Shortest paths on dynamic graphs

Giacomo Nannicini[1,2], Leo Liberti[1]

[1] *LIX, École Polytechnique, F-91128 Palaiseau, France*
Email:giacomon,liberti@lix.polytechnique.fr

[2] *Mediamobile, 10 rue d'Oradour sur Glane, 75015 Paris, France*
Email:giacomo.nannicini@v-trafic.com

March 15, 2008

Abstract

Among the variants of the well known shortest path problem, those that refer to a dynamically changing graphs are theoretically interesting, as well as computationally challenging. Application-wise, there is an industrial need for computing point-to-point shortest paths on large-scale road networks whose arcs are weighted with a travelling time which depends on traffic conditions. We survey recent techniques for dynamic graph weights as well as dynamic graph topology.

1 Introduction

A graph is *dynamic* when some of the graph entities (vertices, edges, weights) change with time. The most usual time-dependent changes are in the edge weights, which can also model edge connection/disconnection if we allow an arc cost to be infinite. Several combinatorial optimization problems on graphs may be defined on dynamic graphs, and in most cases this fact significantly alters the problem definition, so that the employed methods are different in the static and dynamic case (the LEDA algorithmic software system, for example, provides specific data structures for dealing with dynamic graphs [26]). In this paper, we focus on problems arising from finding shortest paths in graphs. Although the Shortest Path Problem (SPP) is one of the best studied combinatorial optimization problems in the literature [1, 37], the dynamic graph variants received much less attention over the years. In this paper, we survey some of the results in this field.

We distinguish two main categories for the SPP on dynamic graphs. The first one is usually called the *time-dependent* variant in the literature: the cost of an arc is a travelling time which is given by a pre-determined function of time, which means that the cost of an arc (u, v) on a path depends on the starting time of the path and on the time already spent to reach u. The second one does not have a common name in literature; it deals with graphs where the cost function changes or is updated after a certain time interval, but the graph is static between two cost function changes. We will call this variant *cost-update*.

The rest of this paper is organized as follows. In Sect. 2, we discuss some pre-1980 developments. In Sect. 3 we investigate some methods for finding SPP trees in graphs with dynamic topology, based on reoptimization. In Sect. 4 we survey recent developments in finding point-to-point shortest paths in large dynamic graphs with changing arc weights. Sect. 5 concludes the paper.

2 Early history

The main direct application of shortest path type problems is into the theory of transportation. A lot of early work was carried out on related topics at the RAND corporation, but it was mostly to do with

transportation network analysis (on dynamic networks where the capacities changed according to traffic congestion) rather than the shortest path to be chosen by any individual driver [4].

The first citation we could find concerning the SPP on dynamic graphs with time-dependent edge weight changes is [8] (a good review of this paper can be found in [14], p. 407): a recursive formula is given to establish the minimum time to travel to a given target starting from a given source at time τ. It is shown that if travel times take on integer positive values then the procedure terminates with the shortest path from all nodes to a given destination. Let $f_i(\tau)$ be the minimum time of travel to a node t starting at node i at time τ, and let $c(i, j, \tau)$ be the travelling time on the arc (i, j) starting from i at time τ; the procedure is based on the formula

$$f_i(\tau) = \min_{j \neq i} \{c(i, j, \tau) + f_j(\tau + c(i, j, t))\} \;\; f_t(\tau) = 0.$$

In [14], Dijkstra's algorithm [12] is extended to the dynamic case, but the FIFO property, which is necessary to prove that Dijkstra's algorithm terminates with a correct shortest paths tree on time-dependent networks, is not mentioned.

The FIFO property states that for each pair of time instants τ, τ' with $\tau < \tau'$:

$$\forall\, (u, v) \in A \;\; c(u, v, \tau) + \tau \leq c(u, v, \tau') + \tau',$$

The FIFO property is also called the *non-overtaking property*, because it basically says that if T_1 leaves u at time τ and T_2 at time $\tau' > \tau$, T_2 cannot arrive at v before T_1 using the arc (u, v). Although FIFO networks are useful for the study of those means of transportation where overtaking is rare (such as trains), modelling of car transportation yields networks which do not necessarily have the FIFO property. For the time-dependent variant of the SPP, the FIFO assumption is usually necessary in order to mantain an acceptable level of complexity: the SPP in time-dependent FIFO networks is polynomially solvable [24], even in the presence of traffic lights [2], while it is NP-hard in non-FIFO networks [32].

Early studies on general transportation networks were mostly motivated by transportation planning, i.e. network analysis in order to optimize investments to improve the current road network; see [21] for a survey. This required to study the effect of modifying a link on the routes chosen by the network's users. A road network was modeled as a graph where each link had an associated travelling time and a capacity, and nodes corresponded to entry points on the road network of particular zones [21]. Thus, only *interzonal* travelling times affected the road network. The number of individuals that chose a particular source-destination pair at each time of the day was supposed to be known by demographical studies or trip generation techniques, and routes were assigned computing the shortest paths tree rooted at each node of the network. The first case to be analyzed is the shortening of a link [25, 27], i.e. its associated travelling time decreases: it is observed that in this situation the length of the shortest path between two nodes s, t will decrease only if the shortest path between s, t passing through the affected arc is shorter than the previous solution. Thus, if (u, v) is the link to be shortened, $d(s, t)$ is the initial cost of the shortest path between two nodes s, t, and $c(u, v)$ is the new cost of arc (u, v), the new shortest path distances can be computed as $d'(s, t) = \min\{d(s, t), d(s, u) + c(u, v) + d(v, t)\}$. The *method of competing links* [20] analyzed the effect of an arbitrary change in the cost of a link in a cutset: the graph was partitioned in two sets Z_1, Z_2, and if we call C the set of arcs connecting the two node sets then the travelling time between two nodes $s \in Z_1, t \in Z_2$ was computed as

$$\min_{(p,q) \in C} (d(s, p) + d(p, q) + d(q, t)),$$

where again $d(i, j)$ is the cost of the shortest path from i to j. As only the costs of arcs in the cutset C were allowed to change, the new shortest paths trees were easily computed.

The first attempts of solving the cost-update variant of the SPP were done through reoptimization techniques: in particular, [28] considers the problem of finding the shortest path cost matrix when only one arc of the input graph changes its cost. The same problem was investigated a few years later in

[13]. [17] addresses the SPP on dynamic graphs where either an arc changes its cost or a different root node is selected, and lays the foundation for future work; it proposes a procedure to reduce the complexity of Dial's implementation [11] of Dijkstra's algorithm. The number of comparisons needed by Dial's implementation depends on the cost of the longest shortest path from the root to all other nodes of the graph; in order to reduce this cost, [17] modifies the length of all arcs with the formula $c'(i,j) = c(i,j) + \pi_i - \pi_j$, where c' is the new cost function, c is the old cost function, and $\pi_i \; \forall i \in V$ is a positive integer such that $c'(i,j) \geq 0 \; \forall (i,j) \in A$. It is noted that a transformation of this kind does not modify which arcs appear on a shortest path, and was first proposed in [31] in order to get non-negative arc costs on graphs with $c(i,j) < 0$ for some $(i,j) \in A$. The interpretation of the vector $(\pi_1, \ldots, \pi_{|V|})$ as a dual feasible solution to the SPP is due to [3].

3 Reoptimization of shortest paths trees

Given a graph where the cost function may be updated, one possible approach to compute shortest paths is the reoptimization of shortest paths trees (SPTs) whenever the cost function changes. Suppose we have already computed the SPT rooted at r; if there is a change in the cost function which is not too severe, the reoptimization of the current SPT may be faster than recomputing everything from scratch. Also, note that selecting a different root node r can be seen as a special case of cost function's change (see [34]).

The first work to address the general case of an arbitrary arc cost change (independently of sign and for any number of arcs at the same time) is [34], which is based on a dual approach to the problem. The SPT problem can be formulated as follows:

$$(SPT_r) \begin{cases} \min \displaystyle\sum_{(i,j)\in A} c_{ij} x_{ij} \\ \displaystyle\sum_{(j,i)\in A} x_{ji} - \sum_{(i,j)\in A} x_{ij} = \begin{cases} -|V|+1, & i = r \\ 1, & \forall\, i \in V \setminus \{r\} \end{cases} \\ x_{ij} \geq 0 \; \forall\, (i,j) \in A, \end{cases}$$

with dual:

$$(DSPT_r) \begin{cases} \max(1-|V|)\pi_r + \displaystyle\sum_{j\in V\setminus\{r\}} \pi_j \\ \pi_j - \pi_i \leq c_{ij} \; \forall\, (i,j) \in A \end{cases}$$

Supposing the optimal solution to $(DSPT_r)$ is the dual vector $\pi^{(r)}$ for a given root node r, if a subset $K \subset A$ of the arcs set has its costs updated, we can compute the new reduced costs of the arcs with respect to the optimum dual vector as $\bar{c}'_{ij} = c'_{ij} + \pi_i^{(r)} - \pi_j^{(r)}$, where c'_{ij} are the new costs. By complementary slackness conditions, for an optimal solution $\bar{c}'_{ij}$ should be non-negative for all arcs, and equal to zero for arcs in the optimal SPT; however, due to the updated costs this condition may fail to hold. The suggested strategy is to reoptimize the arcs with positive updated reduced cost and those with negative updated reduced cost separately, in order to obtain the new optimum. Several ideas are proposed for both situations. For arcs with non-negative updated reduced cost $\bar{c}'_{ij} \geq 0$, the authors suggest using the "dual-hanging" algorithm presented in [33], which considers the forest generated by deleting all arcs of the SPT who do not satisfy the complementary slackness conditions after the change of arc cost, and iteratively "hangs" a tree of this forest to the main fragment, that is the one which contains the root r. Arcs with $\bar{c}'_{ij} = 0$ are used to speed up the process, allowing several sub-trees to be hanged at one time. For arcs with negative updated reduced cost $\bar{c}'_{ij} < 0$, they propose the identification of particular structures, called "starpaths", within the graph formed by all arcs with non positive updated reduced cost; for each starpath, the reoptimization can be done by means of a slight modification of Dijkstra's algorithm, which computes the new optimum dual vector visiting each vertex of the starpath only once.

The shortest path tree reoptimization algorithm requires linear time, if the cost perturbation is linear in the number of arcs [34]; otherwise, time requirements may vary.

4 The point-to-point SPP

A current problem faced by traffic information providers is that of offering GPS terminal enabled drivers a source-destination path subject to the following constraints: (a) the path should be fast in terms of travelling time; (b) the travelling times (weights on the edges) vary according to traffic information being available on part of the road network; (c) the graph topology is fixed; (d) traffic information data are updated at regular time intervals; (e) answers to path queries should be computed in real time. Given these engineering requirements, it is possible to apply to the problem SPP algorithms defined on static graphs as long as the computation speed of a single point-to-point shortest path is much faster than the edge weight update rate. This, in practice, requires point-to-point SPP algorithms that are capable of computing a solution in a few milliseconds in graphs of several million nodes.

In Sect. 4.1 we briefly review Dijkstra's algorithm and its bi-directional variant, which are at the basis of all other algorithms that we will describe in the following. In the rest of this section we survey three methods for finding point-to-point shortest paths on dynamic graphs. The first of these methods (Sect. 4.2) has been conceived only for the cost-update variant of the SPP, while the remaining two (Sect. 4.3 and Sect. 4.4) are also applicable to graphs with time-dependent arc costs.

4.1 Dijkstra's algorithm: uni- and bi-directional

Dijkstra's algorithm (see [12]) solves the single source shortest path problem in static directed graphs with non-negative weights in polynomial time; it also solves the problem in the presence of negative weights, but it may require exponential time in the worst case. Dijkstra's algorithm is a so-called labeling method.

The *labeling method* for the SPP [15] finds shortest paths from the source to all vertices in the graph; the method works as follows: for every vertex v it maintains its distance label $d(v)$, parent node $p(v)$, and status $S(v) = \{\texttt{unreached}, \texttt{explored}, \texttt{settled}\}$. Initially $d(v) = \infty$, $p(v) = NIL$, and $S(v) = \texttt{unreached}$ for every vertex v. The method starts by setting $d(s) = 0$ and $S(s) = \texttt{explored}$; while there are labeled (i.e. explored) vertices, the method picks an `explored` vertex v, relaxes all outgoing arcs of v, and sets $S(v) = \texttt{settled}$. To relax an arc (v, w), one checks if $d(w) > d(v) + c(v, w)$ and, if true, sets $d(w) = d(v) + c(v, w)$, $p(w) = v$, and $S(w) = \texttt{explored}$. If the graph does not contain cycles with negative cost, the labeling method terminates with correct shortest path distances and a shortest path tree; its efficiency depends on the rule to choose a vertex to scan next. We say that $d(v)$ is exact if it is equal to the distance from s to v; it is easy to see that if one always selects a vertex v such that, at the selection time, $d(v)$ is exact, then each vertex is scanned at most once. Dijkstra [12] observed that if the cost function c is non-negative and v is an explored vertex with the smallest distance label, then $d(v)$ is exact; so, we refer to the labeling method with the minimum label selection rule as Dijkstra's algorithm. If c is non-negative then Dijkstra's algorithm scans vertices in nondecreasing order of distance from s and scans each vertex at most once; for the point-to-point SPP, we can terminate the labeling method as soon as the target node is `settled`. The algorithm requires $O(m + n \log n)$ time if the queue is implemented as a heap data structure such as binary heaps or Fibonacci heaps [16].

One basic variant of Dijkstra's algorithm for the point-to-point SPP is bidirectional search; instead of building only one shortest path tree rooted at source node s, we also build a shortest path tree rooted at target node t on the reverse graph $\bar{G} : (V, \bar{A})$ where $(u, v) \in \bar{A} \Leftrightarrow (v, u) \in A$. As soon as one node v becomes `settled` in both searches, we are guaranteed that the concatenation of the shortest $s \to v$ path found in the forward search and of the shortest $v \to t$ path found in the backward search is a shortest $s \to t$ path. Since we can think of Dijkstra's algorithm as exploring nodes in circles centered at s with increasing radius until t is reached (see Fig. 1), the bidirectional variant is faster because it explores nodes

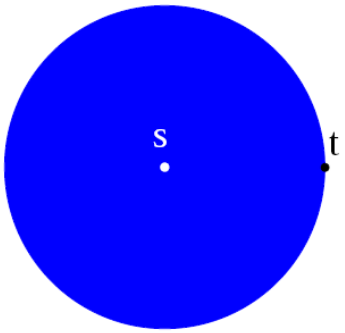

Figure 1: Schematic representation of Dijkstra's algorithm search space

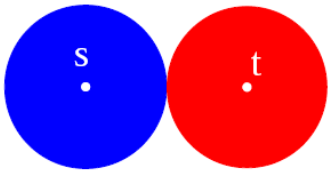

Figure 2: Schematic representation of bidirectional Dijkstra's algorithm search space.

in two circles centered at both s and t, until the two circles meet (see Fig. 2); the area within the two circles, which represents the number of explored nodes, will then be smaller than in the unidirectional case, up to a factor of two.

Dijkstra's algorithm applied to time-dependent FIFO networks has been optimized in various ways [5, 6]. We note here that in the time-dependent scenario bidirectional search cannot be applied, since the arrival time at destination node is unknown. We also remark that all speed-up techniques based on finding shortest paths in Euclidean graphs [38] cannot be applied either, since the typical arc cost function, the arc travelling time at a certain time of the day, does not yield a Euclidean graph.

4.2 Dynamic Node Routing

Separator-based multi-level methods for the SPP have been used by many authors; we refer to [22] for the basic variant. The main idea behind separator-based methods is to define, given a subset of the vertex set $V' \subset V$, the shortest path overlay graph $G' = (V', A')$ with the property that A' is a minimal set of edges such that $\forall u, v \in V'$ the shortest path length between u and v in G' is equal to the shortest path length between u and v in G. In other words, there is an arc $(u, v) \in A'$ if and only if for any shortest path from u to v in G then no internal node of the paths (i.e. all nodes except u and v) belongs to V'. It can be shown that G' is unique [22]. Usually, the set of separator nodes V' is chosen in such a way that the subgraph induced by $V \setminus V'$ consists of small components of similar size. In a bidirectional query algorithm, the components containing source and target node are wholly searched, but starting from the separator nodes only edges of the overlay graph G' are considered. This approach can be generalized and applied in a hierarchical way, building several levels of overlay graphs with node sets $V = V_0 \supseteq V_1 \supseteq \cdots \supseteq V_L$ so that the following property is mantained: $\forall \ell \leq L - 1$, for all node pairs $s, t \in V_\ell$ the part of the shortest path between s and t that lies outside the level ℓ components to which s and t belong is entirely included in the level $\ell + 1$ overlay graph. As the overlay graphs become smaller with the increasing level in the hierarchy, a shortest path computation becomes faster because most of the search for a long-distance s, t path takes place on the highest hierarchy level, and thus fewer nodes are explored. A path on the original

graph can then be reconstructed, because each arc at level ℓ has a unique decomposition as level $\ell - 1$ arcs.

In [36], an arbitrary subset $V' = V'(V)$ of V is considered instead of separator nodes; in practice, the set is chosen in such a way that it contains the most important nodes, i.e. those that appear "more often" on shortest paths. This yields a smaller set V', more uniformly distributed over the whole graph, and thus G' will be smaller, resulting in a smaller space consumption and a faster query algorithm. However, since in this case $V \setminus V'$ is no longer made of small isolated components, the query algorithm is not as simple as in canonical separator-based methods. From a theoretical point of view the same principle holds: we might want to explore nodes from source and target until the queue in Dijkstra's algorithm only contains nodes that are covered by V' (i.e. there is at least one node $v \in V'$ on the shortest path from the root to any leaf of the current partial shortest path tree), and then switch to the overlay graph G', or to a higher level in the overlay graph hierarchy in the case of a multi-level approach. This, however, does not yield good results in practice, because we cannot tell in advance how many nodes we will have to explore until the whole partial shortest path tree is covered by V'. The main challenge is therefore to compute the set of all covering nodes for the partial shortest path tree T rooted at s as quickly as possible.

Many possible strategies are suggested in [36], including an aggressive variant which stops the search whenever a node in V' is encountered, and which yields a superset of the covering nodes. Some other analyzed possibilities may require a greater computational effort, while finding the minimal set of covering nodes. Once the set (or a superset) of all covering nodes for a given level of the overlay graph has been computed, the search can switch to the next level, until the shortest path is found, which is guaranteed to happen at the topmost level. The choice of level node sets $V = V_0, V_1, \ldots, V_L$, where $V_i = V'(V_{i-1})$ for all $i > 0$, is critical for query times: these nodes should correspond to nodes that appear very often on shortest paths, i.e. road network junctions with high-importance, such as highway access points. The Highway Hierarchies algorithm [35] is employed in [36] to choose the node sets.

The main advantage of this approach is that overlay graphs can be computed in a very short time because they only require the application of Dijkstra's algorithm on limited parts of the graph; besides, if a few arc costs change there is no need to recompute the whole overlay graphs, but only a small part of them — the part which is affected by the change. Certainly, if the changed arc does not belong to the partial shortest path tree of a given node, the construction phase from that node need not be repeated. In particular, during the pre-processing phase, we can build for each node v a list of all nodes that can be affected if the cost of one of the outgoing arcs from v changes. If these lists are kept in memory, then one knows exactly which parts of the overlay graphs are affected by the change and must be recomputed. The construction phase is repeated only when necessary; the authors of [36] claim that the update process takes on average up to a few dozen milliseconds for each arc cost change. After the update step the bidirectional query algorithm will correctly compute shortest paths. The total speed-up, with respect to a "pure" bidirectional Dijkstra's algorithm, is of about three orders of magnitude. Due to the inherent bidirectional nature of this algorithm, this approach works only for the cost-update variant of the SPP.

4.3 Guarantee Regions

If we assume that lower and upper bounding arc cost functions $l, u : A \to \mathbb{R}$ are available, a promising approach whose purpose is to speed up a bi-directional Dijkstra searches without the need to update data structures whenever the cost function changes, while at the same time requiring an approximation guarantee on the solution quality, can be found in [29, 30]. Let G_u be the graph $G = (V, A)$ weighted by the upper bounding function u. Given a source node s and a destination node t, the main idea is to compute the shortest path p^* between s and t on G_u, which gives an upper bound on the shortest path cost for that node pair over all possible cost function within the given bounds. By means of this upper bound $u(p^*)$ one can determine, in a pre-processing phase, all nodes that have to be explored when computing the shortest path from s to t in order to obtain a K-approximate solution, where K is a given constant. This can be done in polynomial time by identifying all nodes lying on a path p (from s to t) whose l-weighted cost is strictly lower than $u(p^*)/K$. More precisely, the *guarantee region* for the node

pair s, t and for the approximation constant K is:

$$\Gamma_{st}(K, p^*) = \left\{ v \in V | (v \in p^*) \vee (v \in p = (s, v_1, \ldots, v_n, t) : l(p) < \frac{1}{K} u(p^*) \right\}.$$

As long as guarantee regions depend on each node pair (s, t), this approach is largely impractical. It turns out, however, that the computations can be carried out for a set of departure nodes and a set of arrival nodes, while still mantaining the desired approximation constant. The only difference is that the guarantee region should be valid for any choice of s and t in the respective sets. Thus, the resulting guarantee region will have to be somehow "larger". In practice, if one covers the graph with connected node sets $V_1, \ldots, V_k$, which we will call *clusters*, and defines a central node c_i for each of them, computations can still be carried out in polynomial time if arc costs are non-negative: for $K > 1$, $i \neq j \leq k$ and path p^* which is the shortest path between c_i and c_j on G_u, we define the *clustered guarantee region* between for the cluster pair (i, j) as

$$\Gamma_{V_i V_j}(K, p^*) = \{ v \in V | v \in p^* \cup V_i \cup V_j \vee (v \in q = (c_i, v_1, \ldots, v_n, c_j) : l(q) < \frac{1}{K}(u(p^*) + \sigma_i + \tau_j)) \},$$

where σ_i and τ_i are the costs of the longest shortest path in G_u from v to s_i and respectively from t_i to v over all $v \in V_i$. They can be easily computed with Dijkstra's algorithm. Thus, one can compute in polynomial time a subset of the vertex set which still guarantees to find a K-approximated solution if the Dijkstra's search is constrained to only explore the graph induced by that subset; if these subsets are stored for each possible *cluster* pair, then the computation of the shortest path between any two nodes can be several times faster than a simple application of Dijkstra's algorithm.

Figure 3: Picture of an application developed with the algorithm in Sect. 4.3: a path constrained to be within a guarantee region (red path, travel time: 5 minutes, 20 seconds) compared to the optimal solution (green path, travel time: 5 minutes, 12 seconds).

The speed-ups yielded by this approach are not as spectacular as those obtained by [36]; however, guarantee regions do not require an update step whenever the cost function changes. In particular, since the assumption in [30] is simply that the cost on each arc is always between the lower and the upper bound for that arc, this approach is valid also for computations for both the time-dependent and the cost-update variant of the SPP, as long as the cost of an arc is always within the given bounds. Speed-up with respect to Dijkstra's algorithm is roughly of one order of magnitude, depending on the value of the approximation constant.

4.4 A^* for dynamic scenarios

Goal directed search, also called A^*, is a search technique which is similar to Dijkstra's algorithm, but which adds a potential function to the priority key of each node in the queue. This function applied on a node v should be an estimate of the distance to reach the target from v; A^* then works exactly as Dijkstra's algorithm, but the use of a potential function has the effect of giving priority to nodes that are (supposedly) closer to target node t. If the potential function π is such that $\pi(v) \leq d(v,t)\,\forall v \in V$, where $d(v,t)$ is the distance from v to t, then A^* always finds shortest paths. A^* is guaranteed to explore no more nodes than Dijkstra's algorithm: if $\pi(v)$ is a good approximation from below of the distance to target, A^* efficiently drives the search towards the destination node, i.e. the search space is not a circle centered at s, bu an ellipse directed towards t (see Fig. 4); if $\pi(v) = 0\,\forall v \in V$, A^* behaves exactly like Dijkstra's algorithm, i.e. it explores the same nodes. In [23] it is shown that A^* is equivalent to Dijkstra's algorithm on a graph with reduced costs, i.e. $c_\pi(u,v) = c(u,v) - \pi(u) + \pi(v)$. A^* was first applied in a time-dependent scenario with the FIFO property in [7]; a much more efficient version, presented in [9], makes use of landmarks to compute the potential function.

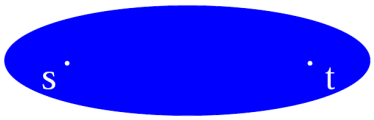

Figure 4: Schematic representation of A^* algorithm search space

Landmarks have first been proposed in [18]; they are a preprocessing technique which is based on the triangular inequality. The basic principle is as follows: suppose we have selected a set $L \subset V$ of landmarks, and we have precomputed distances $d(v,\ell), d(\ell,v)\,\forall v \in V, \ell \in L$; the following triangle inequalities hold: $d(u,v)+d(v,\ell) \geq d(u,\ell)$ and $d(\ell,u)+d(u,v) \geq d(\ell,v)$. Therefore $\pi_t(u) = \max_{\ell \in L}\{d(u,\ell)-d(t,\ell), d(\ell,t)-d(\ell,u)\}$ is a lower bound for the distance $d(u,t)$, and it can be used as a potential function which preserves optimal paths. Bidirectional search can be implemented, using some care in modifying the potential function so that it is consistent for the forward and backward search (see [19]). A^* with the potential function described above is called ALT. It is straightforward to note that, if arc costs can only increase with respect to their original value, the potential function associated with landmarks is still a valid lower bound, and in [9] this idea is applied to a real road network in order to analyse the algorithm's performances. Two main cases are considered: the cost-update scenario and the time-dependent scenario (in a FIFO network).

In the cost-update scenario two different approaches are considered; the first one is to update the preprocessing information, i.e. update distances to and from landmarks whenever an arc cost changes. Required time for this operation is greatly dependent on the number of updated arcs, their relative importance (urban edge, motorway, etc.) and their position with respect to landmarks, but it is a costly operation if several motorway edges are perturbed. The other possible approach is to compute paths without updating distances to and from landmarks; the algorithm's efficiency decreases with respect to the non-dynamic graph case, depending again on the number and type of perturbed edges. The authors of [9] report that, if 1000 motorway edges out of 42.6 millions edges are perturbed, roughly 95% of queries become slower, but ALT still yields an order of magnitude of speed increase with respect to bidirectional Dijkstra.

In the time-dependent scenario bidirectional search cannot be applied, and the algorithm loses efficiency; in this case, ALT yields a speed-up factor beween 2.5 and 5 with respect to unidirectional Dijkstra's algorithm, depending on departure time (ALT behaves better when there is no traffic, i.e. arc costs have very small increases with respect to their initial value).

5 Conclusion

This paper is a (partial) survey of some of the available methods for finding shortest paths in graphs whose topology or arc costs change. The main motivation for this study stems from an industrial application in the traffic information market. Recent algorithms are currently capable of finding point-to-point shortest (or within a given constant tolerance of a shortest) paths on time-dependent graphs or graphs with cost updates, in the order of a second of CPU time in networks comprising tens of millions of nodes and edges.

Acknowledgements

We are grateful to Maria Grazia Scutellà for providing useful suggestions and helpful discussions that increased the quality of this survey.

References

[1] R. Ahuja, T. Magnanti, and J. Orlin. *Network flows: theory, algorithms, and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[2] R. Ahuja, J. Orlin, S. Pallottino, and M. Scutellà. Dynamic shortest paths minimizing travel times and costs. *Networks*, 41(4):197–205, 2003.

[3] M. S. Bazaraa and R. W. Langley. A dual shortest path algorithm. *SIAM Journal on Applied Mathematics*, 26(3):496–501, 1974.

[4] M. Beckmann, C. McGuire, and C. Winsten. Studies in the economics of transportation. Technical Report RM-1488, RAND Corporation, 1955.

[5] L. Buriol, M. Resende, and M. Thorup. Speeding up dynamic shortest path algorithms. *INFORMS Journal on Computing*, accepted for publication.

[6] I. Chabini. Discrete dynamic shortest path problems in transportation applications: complexity and algorithms with optimal run time. *Transportation Research Records*, 1645:170–175, 1998.

[7] I. Chabini and L. Shan. Adaptations of the A^* algorithm for the computation of fastest paths in deterministic discrete-time dynamic networks. *IEEE Transactions on Intelligent Transportation Systems*, 3(1):60–74, 2002.

[8] K. Cooke and E. Halsey. The shortest route through a network with time-dependent internodal transit times. *Journal of Mathematical Analysis and Applications*, 14:493–498, 1966.

[9] D. Delling and D. Wagner. Landmark-based routing in dynamic graphs. In Demetrescu [10].

[10] C. Demetrescu, editor. *WEA 2007 — Workshop on Experimental Algorithms*, volume 4525 of *LNCS*, New York, 2007. Springer.

[11] R. B. Dial. Algorithm 360: shortest-path forest with topological ordering [h]. *Communications of the ACM*, 12(11):632–633, 1969.

[12] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[13] R. Dionne. Étude et extension d'un algorithme de Murchland. *INFOR*, 16:132–146, 1978.

[14] S. Dreyfus. An appraisal of some shortest-path algorithms. *Operations Research*, 17(3):395–412, 1969.

[15] L. R. Ford and D. R. Fulkerson. *Modern Heuristic Techniques for Combinatorial Problems*. Princeton University Press, Princeton, NJ, 1962.

[16] M. Fredman and R. Tarjan. Fibonacci heaps and their use in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

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

[18] A. Goldberg and C. Harrelson. Computing the shortest path: A^* meets graph theory. In *SODA 2005*. SIAM, 2005.

[19] A. Goldberg, H. Kaplan, and R. Werneck. Reach for A^*: Efficient point-to-point shortest path algorithms. In C. Demetrescu, R. Sedgewick, and R. Tamassia, editors, *ALENEX 2005*. SIAM, 2005.

[20] A. Halder. The method of competing links. *Transportation Science*, 4:36–51, 1970.

[21] A. Halder. Some new techniques in transportation planning. *Operational Research Quarterly*, 21:267–278, 1970.

[22] M. Holzer, F. Schulz, and D. Wagner. Engineering multi-level overlay graphs for shortest-path queries. In *SIAM*, volume 129 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 2006.

[23] T. Ikeda, M. Tsu, H. Imai, S. Nishimura, H. Shimoura, T. Hashimoto, K. Tenmoku, and K. Mitoh. A fast algorithm for finding better routes by ai search techniques. In *Proceedings for the IEEE Vehicle Navigation and Information Systems Conference*, pages 291–296, 2004.

[24] D. E. Kaufman and R. L. Smith. Fastest paths in time-dependent networks for intelligent vehicle-highway systems application. *Journal of Intelligent Transportation Systems*, 1(1):1–11, 1993.

[25] P. Loubal. A network evaluation procedure. *Highway Research Record*, 205:96–109, 1967.

[26] K. Mehlhorn and S. Näher. LEDA, a platform for combinatorial and geometric computing. *Communications of the ACM*, 38(1):96–102, 1995.

[27] J. D. Murchland. The effect of increasing or decreasing the length of a single arc on all shortest distances in a graph. *London Businness School, Transport Network Theory Unit*, 1967.

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

[29] G. Nannicini, P. Baptiste, D. Krob, and L. Liberti. Fast point-to-point shortest path queries on dynamic graphs with interval data. In J. Hurink, W. Kern, G. Post, and G. Still, editors, *Proceedings of the 6th Cologne-Twente Workshop on Graphs and Combinatorial Optimization*, Enschede, 2007. University of Twente.

[30] G. Nannicini, P. Baptiste, D. Krob, and L. Liberti. Fast paths in dynamics road networks. In A. Quillot and P. Mahey, editors, *Proceedings of ROADEF 08*, Clermont-Ferrand, 2008. Université Blaise Pascal.

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

[32] A. Orda and R. Rom. Shortest-path and minimum delay algorithms in networks with time-dependent edge-length. *Journal of the ACM*, 37(3):607–625, 1990.

[33] S. Pallottino and M. Scutellà. Dual algorithms for the shortest path tree problem. *Networks*, 29:125–133, 1997.

[34] S. Pallottino and M. Scutellà. A new algorithm for reoptimizing shortest paths when the arc costs change. *Operations Research Letters*, 31(2):149–160, 2003.

[35] P. Sanders and D. Schultes. Engineering highway hierarchies. In *ESA 2006*, volume 4168 of *Lecture Notes in Computer Science*, pages 804–816. Springer, 2006.

[36] P. Sanders and D. Schultes. Dynamic highway-node routing. In Demetrescu [10], pages 66–79.

[37] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, Berlin, 2003.

[38] R. Sedgewick and J. Vitter. Shortest paths in Euclidean graphs. *Algorithmica*, 1(1):31–48, 1986.