Restricted Shortest Path in Temporal Graphs

Sudip Biswas, Arnab Ganguly(✉), and Rahul Shah

School of Electrical Engineering and Computer Science, Louisiana State University, Baton Rouge, USA
{sbiswa7,agangu4}@lsu.edu, rahul@csc.lsu.edu

Abstract. The *restricted shortest path* (RSP) problem on directed networks is a well-studied problem, and has a large number of applications such as in *Quality of Service* routing. The problem is known to be *NP-hard*. In certain cases, however, the network is not static i.e., edge parameters vary over time. In light of this, we extend the RSP problem for general networks to that for *temporal networks*. We present several exact algorithms for this problem, one of which uses heuristics, and is similar to the A^* algorithm. We experimentally evaluate these algorithms by simulating them on both existing temporal networks, and synthetic ones. Furthermore, based on one of the pseudo-polynomial exact algorithms, we derive a fully polynomial time approximation scheme.

1 Introduction

The restricted shortest path (RSP) problem on directed networks is an important problem both from theoretic and application point of view. The practical importance can be mainly attributed to its enormous application in *Quality of Service* (QoS) routing [12], where the objective is to find a path that satisfies various QoS attributes. Other applications include resource planning, airline industry [3], railway networks [16], and etc. One can visualize this problem as follows: a traveller wishes to travel from one place to another in the fastest possible way, but only has a limited amount of money. Therefore, she has to choose a path that provides the fastest travel time, but maintains her budget constraints.

In a large number of applications, however, the network is not static i.e., the edge attributes are dependent on time. An RSP in these networks has to maintain the edge constraints while taking the *temporal* information into account. In continuation with our example of a traveller above, this can be visualized as follows. Suppose, the traveller books multiple flights to reach her destination. Then, the departure of any flight from an intermediate city must be after the flight at which she arrived at this city.

1.1 Related Work

The RSP problem is *NP-hard* even for acyclic graphs [5]. The best known exact algorithms (in terms of complexity) are based on dynamic programming [8].

This research is funded in part by National Science Foundation (NSF) Grants CCF 1218904 and CCF 1527435.

Q. Chen et al. (Eds.): DEXA 2015, Part I, LNCS 9261, pp. 13–27, 2015.
DOI: 10.1007/978-3-319-22849-5_2

Using these (pseudo-polynomial) algorithms, the best known fully-polynomial time approximation scheme (FPTAS) has complexity $O(mn(\log\log n + 1/\epsilon))$ for general graphs [13] and $O(mn/\epsilon)$ for acyclic graphs [4]. Here, n and m are the number of vertices and edges respectively. Other techniques (see [2,10,11,17] and references therein), both exact and approximate, are based on label setting approach, Lagrangian relaxation, K-shortest paths approach, pre-processing schemes, heuristics, and A^* search.

A related problem is that of finding RSP in time varying graphs, where the length and the constraint of an edge depends on a particular time-stamp [1]. In temporal networks, on the other hand, every edge e has a start time, and and an arrival time, say t, which is also the arrival time of a path ending with e. Only the edges starting after (or, at) t can be taken to extend this path. Clearly, the RSP problem on temporal networks is at least as hard as that in static networks.

To the best of our knowledge, RSP problem on temporal graphs has not been previously studied. However, other path problems and connectivity problems in temporal networks have received significant attention. An important result was that of Kempe et al. [9], who showed that classical Menger's theorem is violated for temporal networks, and the problem of computing number of source-destination disjoint paths becomes *NP-complete*. Later, Mertzios et al. [14] showed that an alternative formulation of Menger's theorem holds for all temporal graphs. Recently, Wu et al. [18] showed that under reasonable preprocessing of the graph, single-source shortest paths can be found in $O(n+m)$ time. Travelling salesman problem on temporal networks has also been studied [15].

1.2 Notations and Problem Formulation

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a directed finite temporal graph on n vertices and m edges. Though, the correctness of our algorithms does not depend on n and m, we assume $m \geq n$. Let $\delta^+(v)$ (resp. $\delta^-(v)$) be the set of outgoing edges (resp. incoming edges) of a vertex v, and let $\Delta_{in} = \max\{|\ \delta^-(v)\ ||\ v \in \mathcal{V}\}$. Each edge $e \in \mathcal{E}$ has a source vertex $src(e)$, a target vertex $tgt(e)$, a length $len(e)$, a penalty (i.e., constraint) $pen(e)$, a start time $start(e)$, and an arrival time $end(e)$. We assume that length, penalty, start time and arrival time are all positive and integral. The transit time of an edge is the difference $end(e) - start(e)$.

For any non-empty subset $\mathcal{E}'$ of $\mathcal{E}$, let ith smallest edges of $\mathcal{E}'$ denote those edge in $\mathcal{E}'$ having the ith smallest $end(e)$ value. The graph can be initially preprocessed, such that for any edge $e \in \delta^-(v)$, in $O(1)$ time we can determine its rank among all edges in $\delta^-(v)$; this requires an additional $O(m)$ space.

Throughout this paper, s and g denote the source and destination vertices, and $[\mathsf{t}_\alpha, \mathsf{t}_\beta]$ denotes a time interval. An s-g ***temporal path*** w.r.t $[\mathsf{t}_\alpha, \mathsf{t}_\beta]$ is a path, say $\pi = \langle e_1, e_2, \ldots e_k \rangle$, such that $start(e_1) \geq \mathsf{t}_\alpha$, $start(e_{i+1}) \geq end(e_i)$, $1 \leq i \leq k-1$, and $end(e_k) \leq \mathsf{t}_\beta$, where $\mathsf{s} = src(e_1)$ and $\mathsf{g} = tgt(e_k)$. The start time (resp. arrival time) of π is $start(e_1)$ (resp. $end(e_k)$), and is denoted by $start(\pi)$ (resp. $end(\pi)$). An s-g ***restricted temporal path*** (RTP), say π, w.r.t $[\mathsf{t}_\alpha, \mathsf{t}_\beta]$ and a positive integer P, is an s-g temporal path w.r.t $[\mathsf{t}_\alpha, \mathsf{t}_\beta]$ that has

penalty at most P i.e., $pen(\pi) = \sum_{e \in \pi} pen(e) \leq \mathsf{P}$. An s-g ***restricted shortest temporal path*** (RSTP) is an s-g RTP π that minimizes the length $len(\pi) = \sum_{e \in \pi} len(e)$. Throughout this paper, P is a positive integral upper-bound on the allowed penalty of an s-g temporal path, and unless stated otherwise, the terms RTP and RSTP are used w.r.t $[\mathsf{t}_\alpha, \mathsf{t}_\beta]$ and P. We denote by $\mathsf{B_U}$ (resp. $\mathsf{B_L}$), an upper-bound (resp. lower-bound) on $\mathsf{L_{opt}}$, the length of an s-g RSTP.

Problem 1. (Restricted Shortest Temporal Path). *Let $\mathcal{G}$ be a temporal graph where each edge has a length, a penalty, a start time, and an arrival time.* ***Input:*** *Two vertices s and g, a maximum penalty P, and a time interval $[\mathsf{t}_\alpha, \mathsf{t}_\beta]$.* ***Output:*** *An s-g restricted shortest temporal path with respect to $[\mathsf{t}_\alpha, \mathsf{t}_\beta]$ and P.*

1.3 Minimum Penalty Temporal Path

In order to find an RSTP from s to g w.r.t $[\mathsf{t}_\alpha, \mathsf{t}_\beta]$ and P, we need to first determine whether there exists any RTP for the same parameters. If we are provided with an algorithm (see Lemma 1 below) that computes the minimum penalty temporal path (MPTP) from s to g w.r.t $[\mathsf{t}_\alpha, \mathsf{t}_\beta]$, then we can easily figure out whether there exists an RTP, or not. The actual algorithm is a straightforward adaptation of Dijkstra's algorithm. We omit the algorithm description, and the proof of Lemma 1 due to space constraints.

Lemma 1. *There exists an algorithm that can find the minimum penalty s-g temporal path with respect to $[\mathsf{t}_\alpha, \mathsf{t}_\beta]$, or reports that no such path exists. The complexity is $O(m \log m + m\Delta_{in})$.*

1.4 Organization of the Paper

The rest of the paper is organized as follows. In Sect. 2, we present three pseudo-polynomial exact algorithms. In Sect. 3, we present an A^* algorithm. In Sect. 4, we present an FPTAS. Simulation results of the exact algorithms are presented in Sect. 5. Finally, we conclude the paper in Sect. 6.

2 Exact Algorithms Using Dynamic Programming

In this section, we present several (pseudo-polynomial) algorithms based on dynamic programming. In Sect. 2.1, we first present two algorithms whose complexities are independent of the query time span i.e., t_α and t_β. In Sect. 2.2, we present an algorithm whose complexity depends on the query time span.

2.1 Query Time Span Independent Algorithms

Our approach is to find the minimum penalty by which one can reach a vertex v from s, using a temporal path of length ℓ that arrives at v at time t, where $t \in [\mathsf{t}_\alpha, \mathsf{t}_\beta]$. We incrementally check this for every $\ell \in [0, \mathsf{B_U}]$, and for every vertex v. Once we reach g via a path having penalty at most P, we terminate. Following are important properties of RSTP and MPTP.

Algorithm 1.

procedure DYNAMICLENGTH($\mathsf{s}, \mathsf{g}, \mathsf{P}, \mathsf{B_U}, \mathsf{t}_\alpha, \mathsf{t}_\beta, len, pen, start, end$)
  $open(0) \leftarrow \{\mathsf{s}\}$; $P(\mathsf{s}, 0, \mathsf{t}_\alpha) \leftarrow 0$; $E(\mathsf{s}, 0, \mathsf{t}_\alpha) \leftarrow \varnothing$; add t_α to $\mathcal{T}(\mathsf{s}, 0)$;
  for ($\ell \leftarrow 0; \ell \leq \mathsf{B_U}; \ell \leftarrow \ell + 1$) **do**
    if ($\mathsf{g} \in open(\ell)$) **then return** ℓ;
    end if
    for (every vertex $u \in open(\ell)$ and every edge $e^+ = (u, v) \in \delta^+(u)$) **do**
      let $\ell^+ = \ell + len(e^+)$, $t^+ = end(e^+)$;
      if ($\ell^+ \leq \mathsf{B_U}$ and $t^+ \leq \mathsf{t}_\beta$) **then**
        for (every t in $\mathcal{T}(u, \ell)$) **do**
          let $p^+ = P(u, \ell, t) + pen(e^+)$;
          if ($start(e^+) \geq t$ and $p^+ \leq \mathsf{P}$) **then**
            if ($v \notin open(\ell^+)$) **then** add v to $open(\ell^+)$;
            end if
            if ($t^+ \notin \mathcal{T}(v, \ell^+)$) **then** $P(v, \ell^+, t^+) \leftarrow p^+$; $E(v, \ell^+, t^+) \leftarrow \langle e^+, t \rangle$; add t^+ to $\mathcal{T}(v, \ell^+)$;
            else if ($p^+ < P(v, \ell^+, t^+)$) **then**
              $P(v, \ell^+, t^+) \leftarrow p^+$; $E(v, \ell^+, t^+) \leftarrow \langle e^+, t \rangle$;
            end if
          end if
        end for
      end if
    end for
  end for
  return ∞;
end procedure

Observation 1. *For a vertex v, let π_1 and π_2 be two s to v temporal paths having same length and arriving at v at the same time. If $pen(\pi_1) < pen(\pi_2)$, and an* s-g *RSTP contains π_2, then π_2 can be replaced by π_1 in the RSTP.*

Observation 2. *For any vertex $v \neq \mathsf{s}$, let $\pi = \langle e_1, e_2, \ldots, e_k \rangle$ be an s to v minimum penalty temporal path that arrives at $v = tgt(e_k)$ at time $end(e_k)$. Then, π is a minimum penalty temporal path that arrives at $tgt(e_{k-1})$ at time $end(e_{k-1}) \leq start(e_k)$, and has length $len(\pi) - len(e_k)$.*

Correctness and Complexity. Algorithm 1 presents a detailed implementation of the above observations. Here, $open(\ell)$ is the set of vertices v, such that an RTP from s to v having length ℓ has been discovered. Denote by $P(v, \ell, t)$, the minimum penalty among all s to v RTPs that has length ℓ and arrives at v at time t. Also, $E(v, \ell, t)$ maintains the tuple $\langle e, t' \rangle$, where $e = (u, v)$ is the incoming edge of v on the path defined by $P(v, \ell, t)$, and t' is the arrival time of this path at u. Further, $\mathcal{T}(v, \ell)$ is the set of distinct arrival times t, such that an s-v RTP, having length ℓ, and arriving at v at time t has been discovered.

The correctness follows from Observations 1, 2, and 3.

Observation 3. *The length of an RSTP from s to g w.r.t timespan $[\mathsf{t}_\alpha, \mathsf{t}_\beta]$ and penalty P is $\min\{\ell \mid P(\mathsf{g}, \ell, t) \leq \mathsf{P}$ and $t \in [\mathsf{t}_\alpha, \mathsf{t}_\beta]\}$.*

We maintain each $\mathcal{T}(v,\ell)$ as a bit-array of length at most $|\delta^-(v)|$, where the ith index in the array corresponds to the ith smallest edges in $\delta^-(v)$. Recall that the graph has been preprocessed so that in $O(1)$ time, we can find the rank of any edge $e \in \delta^-(v)$ among all edges in $\delta^-(v)$.If the ith bit is set to 1, then an s-v RTP that uses an ith smallest edge in $\delta^-(v)$ has been found, otherwise not. Likewise, for any edge $e = (u, v)$, the entry $P(v, \ell, t)$, where $t = end(e)$, can also be accessed (and updated) in constant time.

Recall that we assume that there is an RTP w.r.t P and $[\mathsf{t}_\alpha, \mathsf{t}_\beta]$. Therefore, line 3 is executed exactly $\mathsf{L_{opt}}$ times, and for each iteration, line 6 is executed at most m times. Since, the size of $\mathcal{T}(u,\ell)$ is at most $|\delta^-(u)|$, the time required by Algorithm 1 can be bounded by $O(m\Delta_{in}\mathsf{L_{opt}})$.

For every $v \in \mathcal{V}$, $\ell \in [0, \mathsf{B_U}]$, and $t \in \{end(e) \mid e \in \delta^-(v)\}$, the data-structures $P(v, \ell, t)$ requires $O(m\mathsf{B_U})$ space. Likewise, maintaining all $E(v, \ell, t)$ and $\mathcal{T}(v, \ell)$ structures requires $O(m\mathsf{B_U})$ space. Maintaining the structure to find out the rank of any edge e in $\delta^-(v)$ requires $O(m)$ space. Therefore, total space required is bounded by $O(m\mathsf{B_U})$.

Constructing the Path. The RSTP can be constructed easily using the information in $E(\cdot,\cdot,\cdot)$ as follows. Let the s-g RSTP be $\langle v_1, e_1, v_2, \ldots, e_{k-1}, v_k\rangle$, where $v_1 = \mathsf{s}$, $v_k = \mathsf{g}$, and for $1 \le i < k$, $v_i = src(e_i)$ and $v_{i+1} = tgt(e_i)$. Furthermore, let the length (resp. arrival time) of the s-v_i subpath be ℓ_i (resp. t_i). Then, $t_1 = \mathsf{t}_\alpha$, and $\ell_1 = 0$. For $1 < i \le k$, $t_i = end(e_{i-1})$, $\ell_i = \ell_{i-1} + len(e_{i-1})$, where $\ell_k = \mathsf{L_{opt}}$. Clearly, $\varnothing \in E(v_1, 0, \mathsf{t}_\alpha)$, and for every $1 < i \le k$, $\langle e_{i-1}, t_{i-1}\rangle \in E(v_i, \ell_i, t_i)$. Using this information, we can easily construct the s-g RSTP, say π^*, in $O(|\pi^*|)$ time, where $|\pi^*| < n$ denotes the number of edges in π^*.

An alternate algorithm is based on the following somewhat orthogonal approach to Algorithm 1. Constrained by space, we provide a sketch of the algorithm, and refer the reader to [8] for a penalty bounded algorithm for the RSP problem.

We maintain $L(v, p, t)$ i.e., the length of the minimum length s to v temporal path reaching v at time t with penalty p. For every $p \in [0, \mathsf{P}]$, $v \in V$, and $t \in \{end(e) \mid e \in \delta^-(v)\} \cap [\mathsf{t}_\alpha, \mathsf{t}_\beta]$, we maintain and update $L(v, p, t)$ according to Observations 4 and 5. Finally, we return the optimal length based on Observation 6. The correctness claim is immediate. As in the case of Algorithm 1, the path can also be constructed by maintaining additional information. Space and time complexity of $O(m\mathsf{P})$ and $O(m\Delta_{in}\mathsf{P})$ respectively can be derived based on the arguments in complexity analysis of Algorithm 1.

Observation 4. *For a vertex v, let π_1 and π_2 be two s to v temporal paths having same penalty and arriving at v at the same time. If $len(\pi_1) < len(\pi_2)$, then any RSTP from s to g does not contain the path π_2.*

Observation 5. *For any vertex $v \neq \mathsf{s}$, let $\pi = \langle e_1, e_2, \ldots, e_k\rangle$ be an s to v minimum length temporal path that arrives at $v = tgt(e_k)$ at time $end(e_k)$. Then, π is a minimum length temporal path that arrives at $tgt(e_{k-1})$ at time $end(e_{k-1}) \le start(e_k)$, and has penalty $pen(\pi) - pen(e_k)$.*

Observation 6. *The length of an RSTP from s to g w.r.t $[\mathsf{t}_\alpha, \mathsf{t}_\beta]$ and penalty P is $\min\{L(\mathsf{g}, p, t) \mid p \in [0, \mathsf{P}]$ and $t \in [\mathsf{t}_\alpha, \mathsf{t}_\beta]\}$.*

Algorithm 2.

procedure TIMESPAN($\mathsf{s}, \mathsf{g}, \mathsf{P}, \mathsf{t}_\alpha, \mathsf{t}_\beta, len, pen, start, end$)
  $L(v, p, t) \leftarrow \infty$ for every $v \in \mathcal{V}, 1 \leq p \leq \mathsf{P}$ and $t \in [\mathsf{t}_\alpha, \mathsf{t}_\beta]$; $E(\mathsf{s}, 0, \mathsf{t}_\alpha) \leftarrow \varnothing$;
  let $\mathcal{E}'$ be the set of edges e in $\mathcal{E}$, such that $start(e) \geq \mathsf{t}_\alpha$ and $end(e) \leq \mathsf{t}_\beta$;
  sort the edges e of $\mathcal{E}'$ in ascending order based on $end(e)$;
  for (every edge $e = (u, v) \in \mathcal{E}'$) **do**
    for (every entry $L(u, p', t')$ of u with $t' \leq start(e)$) **do**
      let $\ell = L(u, p', t') + len(e), p = p' + pen(e)$;
      if ($p \leq \mathsf{P}$ and $\ell < L(v, p, end(e))$) **then**
        $L(v, p, end(e)) \leftarrow \ell$, $E(v, p, end(e)) \leftarrow \langle e, t' \rangle$;
      end if
    end for
  end for
  return the minimum value in $\{L(\mathsf{g}, p, t) \mid \mathsf{t}_\alpha \leq t \leq \mathsf{t}_\beta$ and $0 \leq p \leq \mathsf{P}\}$;
end procedure

By Lemma 1, verifying the existence of an RTP is achieved in $O(m \log m + m\Delta_{in})$ time. Following the discussions in this section, we arrive at the following theorem.

Theorem 1. *By maintaining data-structures occupying $O(m\mathsf{B}_\mathsf{U})$ and $O(m\mathsf{P})$ space, we can find* s-g *RSTP w.r.t $[\mathsf{t}_\alpha, \mathsf{t}_\beta]$ and P in $O(m \log m + m\Delta_{in}\mathsf{L}_\mathsf{opt})$ and $O(m \log m + m\Delta_{in}\mathsf{P})$ time respectively.*

2.2 Query Time Span Dependent Algorithm

In the previous section, we introduced two exact algorithms with complexities $O(m \log m + m\Delta_{in}\mathsf{P})$ and $O(m \log m + m\Delta_{in}\mathsf{L}_\mathsf{opt})$. Larger values of Δ_{in} and L_opt can be a bottleneck for these algorithms, resulting in inefficient query time. In this section, we tackle those scenarios and propose an $O(m\mathsf{P}(\mathsf{t}_\beta - \mathsf{t}_\alpha))$ time algorithm. Following is yet another useful property of RSTP.

Observation 7. *For a vertex v, let π_1 and π_2 be two s to v temporal paths having equal length and penalty. If $end(\pi_1) < end(\pi_2)$ and an* s-g *RSTP contains π_2, then π_2 can be replaced by π_1 in the RSTP.*

Correctness and Complexity. We use the greedy properties of Observations 4 and 7 at every intermediate vertex v while searching an s-g RSTP. This way the number of useful paths reaching v at a time-stamp t is bounded by P. Let $L(v, p, t)$ be the length of the shortest s to v temporal path reaching v at time t with penalty p. The idea is to store $L(v, p, t)$ for every value of $p \in [0, \mathsf{P}]$, and update it based on Observation 4. By maintaining additional information, as described in the Algorithm 1, the RSTP can be constructed.

Correctness follows from Observations 4, 5, 6, and 7.

The total size of $L(u, \cdot, \cdot)$ can be bounded by $O(\mathsf{P}(\mathsf{t}_\beta - \mathsf{t}_\alpha))$. Therefore, total space required over all vertices is bounded by $O(n\mathsf{P}(\mathsf{t}_\beta - \mathsf{t}_\alpha))$. Since we only

consider the edges with start time and arrival time falling within $[\mathsf{t}_\alpha, \mathsf{t}_\beta]$, we can sort the edges in step 4 by using bucket sort with $(\mathsf{t}_\beta - \mathsf{t}_\alpha)$ buckets. This step can then be performed in $O(m+(\mathsf{t}_\beta-\mathsf{t}_\alpha))$ time. The size of $\mathcal{E}'$ is at most m; thus steps 5–12 require time $O(m\mathsf{P}(\mathsf{t}_\beta - \mathsf{t}_\alpha))$. Finally, returning the optimal length takes $O(\mathsf{P}(\mathsf{t}_\beta - \mathsf{t}_\alpha))$ time, and we have the following theorem.

Theorem 2. *By maintaining data-structures occupying $O(n\mathsf{P}(\mathsf{t}_\beta - \mathsf{t}_\alpha))$ space, we can find* s-g *RSTP w.r.t $[\mathsf{t}_\alpha, \mathsf{t}_\beta]$ and P in $O(m\mathsf{P}(\mathsf{t}_\beta - \mathsf{t}_\alpha))$ time.*

3 A^* Algorithm

We start with a few notations. Let π^* be an s-g RSTP. For a vertex v on π^*, let $g_{len}(\pi^*, v)$ be the length of the s-v subpath of π^*, and $h_{len}(\pi^*, v)$ be the length of the v-g subpath of π^*. Likewise, we define $g_{pen}(\pi^*, v)$ and $h_{pen}(\pi^*, v)$. Clearly, $g_{len}(\pi^*, v)+h_{len}(\pi^*, v) = len(\pi^*) = \mathsf{L}_{\mathsf{opt}}$ and $g_{pen}(\pi^*, v)+h_{pen}(\pi^*, v) = pen(\pi^*)$. Further, if a vertex v' is not on π^*, then $g_{len}(\pi^*, v') = g_{pen}(\pi^*, v') = \infty$ and $h_{len}(\pi^*, v') = h_{pen}(\pi^*, v') = \infty$. For a vertex w (not necessarily on an s-g RSTP),

- $h_{len}(w)$ is an underestimate of the length of the w-g subpath (if any) of π^* i.e., $h_{len}(w) \leq h_{len}(\pi^*, w)$. Likewise, we define $h_{pen}(w)$. Since, every edge has positive length and penalty, we let $h_{len}(w) \geq 0$ and $h_{pen}(w) \geq 0$.
- $open(w)$ is a set of *non-dominated* tuples $x = \langle \ell, p, t, y, e\rangle$, where ℓ and p are the length and penalty of an s-w path, e is the incoming edge of w on this path, and y is the tuple which led to the formation of x. If $w = \mathsf{s}$ then $t = \mathsf{t}_\alpha$, otherwise $t = end(e)$. For any vertex v, a tuple $x = \langle \ell, p, t, y, e\rangle$ in $open(v)$ is *said to be non-dominated* if there is no other $x' = \langle \ell', p', t', y', e'\rangle$ in $open(v)$ such that $t' \leq t$, $\ell' \leq \ell$, and $p' \leq p$. For $y = \langle \ell', p', t', y', e'\rangle \in open(u)$, and an edge $e = (u, w)$, we modify $open(w)$ from y as follows (see lines 9–22 in Algorithm 3). Let $x = \langle \ell' + len(e), p' + pen(e), end(e), y, e\rangle$. If $p' + pen(e) + h_{pen}(w) \leq \mathsf{P}$, $t' \leq start(e)$, $end(e) \leq \mathsf{t}_\beta$, and there is no previously removed tuple from $open(w)$ that dominates x, merge x with the tuples at $open(w)$, keeping only non-dominated ones in $open(w)$. Initially, $open(\mathsf{s})$ contains $\langle 0, 0, \mathsf{t}_\alpha, \varnothing, \varnothing\rangle$, and every other $open(\cdot)$ is empty.
- $closed(w)$ is a set of tuples that have been selected (and removed) from $open(w)$. Initially, every $closed(\cdot)$ is empty. We select a tuple from $open(w)$ as follows (see lines 5–8 in Algorithm 3). If $open(v)$ is not empty, then $openVertices$ contains the vertex v keyed by $\ell_v + h_{len}(v)$, where ℓ_v is the length of the minimum length tuple in $open(v)$. We first remove the vertex w from $openVertices$ with the minimum key, and then remove the minimum length tuple from $open(w)$.

Constructing the Path. The previous tuple y and incoming edge e is maintained in each tuple x so that we can efficiently construct the s-g RSTP π^*. Let, $\pi^* = \langle v_1, e_1, v_2, \ldots, e_{k-1}, v_k\rangle$, where $v_1 = \mathsf{s}$ and $v_k = \mathsf{g}$. For each vertex v_i, $1 \leq i \leq k$, there is a tuple $x_i = \langle \ell_i, p_i, t_i, x_{i-1}, e_{i-1}\rangle$ in $closed(v_i)$. Here, $\ell_1 = p_1 = 0$, $t_1 = \mathsf{t}_\alpha$, and $x_0 = e_0 = \varnothing$. For each x_i, by using the incoming edge e_{i-1} and the previous tuple x_{i-1}, we can construct π^*.

Algorithm 3.

procedure $A^*(\mathsf{s}, \mathsf{g}, \mathsf{t}_\alpha, \mathsf{t}_\beta, \mathsf{P}, len, pen, start, end, h_{len}, h_{pen})$
  $closed(v)$, $open(v) \leftarrow \varnothing$, for all $v \in \mathcal{V}$;
  add $\langle 0, 0, \mathsf{t}_\alpha, \varnothing, \varnothing \rangle$ to $open(\mathsf{s})$; add s to $openVertices$ keyed by $h_{len}(\mathsf{s})$;
  while ($openVertices$ is not empty) **do**
    remove the vertex u from $openVertices$ with minimum key, with ties broken arbitrarily, but always in favour of vertex g; remove the minimum length tuple $x^* = \langle \ell, p, t, x, e \rangle$ from $open(u)$; add x^* to $closed(u)$;
    if $open(u)$ is not empty, then add u to $openVertices$ keyed by $\ell_{min} + h_{len}(u)$, where $\ell_{min} = \min\{\ell' \mid \langle \ell', p', t', x', e' \rangle \in open(u)\}$;
    if ($u = \mathsf{g}$) **then** construct and return the s-g path having length ℓ and penalty p, which arrives at g at $end(e)$;
    end if
    for (each edge $e^+ = (u, v) \in \delta^+(u)$) **do**
      let $p^+ = p + pen(e^+)$, $\ell^+ = \ell + len(e^+)$; $x' = \langle \ell^+, p^+, end(e^+), x^*, e^+ \rangle$;
      if ($p^+ + h_{pen}(v) \leq \mathsf{P}$, $t \leq start(e^+)$ and $end(e^+) \leq \mathsf{t}_\beta$) **then**
        if (there is no tuple in $closed(v)$ that dominates x') **then**
          if (there is no tuple in $open(v)$) **then** add x' to $open(v)$;
            add v to $openVertices$ keyed by $\ell^+ + h_{len}(v)$;
          else if (no triplet in $open(v)$ dominates x') **then**
            remove the triplets from $open(v)$ that are dominated by x'; add x' to $open(v)$; update the key of v in $openVertices$ with $\ell^+_{min} + h_{len}(v)$, where $\ell^+_{min} = \min\{\ell' \mid \langle \ell', p', t', x', e' \rangle \in open(v)\}$;
          end if
        end if
      end if
    end for
  end while
  return $\varnothing$;
end procedure

Remark 1. *We can also prune paths using the transit time information. For any vertex u, let $e = (u, v) \in \delta^+(u)$, and $h_{transit}(v)$ be an underestimate of the minimum total transit time of v to g sub-path of π^*. If $end(e) + h_{transit}(v) > \mathsf{t}_\beta$, then we can ignore the edge e for a tuple selected from open(u). If the transit time of an edge is same as its length, then $h_{transit}$ is same as h_{len}.*

3.1 Correctness and Complexity

In Algorithm 3, by taking temporal information into account, we extend the A^* algorithm of Li et al. [11] for the RSP problem. According to the definition of h_{len} and h_{pen}, both are *admissible* i.e., for any vertex v and any s-g RSTP π^*, we have $h_{len}(v) \leq h_{len}(\pi^*, v)$ and $h_{pen}(v) \leq h_{pen}(\pi^*, v)$. In Lemma 2, we show that given admissible heuristics, Algorithm 3 finds an RSTP w.r.t P and $[\mathsf{t}_\alpha, \mathsf{t}_\beta]$. Due to space constraint, we omit the proofs in this section, which are based on the A^* algorithm for the shortest path problem [7].

Lemma 2. *If h_{len} and h_{pen} are both admissible, then Algorithm 3 finds an* s-g *RSTP w.r.t* $[\mathsf{t}_\alpha, \mathsf{t}_\beta]$ *and* P.

Consistency Assumption. For any two vertices u and v, denote by $L(u, v)$, the length of a minimum length path (not necessarily temporal or restricted) from u to v. The consistency assumption is

$$h_{len}(u) \leq L(u, v) + h_{len}(v)$$

The following theorem summarizes our main result in this section.

Theorem 3. *Suppose h_{len} is consistent and h_{pen} is admissible. Then, Algorithm 3 finds an* s-g *RSTP in time $O(m\bar{L}\Delta_{in} + m\bar{L}\log(n\bar{L}\Delta_{in}))$, where $\bar{L} = \mathsf{L}_{\mathsf{opt}} + \max_{e\in\mathcal{E}}\{len(e)\}$.*

3.2 Obtaining Estimates

For any two vertices w and w', denote by $L(w, w')$, (resp. $P(w, w')$) the length (resp. penalty) of a minimum length (resp. penalty) path from w to w'. For every vertex v, $h_{len}(v) = L(v, \mathsf{g})$, and $h_{pen}(v) = P(v, \mathsf{g})$. Clearly, both $L(v, \mathsf{g})$ and $P(v, \mathsf{g})$ are admissible. Since, $L(u, v)+h_{len}(v)-h_{len}(u) = L(u, v)+L(v, \mathsf{g})-L(u, \mathsf{g}) \geq 0$, h_{len} (and likewise, h_{pen}) is consistent. Observe that $L(v, \mathsf{g})$ (resp. $P(v, \mathsf{g})$) in a graph is same as $L(\mathsf{g}, v)$ (resp. $P(\mathsf{g}, v)$ in the same graph, but with edge directions reversed. Therefore, we can obtain $L(v, \mathsf{g})$ and $P(v, \mathsf{g})$ in $O(m + n\log n)$ time by first reversing the edges of the graph, and then running Dijkstra's algorithm twice, once for edge lengths and then for edge penalties.

Note that the minimum length (resp. penalty) of a v-g temporal path w.r.t $[\mathsf{t}_\alpha, \mathsf{t}_\beta]$ also serves as an admissible $h_{len}(v)$ (resp. $h_{pen}(v)$) of any v-g RSTP. These again can be obtained using suitably modified forms (with the complexity remaining unchanged) of the algorithm in Lemma 1. Also note that the estimates using minimum length and minimum penalty temporal paths is at least as tight an estimate as those using Dijkstra's algorithm.

Since for both these heuristic measures, we need to find shortest (temporal) paths from every vertex v to g, the time required to find the underestimates can be very expensive. To this end, we adapt the landmark based approach in [6] for the shortest path problem to that for the RSTP problem.

Landmark Based Approach. A set of landmarks, denoted by $\mathcal{L}$, is a subset of the set of the vertices of the graph i.e., $\mathcal{L} \subseteq \mathcal{V}$. We show how to compute $h_{len}(v)$ for any vertex v; $h_{pen}(v)$ can be computed similarly. Let $L(v', v'', t_1, t_2)$ be the minimum length of a temporal path from v' to v'' that lies in the interval $[t_1, t_2]$. For any $w \in \mathcal{V}$, let $t^{w+}_{min} = \min\{start(e) \mid e \in \delta^+(w)\}$, $t^{w+}_{max} = \max\{start(e) \mid e \in \delta^+(w)\}$, and $t^{w-}_{max} = \max\{end(e) \mid e \in \delta^-(w)\}$. For $u \in \mathcal{L}$ and $v \in \mathcal{V}$, we have

$$L(v, \mathsf{g}) \geq L(v, u) - L(\mathsf{g}, u)$$
$$L(v, \mathsf{g}, t^{v+}_{min}, \mathsf{t}_\beta) \geq L(v, u, t^{v+}_{min}, t^{u-}_{max}) - L(\mathsf{g}, u, \mathsf{t}_\beta, t^{u-}_{max})$$

Algorithm 4.

procedure TEST($\mathsf{s}, \mathsf{g}, \mathsf{P}, \mathsf{B_L}, \mathsf{B_U}, \mathsf{t}_\alpha, \mathsf{t}_\beta, \epsilon, len, pen, start, end$)
  $\lambda \leftarrow \epsilon \times \mathsf{B_L}/n$;
  $len'(e) \leftarrow \lfloor len(e)/\lambda \rfloor + 1$, for every edge $e \in \mathcal{E}$;
  $B \leftarrow \lfloor \mathsf{B_U}/\lambda \rfloor + n$;
  return DYNAMICLENGTH($\mathsf{s}, \mathsf{g}, \mathsf{P}, B, \mathsf{t}_\alpha, \mathsf{t}_\beta, len', pen, start, end$);
end procedure

procedure APPX($\mathsf{s}, \mathsf{g}, \mathsf{P}, \mathsf{t}_\alpha, \mathsf{t}_\beta, \epsilon, len, pen, start$)
  compute $\mathsf{B_L}$ and $\mathsf{B_U}$ such that $\mathsf{B_U} < n\mathsf{B_L}$;
  while ($\mathsf{B_U} > 2\mathsf{B_L}$) **do**
    $B \leftarrow \sqrt{\mathsf{B_U} \cdot \mathsf{B_L}}$;
    if (TEST($\mathsf{s}, \mathsf{g}, \mathsf{P}, B, B, \mathsf{t}_\alpha, \mathsf{t}_\beta, 1, len, pen, start, end$) $= \infty$) **then** $\mathsf{B_L} \leftarrow B$
    else $\mathsf{B_U} \leftarrow B$
    end if
  end while
  return TEST($\mathsf{s}, \mathsf{g}, \mathsf{P}, \mathsf{B_L}, 2\mathsf{B_L}, \mathsf{t}_\alpha, \mathsf{t}_\beta, \epsilon, len, pen, start, end$);
end procedure

Note that both $L(v, \mathsf{g})$ and $L(v, \mathsf{g}, t^{v+}_{min}, \mathsf{t}_\beta)$ can be used as $h_{len}(v)$. For every $u \in \mathcal{L}$, and every $v \in \mathcal{V}$, we store the following: (i) $L(v, u)$, (ii) $L(v, u, t^{v+}_{min}, t^{u-}_{max})$, and (iii) $L(v, u, t^{v+}_{max}, t^{u-}_{max})$. Total space is bounded by $O(n|\mathcal{L}|)$.

If $\mathsf{t}_\beta \leq t^{\mathsf{g}+}_{max}$, then $L(\mathsf{g}, u, \mathsf{t}_\beta, t^{u-}_{max}) \leq L(\mathsf{g}, u, t^{\mathsf{g}+}_{max}, t^{u-}_{max})$; otherwise, there is no g-u temporal path in the time interval $[\mathsf{t}_\beta, t^{u-}_{max}]$. Let $\mathcal{L}' \subseteq \mathcal{L}$ be the set of vertices in the former case. For any vertex v, we obtain $h_{len}(v)$ as shown below.

$$h_{len}(v) = \max\{0, h'_{len}(v), \max_{u \in \mathcal{L}}\{L(v, u) - L(\mathsf{g}, u)\}\}, \text{ where}$$
$$h'_{len}(v) = \max_{u \in \mathcal{L}'}\{L(v, u, t^{v+}_{min}, t^{u-}_{max}) - L(\mathsf{g}, u, t^{\mathsf{g}+}_{max}, t^{u-}_{max})\}.$$

4 Approximation Algorithm

In this section, we present an FPTAS for the RSTP problem, which is based on the FPTAS for the RSP problem [4,13]. The idea is to scale down the edge lengths, and then use Algorithm 1 on this scaled down instance. Due to lack of space, we omit the proofs in this section, which are similar to that in [4,13].

Let $\mathbb{I}$ be an instance of the RSTP problem. Consider the $TEST$ procedure in Algorithm 4, which scales down both the edge lengths, and an appropriate upper-bound $\mathsf{B_U}$ by λ, and then calls Algorithm 1. Denote this scaled down instance by $\mathbb{I}_\lambda$. For the instance $\mathbb{I}_\lambda$, if there exists an s-g RTP, say π, having length at most the scaled down upper-bound B, then Algorithm 1 returns $len(\pi)$; otherwise, it returns ∞. The following lemma proves important in obtaining the FPTAS.

Lemma 3. *Suppose $TEST$ procedure in Algorithm 4 returns the length of a path π_f. Then, $len(\pi_f) < \mathsf{L_{opt}} + \epsilon \cdot \mathsf{B_L}$*

Clearly, the complexity of the $TEST$ procedure can be bounded by $O(nm\Delta_{in}(1+\frac{\mathsf{B_U}}{\epsilon\cdot\mathsf{B_L}}))$. Further, if $\mathsf{B_U}$ (and hence, $\mathsf{L_{opt}}$) is within a constant factor of $\mathsf{B_L}$, then $TEST$ is an FPTAS for the RSTP problem. The objective, therefore, is to obtain $\mathsf{B_L}$ and $\mathsf{B_U}$ such that $\mathsf{B_U}$ is within a constant factor of $\mathsf{B_L}$. To this end, as described in the following lemma, we first find $\mathsf{B_L}$ and $\mathsf{B_U}$ such that $\mathsf{B_U} < n\mathsf{B_L}$.

Lemma 4. *An upper bound* $\mathsf{B_U}$ *and a lower bound* $\mathsf{B_L}$ *of the length of a RSTP such that* $\mathsf{B_U} < n\mathsf{B_L}$ *can be found in* $O(m\log^2 m + m\Delta_{in}\log m)$ *time.*

Finally, if both $\mathsf{B_U}$ and $\mathsf{B_L}$ are taken in the logarithmic scale, then by using Lemmas 3 and 4, we arrive at the following theorem.

Theorem 4. *APPX procedure in Algorithm 4 is an FPTAS for Problem 1 having complexity* $O(m\Delta_{in}\log m + nm\Delta_{in}(\log\log n + 1/\epsilon))$.

5 Experimental Evaluation

Since, there is no previously known work on finding RSP in temporal graphs, we present a comparative analysis of our proposed exact algorithms by varying different parameters. For simplicity of notation, we denote the algorithms having complexity $O(m\log m + m\Delta_{in}\mathsf{L_{opt}})$, $O(m\log m + m\Delta_{in}\mathsf{P})$ and $O(m\mathsf{P}(\mathsf{t}_\beta - \mathsf{t}_\alpha))$ as $DLen$, $DPen$ and $DTime$ respectively. For Algorithm 3, we use $DijkA^*$, $TempA^*$ and ALT to denote the Dijkstra estimates, temporal estimates and landmark based estimates respectively.

5.1 Settings and Dataset

All experiments are performed using the JAVA programming language on a 2.6 Ghz Intel dual-core E5300 processor and 6 Gb RAM machine running LINUX.

We use the flight dataset (*flight*) located at Bureau of Transportation Statistics (http://www.transtats.bts.gov/). The origin (resp. destination) airport ID has been taken as edge source (resp. edge target). The departure (resp. arrival) time has been taken as the start (resp. end) time of an edge. We take the transit time of an edge as its length, and the distance traversed as the edge penalty.

We also use the well-known temporal datasets located at the Koblenz Large Network Collection (http://konect.uni-koblenz.de/). Specifically, we choose the Enron communication network database (*enron*), YouTube social network database (*youtube*), DBLP co-author index database (*dblp*), and the Wiktionary English authorship database (*wiktionary*). For each dataset, the first two columns are taken as the source and target of an edge, the third column is edge length, and the fourth column is the start time of an edge. The arrival time of an edge is assumed to be its start time plus its length. The penalty of an edge is a randomly assigned integer in the interval [1, 100].

Table 1 presents some statistics of the datasets. Here, **Disksize** and **Loadtime** is the space occupied by the dataset on the disk, and the time required to load the graph onto the memory. The latter includes the time required to create

Table 1. Dataset statistics

Dataset	Disksize	Loadtime	n	m	δ_{avg}	Δ_{in}	$\mathcal{T}$
flight	11.2 Mb	3247 ms	6084	456964	75	29172	2642
enron	25.9 Mb	4693 ms	87273	1148072	13	6166	220364
youtube	257.1 Mb	32994 ms	3223589	9375374	2	61556	203
dblp	518.7 Mb	62880 ms	1314050	18986618	14	2630	72
wiktionary	215.9 Mb	33957 ms	2133892	8998641	4	34536	8459765

the reverse graph so that we can compute minimum length (penalty) heuristics, or minimum temporal length (penalty) heuristics, as discussed in Sect. 3.2. The average degree has been denoted by δ_{avg}, and $\mathcal{T}$ is the number of distinct arrival times. In every dataset barring the *flight* dataset, every edge has unit length (or, transit time); therefore, $\mathcal{T}$ is also the set of distinct start times.

5.2 Algorithms on *flight* dataset

In this set of experiments, we use the flight dataset to evaluate the effect of varying query time span and that of varying P on running time. For Algorithm 1, the upper-bound $\mathsf{B_U}$ on the length of the RSTP, was taken to be length of the minimum penalty temporal path from s to g. Each experiment was run for 100 randomly selected s-g pairs, and the average time was taken.

Varying Query Time Span. In Fig. 1(a), we show the average running times for time spans $\mathsf{t}_\beta - \mathsf{t}_\alpha = 80, 100, 120, 140, 160, 180, 200$ and fixed penalty restriction $\mathsf{P} = 150$. In terms of performance, Algorithm *DTime* shows a linear progression with increasing time span. Running time of Algorithms *DLen* and *DPen* stay stable, which supports their theoretical time bounds. Also, as expected from the theoretical bounds, the running time of *DTime* is lower than both *DLen* and *DPen* for lower time span values. Running time for the heuristic estimate based algorithms are significantly lower.

Varying Query Penalty. In Fig. 1(b), we use query penalties $\mathsf{P} = 160, 180, 200, 220, 240, 260, 280$ and fixed time span $\mathsf{t}_\beta - \mathsf{t}_\alpha = 60$. Again only Algorithm *DTime* is affected and shows significant increase in running time, while Algorithms *DLen* and *DPen* remain stable. Running time for the heuristic estimate based algorithms are negligible compared to the other exact algorithms.

The running time of *DLen* can be attributed to its theoretical complexity; however, *DPen* algorithm shows an anomaly, as it should linearly scale with P. A possible explanation is that for most values of $p \in [1, \mathsf{P}]$, $open(p)$ is empty or contains very few vertices. Therefore, the running time becomes negligible when compared to that of *DTime*, in which every edge with arrival time t is scanned once for every $t \in [\mathsf{t}_\alpha, \mathsf{t}_\beta]$ and every $p \in [0, \mathsf{P}]$.

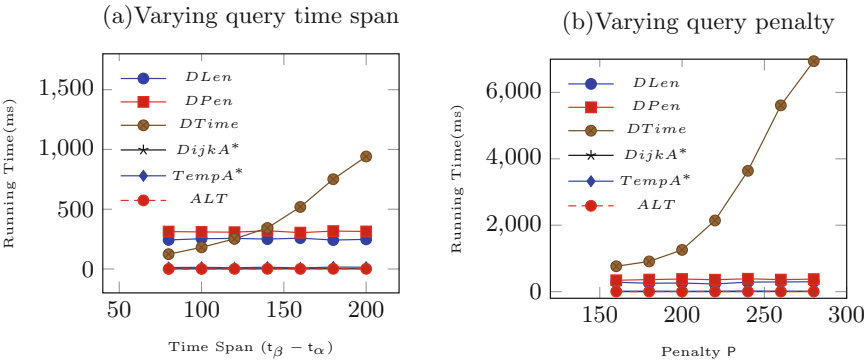

Fig. 1. Running time for varying query time span and penalty restriction.

5.3 A^* Algorithm on KONECT Dataset

The space occupied by the algorithms in Sect. 2 can be very high depending on n, Δ_{in}, P, $\mathsf{B_U}$, and $(\mathsf{t}_\beta - \mathsf{t}_\alpha)$. For e.g., the *dblp* dataset has 1314050 vertices, and average degree is 14. Therefore, the space occupied by only $P(\cdot,\cdot,\cdot)$ structure in Algorithm 1, even for $\mathsf{B_U} = 10$ can be as bad as 5.5 Gb. Additional space required by other data structures, and the graph itself, makes these algorithms impractical for large datasets. In fact, we observed that these algorithms suffer from heap size overflow exception for every KONECT dataset. The strength of the A^* algorithm lies in the fact that it prunes those paths which can never lead to an s-g RSTP not only based on $[\mathsf{t}_\alpha, \mathsf{t}_\beta]$, but also on the dominance criteria among tuples, and the penalty underestimate $h_{pen}(v)$. Since, in these data sets, the edge transit time is also its length, we also prune paths based on Remark 1.

As expected using a tighter heuristic of minimum temporal lengths (penalty), the path finding time (i.e., **Total - Est.** time) is smaller than that using minimum length (penalty) estimates. However, note that in Table 2, the bulk of time in computing the s-g RSTP (or determining that there is none) is due to computing the estimates. Therefore, the objective is to reduce this estimate computation time at the expense of the path computing time. This is achieved using estimates based on landmarks, which along with the corresponding values of $L(\cdot,\cdot)$, $L(\cdot,\cdot,\cdot,\cdot)$, $P(\cdot,\cdot)$, and $P(\cdot,\cdot,\cdot,\cdot)$ is precomputed and stored. Therefore, we only need to compute $h_{len}(v)$ and $h_{pen}(v)$ based on these stored values, as discussed in Sect. 3.2, whenever a tuple is added to $open(v)$ for the first time. It can be seen that in most cases, the landmark based approach is much faster.

We chose the landmarks randomly, and the size of the landmark set $\mathcal{L}$ is fixed at 10. Furthermore, $\mathsf{t}_\alpha = \min\{start(e) \mid e \in \delta^+(\mathsf{s})\}$ and $\mathsf{t}_\beta = \max\{end(e) \mid e \in \delta^-(\mathsf{g})\}$. Each experiment was repeated for 100 s-g pairs, and then the average was taken. We observed that on increasing the size of $\mathcal{L}$, the running time increases, and in fact in few cases, even for an empty $\mathcal{L}$ (i.e., with no heuristic information), Algorithm 3 performs better than using minimum (temporal) length and penalty

Table 2. Running time of Algorithm 3 on the KONECT datasets for various P values, and using various heuristic estimates. Time required to compute the minimum length (penalty) estimates are listed under the **Est.** sub-column of the **Dijkstra** column. Likewise, the time required to compute the minimum temporal length (penalty) estimates are listed under the **Est.** sub-column of the **Temporal** column. The total time required to find the paths are listed under the respective **Total** columns. Time required to find the paths by using *landmarks* are listed under the **ALT** column.

P	*enron*					*youtube*				
	Dijkstra		**Temporal**		**ALT**	**Dijkstra**		**Temporal**		**ALT**
	Est.	**Total**	**Est.**	**Total**		**Est.**	**Total**	**Est.**	**Total**	
200	28.79	31.13	87.09	90.67	56.02	518.54	528.41	506.50	523.19	23.99
400	19.39	29.70	57.58	60.45	60.37	463.12	477.62	455.69	468.52	35.87
600	27.22	49.54	56.85	64.03	44.63	458.16	468.80	403.73	416.09	13.72
800	23.16	23.62	52.60	52.93	49.01	345.66	355.79	374.92	388.84	15.17
1000	26.70	60.85	63.59	76.08	43.24	481.94	494.27	388.85	402.06	15.77
Avg	25.06	38.97	63.55	68.84	50.66	453.49	464.99	425.94	439.74	20.91

P	*dblp*					*wiktionary*				
	Dijkstra		**Temporal**		**ALT**	**Dijkstra**		**Temporal**		**ALT**
	Est.	**Total**	**Est.**	**Total**		**Est.**	**Total**	**Est.**	**Total**	
200	3111.96	3128.84	2052.05	2057.07	72.47	47.66	55.31	28.04	35.33	11.91
400	3572.06	4120.01	3063.62	3097.18	2057.39	67.71	102.2	88.98	120.26	190.13
600	3374.16	4094.12	1994.92	2016.98	1515.96	40.39	48.19	26.27	35.51	13.22
800	3735.48	4205.83	1945.71	2089.19	1737.14	65.54	98.05	84.91	115.92	43.30
1000	3358.02	4552.57	1898.25	1908.88	1727.06	53.05	61.61	32.41	40.91	12.06
Avg	3430.34	4020.28	2190.91	2233.86	1422.01	54.87	73.08	52.13	69.59	54.13

estimates. Also for the given datasets, we observed that the choice of $\mathtt{t}_\alpha$ and $\mathtt{t}_\beta$ does not affect the running time of Algorithm 3 to a great extent.

6 Conclusion

In this paper, we extend the restricted shortest path problem in static networks to that for temporal networks. We present various exact algorithms, discuss their complexities, and experimentally evaluate them. Furthermore, using one of these exact algorithms, we show how to obtain an FPTAS. For the A^*-like algorithm, we show how to obtain various underestimates, one of which uses landmark based techniques. However, the selection of the landmarks has been done randomly. As a future effort, we would like to extend this by adapting "more intelligent" landmark selection strategies. Also, we have not discussed lower bounds of the complexities of the problem. It would be interesting to study whether algorithms with better complexities exist for the RSTP problem.

References

1. Cai, X., Kloks, T., Wong, C.K.: Time-varying shortest path problems with constraints. Networks **29**(3), 141–150 (1997)
2. Carlyle, W.M., Royset, J.O., Kevin Wood, R.: Lagrangian relaxation and enumeration for solving constrained shortest-path problems. Networks **52**(4), 256–270 (2008)
3. Day, P.R., Ryan, D.M.: Flight attendant rostering for short-haul airline operations. Oper. Res. **45**(5), 649–661 (1997)
4. Ergun, F., Sinha, R., Zhang, L.: An improved FPTAS for restricted shortest path. Inf. Process. Lett. **83**(5), 287–291 (2002)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co., New York (1979)
6. Goldberg, A.V., Harrelson, C.: Computing the shortest path: a search meets graph theory. In: Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, pp. 156–165. Society for Industrial and Applied Mathematics, Philadelphia (2005)
7. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968)
8. Hassin, R.: Approximation schemes for the restricted shortest path problem. Math. Oper. Res. **17**(1), 36–42 (1992)
9. Kempe, D., Kleinberg, J.M., Demers, A.J.: Spatial gossip and resource location protocols. In: Proceedings on 33rd Annual ACM Symposium on Theory of Computing, Heraklion, Crete, Greece, 6–8 July 2001, pp. 163–172 (2001)
10. Korkmaz, T., Krunz, M.: Multi-constrained optimal path selection. In: Proceedings IEEE INFOCOM 2001, The Conference on Computer Communications, Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies, Twenty Years into the Communications Odyssey, Anchorage, Alaska, USA, 22–26 April 2001, pp. 834–843 (2001)
11. Li, Y., Harms, J., Holte, R.: Fast exact multiconstraint shortest path algorithms. In: IEEE International Conference on Communications, ICC 2007, June 2007, pp. 123–130 (2007)
12. Lorenz, D., Orda, A.: QoS routing in networks with uncertain parameters. IEEE/ACM Trans. Netw. **6**(6), 768–778 (1998)
13. Lorenz, D.H., Raz, D.: A simple efficient approximation scheme for the restricted shortest path problem. Oper. Res. Lett. **28**(5), 213–219 (2001)
14. Mertzios, G.B., Michail, O., Chatzigiannakis, I., Spirakis, P.G.: Temporal network optimization subject to connectivity constraints. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part II. LNCS, vol. 7966, pp. 657–668. Springer, Heidelberg (2013)
15. Michail, O., Spirakis, P.G.: Traveling salesman problems in temporal graphs. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014, Part II. LNCS, vol. 8635, pp. 553–564. Springer, Heidelberg (2014)
16. Nachtigall, K.: Time depending shortest-path problems with applications to railway networks. Eur. J. Oper. Res. **83**(1), 154–166 (1995)
17. Van Mieghem, P., Kuipers, F.: Concepts of exact QoS routing algorithms. IEEE/ACM Trans. Netw. **12**(5), 851–864 (2004)
18. Wu, H., Cheng, J., Huang, S., Ke, Y., Lu, Y., Xu, Y.: Path problems in temporal graphs. Proc. VLDB Endow. **7**(9), 721–732 (2014)

Springer

http://www.springer.com/978-3-319-22848-8

Database and Expert Systems Applications
26th International Conference, DEXA 2015, Valencia, Spain, September 1-4, 2015, Proceedings, Part I
Chen, Q.; Hameurlain, A.; Toumani, F.; Wagner, R.; Decker, H. (Eds.)

2015, XXXI, 578 p. 175 illus., Softcover
ISBN: 978-3-319-22848-8