The shortest path tour problem: problem definition, modeling, and optimization

Paola Festa

Department of Mathematics and Applications, University of Napoli FEDERICO II
Compl. MSA, Via Cintia, 80126 Napoli, Italy. Email: `paola.festa@unina.it`

Abstract

The shortest path tour problem (SPTP) consists of finding a shortest path from a given origin node s to a given destination node d in a directed graph with nonnegative arc lengths with the constraint that the optimal path P should successively pass through at least one node from given node subsets $T_1, T_2, \ldots, T_N$, where $\bigcap_{k=1}^{N} T_k = \emptyset$.

In this paper, it will proved that the SPTP belongs to the complexity class **P** and several alternative techniques will be presented to exactly solve it.

Keywords: *Shortest path problems, network flow problems, network optimization, combinatorial optimization.*

1 Introduction

In this paper, a new variant of shortest path problems is described: the shortest path tour problem (SPTP). Let $G = (V, A, C)$ be a directed graph, where

- V is a set of nodes, numbered $1, 2, \ldots, n$, $A = \{(i,j) |\ i, j \in V\}$ is a set of m arcs, and $C : E \rightarrow \mathbb{R}^+ \cup \{0\}$ is a function that assigns a nonnegative length to any arc $(i,j) \in A$;
- For each node $i \in V$, let $FS(i) = \{j \in V \mid (i,j) \in A\}$ and $BS(i) = \{j \in V \mid (j,i) \in A\}$ be the *forward star* and *backward star* of node i, respectively;
- A *simple path* $P = \{i_1, i_2, \ldots, i_k\}$ is a walk without any ripetition of nodes;
- The length $L(P)$ of any path P is defined as the sum of lengths of the arcs connecting consecutive nodes in the path.

Then, the SPTP can be stated as follows:

Definition 1 *The SPTP consists of finding a shortest path from a given origin node $s \in V$ to a given destination node $d \in V$ in the graph G with the constraint that the optimal path P should successively pass through at least one node from given node subsets $T_1, T_2, \ldots, T_N$, where $\bigcap_{k=1}^{N} T_k = \emptyset$.*

In more detail, P starts at $s \in T_1$, moves to some node in T_2 (possibly through some intermediate nodes that are not in T_2), then moves to some node in T_3 (possibly through some intermediate nodes that are not in T_3, but may be in T_1 and/or in T_2), etc., then finally it moves to $d \in T_N$ (possibly through some intermediate nodes not equal to d). The shortest path tour problem and the idea behind it were given in 2005 as Exercise 2.9 in Bertsekas's Dynamic Programming and Optimal Control book [3],

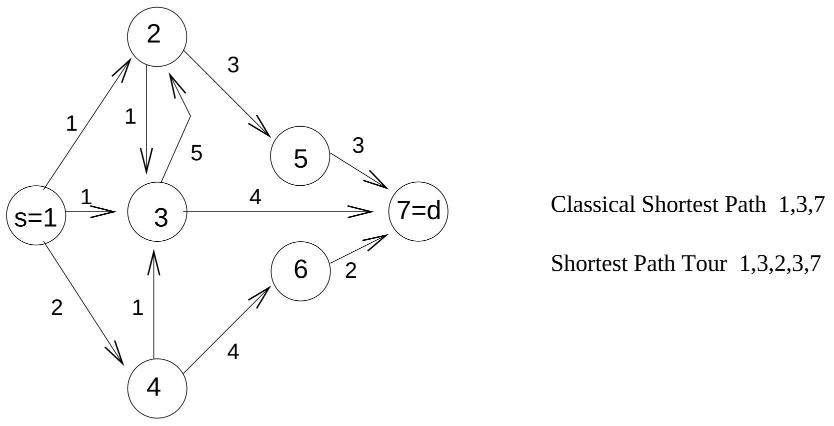

Figure 1: *A SPTP instance on a small graph G.*

where it is asked to formulate it as a dynamic programming problem. This problem has several practical applications, such as for example in warehouse management or control of robot motions. In both cases, there are precedence constraints to be satisfied. In the first case, assume that an order arrives for a certain set of N collections of items stored in a warehouse. Then, a vehicle has to collect at least an item of each collection of the order to ship them to the costumers. In control of robot motions, assume that to manufacture workpieces, a robot has to perform at least one operation selected from a set of N types of operations. In this latter case, operations are associated with nodes of a directed graph and the time needed for a tool change is the distance between two nodes.

The description and the properties of the problem are provided in Section 2, while in Section 3 several alternative techniques will be presented to exactly solve it. Finally, in Section 4, preliminary computational results are presented to demonstrate empirically which algorithms result more efficient in finding an optimal solution to several different problem instances.

2 Problem description

Let us consider the small graph $G = (V, A, C)$ depicted in Figure 2, where $V = \{s = 1, 2, \dots, 7 = d\}$. It is easy to see that $P = \{1, 3, 7\}$ is the shortest path from node 1 to node 7 and has length 5. Let us now define on the same small graph G the SPTP instance characterized by $N = 4$ and the following node subsets $T_1 = \{s = 1\}$, $T_2 = \{3\}$, $T_3 = \{2, 4\}$, $T_4 = \{d = 7\}$. The shortest path tour from node 1 to node 7 is the path $P_T = \{1, 3, 2, 3, 7\}$ which has length 11 and is not simple, since it passes twice through node 3. When dealing with any given optimization problem (such as SPTP), one is usually interested in classifying it according to its complexity in order to be able to design an algorithm that solves the problem finding the best compromise between solution quality and computational time required to find that solution. In classifying a problem according to its complexity, polynomial-time reductions are helpful. In fact, deciding the complexity class of an optimization problem Pr becomes easy once a polynomial-time reduction to a second problem $\bar{P}r$ is available and the complexity of $\bar{P}r$ is known, as stated in the following Definition 2 and Theorem 1, whose proof is reported in [15] and several technical books, included [16].

Definition 2 *An optimization problem Pr is reducible to an optimization problem $\bar{P}r$ ($Pr <_P \bar{P}r$) if there exists a polynomial-time computable function f such that*

$$x \text{ is an instance of } Pr \iff f(x) \text{ is an instance of } \bar{P}r; \tag{1}$$

f is called the reduction function *and a polynomial-time algorithm $\mathcal{A}$ that computes f is called* a reduction algorithm.

```
algorithm SPTPReduction($V$,$A$,$C$,$N$,$(T_k)_{k=1,2,\dots,N}$)
1 $V' := \{1, 2, \dots, N * n\}$; $A' := \emptyset$;
2 for $k = 1$ to $N - 1 \rightarrow$
3     for $v = 1$ to $n \rightarrow$
4         for each $w \in FS(v) \rightarrow$
5             if $(w \in T_{k+1})$ add($A'$,$(v + (k-1) * n, w + k * n)$,$c_{vw}$);
6             else add($A'$,$(v + (k-1) * n, w + (k-1) * n)$,$c_{vw}$);
7 for $v = 1$ to $n \rightarrow$
8     for each $w \in FS(v) \rightarrow$ add($A'$,$(v + (N-1) * n, w + (N-1) * n)$,$c_{vw}$);
9 return $(V'$,$A'$,$C')$;
end SPTPReduction
```

Figure 2: *Pseudo-code of a polynomial reduction algorithm.*

Theorem 1 *Let Pr and $\bar{P}r$ be any two optimization problems such that $Pr <_P \bar{P}r$, then $\bar{P}r$ in the complexity class of polynomially solvable problems $\mathbf{P}$ implies $Pr \in \mathbf{P}$.*

Hence, Theorem 1 guarantees that problem Pr and problem $\bar{P}r$ belong to the same complexity class, or in other words problem Pr is no harder than problem $\bar{P}r$. Theorem 2 uses a polynomial-time reduction and theoretical result of Theorem 1 to prove that SPTP belongs to the complexity class $\mathbf{P}$, since it reduces to a single source-single destination shortest path problem (SPP).

Theorem 2 *SPTP belongs to the complexity class $\mathbf{P}$.*

Proof: To prove the thesis, a polynomial-time reduction algorithm must be found that transforms any SPTP instance into a single source - single destination SPP instance and viceversa.

It is trivial to show that any SPP instance $< G = (V, A, C), s, d >$ can be polynomially transformed in the SPTP instance $< G = (V, A, C), s, d, N, \{T_i\}_{k=1,\dots,N} >$, where $N = 1$ and $T_1 = V$.

Viceversa, there exists a polynomial-time reduction algorithm that transforms any SPTP instance $< G = (V, A, C), s, d, N, \{T_k\}_{k=1,\dots,N} >$ into a single source - single destination SPP instance $< G' = (V', A', C'), s, d' = d + (N-1) \cdot n >$, where G' is a multi-stage graph with N stages, each replicating G. Figure 2 depicts the pseudo-code of the reduction algorithm `SPTPReduction` that performs the following operations:

i. (line 1) $V' := \{1, 2, \dots, N \cdot n\}$; $A' := \emptyset$;

ii. (lines 2–6) at each iteration, an arc (a, b) is added to A'. In particular, for each stage $k \in \{1, \dots, N-1\}$, for each node $v \in \{1, \dots, n\}$, and for each adjacent node $w \in FS(v)$, $(a, b) = (v + (k-1) \cdot n, w + k \cdot n)$ with length c_{vw}, if $w \in T_{k+1}$; $(a, b) = (v + (k-1) \cdot n, w + (k-1) \cdot n)$ with length c_{vw}, otherwise.

It is easy to see that $|A'| = N \cdot m$ and therefore the computational complexity of `SPTPReduction` is $O(N \cdot m)$. Note that, since $\bigcap_{k=1}^{N} T_k = \emptyset$, it results that $N \leq n$, and therefore, the worst case computational complexity is $O(n \cdot m)$. Figure 3 depicts the multi-stage graph G' corresponding to the SPTP instance on the small graph G of Figure 2. Note that, $|V'| = N \cdot n = 4 \cdot 7 = 28$, $|A'| = N \cdot m = 4 \cdot 11 = 44$, $s = 1$, $d' = d + (N-1) \cdot n = 7 + 3 \cdot 7 = 28$.

We now claim that in G', as constructed above, there is a path P' from s to d' of length K if and only if in G there is a path tour P_T from s to d of length K. Suppose that in G' there is a path P' of length K from s to d'. Because P' connects $s \in T_1$ to $d' \in T_N$, for each $k = 1, \dots, N-1$ there must be in P' necessarily at least one arc connecting two nodes in consecutive stages k and $k+1$. Therefore, it follows that P' consists of at least one node in each of the N stages, so corresponding to a path tour P_T of length K in G that successively passes through at least one node from the given node subsets $T_1, T_2, \dots, T_N$.

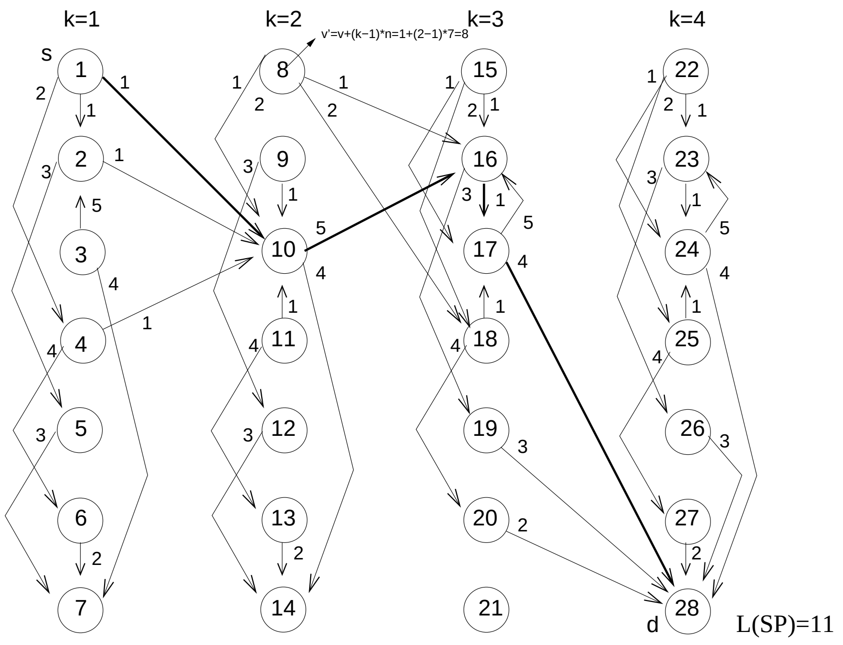

Figure 3: *The multi-stage graph G' corresponding to the SPTP instance on G of Figure 2.*

Viceversa, suppose that in G there is a path tour P_T of length K that successively passes through at least one node from the given node subsets $T_1, T_2, \ldots, T_N$. Then, by construction, for each arc in P_T connecting in G two nodes belonging to consecutive subsets, there exists in the simple path P' an arc connecting in G' two nodes belonging to consecutive stages, till finally moving to d' in the last stage N. ■

3 Several alternative algorithms

Once obtained in polynomial-time the multi-stage graph G', any shortest path algorithm can be applied to solve the resulting SPP. Although the huge number of state-of-the-art algorithms for the SPP, there does not exist a *best* method that outperforms all the others. In fact, recent research lines tend to develop techniques designed *ad hoc* for solving special structured shortest path problems: either a special network topology or a special cost structure. Exhaustive surveys of the most interesting and efficient shortest path algorithms, important for their computational time complexity or for their practical efficiency, can be found among others in [1], [8, 9, 7], [11], [10], [12], [14], and [17].

We have designed the following algorithms:

- a dynamic programming algorithm, as suggested in [3];
- a Dijkstra like algorithm ([13]) that uses a binary heap to store the nodes with temporary labels;
- several Auction like algorithms ([2, 4]): a *forward*, a *backward*, and a *combined for/backward* version.

To describe the dynamic programming approach we will use a slight different notation to represent an *expanded graph* $G' = (V', A', C')$.

For each node $i \in V$, V' contains $N+1$ nodes $(i, 0)$, $(i, 1)$, …, (i, N). The meaning of being in node (i, k), $k = 1, 2, \ldots, N$, is that we are at node i and have already successively visited the sets $T_1, \ldots, T_k$, but not yet the sets $T_{k+1}, \ldots, T_N$. The meaning of being in node $(i, 0)$ is that we are at node i and have not yet visited any node in the set T_1. For each arc $(i, j) \in A$ and for each $k = 0, 1, \ldots, N-1$, we introduce in A' an arc from (i,k) to (j,k), if $j \notin T_{k+1}$ or an arc from (i,k) to $(j,k+1)$, if $j \in T_{k+1}$. Moreover, for each arc $(i,j) \in A$ we introduce in A' an arc from (i,N) to (j,N). Once obtained the expanded graph G', the SPTP is equivalent to find a shortest path from $(s,0)$ to (d,N).

Let $D^{r+1}(i,k)$, $k = 0,1,\ldots,N$, be the shortest distance from (i,k) to the destination node (d,N) using r arcs or less.

For $k = 0,1,\ldots,N-1$ the DP iteration is the following:

$$\begin{aligned} D^{r+1}(i,k) &= \min_{(i,j)\in A} \left\{ \min_{j\notin T_{k+1}} \{c_{ij} + D^r(j,k), \min_{j\in T_{k+1}} \{c_{ij} + D^r(j,k+1)\} \right\} \\ D^{r+1}(i,N) &= \begin{cases} \min_{(i,j)\in A} \{c_{ij} + D^r(j,N)\}, & \text{if } i \neq d; \\ 0, & \text{if } i = d. \end{cases} \end{aligned} \tag{2}$$

Initial conditions are the following:

$$D^0(i,k) = \begin{cases} \infty, & \text{if } (i,k) \neq (d,N); \\ 0, & \text{if } (i,k) = (d,N). \end{cases}$$

Theorem 3 *An algorithm implementing the above DP iteration terminates after a finite number of iterations with an optimal solution and its computational complexity is $O(N^2 \cdot n \cdot m)$, and $O(n^3 \cdot m)$ in the worst case.*

Proof: Given the nonnegativity of arc lengths, for each r and for each node (i,k), $D^{r+1}(i,k) \leq D^r(i,k)$, since there are more candidate paths as the number of arcs increases. Moreover, a shortest path from node (i,k) to node (d,N) that uses at most $r+1$ arcs consists of an initial arc from (i,k) to node (j,l) $(l \in \{k, k+1\})$ and a path $\overline{P}$ from (j,l) to (d,N) that crosses at most r arcs. Obviously, DP iteration (2) follows from considering that $\overline{P}$ should be chosen as short as possible and its length is therefore $D^r(j,l)$ and that node (j,l) should also be chosen in the most profitable way.

Since there are no negative length cycles, there exists a shortest path having at most $|V'| - 1$ arcs, and therefore the number of needed iterations is at most $|V'| = n \cdot (N+1)$.

Finally, it can be easily seen that the computational complexity of an algorithm implementing the DP iteration (2) is $O(N^2 \cdot n \cdot m)$, and $O(n^3 \cdot m)$ in the worst case. ∎

4 Experimental results and future work

In this section, we report on some preliminary computational experiments designed to determine which algorithm among those proposed seems to be more effective to solve the SPTP.

The following algorithms have been implemented in C language and run on a Pentium 4 with 2400 Mhz and 512 Mb of RAM:

- `Dijkstra` forward with binary heap;
- standard `Auction forward`;
- standard `Auction backward`;
- combined for/backward `Auction fb`;
- `DP Dijkstra`, DP with Dijkstra as initialization phase;
- `DP Auction`, DP with Auction as initialization phase;
- `DP Dijkstra`, DP with Dijkstra as initialization phase;
- `DP Auction back`, DP with Auction backward as initialization phase;

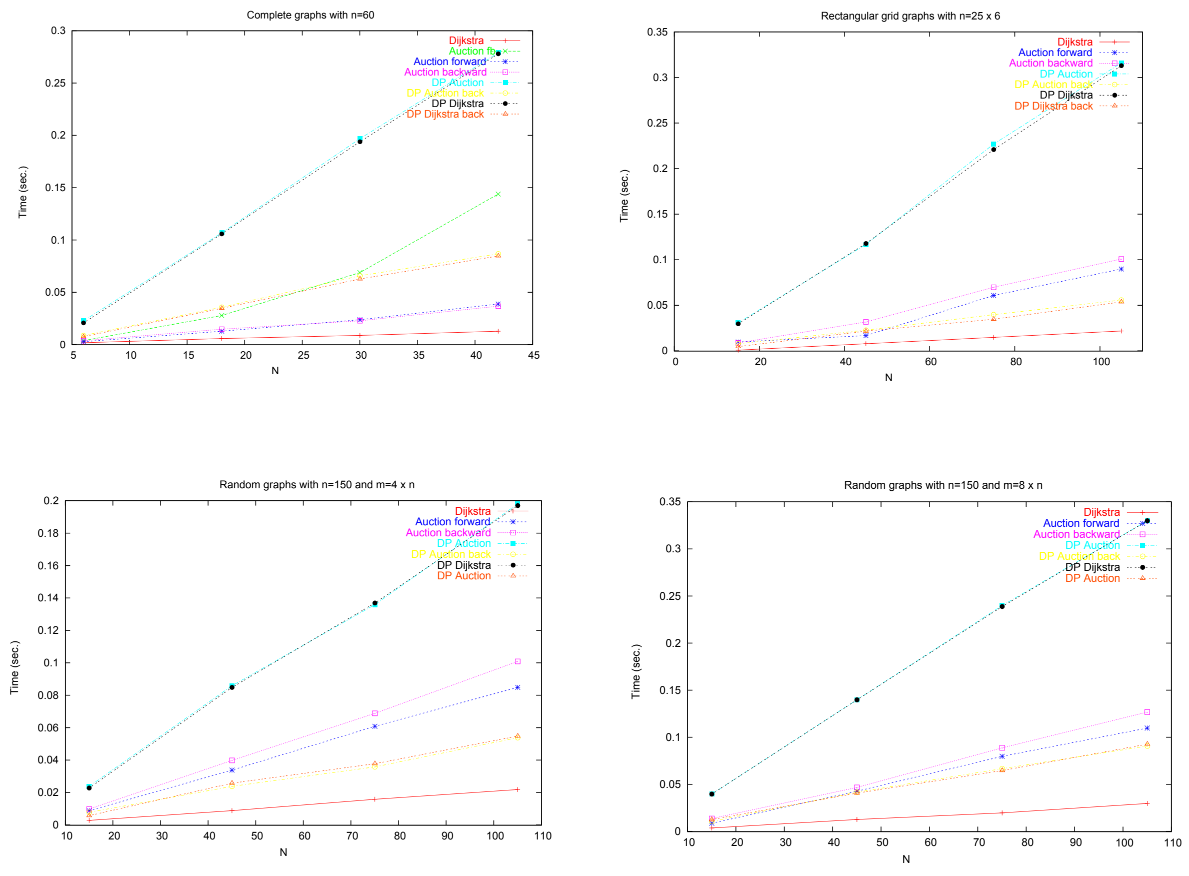

Figure 4: *For each problem family,* 10 *different instances have been generated for each possible value of* $N \in \{10\%n, 30\%n, 50\%n, 70\%n\}$ *and the mean time (in sec.) required to find an optimal solution has been stored and plot.*

- `DP Dijkstra back`, DP with Dijkstra backward as initialization phase.

The algorithm simply implementing DP iteration (2) has been also coded and it is has been revealed too time consuming and not very efficient. The algorithm can be made more efficient by observing that to calculate $D(i,k)$ for all i, it is not needed to know $D(i,k-1),\ldots,D(i,0)$, but it is sufficient to know just $D(j,k+1)$ for $j \in T_{k+1}$. Thus, we have calculated first $D(i,N)$ using a standard shortest path computation (`DP Dijkstra`, `DP Auction`, `DP Dijkstra`, `DP Auction back`, `DP Dijkstra back`).

The algorithms above listed have been tested on the following graph families: 1) complete graphs with $n \in \{60, 100\}$; 2) square grids with $n = 10 \times 10$ and rectangular grids with $n = 25 \times 6$; 3) random graphs with $n = 150$ and $m \in \{4 \times n, 8 \times n\}$.

For each problem family, 10 different instances have been generated for each possible value of $N \in \{10\%n, 30\%n, 50\%n, 70\%n\}$ and the mean time (in sec.) required to find an optimal solution has been stored and plot in Figure 4. Looking at the preliminary results, it seems that Dijkstra's algorithm outperforms all the competitors. Nevertheless, further experiments are needed on a wider set of instances and further investigation is planned in the next future in order to implement and test a collection of different algorithms by using path cost upper bounds, by mixing Auction and *graph collapsing* ([5]) and *virtual sources* ([6]) ideas. Moreover, it would be interesting to study some variants of the SPTP and their complexity where the constraint $\bigcap_{k=1}^{N} T_k = \emptyset$ is relaxed and/or arc capacity constraints are added.

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms and Applications.* Prentice-Hall Englewood Cliffs, 1993.

[2] D.P. Bertsekas. An auction algorithm for shortest paths. *SIAM Journal on Optimization*, 1:425–447, 1991.

[3] D.P. Bertsekas. *Dynamic Programming and Optimal Control. 3rd Edition*, volume I. Athena Scientific, 2005.

[4] D.P. Bertsekas, S. Pallottino, and M.G. Scutellà. Polynomial auction algorithms for shortest paths. *Computational Optimization and Application*, 4:99–125, 1995.

[5] R. Cerulli, P. Festa, and G. Raiconi. Graph collapsing in shortest path auction algorithms. *Computational Optimization and Applications*, 18:199–220, 2001.

[6] R. Cerulli, P. Festa, and G. Raiconi. Shortest path auction algorithm without contractions using virtual source concept. *Computational Optimization and Applications*, 26(2):191–208, 2003.

[7] B.V. Cherkassky and A.V. Goldberg. Negative-cycle detection algorithms. *Mathematical Programming*, 85:277–311, 1999.

[8] B.V. Cherkassky, A.V. Goldberg, and T. Radzik. Shortest path algorithms: Theory and experimental evaluation. *Mathematical Programming*, 73:129–174, 1996.

[9] B.V. Cherkassky, A.V. Goldberg, and C. Silverstein. Buckets, heaps, lists, and monotone priority queues. *SIAM Journal of Computing*, 28:1326–1346, 1999.

[10] E.V. Denardo and B.L. Fox. Shortest route methods: 2. group knapsacks, expanded networks, and branch-and-bound. *Operational Research*, 27:548–566, 1979.

[11] E.V. Denardo and B.L. Fox. Shortest route methods: reaching pruning, and buckets. *Operational Research*, 27:161–186, 1979.

[12] N. Deo and C. Pang. Shortest path algorithms: taxonomy and annotation. *Networks*, 14:275–323, 1984.

[13] E. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1:269–271, 1959.

[14] G. Gallo and S. Pallottino. Shortest path methods: A unified approach. *Math. Programming Study*, 26:38–64, 1986.

[15] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations.* Plenum Press, 1972.

[16] C.H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: Algorithms and complexity.* Prentice-Hall, 1982.

[17] D.R. Shier and C. Witzgall. Properties of labeling methods for determining shortest path trees. *Journals of Research of the National Bureau of Standards*, 86:317–330, 1981.