Pesquisa Operacional (2017) 37(3): 487-508

Printed version ISSN 0101-7438 / Online version ISSN 1678-5142
www.scielo.br/pope
doi: 10.1590/0101-7438.2017.037.03.0487

SHORTEST PATHS ON DYNAMIC GRAPHS: A SURVEY

Daniele Ferone, Paola Festa*, Antonio Napoletano and Tommaso Pastore

Received April 31, 2017 / Accepted October 23, 2017

ABSTRACT. This paper provides an overview of the state-of-the art and the current research trends concerning shortest paths problem on dynamic graphs. The discussion is divided in two main topics: reoptimization and time-dependent shortest paths. Reoptimization consists in the solution of a sequence of shortest path problems in which each instance slightly differs from the previous one. The reoptimization tackles this problem wisely using information stored in an optimal solution previously computed. On the other hand, shortest path problems on time-dependent graphs are characterized by a weight function which not only depends upon the arcs but changes in time according to a certain time horizon.

Keywords: Shortest Path, Network Optimization, Network Flows, Structure path constraints, Labelling methods, Auction Method.

1 INTRODUCTION

One of the most iconic algorithms in combinatorial optimization is due to Dijkstra [21], who in 1959 devised a label setting algorithm for the *shortest path problem (SPP)*. Since then, the SPP established itself as one of the most representative problems of operations research. Indeed, even today, many combinatorial optimization problems (COPs) require the solution of SPP as sub-task. Some examples of these problems are Maximum-Flow Minimum-Cost Problems [2], Vehicle Routing Problems [64], and several other variations of the SPP, spanning from problems on time-dependent graphs [49, 50] to general constrained SPP [25, 57].

Reoptimizing shortest paths on dynamic graphs consists in solving a sequence of shortest path problems, where each problem differs only slightly from the previous one, because the origin node has been changed, some arcs have been removed from the graph, or the cost of a subset of arcs has been modified. Each problem could be simply solved from scratch, independently from the previous one, by using either a *label-correcting* or a *label-setting* shortest path algorithm. Nevertheless, a clever way to approach it is to design ad hoc algorithms that efficiently use information resulting from previous computations.

*Corresponding author.
Department of Mathematics and Applications "R. Caccioppoli", University of Napoli Federico II, Italy.
E-mails: daniele.ferone@unina.it; paola.festa@unina.it; antonio.napoletano2@unina.it; tommaso.pastore@unina.it

Another type of dynamic graph is the *time-dependent graph*, introduced in Cooke & Halsey [13], where the characteristic cost function w is defined for each edge (i, j), as $w_{ij}(t)$, where t is a time variable in a time domain T. The value $w_{ij}(t)$ specifies how much time it takes to travel from node i to node j, if departing from i at the time t. Most of the solution strategies for problems on dynamic graphs (often called networks) have been adopted to solve instances of shortest path problems on time-dependent graphs.

In the wholeness of its variations, shortest path problems on dynamic graphs appear in a wide variety of contexts and application settings, including logistics, telecommunications, transportation, urban traffic, and transit planning.

The remainder of the paper is organized as follows. In Section 2 the shortest path problems are formally introduced and the classical approaches to solve them are presented. Sections 3 and 4 analyze the scientific literature of reoptimization and time-dependent graphs, respectively. Conclusions and final remarks are given in Section 5.

2 MATHEMATICAL FORMULATIONS AND CLASSICAL APPROACHES

2.1 Mathematical Formulation of the Shortest Paths Problem

In this section, the mathematical formulation for all types of shortest path problems is given. Indeed, SPPs can be classified in three different sub-categories: shortest path point-to-point (P2P), shortest path tree (SPT), and all pairs shortest paths (APSP).

All these problems rely on the following notation. Let $G = (V, A)$ be a directed weighted graph, where:

- $V = \{1, 2, \ldots, n\}$ is a set of nodes;
- $A \subseteq \{(i, j) \in V \times V \| i, j \in V \wedge i \neq j\}$ is a set of m arcs;
- $w : A \to \mathbb{R}^+$ is a function that assigns a non-negative cost w_{ij} to each arc $(i, j) \in A$.

Furthermore, for each $i = 1, \ldots, n$, let

- $FS(i) = \{j \in V \| (i, j) \in A\}$ be the *forward star* of node i;
- $BS(i) = \{j \in V \| (j, i) \in A\}$ be the *backward star* of node i.

2.2 Shortest path point-to-point problem (P2P)

The problem consists in finding a shortest path $P^* = (v_1, v_2, \ldots, v_h)$ from a source node $v_1 = s$ to a destination node $v_h = t$, with $s, t \in V$. Introducing m Boolean decision variables, x_{ij}, $\forall\, (i, j) \in A$, such that:

$$x_{ij} = \begin{cases} 1, & \text{if } (i, j) \text{ belongs to } P^*, \\ 0, & \text{otherwise,} \end{cases}$$

the mathematical formulation of the (P2P) problem is the following:

$$\text{(P2P)} \qquad z = \min \sum_{(i,j)\in A} w_{ij}x_{ij}$$

subject to:

$$\text{(P2P-1)} \qquad \sum_{j\in BS(i)} x_{ji} - \sum_{j\in FS(i)} x_{ij} = b_i, \qquad \forall i \in V$$

$$\text{(P2P-2)} \qquad x_{ij} \in \{0, 1\}, \qquad \forall (i, j) \in A,$$

with $b_i = -1$ for $i = s$, $b_i = 1$ for $i = t$, and $b_i = 0$ otherwise.

2.3 Shortest path tree problem (SPT)

Given a node $r \in V$, named *root*, the goal of the problem is to find a shortest path from r to all other nodes $i \in V$, $i \neq r$. Defining m Boolean decision variables x_{ij}, $\forall\ (i, j) \in A$, the mathematical formulation of the (SPT) problem is the following:

$$\text{(SPT)} \qquad z = \min \sum_{(i,j)\in A} w_{ij}x_{ij}$$

subject to:

$$\text{(SPT-1)} \qquad \sum_{j\in BS(i)} x_{ji} - \sum_{j\in FS(i)} x_{ij} = b_i, \quad \forall\ i \in V;$$

$$\text{(SPT-2)} \qquad x_{ij} \geq 0, \qquad \forall\ (i, j) \in A.$$

where $b_i = -n + 1$ for $i = r$, and, $b_i = 1$ for $i \neq r$.

2.4 All pairs shortest path problem (APSP)

The aim of this problem is to find all the point-to-point shortest paths between each pair of nodes $i, j \in V$, $i \neq j$. Its mathematical formulation can be easily obtained starting from the mathematical formulation of the SPT. With few modifications, the following model can be obtained:

$$\text{(APSP)} \qquad z = \min \sum_{k=1}^{n} \sum_{(i,j)\in A} w_{ij}x_{ij}^k$$

subject to:

$$\text{(APSP-1)} \qquad \sum_{j\in BS(i)} x_{ji}^k - \sum_{j\in FS(i)} x_{ij}^k = b_i^k, \quad \forall i \in V, \forall k \in V;$$

$$\text{(APSP-2)} \qquad x_{ij}^k \geq 0, \qquad \forall (i, j) \in A,\ \forall k \in V,$$

where $b_i^k = -n + 1$ for $i = k$, and, $b_i^k = 1$ for $i \neq k$.

2.5 Labeling Methods

A first successful attempt to solve the SPP was originally proposed by Ford Jr [28] and Ford Jr & Fulkerson [29], although the most famous algorithm to solve P2P and SPT is a labeling method proposed by Dijkstra [21], whose pseudo-code is reported in Figure 1. Let $s \in V$ be the source node in a graph G, to find a shortest path from s to each other $v \in V$, $i \neq s$, Dijkstra's algorithm maintains and updates for each node $v \in V$:

- `dist[v]`, the distance of v from the source node s;
- `pred[v]`, the predecessor of the node v in the incumbent path from s to v.

In addition, the following sets are used: S and Q, that are the sets of visited and unvisited nodes, respectively.

The algorithm starts with an initialization phase (lines 2-5), where the vectors `pred`, `dist` and the sets S and Q are initialized. Afterwards, while set Q is nonempty, the algorithm selects an unvisited node v, relaxes all the edges in $FS(v)$, and insert v in S. The relaxation operation is described in lines 10-12.

If the weight function w is non-negative, the algorithm always terminates with the correct shortest path distances stored in `dist[]`, and shortest path tree in `pred[]`. The following theorem holds:

Theorem 1. *If the cost function w is non-negative, then Dijkstra's algorithm visits nodes in non-decreasing order of their distances from the source, and visits each node at most once.*

Algorithm 1 – Dijkstra's algorithm

```
1:  procedure DIJKSTRA(G = (V, A), s)
2:      dist[s] := 0; pred[s] := NULL; S := ∅;
3:      for all v ∈ V \ {s} do
4:          dist[v] := +∞; pred[v] := NULL;
5:      Q ← V
6:      while Q ≠ ∅ do
7:          u := extract_min(Q);
8:          S := S ∪ {u};
9:          for all v ∈ FS(u) do
10:             if dist[v] > dist[u] + w_uv then
11:                 dist[v] := dist[u] + w_uv;
12:                 pred[v] := u;
```

In the case of P2P problem, *bidirectional versions* of Dijkstra's algorithm were proposed in [14, 22, 53]. The bidirectional framework is based on the consideration that if s and t are the source and the destination node, respectively, then it is possible to run the algorithm into two opposite

directions: the first from node s to t, called the *forward search*, and the latter from t to s, the *backward search*. The backward search operates on the *reverse graph*, obtained from G reversing the direction of each arc in A. The algorithm terminates when the two paths meet.

Hart et al. [36] proposed another labeling method for SPP: an informed search algorithm called A^*. It refines the Dijkstra's method, using a best first paradigm, firstly exploring sub-paths which appear to lead most quickly to the solution.

The estimation of the most promising sub-paths is carried out by means of a potential function π_t. Let $\pi_t : V \to \mathbb{R}^+$ be a non-negative function, giving an estimate on the distance from each node v to t. The A^* search uses a new set L, which contains all the nodes that are relaxed at least once and whose label is not permanent. It selects a node $v \in L$ with the smallest value of $k(v) = d(s, v) + \pi_t(v)$, where $d(s, v)$ is the shortest distance from s to v.

A potential function π_t is defined to be *feasible* if $d_\pi(u, v) = d(u, v) - \pi_t(u) + \pi_t(v)$ is non-negative for each arc $(u, v) \in A$. Goldberg & Harrelson [34] showed that A^* search with a feasible non-negative potential function visits no more nodes than Dijkstra's algorithm.

In order to define π_t, in the Euclidean domain, it is possible to use the canonical Euclidean distance to establish a lower bound. Such computation is carried out by means of a method based on the concept of landmarks selection [34] and the triangle inequality.

3 REOPTIMIZATION

Nowadays, in the era of big data and huge networks, there is a rising need of well performing algorithms, able to handle the massive amount of available information. One of the suitable approaches to tackle this complexity is to reuse information already computed, in order to reduce the computational time needed to obtain an optimal solution. In the context of SPP, previous information can be reused while tackling a problem which differs only slightly from another SPP previously solved. This occurrence can happen with one of the following changes in the network:

- the origin node has been changed;
- some nodes have been added or removed;
- some arcs have been added or removed;
- some arcs weight have been increased or decreased.

This problem can be addressed as a shortest path reoptimization problem [26], which consists in solving a sequence of shortest path problems, where the k^{th} problem marginally differs from the $(k-1)^{th}$ one.

In the first case, we say that there was a root change from the $(k-1)^{th}$ problem to the k^{th} problem, in the remaining cases we say that the graph is dynamic. Moreover, for what concerns problems on dynamic graphs, they can be classified according to the type of changes that can

occur on the network. A dynamic graph is said to be fully dynamic if both insertion and deletion of either edges or nodes are allowed.

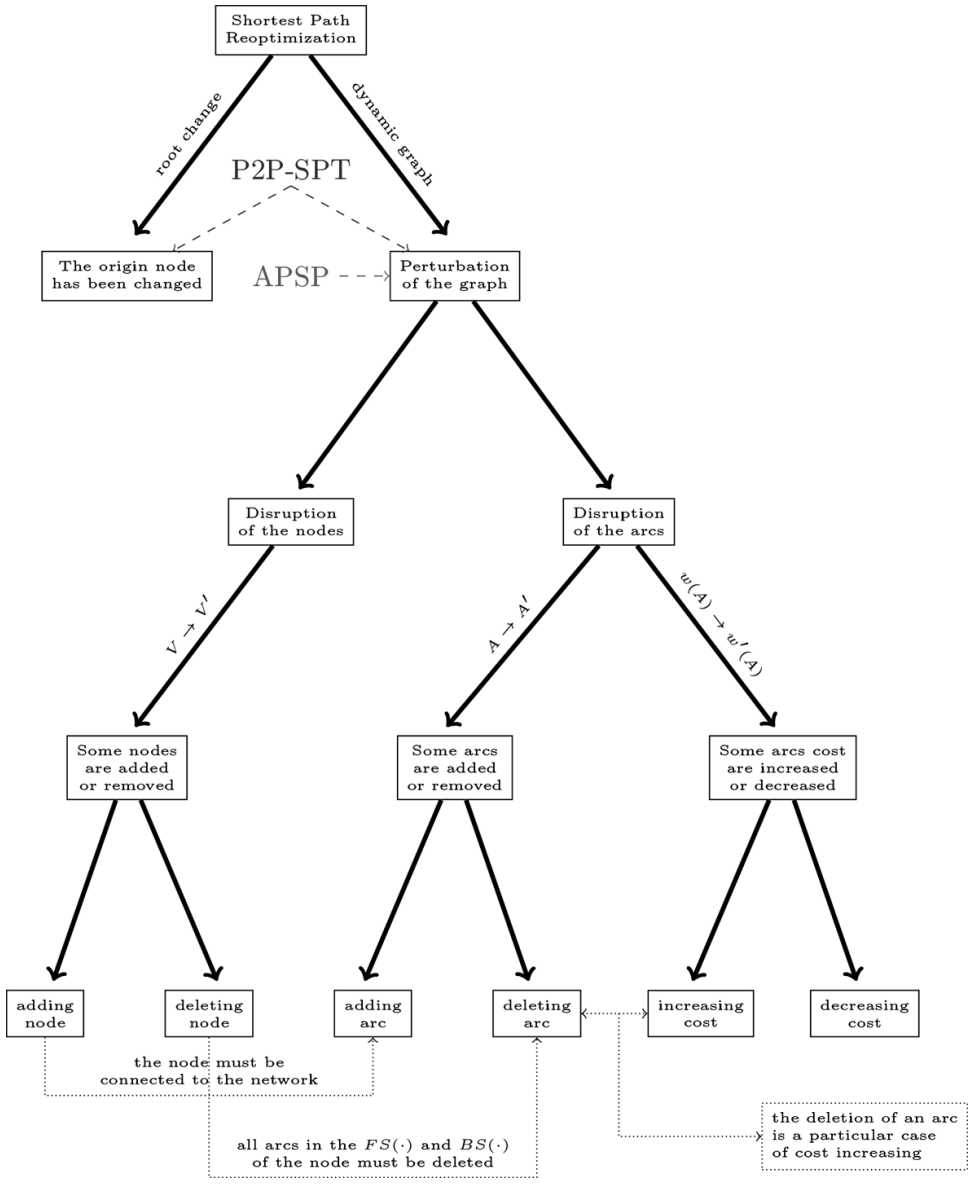

Figure 1 – Reoptimization problems hierarchy.

In Figure 1, we propose a diagram of the interplays among all possible cases of shortest path reoptimization. The root change reoptimization problem does not admit further sub-cases. For what concerns dynamic graphs, the problems can be classified in two different branches, depending on whether the changes involve nodes or arcs. The only possible changes involving a

node are addition or removal. On the other hand, the arcs can be either added/removed or their cost can be increased/decreased. It is worthy to note that the case of changes involving nodes implies also arcs insertion or deletion.

Without loss of generality, we can consider the input graph $G = (V, A)$ as a complete graph. Indeed, if G is not complete, for each pair of nodes i and j, such that $(i, j) \notin A$, it can be always added a dummy arc from i to j with $w_{ij} = +\infty$.

This operation allows the following considerations:

- arc removal in the graph can be seen as a special case of arc cost increasing. If an arc (i, j) is deleted, then the cost w_{ij} increases to $+\infty$.
- Arc insertion can be seen as a special case of arc cost decreasing. If an arc (i, j), must be inserted, then the cost w_{ij} is decreased from $+\infty$ to the new cost k.

3.1 Root change

The purpose of this paragraph is to show how, in the case of root change for the SPT problem, it is possible to obtain a well performing algorithm making wise use of the information stored in a SPT previously computed. Such result relies on some remarkable theoretical properties proven by Gallo [32], starting from the assumption that a single root shortest path tree problem has been solved.

Let $G = (V, A)$ be a complete directed graph, and let T_r be a shortest path tree rooted at node r, i.e., a tree that contains a shortest path from r to each node $v \in V,\ v \neq r$. Let s be a node of V, $s \neq r$, and T_s be a SPT rooted at node s.

The following propositions show how the knowledge of T_r provides useful informations on T_s. Let $T_r(h)$ denote the subtree of T_r which contains node h together with all its descendants, then

Proposition 1. *$T_r(s) \subseteq T_s$ and $d(s, j) = d(r, j) - d(r, s)$, for any $j \in T_r(s)$.*

Henceforth, the paths contained in the subtree of T_r rooted in s still remain optimal shortest paths from s to its descendants. This result shows how a wise handling of the old solution is likely to be the most efficient strategy, since – especially when the new root s is close to r – a consistent part of the previously optimal tree T_r will remain optimal.

As formerly stated, beyond the theoretical insight given by Proposition 1, the information provided by T_r can be employed in order to reduce the computational time needed to solve the new SPT problem, improving the classical Dial's implementation of Dijkstra's Algorithm (DDA) [19, 20]. In Dial's implementation, in fact, one of the most time consuming tasks consists in the identification of the minimum temporary node cost, due to the high number of comparisons to be performed. This number of comparisons strongly depends on the maximum weight among the arcs, $w_{\max} = \max_{(i,j)\in A} w_{ij}$. Propositions 2 and 3 show how to reduce $w_{\max}$ without changing the sets of feasible and optimal solutions.

Let $\pi_1, \pi_2, \ldots, \pi_n$ be integer numbers such that

$$l_{ij} = w_{ij} + \pi_i - \pi_j \geq 0 \quad \forall (i, j) \in A. \tag{1}$$

Proposition 2. *The problem of finding the SPT from s with arc lengths w_{ij} is equivalent to the problem of finding the SPT with arc lengths l_{ij} given by* (1).

Proof. Let P be a generic path from a node k to a node h of $G = (V, A)$, and let $W(P)$ and $L(P)$ be the lengths of P pertaining to the length functions w and l, respectively. Then, it holds that

$$L(P) = \sum_{(i,j)\in P} l_{ij} = \sum_{(i,j)\in P} (w_{ij} + \pi_i - \pi_j) = W(P) + (\pi_k - \pi_h). \tag{2}$$

Henceforth, the lengths $W(P)$ and $L(P)$ only differ by a constant depending only on the source and the destination nodes of the path.

Ultimately, a shortest path tree T_r with respect to the arc length w_{ij} will remain optimal with arc length l_{ij}. □

A straightforward consequence of Proposition 2 is that lengths w_{ij} can be replaced by lengths l_{ij}, and once the new shortest length $d'(s, h)$, $h \in V$, is found, the value $d(s, h)$ can be obtained as follows:

$$d(s, h) = d'(s, h) - \pi_s + \pi_h. \tag{3}$$

The results outlined above suggest that an appropriate choice of the integers π_i might decrease the distance from the source to the farthest node, and thus also the computation time required by DDA. Indeed, when selecting $\pi_j = d(s, j)$ one has $d'(s, j) = 0$, for all $j \in V$, thus obtaining the validity of the following proposition:

Proposition 3. *Let be $\pi_j = d(r, j)$, for all $j \in T_r$, and let be the arc lengths defined as in* (1). *Let h be one of the farthest node from the origin s, then:*

$$d'(s, h) = d(r, s) + d(s, r). \tag{4}$$

From Proposition 3 it follows that if nodes r and s are close enough, the computational effort required by DDA can be strongly reduced by a cost modification of type (1), with the vector π given by $\pi_j = d(r, j)$, for all $j \in T_r$.

As reported in Gallo [32], in terms of Linear Programming, such new costs correspond to the reduced costs relative to a dual feasible, but primal unfeasible, basis. This interpretation of the vector $(\pi_1, \pi_2, \ldots, \pi_n)$ as a dual feasible solution for the Shortest Path Problem is due to Bazaraa & Langley [6].

Deriving a cost reduction in a similar fashion, in 1982 Gallo & Pallottino [33] devised an algorithm which outperforms both the one proposed in [32] and classical from scratch optimization

techniques. This algorithm refines the classical label setting paradigm by partitioning the nodes of the graph in three distinct sets: NT, NP, and NQ. As in a classical label setting algorithm, NT and NP are the set of nodes whose labels are temporary and permanent, respectively. While, the nodes in NQ are those nodes such that $d(s, v)$=$d(s, p(v))$. This property ensures that such nodes can be inserted straightaway in NP without further comparisons, thus speeding up the execution of the algorithm. In [33], it has been noted how in any reoptimization problem instances a large share of nodes of V is likely to be found in NQ.

The observations made by Gallo are the starting point of the work of Florian et al. [27]: the shortest path tree rooted at node r is an optimal solution to a corresponding linear program, but when a successive new source s is considered, the previous tree is a dual feasible and primal infeasible solution for the new problem. The approach proposed by Florian et al. [27] consists in the adaptation of the dual simplex method to compute the shortest paths from the new root s. It is shown in Florian et al. [27] that the proposed algorithm runs at most in $O(n^2)$.

In order to evaluate the performances of their method, the authors tested their code on graphs representing the regional roads of the cities of Vancouver and Winnipeg, as in Gallo [32].

The experimental evaluations proposed in these works show how Gallo [32] is slightly better performing than Dijkstra's from scratch technique, meanwhile Gallo & Pallottino [33] further improves the previous results. Although, among all, the algorithm proposed by Florian et al. [27] appears to outperform the other competitive methods.

In Ferone et al. [26], a novel dual approach based on an auction framework [8] is presented. Given a shortest paths tree T_r, and a new source s, all the arcs in $BS(s)$ are deleted and the nodes of the tree $T_r(s)$ are included in a priority queue Q, containing the nodes ordered according to the reduced costs in decreasing order. Q is analyzed with a strongly polynomial auction algorithm [10]. During the extraction of the nodes from Q, when the algorithm reaches a node $j \in T_r$, it moves the sub-tree $T_r(j)$ from T_r to T_s, and its nodes are included in Q. The algorithm terminates when Q becomes empty.

3.2 Arc Cost Change

Given a shortest paths tree, T_r, the problem of arc cost change consists in recalculating tree T_r when a new weight is assigned to one or more (batch updates) arcs. As extensively discussed in Gallo [32], the results of Theorem 4 can be used to derive algorithms that reoptimize the solution of a SPP after a change of the cost of a single arc.

Theorem 4 (Gallo [32]). *Let T_r be a shortest paths tree, w'_{uv} be the new cost of the arc $(u, v) \in A$, and $S(u) = \{v \in V \| d(r, v) \leq d(r, u)\}$. Denoting with T'_r the new shortest paths tree, the following properties hold:*

(i) if $w'_{uv} < w_{uv}$ and $(u, v) \in T_r$, then

$$T_r(v) \subseteq T'_r(v),$$
$$j \in T_r(v) \Rightarrow d'(r, j) = d(r, j) - \left(w_{uv} - w'_{uv}\right),$$
$$j \in S(u) \Rightarrow d'(r, j) = d(r, j);$$

(ii) if $w'_{uv} < w_{uv}$ *and* $(u, v) \notin T_r$, *then*

$$T_r(v) \subseteq T'_r(v),$$
$$j \in S(u) \Rightarrow d'(r, j) = d(r, j);$$

(iii) if $w'_{uv} > w_{uv}$ *and* $(u, v) \in T_r$, *then*

$$j \notin S(u) \Rightarrow d'(r, j) = d(r, j);$$

(iv) if $w'_{uv} > w_{uv}$ *and* $(u, v) \notin T_r$, *then*

$$T_r(v) = T'_r(v),$$
$$d'(r, j) = d(r, j) \quad \forall j \in V.$$

In other words, a decrease in the cost of the arc (u, v) (cases (i) and (ii)) implies that the sub-tree $T_r(v)$ remains part of the optimal solution and the optimal distances of its nodes accordingly decreased where necessary. Case (iii) tackles the cost increase when (u, v) is part of the solution, stating that the optimal distances are preserved for each $j \notin S(u)$. Finally, in case (iv) the entire solution remains optimal.

Pallottino & Scutellà [55] assume that a shortest paths tree T_r has been determined and address the problem of computing the shortest paths tree when new costs are given to a subset K of the arcs of G, either lower or higher than the old ones. The reoptimization framework used is based on the determination of 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 such decomposition. Pallottino & Scutellà [55] also compute the time complexity of the algorithm as a function of both the input size and the overall cost perturbations. Nevertheless, in the same work it has not been proposed an implementation of any kind for this framework, henceforth numerical results are not available.

One of the most important works in shortest path reoptimization in case of arc cost changes is Ramalingam & Reps [58], that proposes the so called *DynamicSWSF-FP* algorithm. At any time, it is defined

$$rhs(v) = \min_{x \in BS(v)} \left\{\hat{d}_x + w'_{xv}\right\},$$

where $\hat{d}_x$ is the distance of x from root r in the current intermediate tree $\hat{T}$. In the proposed algorithm, each node is processed differently according to whether $rhs(v)$ is greater than (*underconsistent node*), equal to (*consistent node*), or less than (*overconsistent node*) $\hat{d}_v$. The algorithm processes inconsistent nodes, opportunely stored in a heap H, in a nondecreasing order of

key values, where $key(v) = \min\{\hat{d}_v, rhs(v)\}$. Let q be the inconsistent node currently selected. If it is underconsistent, then the algorithm sets $\hat{d}_q = +\infty$; otherwise, it sets $\hat{d}_q = key(q)$, its forward star is relaxed, and q is removed from the heap. The algorithm terminates when the set of inconsistent nodes is empty.

Buriol et al. [9] empirically showed how the algorithm devised by Ramalingam and Reps outperforms an optimization from scratch by means of Dijkstra's algorithm. Furthermore, they proposed a new technique to improve computational times of the approach described above. This technique is called *Reduced-Heap*, because it reduces the number of nodes of Q (inconsistent nodes) to be inserted in the heap H.

Buriol et al. [9] use the reverse graph representation, so a shortest path tree T_r is obtained as the tree of shortest paths from every node to the single target r. Let $a = (u, v)$ be the arc whose cost is increased by an amount Δ. The effective increase ∇ of the node distance d_u is then computed, and the distances of nodes $x \in Q$ are updated to $d_x + \nabla$. Afterwards, the only nodes of Q to be inserted in the heap H are the nodes $\{x \in Q | \exists y \in FS(x) \colon d_x > d_y + w_{xy}\}$, which is the set of nodes x for which there exists a path shorter than $d_x + \nabla$. The method can be applied to the case of a single arc cost decrease in a very similar and straightforward way. This permits, in many cases, to reduce the computational time of the re-optimization algorithm.

While Buriol et al. [9]'s algorithm is able to manage the cost update of a single arc, Chan & Yang [12] proposed a comparison between several algorithms devised to handle multiple arc cost updates. The first algorithm is a dynamic version of the classic Dijkstra's algorithm. Given the original shortest paths tree T_r, after the arc cost changes occur, all the updated arcs are removed from T_r, and the set $\bar{N}$, the set of nodes that are not reachable anymore from the root r, is computed. The nodes that have an arc connecting to a node in $V \setminus \bar{N}$ are added in an heap H and a Dijkstra-like algorithm is used to update labels.

The main accomplishment made by Chan & Yang [12] is the *MFP* algorithm. It is an extension of algorithm by Ramalingam & Reps [58] in order to handle optimization in fully dynamic networks. The algorithm has been improved to avoid unnecessary rhs value recomputations and also simplifying computation when it is possible. When an overconsistent node v is extracted, for each child q, MFP recomputes $\hat{rhs}(q) = \min\{\hat{rhs}(q), \hat{d}_v + w'_{vq}\}$. When an underconsistent node u is extracted, MFP reevaluates the rhs values only on children where $rhs(v) = \hat{d}_u + w'_{uv}$.

Another approach to handle batch updates is proposed by D'Andrea et al. [23]. They use the concept of *accounting function* to estimate and potentially reduce the computational complexity of the reoptimization algorithm. An accounting function f for G is a function that, for each arc $(x, y) \in A$, determines either node x or node y as the *owner* of the arc; f is k-bounded, if k is the maximum over all nodes x of the cardinality of the set of arcs owned by x.

Moreover, if β is a batch of arc update operations, $Q(\beta)$ is defined as the set of affected nodes caused by the execution of all the operations in β, simultaneously. If the input graph admits a k-bounded accounting function, then the algorithms proposed by D'Andrea et al. have the following theoretical properties:

1. the algorithm is able to process only a batch of arc deletions and weight increases. Its worst case time complexity is $O((|\beta| + |Q(\beta)| \cdot k) \cdot \log n)$;

2. the function that solves the dynamic shortest path of only arc insertion and arc cost decrease has a time complexity of $O(|\beta| + |Q(\beta)| \cdot \max\{k, k^*\} \cdot \log n)$ in the worst case, where k^* is the minimum integer such that a k^*-bounded accounting function exists in the graph after β;

3. finally, the combined method to solve a mixed sequence $B = (\beta_1, \ldots, \beta_h)$ of incremental and decremental batches is shown to require $O((|B| + |\hat{Q}| \cdot k) \cdot \log n)$ overall time, where $|B| = \sum_{\beta_i \in B} |\beta_i|$ and $\hat{Q} = \sum_{\beta_i \in B} |Q(\beta_i)|$.

Narváez et al. [51] establish a framework to manage a variety of well known strategies for the dynamic SPT. Unlike previous work bas:w ed on Dijkstra's algorithm only, the proposed framework also yields the dynamic version of Bellman-Ford [7, 28] and D'Esopo-Pape [56] ones. They consider both arc cost increasing and arc cost decreasing reoptimization case, but the update affects only one arc. In [52], the same authors provide an extension which allows to manage the arc cost change in the case of multiple updates. Although in practice it would seem a very efficient approach, Chan & Yang [12] proved that it is not correct in some case of multiple arc cost increases.

Thomas & White [63] proposed a mathematical model for the problem named "dynamic shortest path with anticipation". It frequently occurs on road networks, where some arcs are congested due to extraordinary events (i.e., car accidents), and a driver can choose to change its route to destination. Nevertheless, if the time to solve the congestion is known, it can be preferable to remain on the same road.

Nannicini et al. [48] proposed a Polynomial-Time Approximation Scheme (PTAS) heuristic for the P2P on dynamic road networks. The algorithm is based on Dijkstra-type searches performed on clusters nodes. The clusters are defined in order to give a bound on the solution performance and to speed up the search to be practically useful within the given time constraints.

Tretyakov et al. [65] try to tackle the SPT reoptimization in the fully dynamic context proposing an approximation method based on landmarks estimation. They justified this approach arguing that the classical exact method are unable to efficiently address instances based on very large graph representing, for example, social networks with hundreds of millions of users and billions of connections. Two improvements to existing landmark-based estimation methods for undirected graph are described. The first improvement is based on the maintaining a shortest paths tree to store the paths between each landmark and every node in the graph, while the second adopts a greedy approach to select the landmarks which provide the best coverage of all shortest paths in a random sample of vertex pairs.

Hong et al. [39] proposed a disk-based approach to discover shortest paths over large dynamic graphs. The disk-based approach divides a large graph into memory-sized subgraphs and loads them into memory to partial processing. They used a relational database (RDB) to maintain an

index table that stores the pre-computed shortest segments with distances shorter than a given threshold. This table is used to compute the complete shortest paths using several relational operations. When an arc cost change occurs, only the shortest segments involved in the index table are recomputed.

Apart for what described for the SPT, the reoptimization has been considered also for the *All Pairs Shortest Paths* (APSP), whose goal is to find the shortest paths between all pairs of nodes in a graph $G = (V, A)$. There are several algorithms that solve the static version of this problem. The algorithm by Fredman & Tarjan (1987) using Fibonacci heaps has a running time of $O(mn + n^2 log n)$, but the best asymptotic bound has been obtained by Takaoka [62], whose algorithm solves APSP in $O(n^3\sqrt{\log\log n/\log n})$.

In the fully dynamic APSP, we wish to maintain a directed graph $G = (V, A)$ with real-valued arc costs under an intermixed sequence of **Update**(x, y, w') operations, which modify the cost of arc (x, y) to the real value w'.

The first dynamic algorithm for handling all pairs shortest paths in a graph was presented by King [45], with the constraint that all costs are positive integers and bounded by a value b. Initially, they proved that any shortest paths tree, where the length of each path is at most δ, can be maintained in time $O(m\delta)$, where m is the cardinality of the arcs set, during a sequence of arbitrary number of arc deletions. While, still considering arc deletions only, all pairs shortest paths can be maintained with a forest of n shortest paths trees of depth nb. Finally, they showed how to maintain all pairs shortest paths both in case of arc deletion and insertion, where the insertion/deletion is allowed only for arcs with maximum weight δ.

In Demetrescu & Italiano [18], the authors use the equivalence between APSP and matrix multiplication on the $\{\min, +\}$ semiring [1]. Assuming that each arc can assume at most S different real values (each arc can have a different set of real values), they use several data structures to handle both decreasing costs of arcs incident on a single node i, or increasing costs of any arc of the graph. Any **Update**(x, y, w') operation can be supported in $O(n^{2.5}\sqrt{S\log^3 n})$ amortized time. If only decreasing operations are performed in an operation sequence of length $\Theta(n^2)$, the amortized cost per update is $O(S \cdot n \log^3 n)$.

4 SHORTEST PATHS ON TIME-DEPENDENT NETWORKS

While reoptimization tries to handle sudden and unpredictable changes in the graph, in shortest path problems on time-dependent networks (TDSPP), arc costs are known beforehand and given as a function of time. The growing interest in time-dependent SPP shown in literature can be addressed to the key role that these problems play in logistics, due to their high suitability to model congestion and time delay in transit planning.

One of the early works on the subject is due to Cooke & Halsey [13], who proposed an iterative scheme as extension of Bellman's principle of optimality [7] for the TDSPP, presenting an evaluation of the theoretical computational complexity, although not actually reporting any

computational results. Few years later, Dreyfus [22] extended Dijkstra's algorithm to the case of time-dependent scenarios, but the use of this technique is restricted to graph $G = (V, A)$ showing the FIFO property. The FIFO property says that, given $(i, j) \in A$, if two paths X and Y use the arc (i, j) and X leaves i at the time τ' and Y leaves i at the time τ'', with $\tau' < \tau''$, then Y cannot reach j before X. Orda & Rom [54] proved that the TDSPP is NP-hard in non-FIFO networks. On the contrary, the problem appears to be polynomially solvable when G shows the FIFO property, as proved in Kaufman & Smith [44].

Ziliaskopoulos & Mahmassani [67] implemented an algorithm that can be assimilated to a static label correcting technique, once again based on the Bellman's principle of optimality. In their algorithm, the path is calculated starting from the destination node operating backward, and uses the double-ended queue list technique [2] to reduce the number of label corrections. Their approach can handle graph as large street in urban transportation networks, where in the peak period the time costs are discretized into small intervals. Moreover, this technique can be used on graphs which do not present the FIFO property, being thus suitable to the solution of problems arising in contexts different from transportation planning, such as equipment replacement policy, capacity planning and communication networks.

An algorithm with excellent performances in case of time-dependent scenarios is the SHARC-routing (Shortcut+ArcFlags) algorithm: a fast and robust approach for unidirectional routing in large networks. This method was proposed by Bauer & Delling [4] and can be considered an adaptation of the work of Sanders & Schultes [59] to the techniques described in Lauther [46], Möhring et al. [47], and Hilger et al. [37]. This algorithm is made up of two phases: a prepocessing phase and an arc-flags phase. In the former, the technique iteratively constructs a contraction-based hierarchy, and subsequently, in the arc-flags phase, it automatically sets arc-flags for arcs removed during contraction.

In Nannicini et al. [49], a novel approach based on A^* with landmarks (ALT) is described for the computation of P2P on time-dependent road networks. It is defined also an interval of time instants T, a departure time $\tau_0 \in T$, in order to redefine the cost function as $w^T : A \times T \to \mathbb{R}^+$, and with the goal to find a minimum *time-dependent cost* $\gamma_{\tau_0}(p)$, for a path $p = (s = v_1, \ldots, v_k = t)$, defined recursively:

$$\gamma_{\tau_0}(v_1, v_2) = w^T_{v_1 v_2}(\tau_0); \tag{5}$$

$$\gamma_{\tau_0}(v_1, \ldots, v_i) = w^T_{v_{i-1} v_i}(\tau_0 + \gamma_{\tau_0}(v_1, \ldots, v_{i-1})); \tag{6}$$

for $i = 2, \ldots, k$. Where λ is a lower bound function, such that $\forall (i, j) \in A,$ and $\tau \in T$, $\lambda(i, j) \leq w_{i,j}(\tau)$, and $\lambda(p) = \sum_{i=1}^{k-1} \lambda(v_i, v_{i+1})$. This algorithm is based on the A^* search, in particular on the time-dependent A^* from the source using a set of nodes defined by a time-independent A^* from the target. The forward search is performed on G, while the backward one is performed on G_λ.

Delling & Wagner [16] adapted the algorithm by Goldberg & Harrelson [34] to handle time-dependent graphs. They show that the landmarks found in the pre-processing phase still produce

correct results if the costs of the arcs do not drop below their initial value. Therefore no updates on the pre-processing are required in presence of small changes.

Inherently to the A^* search on time-dependent road networks, some relevant strategies were developed in Chabini & Lan [11], Goldberg & Harrelson [34], Kanoulas et al. [43], Huang et al. [41] and [49]. The basic idea of this technique was a starting point for several subsequent works, see Batz et al. [3] and Delling & Nannicini [15] for example. Moreover, Nannicini et al. [50] present an further elaboration, which can be considered the first attempt to tackle the TDSPP in a bidirectional fashion.

It is well known that in the solution of the SPP the Dijkstra's framework proceeds visiting nodes in circular areas of increasing size, and for this reason should be under-performing according to the computational time in applications with huge and dense networks. In this regards, in the last decades more speed-ups techniques of this framework have been presented. Accordingly to Holzer et al. [38], these can be classified in four different categories.

Goal directed search techniques: where the cost of the arc, linked to the nodes whose probability of belonging to the shortest path is lower, are increased. The aforementioned A^* star search belongs to this category.

Bidirectional search: a second search which starts simultaneously from the destination node is carried over. Approaches of this type, previously described, are those proposed in Ahuja et al. [2] and Nannicini et al. [50].

Bounding boxes: it is adopted a criterion of selection of nodes set. If a group of nodes can be all inserted in a shortest path then the set is considered, otherwise the set is discarded. This strategy has been used in Ertl [24], Wagner & Willhalm [66], Gutman [35], Lauther [46] and Fu et al. [31].

Multi-level approach: introduced by Schulz et al. [60], they are based on the overlay graph concept. Starting from a graph $G = (V, A)$, a *multi-level graph* $\mathcal{M}$ is computed as follows: A is extended by multiple levels of arcs, and for each pair of nodes $s, t \in V$ exists a subgraph of $\mathcal{M}$ smaller than G such that the shortest distance from s to t in the sub-graph is equal to the shortest distance from s to t in G. Some works which adopt this approach were presented by Bauer et al. [5], Delling et al. [17] and Holzer et al. [38].

Hu & Chiu [40] proposed an algorithm to find K shortest paths on time-dependent graphs. The K paths should not be highly overlapped and their travel times should be comparable. If these two conditions hold, a traveler can choose one of the proposed paths taking in account its personal constraints. The algorithm iteratively finds the K shortest paths, updating opportunely the network and the arcs costs between two consequent search queries.

Sun et al. [61] proposed two algorithms. The first algorithm is a Heap-based Bellman-Ford algorithm to solve the *Query_FiST* problem, where a driver starting at a given time wants to reach

the destination in the smallest possible time. The second algorithm is an Extended Bellman-Ford algorithm that solves the *Query_BeST*, choosing the best starting time to avoid congestion. The experimental results show that their algorithms outperform older approaches and are able to solve in few seconds instances with more than 10000 nodes and 20000 arcs.

A new interesting research branch for shortest path problems makes use of neural networks. Huang et al. [42] proposed an approach that models the time dependent graph as a neural network and use the auto-waves generated from the neurons to find the shortest paths.

5 CONCLUSIONS

In this survey, starting from the origin till the current state of the art, two different kinds of shortest path problem on dynamic networks have been analyzed: reoptimization of shortest paths and time-dependent SPP. The onset of these problems can be traced back respectively to the works of Gallo [32] and Cooke & Halsey [13], both inspired by the seminal algorithm of Dijkstra [21] and Ford Jr [28] for the shortest path problem. In Tables 1, 2, and 3 the most relevant works are listed, respectively surveyed in this paper for root change, arc cost change and time dependent shortest path. More specifically, each table reports the problem in question, the reference, the computational complexity of the proposed algorithm (if specified), and the year of publication. For the three main problem categories, {P2P, SPT and APSP}, the following notations have been adopted: a^+ and a^- are respectively used to indicate cost increase and decrease for a single arc, while A^+ and A^- are similarly used for batch updates, addressing respectively cost increase and decrease for a whole a set of arcs.

Table 1 – Root change chronology. SPT indicates shortest path tree reoptimization.

Problem	Reference	Complexity	Year
SPT	Gallo [32]	$o(m + n \cdot d_{\max})$ $d_{\max}$ is an upper bound on the length of the maximum path.	1980
SPT	Florian et al. [27]	$\max\{O(n^2 \cdot \log k),\ O(m \cdot \log k)\}$ k is the largest number of nodes with temporary labels.	1981
SPT	Gallo & Pallottino [33]	$O(n^2)$	1982
SPT	Ferone et al. [26]	$O\left(\frac{n}{2} \cdot (n+1) \cdot \log n\right)$	2016

Reoptimization of shortest paths tackles a polynomially solvable combinatorial optimization problem in a context in which a previous optimal solution to a closely related instance is known. Most of the scientific effort has been devoted to the design of optimal techniques, in the attempt to devise increasingly performing implementations of algorithms with a common semantic core. It is from this perspective that in future works it might be of great interest an in-depth study of innovative data structures, as major step in the progress towards this goal.

Table 2 – Arc cost change chronology.

Problem	Reference	Complexity	Year
SPT $\{a^+, a^-\}$	Gallo [32]	$o(m + n \cdot d_{\max})$	1980
P2P $\{A^+, A^-\}$	Ramalingam & Reps [58]	$O(\|\|\delta\|\| \log \|\|\delta\|\|)^2$ δ is a measure of the change in the input and output.	1996
APSP $\{A^+, A^-\}$	King [45]	$O\left(n^{2.5} \cdot \sqrt{b \cdot \log n}\right)$ b is an upper bound on the weight of the arcs.	1999
SPT $\{a^+, a^-\}$	Narváez et al. [51]	$O\left(d_{\max} \cdot \delta_d^3\right)$ complexity concerns Bellman-Ford version, δ_d denote the minimum number of nodes that must change their distance or parent attributes (or both).	2000
SPT $\{A^+, A^-\}$	Narvaez et al. [52]	$O(\delta_e \cdot \ln(\delta_n))$ with binary heap, where δ_e and δ_n are the maximum number of key decrements and the maximum size of the heap, respectively.	2001
SPT $\{A^+, A^-\}$	Pallottino & Scutellà [55]	$O(hm + \min\{hn \log n, C_p\} + \min\{n \log n, C_d, kn\}$ Let K^+ and K^- the set of the edge incremented and decremented, then C_p and C_d measure the cost perturbation due to the arcs in K^+ and K^-, $k \leq \min\{n - n_r, C_d\}$, where n_r is the dimension of the first fragment of the new solution, while h is the number of primal phases of the algorithm.	2003
P2P $\{A^+, A^-\}$	Thomas & White [63]	$O(2^L)$ $L = \|A^I\|$, where $A^I \in A$ is a set of observed arcs.	2007
SPT $\{a^+, a^-\}$	Buriol et al. [9]	see Ramalingam & Reps (1996)	2008
P2P $\{A^+, A^-\}$	Nannicini et al. [48]	$O(m + n \cdot \log n)$ with Fibonacci's heap	2008
SPT $\{A^+, A^-\}$	Chan & Yang [12]	complexity for each unit operation	2009
SPT $\{A^+, A^-\}$	Tretyakov et al. [65]	$O(k^2 \cdot D^2)$ k is the number of landmarks, D is the diameter of the graph.	2011
P2P $\{A^+, A^-\}$	D'Andrea et al. [23]	$O\left(\left(\|B\| + \|\hat{\Delta}(G, B)\|\right) \cdot k\right) \cdot \log n\right)$ let β be a batch of updates operations, $\|B\| = \sum_{\beta_i \in B} \|\beta_i\|$, $\|\hat{\Delta}(G, B)\|$ is the sum over all batches of B of the number of vertices affected by each of these batches, and $k = O\left(\sqrt{m}\right)$.	2013
P2P $\{A^+, A^-\}$	Hong et al. [39]	$O\left(m + m_{out} \cdot m_{in}\right)$ m_{in} is the average value of the in degree, and m_{out} the average value of the out degree.	2017

Table 3 – Time dependent chronology. Since the arcs have a time-depend cost in this case are considered both the case of cost increasing and cost decreasing.

Problem	Reference	Complexity	Year
APSP	Cooke & Halsey [13]	$O(n^3 \cdot M)$ travel time on the arcs are defined in multiple of positive unit of time δ for every time step of the discrete scale $S_M = \{t_0, t_0 + \delta, t_0 + 2\delta, \ldots, t_0 + M\delta\}$, M is chosen so that the travel times are defined for any $t \in S_M$	1966
APSP	Dreyfus [22]	$O(n^3 \cdot M)$	1969
APSP	Orda & Rom [54]	$O^f(m \cdot n)$ O^f expresses a functional complexity: an algorithm operating on functions has a functional complexity $\alpha(q)$, denoted $O^f(\alpha(q))$, if there is a constant $k > 0$ such that for an input of dimension q the number of simple function operations performed by the algorithm is bounded by $k\alpha(q)$.	1990
APSP	Kaufman & Smith [44]	$O(n^2)$ considering a consistency condition. Otherwise, see Cooke & Halsey (1966).	1993
APSP	Ziliaskopoulos & Mahmassani [67]	$O(n \cdot M \cdot d^{1.4})$ d is the average in degree of a node in the graph.	1993
APSP	Demetrescu & Italiano [18]	$O(S \cdot n^{2.5} \cdot log^3 n)$ S is a bound on the different real values that each arc can assume.	2006
P2P, SPT	Delling & Wagner [16]	—	2007
P2P	Nannicini et al. [49]	—	2008
P2P	Batz et al. [3]	$O(\lvert f_{uv}\rvert + \lvert f_{vw}\rvert)$ $f_{uw}(t) = (t + f_{uv}(t))$ for a path $\langle u, v, w\rangle$. In general, $f(t)$ is a weight function, associated to the arc, that specifies the travel time at the endpoint of the arc when the arc is entered at time t.	2009
P2P	Bauer & Delling [4]	—	2009
P2P	Nannicini et al. [50]	—	2011
K-SPP (K-shortest path)	Hu & Chiu [40]	—	2015
P2P	Huang et al. [42]	$O(c \cdot \sum_{i=1}^{n} z_i)$ c is a constant value, z_i is the time window considered for each neuron. The work presents a time-dependent neural network.	2017
APSP	Sun et al. [61]	$O(k \cdot m \cdot \log(n))$ k is a constant value.	2017

Time-dependent SPP are shortest path problems whose arc cost function depends on both the arcs and the current instant belonging to a given time horizon. This line of research is of high interest given its suitability to model the occurrence of congestion in real transit planning scenarios. As indicated in the scientific literature, when the instance does not exhibit the FIFO property, the time-dependent SPP is NP-hard. Henceforth, in the near future it can be particularly appealing to explore the possibility of the solution of such intractable instances, and preferably those arising from case studies, by means of heuristic and/or meta-heuristic techniques.

REFERENCES

[1] AHO A, HOPCROFT J & ULLMAN J. 1974. *The design and analysis of computer algorithms*. Addison-Wesley series in computer science and information processing. Addison-Wesley Pub. Co.

[2] AHUJA RK, MAGNANTI TL & ORLIN JB. 1993. Network flows: theory, algorithms & applications.

[3] BATZ GV, DELLING D, SANDERS P & VETTER C. 2009. Time-dependent contraction hierarchies. In: *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, pages 97–105. Society for Industrial and Applied Mathematics.

[4] BAUER R & DELLING D. 2009. Sharc: Fast and robust unidirectional routing. *Journal of Experimental Algorithmics (JEA)*, **14**(4).

[5] BAUER R, DELLING D, SANDERS P, SCHIEFERDECKER D, SCHULTES D & WAGNER D. 2008. Combining hierarchical and goal-directed speed-up techniques for Dijkstra's algorithm. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5038 LNCS, pages 303–318.

[6] BAZARAA M & LANGLEY R. 1974. A dual shortest path algorithm. *SIAM Journal on Applied Mathematics*, **26**(3): 496–501.

[7] BELLMAN R. 1958. On a routing problem. *Quarterly of Applied Mathematics*, **16**: 87–90.

[8] BERTSEKAS DP. 1991. An auction algorithm for shortest paths. *SIAM Journal on Optimization*, **1**(4): 425–447.

[9] BURIOL LS, RESENDE MG & THORUP M. 2008. Speeding up dynamic shortest-path algorithms. *INFORMS Journal on Computing*, **20**(2): 191–204.

[10] CERULLI R, FESTA P & RAICONI G. 2003. Shortest path auction algorithm without contractions using virtual source concept. *Computational Optimization and Applications*, **26**(2): 191–208.

[11] CHABINI I & LAN S. 2002. 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.

[12] CHAN EP & YANG Y. 2009. Shortest path tree computation in dynamic graphs. *IEEE Transactions on Computers*, **58**(4): 541–557.

[13] COOKE KL & HALSEY E. 1966. The shortest route through a network with time-dependent internodal transit times. *Journal of mathematical analysis and applications*, **14**(3): 493–498.

[14] DANTZIG G. 1963. *Linear Programming and Extensions*. Princeton University Press.

[15] DELLING D & NANNICINI G. 2008. Bidirectional core-based routing in dynamic time-dependent road networks. In: *International Symposium on Algorithms and Computation*, pages 812–823. Springer.

[16] DELLING D & WAGNER D. 2007. Landmark-based routing in dynamic graphs. In: *International Workshop on Experimental and Efficient Algorithms*, pages 52–65. Springer.

[17] DELLING D, HOLZER M, MÜLLER K, SCHULZ F & WAGNER D. 2009. High-performance multi-level routing. American Mathematical Society Providence, RI, **74**: 73–92.

[18] DEMETRESCU C & ITALIANO GF. 2006. Fully dynamic all pairs shortest paths with real edge weights. *Journal of Computer and System Sciences*, **72**(5): 813–837.

[19] DIAL R, GLOVER F, KARNEY D & KLINGMAN D. 1979. A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees. *Networks*, **9**(3): 215–248.

[20] DIAL RB. 1969. Algorithm 360: Shortest-path forest with topological ordering [h]. *Communications of the ACM*, **12**(11): 632–633.

[21] DIJKSTRA EW. 1959. A note on two problems in connexion with graphs. *Numerische mathematik*, **1**(1): 269–271.

[22] DREYFUS SE. 1969. An appraisal of some shortest-path algorithms. *Operations research*, **17**(3): 395–412.

[23] D'ANDREA A, D'EMIDIO M, FRIGIONI D, LEUCCI S & PROIETTI G. 2013. Dynamically maintaining shortest path trees under batches of updates. In: *International Colloquium on Structural Information and Communication Complexity*, pages 286–297. Springer.

[24] ERTL G. 1998. Shortest path calculation in large road networks. *OR Spectrum*, **20**(1): 15–20.

[25] FERONE D, FESTA P, GUERRIERO F & LAGANÀ D. 2016. The constrained shortest path tour problem. *Computers & Operations Research*, **74**: 64–77.

[26] FERONE D, FESTA P, NAPOLETANO A & PASTORE T. 2016. Reoptimizing shortest paths: From state of the art to new recent perspectives. In: *Transparent Optical Networks (ICTON), 2016 18th International Conference on*, pages 1–5. IEEE.

[27] FLORIAN M, NGUYEN S & PALLOTTINO S. 1981. A dual simplex algorithm for finding all shortest paths. *Networks*, **11**(4): 367–378.

[28] FORD JR LR. 1956. Network flow theory. Technical report, DTIC Document.

[29] FORD JR LR & FULKERSON DR. 2015. *Flows in networks*. Princeton university press.

[30] FREDMAN ML & TARJAN RE. 1987. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, **34**(3): 596–615, July.

[31] FU L, SUN D & RILETT LR. 2006. Heuristic shortest path algorithms for transportation applications: state of the art. *Computers & Operations Research*, **33**(11): 3324–3343.

[32] GALLO G. 1980. Reoptimization procedures in shortest path problem. *Rivista di matematica per le scienze economiche e sociali*, **3**(1): 3–13.

[33] GALLO G & PALLOTTINO S. 1982. A new algorithm to find the shortest paths between all pairs of nodes. *Discrete Applied Mathematics*, **4**(1): 23–35.

[34] GOLDBERG AV & HARRELSON C. 2005. Computing the shortest path: A* search meets graph theory, pages 156–165.

[35] GUTMAN RJ. 2004. Reach-based routing: A new approach to shortest path algorithms optimized for road networks. *ALENEX/ANALC*, **4**: 100–111.

[36] HART PE, NILSSON NJ & RAPHAEL B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, **4**(2): 100–107.

[37] HILGER M, KÖHLER E, MÖHRING RH & SCHILLING H. 2009. Fast point-to-point shortest path computations with arc-flags. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, **74**: 41–72.

[38] HOLZER M, SCHULZ F & WAGNER D. 2009. Engineering multilevel overlay graphs for shortest-path queries. *Journal of Experimental Algorithmics (JEA)*, **13**: 5.

[39] HONG J, PARK K, HAN Y, RASEL MK, VONVOU D & LEE Y-K. 2017. Disk-based shortest path discovery using distance index over large dynamic graphs. *Information Sciences*, **382-383**: 201–215.

[40] HU X & CHIU Y-C. 2015. A Constrained Time-Dependent K Shortest Paths Algorithm Addressing Overlap and Travel Time Deviation. *International Journal of Transportation Science and Technology*, **4**(4): 371–394.

[41] HUANG B, WU Q & ZHAN F. 2007. A shortest path algorithm with novel heuristics for dynamic transportation networks. *International Journal of Geographical Information Science*, **21**(6): 625–644.

[42] HUANG W, YAN C, WANG J & WANG W. 2017. A time-delay neural network for solving time-dependent shortest path problem. *Neural Networks*, **90**: 21–28. ISSN 0893-6080.

[43] KANOULAS E, DU Y, XIA T & ZHANG D. 2006. Finding fastest paths on a road network with speed patterns. In: *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 10–10. IEEE.

[44] KAUFMAN DE & SMITH RL. 1993. Fastest paths in time-dependent networks for intelligent vehicle-highway systems application. *Journal of Intelligent Transportation Systems*, **1**(1): 1–11.

[45] KING V. 1999. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 81–89. IEEE.

[46] LAUTHER U. 2004. An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background. *Geoinformation und Mobilität-von der Forschung zur praktischen Anwendung*, **22**: 219–230.

[47] MÖHRING RH, SCHILLING H, SCHÜTZ B, WAGNER D & WILLHALM T. 2007. Partitioning graphs to speedup dijkstra's algorithm. *Journal of Experimental Algorithmics (JEA)*, **11**: 2–8.

[48] NANNICINI G, BAPTISTE P, KROB D & LIBERTI L. 2008. Fast paths in dynamic road networks. *Proceedings of ROADEF*, **8**: 1–14.

[49] NANNICINI G, DELLING D, LIBERTI L & SCHULTES D. 2008. Bidirectional A* search for time-dependent fast paths. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5038 LNCS, pages 334–346.

[50] NANNICINI G, DELLING D, SCHULTES D & LIBERTI L. 2012. Bidirectional a* search on time-dependent road networks. *Networks*, **59**(2): 240–251.

[51] NARVÁEZ P, SIU K-Y & TZENG H-Y. 2000. New dynamic algorithms for shortest path tree computation. *IEEE/ACM Transactions on Networking (TON)*, **8**(6): 734–746.

[52] NARVAEZ P, SIU K-Y & TZENG H-Y. 2001. New dynamic spt algorithm based on a ball-and-string model. *IEEE/ACM Transactions on Networking (TON)*, **9**(6): 706–718.

[53] NICHOLSON TAJ. 1966. Finding the shortest route between two points in a network. *The computer journal*, **9**(3): 275–280.

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

[55] PALLOTTINO S & SCUTELLÀ MG. 2003. A new algorithm for reoptimizing shortest paths when the arc costs change. *Operations Research Letters*, **31**(2): 149–160, mar 2003.

[56] PAPE U. 1974. Implementation and efficiency of moore-algorithms for the shortest route problem. *Mathematical Programming*, **7**(1): 212–222.

[57] PUGLIESE LDP & GUERRIERO F. 2013. A survey of resource constrained shortest path problems: Exact solution approaches. *Networks*, **62**(3): 183–200.

[58] RAMALINGAM G & REPS T. 1996. An incremental algorithm for a generalization of the shortest-path problem. *Journal of Algorithms*, **21**(2): 267–305.

[59] SANDERS P & SCHULTES D. 2006. Engineering highway hierarchies. In: *European Symposium on Algorithms*, pages 804–816. Springer.

[60] SCHULZ F, WAGNER D & ZAROLIAGIS C. 2002. Using multi-level graphs for timetable information in railway systems. *Algorithm Engineering and ...* , **00104**: 43–59.

[61] SUN Y, YU X, BIE R & SONG H. 2017. Discovering time-dependent shortest path on traffic graph for drivers towards green driving. *Journal of Network and Computer Applications*, **83**: 204–212.

[62] TAKAOKA T. 1992. A new upper bound on the complexity of the all pairs shortest path problem. *Information Processing Letters*, **43**(4): 195–199.

[63] THOMAS BW & WHITE CC. 2007. The dynamic shortest path problem with anticipation. *European Journal of Operational Research*, **176**(2): 836–854.

[64] TOTH P & VIGO D. 2014. *Vehicle routing: problems, methods & applications*. SIAM.

[65] TRETYAKOV K, ARMAS-CERVANTES A, GARCÍA-BAÑUELOS L, VILO J & DUMAS M. 2011. Fast fully dynamic landmark-based estimation of shortest path distances in very large graphs. In: *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1785–1794. ACM.

[66] WAGNER D & WILLHALM T. 2003. Geometric speed-up techniques for finding shortest paths in large sparse graphs. In: *European Symposium on Algorithms*, pages 776–787. Springer.

[67] ZILIASKOPOULOS AK & MAHMASSANI HS. 1993. Time-dependent, shortest-path algorithm for real-time intelligent vehicle highway system applications. *Transportation research record*, pages 94–94.