informatica, Università di Pisa, pallo@di.unipi.it

Maria Grazia Scutellà
Dipartimento di Informatica, Università di Pisa, scut@di.unipi.it

Address: Corso Italia 40, 56125 Pisa, Italy.
Tel: +39 050 2212700
Fax: +39 050 2212726

A new algorithm for reoptimizing shortest paths when the arc costs change

Stefano Pallottino
Dipartimento di Informatica, Università di Pisa, pallo@di.unipi.it

Maria Grazia Scutellà
Dipartimento di Informatica, Università di Pisa, scut@di.unipi.it

Abstract

We propose an algorithm which reoptimizes shortest paths in a very general situation, that is when any subset of arcs of the input graph is affected by a change of the arc costs, which can be either lower or higher than the old ones. This situation is more general than the ones addressed in the literature so far.

Keywords: shortest paths, reoptimization, flow algorithms.

1 Reoptimizing shortest paths

In large scale transportation systems, often it is required to solve a sequence of shortest path problems, where the $(k+1)^{th}$ problem differs only slightly from the k^{th} one. In practice, each problem in the sequence is usually solved from the beginning, independently of the previous one.

The first attempts of exploiting reoptimization techniques were made by Murchland [12] and Dionne [3], who investigated the problem of finding the shortest cost matrix when the cost of one arc of the input graph changes. The same problem was addressed by Goto and Sangiovanni-Vincentelli [10], who proposed a method based on an LU decomposition of the shortest cost matrix. Although of some theoretical interest, these approaches require a very large amount of memory, which prevents from their use in practical applications.

The first efficient reoptimization strategies have been proposed by Gallo in [7], where the following problem is addressed: assume that a shortest path tree relative to a given origin r, say T^*_r, has been determined; we want to find a new shortest path tree when either the origin is changed, or exactly one arc is given a new cost, lower than the old one. Another efficient approach has been studied by Fujishige [6] for the case in which a set of arcs incident to a common node is given a new cost, lower than the old one.

Based on Gallo's reoptimization strategies, further efficient algorithms have been proposed in the literature for the situation in which the origin node is changed. The pioneering works are the algorithm of Gallo and Pallottino [8], that fully exploits the property that, after the computation of T^*_r, the arc *reduced costs* are non-negative (see for instance Nemhauser [13] and Lawler [11]), and the dual simplex algorithm of Florian, Nguyen and Pallottino [4]. Recently, a new dual algorithm has been proposed by Nguyen, Pallottino and Scutellà [14]. Unlike the previous algorithms, which follow a Dijkstra's like node inserting order to reconstruct the shortest path tree [2], the algorithm in [14] may follow a more general insertion policy.

Despite the strong relevance in several application contexts, no paper has yet focused on the more general situation, where any subset of arcs of the input graph can be affected by a change of the arc costs, which can be either lower or higher than the old ones[1]. As observed in [7], this

[1] If the new costs are lower than the old ones, the problem can be solved by iteratively calling either the algorithm in [7] for each single arc affected by the change of the cost, or the algorithm in [6] for each set of arcs incident to a common node.

situation is more general than the case in which the origin node is changed, since the change of the origin can be reduced to the change of the cost of a single arc. The general reoptimization problem will be addressed in this paper, whose aim is to provide a methodological framework to address shortest path reoptimization issues in the presence of any possible change of the arc costs.

The paper is organized as follows. Section 2 contains a formal definition of the problem, while section 3 presents the algorithm we suggest to reoptimize the shortest paths, together with a discussion of its main properties.

2 The problem

Let $G = (N, A)$ be a directed graph, where N is the set of *nodes*, of cardinality n, and A is the set of *arcs*, of cardinality m. Let $c : A \to R$ be a function which assigns a cost c_{ij} to each arc $(i,j) \in A$. For simplicity, we will assume that G is strongly connected and does not have parallel arcs.

Given an *origin* or *root* node $r \in N$, the *shortest path tree problem* (SP_r) consists of finding a directed tree T_r^* such that the path from r to i in T_r^* is a minimum cost path from r to i in G. By our previous assumption, T_r^* is a spanning tree. In order to guarantee a finite solution to the problem, we will assume that no directed cycle with negative cost exists in G.

Using a linear programming notation, the problem and its dual can be formulated as follows:

$$
\begin{aligned}
(SP_r) \quad \min \quad & \sum_{(i,j)\in A} c_{ij} x_{ij} \\
& \sum_{(j,i)\in BS(i)} x_{ji} - \sum_{(i,j)\in FS(i)} x_{ij} = \begin{cases} -n+1, & i = r; \\ \phantom{-n+{}}1, & i \in N \setminus \{r\}; \end{cases} \\
& x_{ij} \geq 0 \qquad\qquad\qquad \forall (i,j) \in A
\end{aligned}
$$

$$
\begin{aligned}
(DSP_r) \quad \max \quad & (1-n)\pi_r + \sum_{j \neq r} \pi_j \\
& \pi_j - \pi_i \leq c_{ij} \qquad\qquad \forall (i,j) \in A,
\end{aligned}
$$

where $FS(i)$ and $BS(i)$ denote the *forward star* and the *backward star* of i, respectively:

$$FS(i) = \{(u,v) \in A : u = i\}, \qquad BS(i) = \{(u,v) \in A : v = i\}.$$

In the formulation of the dual problem, π_i denotes the dual variable associated with i (usually referred to as the *price* or *potential* of i).

Given a potential vector π, and using the *reduced cost* $\bar{c}_{ij} = c_{ij} + \pi_i - \pi_j$ associated with each arc $(i,j) \in A$, the dual feasibility can be equivalently restated as

$$\bar{c}_{ij} \geq 0, \qquad \forall (i,j) \in A.$$

Now, assume that a shortest path tree T_r^* has been determined. We want to compute the shortest path tree when a subset K of the arcs of G is given a new cost (either lower or higher than the old ones). No assumption is made here on the structure of the subgraph induced by the arcs of K.

The problem of exploiting the knowledge of T_r^* to determine the new shortest path tree will be investigated in the following section, where several reoptimization issues will be discussed, and a new reoptimization algorithm will be proposed. As usual, we will assume that no negative directed cycle is created in G after the change of the costs of the arcs of K.

3 The reoptimization approach

Let $T_r^* = (N, A_r)$ be a shortest path tree for the original arc costs c_{ij}, and $\pi^{(r)}$ denote the corresponding optimum dual vector. The optimality conditions of linear programming state that:

$$\bar{c}_{ij} \geq 0, \quad \forall (i,j) \in A, \quad \text{dual feasibility,} \tag{1}$$

$$\bar{c}_{ij} = 0, \quad \forall (i,j) \in A_r, \quad \text{complementary slackness conditions (CSC),} \tag{2}$$

where $\bar{c}_{ij} = c_{ij} + \pi_i^{(r)} - \pi_j^{(r)}$ denotes here the reduced cost of the arc (i,j) with respect to the optimum dual vector $\pi^{(r)}$.

Let c'_{ij} be the new cost of the arc $(i,j), \forall (i,j) \in A$ ($c'_{ij} = c_{ij}$ if $(i,j) \notin K$), and $\bar{c}'_{ij} = c'_{ij} + \pi_i^{(r)} - \pi_j^{(r)}$ be the reduced cost of (i,j) after the change of its cost. From (1), it is $\bar{c}'_{ij} = \bar{c}_{ij} \geq 0$, $\forall (i,j) \in A \setminus K$, whereas $\bar{c}'_{ij}$ can be negative if $(i,j) \in K$. Moreover, also in the case in which the dual feasibility is preserved after the change of the arc costs, that is $\bar{c}'_{ij} \geq 0, \forall (i,j) \in A$, it may happen that the dual vector $\pi^{(r)}$ does not satisfy the complementary slackness conditions (2) with respect to the tree T_r^*, which can be not optimal anymore.

In order to determine the new shortest path tree, we can change the costs of the arcs in K, and apply a standard shortest path tree algorithm. Alternatively, we can consider the modified reduced costs $\bar{c}'_{ij}$, and reoptimize T_r^* using these modified reduced costs. In fact, as observed in [13], the shortest path tree can be computed by replacing the arc costs c'_{ij} by any transformation of type $c'_{ij} + \pi_i - \pi_j$, where $\pi = [\pi_i]$ is a real vector of size n. In particular, the use of the modified reduced costs $\bar{c}'_{ij}$ guarantees to have non-negative arc costs for each $(i,j) \in A \setminus K$, and zero arc costs for each $(i,j) \in A_r \setminus K$. This property will be exploited to reoptimize T_r^*.

In the following, we will propose an algorithmic approach to reoptimize T_r^*. The reoptimization framework we address determines a suitable decomposition of the arc set K into disjoint subsets, and performs subsequent phases, where each phase reoptimizes with respect to the change of the arc costs of one subset of the decomposition. More precisely, each phase receives in input the current shortest path tree and the current optimum dual vector (which, at the beginning, are T_r^* and $\pi^{(r)}$), modifies the costs of one subset of K, and then performs the related reoptimization using the modified reduced costs $\bar{c}'_{ij}$, by producing an updated shortest path tree and an updated optimum dual vector, which will be the input of the next phase. At the end of the last phase, where the last subset of K is taken into consideration, the updated shortest path tree and the updated optimum dual vector are the shortest path tree and the optimum dual vector relative to the change of the costs of all the arcs in K (*reoptimized shortest path tree* and *reoptimized dual vector*).

The reoptimization framework above described strongly depends on how the decomposition of K is determined. In the next sections, firstly we will describe some simple cases, in which each subset of the decomposition contains either a single arc or a star. In the case in which the arc costs can only decrease, these choices correspond to the algorithm in [7] and to the algorithm in [6], respectively. Then we will propose a more involved and, hopefully, more efficient choice for the decomposition of K, which is at the basis of the proposed reoptimization algorithm.

To simplify the notation, hereafter the current shortest path tree and the current optimum dual vector, received in input by each phase, will be denoted by T_r^* and $\pi^{(r)}$. Moreover, the corresponding reduced costs will be denoted by $\bar{c}_{ij}$. Also the reoptimized shortest path tree, the reoptimized dual vector and the reoptimized reduced costs will be denoted by the same notation.

3.1 Two simple cases

The simplest way to decompose the set K is to consider one arc per phase [7]. Clearly, the resulting algorithm can be impractical when the cardinality of K is very large. Another simple choice is to subdivide K into disjoint "stars" of arcs having a common incident node [6]. Although the number of the phases is always limited by n, also the stars choice can be poor in practice, especially when the change of the costs is spread in the whole graph.

Let us first consider the decomposition of K in single arcs. Consider the current shortest path tree $T_r^* = (N, A_r)$, the current optimum dual vector $\pi^{(r)}$ and the related reduced costs $\bar{c}_{ij}, \forall (i,j) \in A$.

Select an arc $(u,v) \in K$ and modify its cost, by ignoring the change of the cost of each arc other than (u,v). Assume that, as a result of the change of the cost of (u,v), it is $\bar{c}'_{uv} < 0$ (remember that, from (1), it is $\bar{c}'_{ij} = \bar{c}_{ij} \geq 0$, $\forall (i,j) \neq (u,v)$). In this case, the algorithm in [7] reoptimizes T_r^* by performing the following procedure, which uses the modified reduced costs $\bar{c}'_{ij}$, $\forall (i,j) \in A$, as the arc costs:

- initialize the label of each node other than v to 0, and set the label of v to the negative (permanent) value $\bar{c}'_{uv}$: $[d_i := 0, \forall i \neq v;\ d_v := \bar{c}'_{uv}]$; set the predecessor of v to u $[p_v := u]$, while maintaining the other predecessors unchanged;

  initialize the set of the candidate nodes Q with node v: $[Q := \{v\}]$;

- call a *Dijkstra's shortest path procedure* ([1],[2]) starting with the set Q previously initialized, in order to propagate the negative label of v; at each iteration, this procedure extracts a node i with the minimum label from Q, and checks the Bellman's optimality conditions for each arc (i,j) in the forward star of i: [**if** $d_i + \bar{c}'_{ij} < d_j$ **then begin** $d_j := d_i + \bar{c}'_{ij}; p_j := i; Q := Q \cup \{j\}$ **end**].

Observe that each node j inserted to Q has a negative label (in fact, at the beginning each label is 0, except for node v). In particular, d_j measures the contraction of the shortest path from r to j due to the reduction of the cost of (u,v). As a proof of this statement, observe that Dijkstra's procedure reduces the label of each node, in the subtree of T_r^* rooted at v, by the amount $\bar{c}'_{uv}$. Also nodes out of this subtree can be affected by the same cost reduction. However, no node along the path linking r to v in T_r^* can be affected by this label reduction, unless a negative cost cycle exists in G after the change of the cost of (u,v).

Due to the meaning of the node labels, at termination the updated dual vector $\pi^{(r)}$ is given by:

$$\pi_i^{(r)} := \pi_i^{(r)} + d_i, \quad \forall i \in N. \tag{3}$$

In practice, $\pi_i^{(r)}$ needs to be updated only for the nodes i having a negative label d_i, that is the nodes which have been inserted to the set Q. The updated shortest path tree is returned by the above procedure via the predecessor vector $p = [p_i]$.

Since $\bar{c}'_{ij} \geq 0$ for each $(i,j) \neq (u,v)$, each node is inserted to Q at most once. In other words, Dijkstra's procedure behaves as a permanent shortest path algorithm, that is it sets the label of each selected node to its permanent (optimum) value. The shortest path tree can thus be reoptimized with respect to (u,v) in $O(m + nlogn)$ time if the Fibonacci heap data structure is used to implement the set Q [5]. Hereafter we will refer to a Dijkstra's procedure when performing with a permanent behavior as a *Dijkstra's permanent procedure*.

Assume now that, as a result of the change of the cost of (u,v), it is $\bar{c}'_{uv} > 0$. If $(u,v) \notin A_r$, then the tree T_r^* is still optimum. Otherwise, that is $(u,v) \in A_r$, it is possible either to compute the new shortest path tree by calling a Dijkstra's permanent procedure with respect to the costs $\bar{c}'_{ij}$, as suggested in [7], or to reoptimize T_r^* via a dual-ascent operation, like the hanging-dual operation in [14, 15], briefly reviewed in the next section. In both cases, the cost perturbation of the arc (u,v) can be managed in $O(m + nlogn)$ time.

The approach proposed in [6] is similar to the one just reviewed. In that paper, the assumption is made that each arc in K reduces its cost, and the subset of K taken into consideration in each phase is a star relative to a single node i (both ingoing and outgoing arcs). By looking at the arcs entering i, the algorithm updates the label d_i to its optimal value, which is zero if no entering arc has a negative reduced cost after the change of its cost, and it is the minimum among the negative reduced costs otherwise: $[d_i := \min\{\bar{c}'_{ji} : (j,i) \in BS(i)\}]$.

Then, for each arc (i, j) leaving node i, the algorithm updates the label of node j to $d_i + \bar{c}'_{ij}$, if this value is negative (the predecessors of node i and of node j are properly updated). Each node j with a negative label is inserted to the set of the candidate nodes Q, and a Dijkstra's procedure is then applied to propagate the negative labels of the nodes in Q, using the reduced costs $\bar{c}'_{ij}$ as the arc costs. At termination, the predecessor vector $p = [p_i]$ defines the updated shortest path tree, and the updated dual vector can be computed via (3).

As for the case of single arc, it is easy to prove that Dijkstra's algorithm exhibits a permanent behavior. The shortest path tree can thus be reoptimized with respect to the arcs in the star of node i in $O(m + nlogn)$ time.

3.2 The new algorithm

The reoptimization algorithm we propose is inspired by the following observations, which generalize some of the properties exploited in [6, 7].

Observation 1: If the set K is such that $\bar{c}'_{ij} \geq 0, \forall (i, j) \in A$, then $\pi^{(r)}$ is still a feasible dual vector after the change of the arc costs. In this case, in order to restore the complementary slackness conditions (2), it is possible to reoptimize T^*_r by exploiting the dual feasibility (1) via a dual-ascent approach.

Observation 2: If the set K is such that $\bar{c}'_{ij} < 0, \forall (i, j) \in K$, then it may be convenient to decompose K into disjoint subsets, in such a way that the graph structure induced by each subset enables to reoptimize with respect to the arcs in the subset via a Dijkstra's permanent procedure, as in [6, 7].

Based on the above observations, the reoptimization algorithm firstly partitions the set K into two subsets, K^+ and K^-, as follows:

$$K^+ = \{(i, j) \in K : \bar{c}'_{ij} \geq 0\}; \qquad K^- = \{(i, j) \in K : \bar{c}'_{ij} < 0\}.$$

If $K^+ \neq \emptyset$, then the algorithm performs its first phase on the subset K^+. From Observation 1, the first phase is a "dual" phase. If $K^- = \emptyset$, the tree and the dual vector returned by the first phase, T^*_r and $\pi^{(r)}$, respectively, are the reoptimized shortest path tree and the reoptimized dual vector, and the algorithm terminates. Otherwise, the algorithm updates the set K^- if necessary, and dynamically decomposes K^- into a certain number h of subsets, $K^-_1, K^-_2, \ldots, K^-_h$, by performing a "primal" phase for each determined subset. In each primal phase, the reoptimization with respect to the arcs in the considered subset is performed via a Dijkstra's permanent procedure, according to Observation 2. Both the dual phase and the primal phases will be described in detail in the next sections.

3.2.1 The "dual" phase

The dual phase receives in input $T^*_r = (N, A_r)$, $\pi^{(r)}$ and $\bar{c}_{ij}, \forall (i, j) \in A$, as well as the new costs $c'_{ij}, \forall (i, j) \in K^+$. Then, it computes the updated reduced costs $\bar{c}'_{ij}, \forall (i, j) \in K^+$: if $\bar{c}'_{ij} = 0, \forall (i, j) \in A_r$, then the complementarity slackness conditions (2) are satisfied by the dual vector $\pi^{(r)}$. Therefore, T^*_r is the reoptimized shortest path tree.

Otherwise, that is if $\bar{c}'_{ij} > 0$ for at least one arc $(i, j) \in A_r$, then a reoptimization is performed with respect to the arcs in K^+ by suitably adapting the "dual-hanging" approach described in [15], already used in [14] to reoptimize in the case of the root node change.

More in detail, the dual phase considers the forest $F^*_r = (N, A_F)$ obtained from T^*_r by removing all the arcs which do not satisfy the complementarity slackness conditions (2) after the change of the arc costs. That is, $A_F = \{(i, j) \in A_r : \bar{c}'_{ij} = 0\}$.

By construction, the subtree $T(r) = (N(r), A(r))$ of F^*_r, which is rooted at r, is a fragment of the reoptimized shortest path tree. The idea is to reconnect the other subtrees to $T(r)$ by performing some suitable hanging operations. Let us briefly explain what *hanging operation* means; for more details, we refer the interested reader to [14]. Consider the cut $(N(r), \bar{N}(r))$, where $\bar{N}(r) = N \backslash N(r)$, and the following two sets of arcs:

- the set of the *border arcs* $A^+ = \{(i,j) \in A : i \in N(r), j \in \bar{N}(r)\}$;
- the set of the *external arcs* $A^E = \{(i,j) \in A : i \in \bar{N}(r), j \in \bar{N}(r)\}$.

For each node $j \in \bar{N}(r)$, denote by δ_j and α_j the following values:

$$\begin{aligned} \delta_j = &\quad \min\{\bar{c}'_{ij} : (i,j) \in BS(j) \cap A^+\}; \\ \alpha_j = &\quad \min\{\bar{c}'_{ij} : (i,j) \in BS(j) \cap A^E\}. \end{aligned}$$

Moreover, for each node $j \in \bar{N}(r)$ denote by $N^+(j)$ and $N^-(j)$ the sets of the nodes which are ancestors and descendants of j in F_r^*, respectively[2]. Then let:

$$\begin{array}{ll} \delta_j^+ = \min\{\delta_i : i \in N^+(j)\} & \alpha_j^+ = \min\{\alpha_i : i \in N^+(j)\} \\ \delta_j^- = \min\{\delta_i : i \in N^-(j)\} & \alpha_j^- = \min\{\alpha_i : i \in N^-(j)\} \end{array}$$

be the minimum of the δ's and of the α's values by looking at the ancestors and at the descendants of j in F_r^*, respectively (these values are assumed to be ∞ when the corresponding sets are empty).

Finally, define the *gap* Δ relative to the cut $(N(r), \bar{N}(r))$ as:

$$\Delta = \min\{\delta_j : j \in \bar{N}(r)\},$$

and denote by w a node such that $\delta_w = \Delta$.

By exploiting the above definitions, a node $v \in \bar{N}(r)$ is said to be *hangable* if:

$$\delta_v \leq \min\{\delta_v^+, \delta_v^-\} \qquad \text{and} \qquad \delta_v \leq \Delta + \min\{\alpha_v^+, \alpha_v^-\}.$$

If v is hangable, then the dual phase "hangs" to $T(r)$ the subtree $T(v) = (N(v), A(v))$ of F_r^* rooted at v, through the arc $(u,v) \in A^+$ such that $\bar{c}'_{uv} = \delta_v$. In this way, $T(r) = (N(r), A(r))$ enlarges as follows:

$$N(r) := N(r) \cup N(v); \qquad A(r) := A(r) \cup A(v) \cup \{(u,v)\}.$$

After the hanging, the dual vector is updated by traversing $T(v)$ and setting:

$$\pi_j^{(r)} := \pi_j^{(r)} + \delta_v, \quad \forall j \in N(v).$$

Observe that the set of the hangable nodes is never empty. In particular, it always contains node w. Moreover, more nodes can be hangable simultaneously, and their subtrees can be hanged in parallel to $T(r)$.

At the end of the hanging operations, the arc sets A^+ and A^E are updated, together with the δ's and the α's values, and new hanging operations are performed. The dual phase terminates when the last subtree has been hanged to $T(r)$. When this happens, $T(r)$ is the reoptimized shortest path tree with respect to the arcs in K^+, and it will be provided in input to the primal phases.

The correctness and the time complexity of the dual phase follow from the correctness and the time complexity of the hanging algorithms [14, 15]. In particular, consider the list formed by the groups of subtrees which are hanged in parallel to $T(r)$, and denote by k the cardinality of this list. Observe that $k \leq n - n_r$, where $n_r = |N(r)|$ is the size of the first fragment of the reoptimized shortest path tree. Then:

Theorem 1: The dual phase runs in $O(m + kn)$ time.

[2] We assume that node j belongs to $N^+(j)$ and to $N^-(j), \forall j \in \bar{N}(r)$.

In fact, it is possible to implement the dual phase in such a way to compute and update the δ's and the α's values in $O(m)$ time; moreover, it is possible to compute the δ^+'s, δ^-'s, α^+'s and the α^-'s values in $O(n)$ time for each group of subtrees which are hanged in parallel to $T(r)$.

Since the dual vector $\pi^{(r)}$ can change as a result of the dual phase, it may happen that some arcs $(i,j) \in K^-$ have no longer a negative reduced cost after the change of their cost, that is $\bar{c}'_{ij} \geq 0$. Observe that this is possible only for arcs out of the tree $T(r)$ returned by the dual phase. In this case, before moving to the primal phases, the algorithm changes the cost of each arc (i,j) of this type without modifying $T(r)$ (it is still the current shortest path tree), and it removes (i,j) from the set K^-.

3.2.2 The primal phases

The idea behind the primal phases is to exploit the graph structure induced by the arcs $(i,j) \in A$ with $\bar{c}'_{ij} \leq 0$, in order to decompose the set K^- into a certain number h of suitable subsets, $K_1^-, K_2^-, \ldots, K_h^-$. More precisely, we suggest to generalize the ideas in [6] as follows.

Let $G_0^- = (N, A_0^-)$ be the partial graph of G formed by all the arcs with a non positive reduced cost. Formally, $A_0^- = \{(i,j) \in A : \bar{c}'_{ij} \leq 0\}$.

Observe that G_0^- contains both the arcs in K^-, and the arcs with a zero reduced cost. As explained in the following section, the arcs with a zero reduced cost will be used to aid the computation of a "small" number of subsets in the decomposition of K^-. Observe also that we can construct G_0^- in such a way to be acyclic. In fact, if G_0^- contains a cycle, then this cycle must have a zero reduced cost, since we have assumed that no negative cycle is created in G after the change of the arc costs. In this case, we can unfold all the zero cost cycles, by removing one arc of each cycle from G_0^-.

Now, we want to dynamically decompose K^- into subsets, $K_1^-, K_2^-, \ldots, K_h^-$, in such a way that each subset K_j^-, $j = 1, \ldots, h$, belongs to a special subgraph of G_0^-, $G(K_j^-)$, satisfying the following definition:

Definition 1: A *Dijkstrable subgraph* is a subgraph of G_0^- whose set of arcs can be reoptimized via a single call to a Dijkstra's permanent procedure, independently of the (non-negative) costs of the arcs not belonging to the subgraph.

In other words, the structure of $G(K_j^-)$, $j = 1, \ldots, h$, must be such that the reoptimization due to the change of the costs of the arcs in $G(K_j^-)$ can be performed via a permanent labelling algorithm, that is a Dijkstra's algorithm.

More in detail, each primal phase performs the following operations:

- compute a Dijkstrable subgraph $G(K_j^-) = (N(K_j^-), A(K_j^-))$ of G_0^-, which contains at least one arc in K^-; let $K_j^- = A(K_j^-) \cap K^-$;
- change the costs of the arcs in K_j^-, by ignoring the change of the costs of the other arcs in K^- (therefore, $\bar{c}'_{ij} = \bar{c}_{ij}$ for each $(i,j) \in K^- \setminus K_j^-$);
- initialize the label of each node $i \in N$ to 0: $[d_i := 0, \forall i \in N]$;

  visit the acyclic graph $G(K_j^-)$ in a topological order, and set the label d_v of each traversed node $v \in N(K_j^-)$ to its permanent value:

  $$d_v := \min\{d_u + \bar{c}'_{uv} : (u,v) \in BS(v) \cap A(K_j^-)\}; \tag{4}$$

  accordingly, update the predecessor $p = [p_i]$; then, initialize the set of the candidate nodes Q with the nodes having a negative label: $[Q := \{v \in N(K_j^-) : d_v < 0\}]$;

- call a Dijkstra's shortest path procedure using the set Q previously initialized, in order to propagate the negative labels of the nodes in $N(K_j^-)$; as a result of the reoptimization, update the dual vector $\pi^{(r)}$ according to (3);
- update the arc reduced costs $\bar{c}'_{ij}$; consequently, update the set K^-, by removing all the arcs which have a non negative reduced cost; then update the graph G_0^-.

The primal phases terminate when K^- becomes empty. The following properties can be easily proved.

Lemma 1: At the end of each primal phase, no arc $(i,j) \notin K^-$ has a negative reduced cost; moreover, the cardinality of K^- strictly decreases.

Therefore:

Corollary 1: The reoptimization algorithm performs no more than $|K^-|$ primal phases.

Now, let us investigate how to efficiently discover Dijkstrable subgraphs.

3.2.3 About Dijkstrable subgraphs and their computation

Let us illustrate the main properties of the Dijkstrable subgraphs via the following examples.

Example 1
Consider the subgraph $G(K_j^-) = (N(K_j^-), A(K_j^-))$ depicted in figure 1, where the integers associated with the arcs denote the current reduced costs.

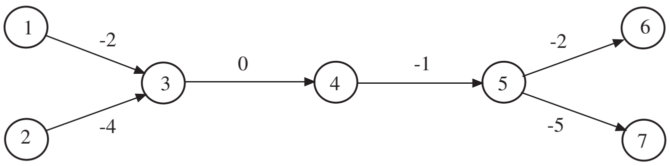

Figure 1: A Dijkstrable subgraph

If we visit $G(K_j^-)$ in a topological order, and perform the operations (4), then we obtain the set of (permanent) labels $d = [0, 0, -4, -4, -5, -7, -10]$. Let us initialize the set of the candidate nodes Q with the nodes having a negative label, i.e., $Q = \{3, 4, 5, 6, 7\}$. If we apply a Dijkstra's shortest path procedure starting from Q, we can determine the optimal labels of the nodes out of $G(K_j^-)$ by examining each node and each arc at most once (recall that all the arcs out of $G(K_j^-)$ have a non-negative reduced cost). Therefore, this is an example of a Dijkstrable subgraph, corresponding to the set $K_j^- = \{(1,3), (2,3), (4,5), (5,6), (5,7)\}$. Observe that the arc $(3,4)$, having a zero reduced cost, is a link between the two subgraphs induced by the arcs $\{(1,3), (2,3)\}$ and $\{(4,5), (5,6), (5,7)\}$, belonging to K^-: it allows to manage simultaneously the two subsets of arcs.

Example 2
Consider now the subgraph $G(K_j^-) = (N(K_j^-), A(K_j^-))$ depicted in figure 2.

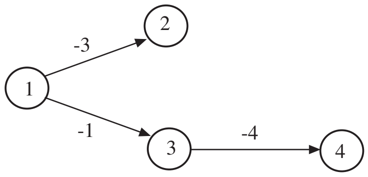

Figure 2: A subgraph which is not Dijkstrable

If we perform a topological visit and apply (4), we obtain $d = [0, -3, -1, -5]$ and $Q = \{2, 3, 4\}$. Now, assume that, out of this subgraph, there exists the arc $(2, 3)$ with reduced cost $\bar{c}'_{23} = 1$. In this case, if we call a Dijkstra's shortest path procedure, then node 4 is examined twice (the second time, when node 3 with label $d_3 = -2$ is extracted from Q). So, according to Definition 1, this subgraph is not a Dijkstrable subgraph.

Example 3
Figure 3 shows another example of subgraph which is not Dijkstrable.

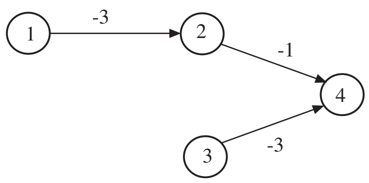

Figure 3: A subgraph which is not Dijkstrable

In this case, the topological visit returns $d = [0, -3, 0, -4]$, and $Q = \{2, 4\}$. Also in this situation, the presence of an arc $(2, 3)$ with reduced cost $\bar{c}'_{23} = 1$ would cause the reinsertion of node 4 in Q, with label $d_4 = -5$.

From the above examples, a Dijkstrable subgraph cannot contains subgraphs which are isomorphic to the ones in figures 2 and 3. As a consequence, the most complex structure which seems to be a candidate to figure out as a Dijkstrable subgraph is the following:

Definition 2: A *starpath centered at i and j* is a subgraph of $G_0^- = (N, A_0^-)$ consisting of a directed path from node i to node j, plus the two sets of arcs (stars):

$$\bar{B}S(i) = BS(i) \cap K^-, \quad \bar{F}S(j) = FS(j) \cap K^-.$$

The subgraph in figure 1 is a starpath centered at 3 and 5. Observe that the definition of starpath includes the star as its special case ($i = j$).

Consider now a starpath centered at i and j, and denote by (v, i) an arc of $\bar{B}S(i)$ with the minimum reduced cost ($v = i$ if $\bar{B}S(i) = \emptyset$):

$$\bar{c}'_{vi} = \min\{\bar{c}'_{ui} : (u, i) \in \bar{B}S(i)\}. \tag{5}$$

Moreover, denote by q the end node of an arc $(j, q) \in \bar{F}S(j)$ with the minimum reduced cost, if this set is not empty ($q = j$ otherwise):

$$\bar{c}'_{jq} = \min\{\bar{c}'_{ju} : (j, u) \in \bar{F}S(i)\}. \tag{6}$$

Refer to the path from v to q in the starpath, P_{vq}, as the *support* of the starpath. In figure 1, it is $v = 2$ and $q = 7$, and the support of the starpath is P_{27}.

Lemma 2: Consider a starpath centered at i and j and its support path P_{vq}. Then P_{vq} belongs to the shortest path tree obtained by reoptimizing with respect to the arcs of the starpath.

Proof: Since no negative cycle is created in G after the change of the arc costs, the reoptimized shortest path from r to i includes the arc (v, i), and the reoptimized shortest path from r to q is given by the concatenation of the reoptimized shortest path from r to i and of the directed path from i to q in the starpath. Therefore, P_{vq} belongs to the reoptimized shortest path tree. □

Corollary 2: Any starpath centered at i and j is a Dijkstrable subgraph.

Proof: It follows from Lemma 2 and from the property that the arcs out of the starpath have a non-negative (reduced) cost. □

In order to compute a starpath, we suggest two alternative procedures.

Consider the graph $G_0^- = (N, A_0^-)$ previously defined. Observe that it contains both the arcs in K^- and the arcs of the last computed shortest path tree, since these arcs have a zero reduced cost[3]. Therefore, G_0^- is formed by a spanning tree rooted at r, plus some additional arcs.

The first procedure consists of performing a depth-first search visit of G_0^- starting from the root r. The visit terminates when a leaf of the current tree T_r^* is reached, and the path from r to that leaf contains at least one arc belonging to K^-; let (j, u) denote the last arc of K^- along this path. When this happens, the procedure moves back from j to r by following the predecessors set up by the visit procedure, and computes the center node i (possibly coinciding with j) and the origin node v (possibly coinciding with i) of a support path P_{vq}, where q satisfies (6). In this way the procedure discovers a starpath centered at i and j, in $O(|A_0^-|) = O(m)$ time.

An alternative procedure consists of computing a shortest path tree, rooted at r, in the acyclic graph G_0^-. As known from the literature, this can be performed in $O(|A_0^-|) = O(m)$ time via a topological visit of G_0^-. Once determined the tree, the procedure selects a node q with the minimum (negative) label and with an entering arc (j, q) belonging to K^-, and computes a starpath by moving back from q towards the root r along the computed tree. Also in this case, a starpath can be determined in $O(|A_0^-|) = O(m)$ time.

Now, if we select a starpath during each primal phase, and denote by ν the number of nodes of G_0^- having at least one incident arc in K^-, we can refine Corollary 1 as follows:

Lemma 3: The number of the primal phases performed by the reoptimization algorithm is:

$$h \leq \min\{\nu, |K^-|\}.$$

Since the dominant computational component of each primal phase is the call to a Dijkstra's shortest path procedure, if we choose the "Fibonacci heap" implementation of Dijkstra's algorithm we get:

Theorem 2: The overall time complexity of the primal phases is $O(h(m + nlogn))$.

From Theorems 1 and 2:

Theorem 3: The time complexity of the proposed reoptimization algorithm is $O(h(m + nlogn) + kn)$, where $h \leq \min\{\nu, |K^-|\}$ and $k \leq n - n_r \leq n$.

4 Conclusions

We have proposed a new algorithm which reoptimizes shortest paths when any subset of arcs of the input graph is affected by a change of the arc costs, which can be either lower or higher than the old ones.

Although presented here as a pure methodological approach, we think that the proposed algorithm can be competitive in practice. In fact, quite frequently, the perturbation of the costs in practical applications does not involve the entire network under consideration, but small and structured portions of it. For example, consider a suburban area affected by car congestion during the peak morning period. If one analyzes the related transportation network, and computes the shortest paths from all the origin points of that area to the Central Business District (CBD) every $15'$, then one can observe that the arcs whose cost changes form an irregular "ring" centered at the CBD, which becomes closer and closer to it as the time passes. Therefore, only a small, structured, subset of the arcs tend to perturb their cost. Another typical example is the congestion along a very crowded highway: the portion of the transportation network whose arcs show a cost perturbation

[3]W.l.o.g. we assume here that, if zero cost cycles are present in G_0^-, the arcs removed from G_0^- to unfold these cycles and to convert G_0^- in an acyclic graph do not belong to the current shortest path tree.

forms generally a long stream (the highway itself and the surrounding routes), easy to decompose in few, long, starpaths.

An interesting theme of research will be to trace out the border which separates set of instances for which the proposed reoptimization algorithm is efficient in practice, from those instance for which our reoptimization procedure looses with respect to more standard optimization approaches.

Acknowledgements. The research has been supported by INDAM-GNAMPA and MURST Cofinanziato 2000.

References

[1] Ahuja, R. K., Magnanti, T. L. and Orlin, J. B. (1993). *Network flows: Theory, algorithms, and applications.* Prentice Hall, Englewood Cliffs, NJ.

[2] Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs". *Numerishe Matematik* 1, 269–271.

[3] Dionne, R. (1978). "Étude et extension d'un algorithme de Murchland". *INFOR* 16, 132–146.

[4] Florian, M., Nguyen, S. and Pallottino, S. (1981). "A dual simplex algorithm for finding all shortest paths". *Networks* 11, 367–378.

[5] Fredman, M. L. and Tarjan, R. E. (1987), "Fibonacci heaps and their uses in improved network optimization algorithms". *Journal of the ACM* 34, 596–615.

[6] Fujishige, S. (1981). "A note on the problem of updating shortest paths". *Networks* 11, 317–319.

[7] Gallo, G. (1980). "Reoptimization procedures in shortest path problems". *Rivista di Matematica per le Scienze Economiche e Sociali* 3, 3–13.

[8] Gallo, G. and Pallottino, S. (1982). "A new algorithm to find the shortest paths between all pairs of nodes". *Discrete Applied Mathematics* 4, 23–35.

[9] Gallo, G. and Pallottino, S. (1988). "Shortest path algorithms". *Annals of Operations Research* 13, 3–79.

[10] Goto, S. and Sangiovanni-Vincentelli, A. (1978). "A new shortest path updating algorithm". *Networks* 8, 341–372.

[11] Lawler, E. L. (1976). *Combinatorial optimization: Networks and matroids.* Holt, Rinehart and Winston, New York.

[12] Murchland, J. D. (1970). *A fixed matrix method for all shortest distances in a directed graph and for the inverse problem.* Ph.D. Thesis, Univ. of Karlsruhe.

[13] Nemhauser, G. (1972). "A generalized permanent label setting algorithm for the shortest path between specified nodes". *Journal of Mathematical Analysis and Applications* 38, 328–334.

[14] Nguyen, S., Pallottino S. and Scutellà M. G. (2001). "A new dual algorithm for shortest path reoptimization". Forthcoming in (M. Gendreau and P. Marcotte, eds.) *Transportation and Network Analysis - Current Trends*, Kluwer.

[15] Pallottino S. and Scutellà M. G. (1997). "Dual algorithms for the shortest path tree problem"., *Networks* 29, 125–133.