Space and time tradeoffs for the k shortest simple paths problem

Ali Al Zoobi, David Coudert and Nicolas Nisse
Université Côte d’Azur, Inria, CNRS, I3S

June 18, 2020
1 Introduction

2 k shortest simple paths problem

3 k shortest simple paths algorithms:
   - Yen’s algorithm
   - Kurz and Mutzel’s algorithm

4 Our contribution:
   - speeding up Kurz and Mutzel’s algorithm
   - space time tradeoff
Motivation

A shortest path is not enough!

- A shortest path may be affected
- Some constraints can be added
  - Bounded delay, cost ...
  - A user may prefer the coast road ...
- User likes diversity!

Give the user a set of 'good' choices
Motivation

Sometimes, it is hard to specify constraints that a path should satisfy.

Applications:
- bioinformatics: biological sequence alignment
- natural language processing
- list decoding
- parsing
- network routing
- many more ...

Figure: aligning two DNA sequences
**Definition**

Input:
- Directed weighted graph $D = (V, A)$ with $w : A \rightarrow \mathbb{R}^+$,
- Two terminals $s$ and $t$ and an integer $k$

Output:
- $k$ paths $P_1, P_2, \ldots, P_k$ from $s$ to $t$ such that $w(P_i) \leq w(P_{i+1})$, $1 \leq i < k$ and $w(P_k) \leq w(Q)$ for all other $s$-$t$ paths $Q$

where $w(P) = \sum_{e \in A(P)} w(e)$
simple vs not simple

Figure: P is simple, Q is not simple

Definition (simple path)

a path is simple if and only if it has no repeated vertices
Complexity of the problem

**Theorem (Eppstein '97)**

The problem of finding \( k \) shortest paths can be solved in time \( O(m + n \log n + k) \)
Complexity of the problem

**Theorem (Eppstein ’97)**

*The problem of finding $k$ shortest paths can be solved in time $O(m + n \log n + k)$*

**Theorem (Yen ’71)**

*The problem of finding $k$ shortest simple paths can be solved in time $O(\text{kn}(m + n \log n))$*
Complexity of the problem

Theorem (Eppstein '97)

The problem of finding $k$ shortest paths can be solved in time $O(m + n \log n + k)$

Theorem (Yen '71)

The problem of finding $k$ shortest simple paths can be solved in time $O(kn(m + n \log n))$

Theorem (Williams and Williams '10)

All-Pairs-Shortest-Paths (APSP) $\prec_{(m,n)} 2$-SSP \iff \tilde{O}(n.m)$ for 2-SSP
Yen’s algorithm (the algorithm)

Yen’s idea:

- A second shortest simple path is a shortest simple deviation from a shortest path

Complexity: $O(kn \underbrace{(m + n\log n)}_{\text{Complexity of finding one SP}})$
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen's algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen's algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Yen’s algorithm (example)
Algorithm engineering

- 9th DIMAC’S implementation challenge followed by a set of improvements
- Feng 2014 (speed up the computation of deviations)
- Kurz and Mutzel 2016 (larger memory consumption)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yen</td>
<td>80</td>
</tr>
<tr>
<td>Feng</td>
<td>30</td>
</tr>
<tr>
<td>Kurz and Mutzel</td>
<td>1.15</td>
</tr>
</tbody>
</table>

Table: COL network ($n \approx 500,000; m \approx 1,000,000$ and $k = 300$)

- We proposed two improvements of Kurz and Mutzel’s algorithm
  - Up to twice faster with the same memory consumption
  - A time-space trade-off
Kurz and Mutzel’s algorithm (a path representation)

An $s$-$t$ path can be represented as a sequence of arcs and shortest path trees

Figure: $G$
Kurz and Mutzel’s algorithm (a path representation)

\[ P = \{s, v_3, v_1, v_2, v_3, v_4, v_2, t\} \] can be represented as \((T_0, e_1, T_0, e_2, T_1)\)

Note: \(P\) is not simple
Kurz and Mutzel’s algorithm (the algorithm)

**Algorithm 1 Kurz – Mutzel** \((G, s, t, k)\)

1: \(C \leftarrow \{(T_0)\} \quad \text{// where } T_0 \text{ is a shortest path tree of } G\)
2: \textbf{while } \(C\) is not empty \textbf{do}
3: \(P \leftarrow \text{extractMin}(C)\)
4: \textbf{if } \(P\) is simple \textbf{then}
5: \quad \text{add } \(P\) to the output
6: \quad \text{add the extensions of } \(P\) to \(C\)
7: \textbf{else}
8: \quad \textbf{if } \(P\) can be “repaired” \textbf{then}
9: \quad \quad \text{repair } \(P\) into a simple path and add it to \(C\)
Kurz and Mutzel’s algorithm (example)

Figure: $G$

Figure: $T_0$

Figure: the heap $C$

Output: $s, v_1, v_2, t$
Kurz and Mutzel’s algorithm (example)

**Figure: G**

**Figure: the heap C**

**Output: \( s, v_1, v_2, t \)**
Kurz and Mutzel’s algorithm (example)

**Figure:** $G$

$S$ \rightarrow $V_1$ \rightarrow $V_2$ \rightarrow $t$

$S$ \rightarrow $V_1$ \rightarrow $V_2$ \rightarrow $t$

$V_3$ \rightarrow $V_4$

$e_1 = 1$, $e_2 = 2$

$e_3 = 6$

**Output:** $s, v_1, v_2, t$

**Figure:** the heap $C$

$(T_0)$

$s, v_1, v_2, t$

$w=3$

$(T_0, e_1, T_0)$

$s, v_3, v_1, v_2, t$

$w=4$

$(T_0, e_2, T_0)$

$s, v_1, v_2, v_3, v_1, v_2, t$

$w=7$

**Figure:** $T_0$
Kurz and Mutzel’s algorithm (example)

Figure: $G$

Figure: $T_0$

Figure: the heap $C$

Output: $s, v_1, v_2, t$

$S, v_3, v_1, v_2, t$

$S, v_3, v_1, v_2, t$

$S, v_3, v_1, v_2, t$

$S, v_3, v_1, v_2, t$

$S, v_3, v_1, v_2, t$

$S, v_3, v_1, v_2, t$

$S, v_3, v_1, v_2, t$
**Kurz and Mutzel’s algorithm (example)**

**Figure: $G$**

- $S$ to $V_1$: $e_1$ with weight 1
- $V_1$ to $V_2$: $e_2$ with weight 2
- $V_2$ to $V_3$: $e_3$ with weight 6
- $V_3$ to $V_4$: $e_3$ with weight 6
- $V_4$ to $T$: $e_1$ with weight 1
- $V_1$ to $T$: $e_2$ with weight 1

**Figure: the heap $C$**

- $(T_0)$: $S, V_1, V_2, t$
- $(T_0, e_1, T_0)$: $S, V_3, V_1, V_2, t$
- $(T_0, e_2, T_0)$: $S, V_1, V_2, V_3, V_1, V_2, t$
- $(T_0, e_1, T_0, e_3, T_0)$: $S, V_3, V_4, t$
- $(T_0, e_1, T_0, e_2, T_0)$: $S, V_3, V_1, V_2, V_3, V_1, V_2, t$

**Output:** $S, V_3, V_1, V_2, t$

---

**K Shortest Paths Problem**

Al Zoobi, Coudert and Nisse  
June 18, 2020
Kurz and Mutzel’s (’16) improvements

- Verify if a path is simple or not in a pivot step
- Using a Lazy Dijkstra (stop once the path is constructed)
- Split the heap $C$ into two ($C_{simple}$ and $C_{not\-simple}$)
- ...

...
Our first improvement

Once a no simple path \( P = (T_0, e_0, \cdots, T_h, e_h, T_h) \) is extracted, the algorithm repair \( P \) \( \Rightarrow \) computing a new SP tree \( T' \)

- Remark: \( T' \) and \( T_h \) are “similar” and \( T_h \) is computed and stored
- Instead of computing \( T' \) from scratch
  - Compute \( T' \) starting from \( T_h \)
Our first improvement - evaluation

Evaluation of the improvement on Dimac’s routing networks

- A speed up by a factor of 1.5 to 2 on average

Publicly available: https://gitlab.inria.fr/dcoudert/k-shortest-simple-paths
Space-time trade off

In practice, memory consumption is a **BIG** issue

- One shortest path tree of a graph of one million vertices needs $\approx 1MB$
- For big values of $k$, a large number of shortest path trees should be stored
- One may use Feng’s algorithm, but it is too slow.

**Goal:** space time tradeoff
Space-time trade off

Let $P = (s, v_1, \ldots, v_i, \ldots, t)$ be a path extracted from $C$, and let $E = \{e_1, \ldots, e_{\text{min}}, \ldots, e_p\}$ be the set of deviations tailing at $P$ and leading to a no simple extension, with $e_{\text{min}}$ the deviation with the smallest weight lower bound.

Kurz-Mutzel’s algorithm:
- May computes independently a new tree for each vertex with a deviation tailing at it
- Each tree remains in the memory till the end of the execution
Remark: All these trees are "similar"

The improvement:

- Compute simultaneously the simple extensions at $e_{min}, \ldots, e_p$
- Add them to $C$ with their real cost as a key
- Add the extensions at $e_1, \ldots, e_{min-1}$ to $C$ with the weight of the deviation at $e_{min}$ as a key
- Only the tree of the extensions at $e_{min}$ is saved
**Space-time trade off**

Consequences:

- All of these no simple extension after $e_{min}$ have a higher key in $C$ and their extraction could be (hopefully) skipped.
- Their corresponding trees are freed from the memory.
- Unfortunately, some trees may be re-computed.
Space-time trade off - evaluation

Evaluation of the improvement on the routing network of Rome
\((n \approx 3000, m \approx 9000 \text{ and } k = 10,000)\)

- A space reduction by a factor of 1.5 to 2 on average with a comparable running time
The goal:
- A tree is stored in the memory if and only if it will be used during the execution of the algorithm
  - No trees re-computation
  - Less space consuming
- **Remark:** Only trees used for (relatively) shortest paths are re-used
  - Store a tree only if it is used to extend a path that is shorter than a threshold value
- Analyse other parameters (number of hops of paths, size of the input graph) in order to know which algorithm is better for each input

**Questions ?**