On the k shortest simple paths : A faster algorithm with low memory consumption

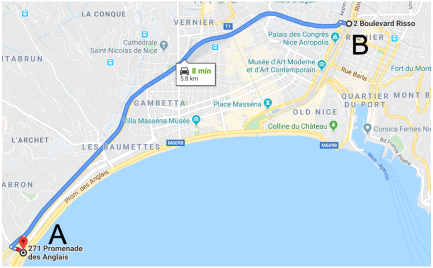
Ali Al Zoobi, David Coudert and Nicolas Nisse
Université Côte d'Azur, Inria, CNRS, I3S

November 13, 2020

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 ...

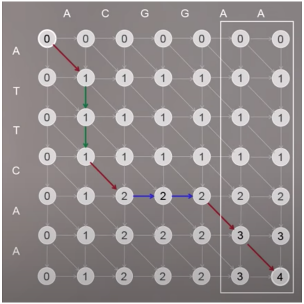


Figure: aligning two DNA sequences

k shortest paths problem

Definition

Input:

- Directed weighted graph $D = (V, A)$ with $w : A \to \mathbb{R}^+$,
- 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 $w(P_i) \leqslant w(P_{i+1})$, $1 \leqslant i < k$ and $w(P_k) \leqslant w(Q)$ for all other s-t paths Q

  where $w(P) = \sum_{e \in A(P)} w(e)$

simple vs not simple

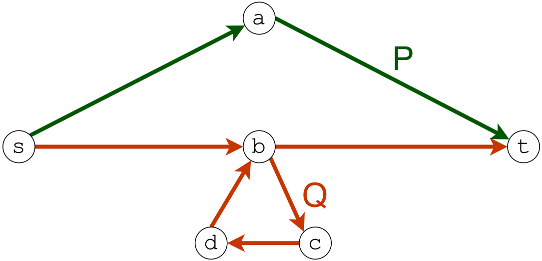


Figure: P is simple, Q is not simple

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 $O(kn(m + n\log n))$

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 ($\Leftrightarrow \tilde{O}(n.m)$ for 2-SSP)*

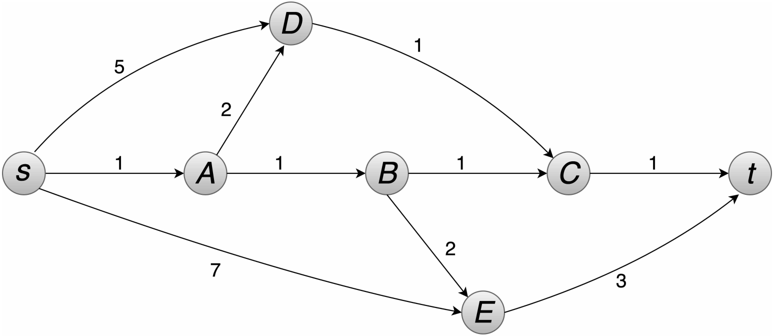
Yen's algorithm (the algorithm)

Yen's idea:

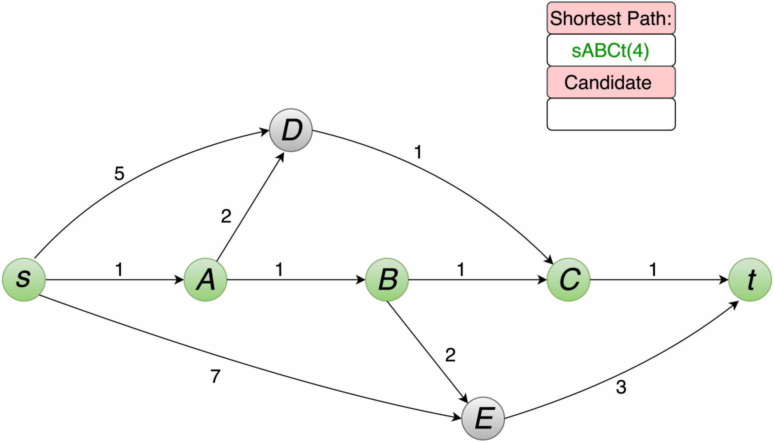
- A second shortest simple path is a shortest simple detour from a shortest path

Complexity: $O(kn \underbrace{(m + nlogn)}_{\text{Complexity of finding one SP}})$

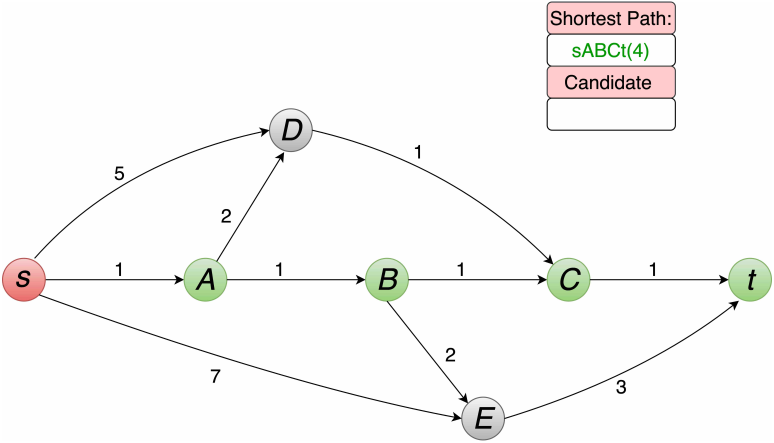
Yen's algorithm (example)



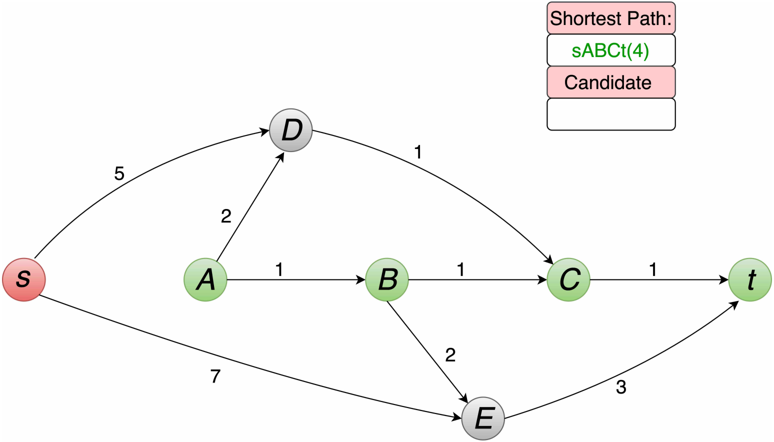
Yen's algorithm (example)



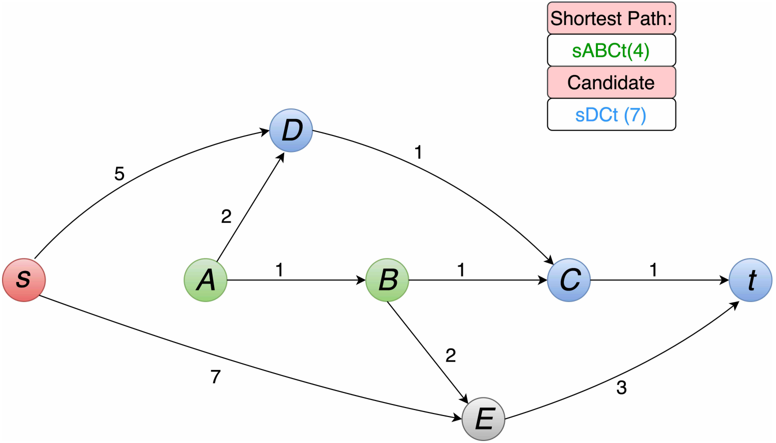
Yen's algorithm (example)



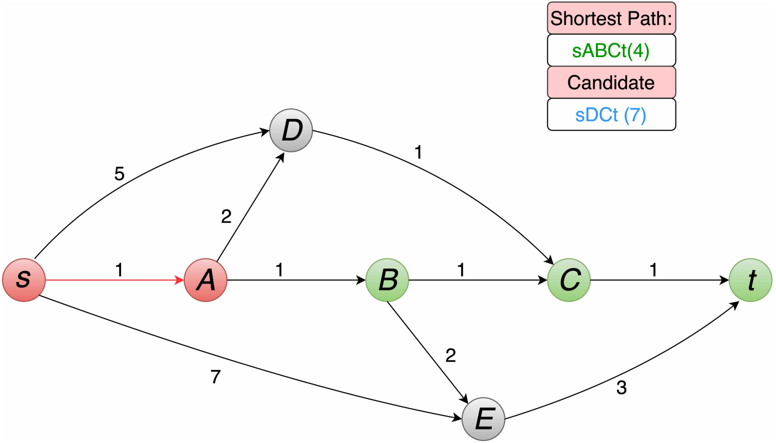
Yen's algorithm (example)



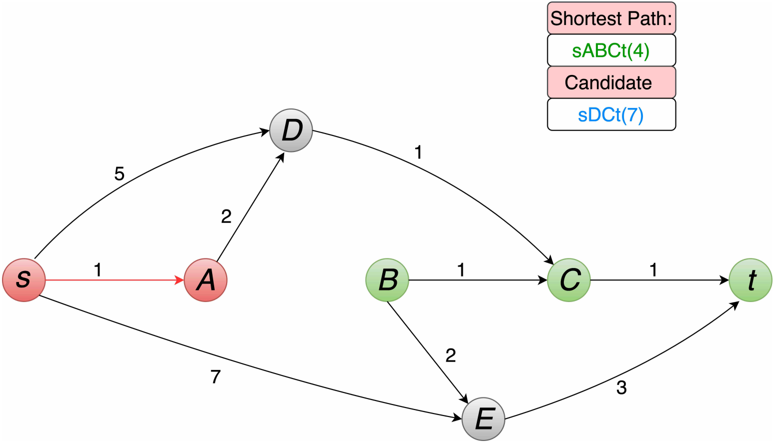
Yen's algorithm (example)



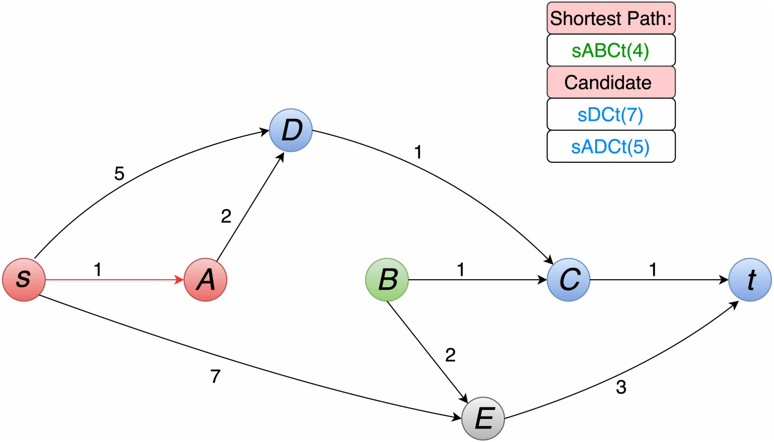
Yen's algorithm (example)



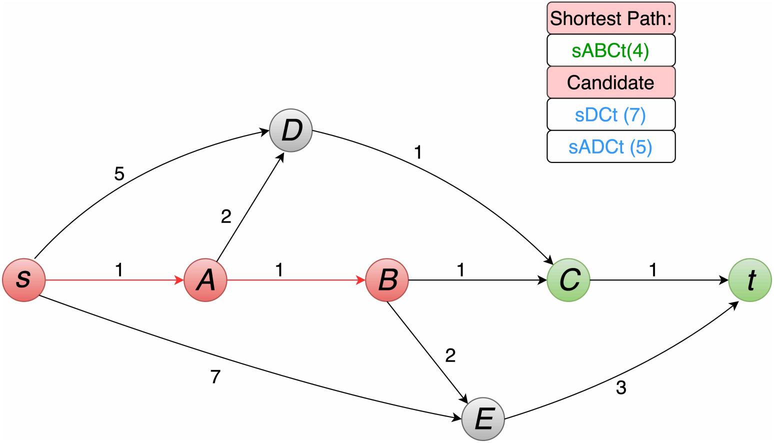
Yen's algorithm (example)



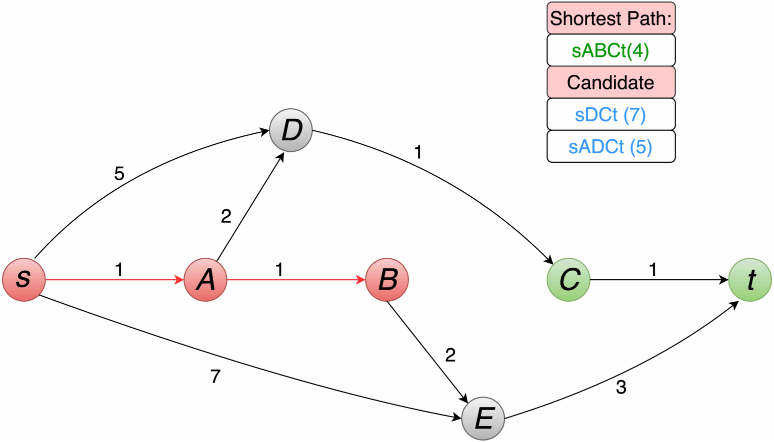
Yen's algorithm (example)



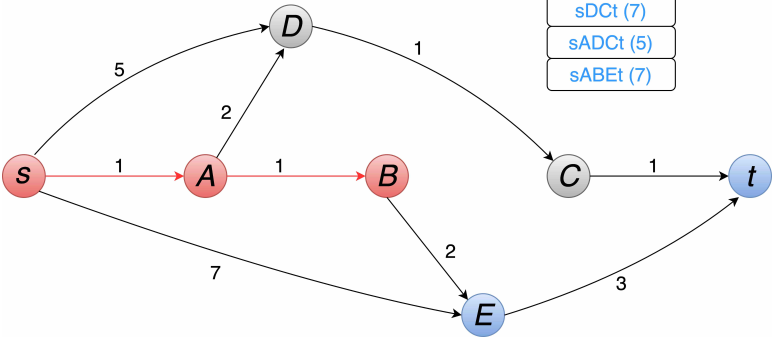
Yen's algorithm (example)



Yen's algorithm (example)

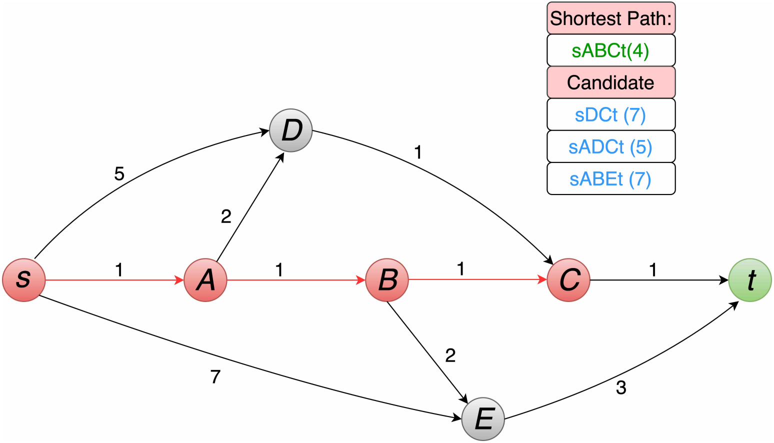


Yen's algorithm (example)

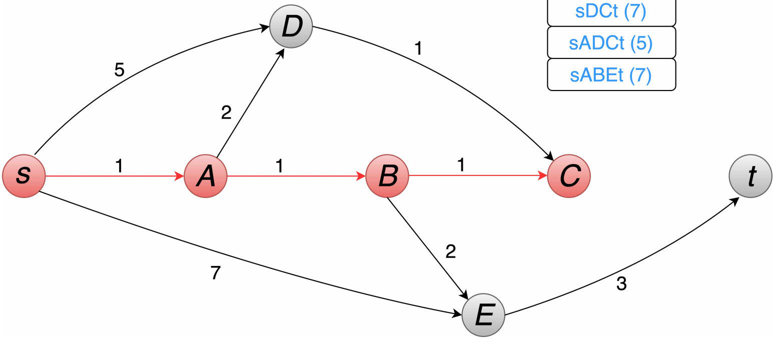


Shortest Path:
sABCt(4)
Candidate
sDCt (7)
sADCt (5)
sABEt (7)

Yen's algorithm (example)

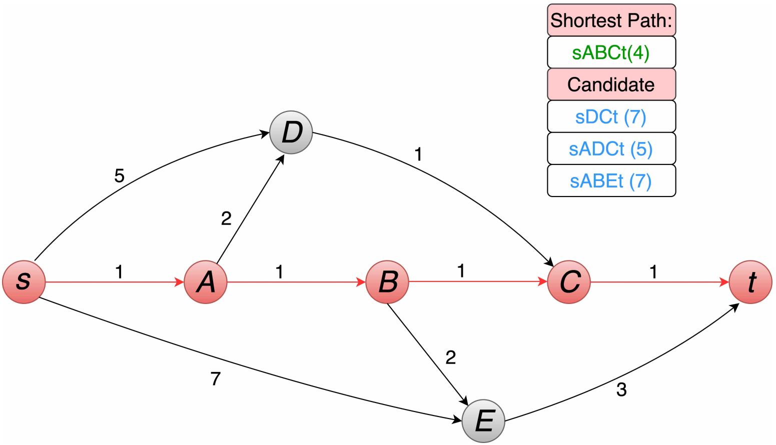


Yen's algorithm (example)

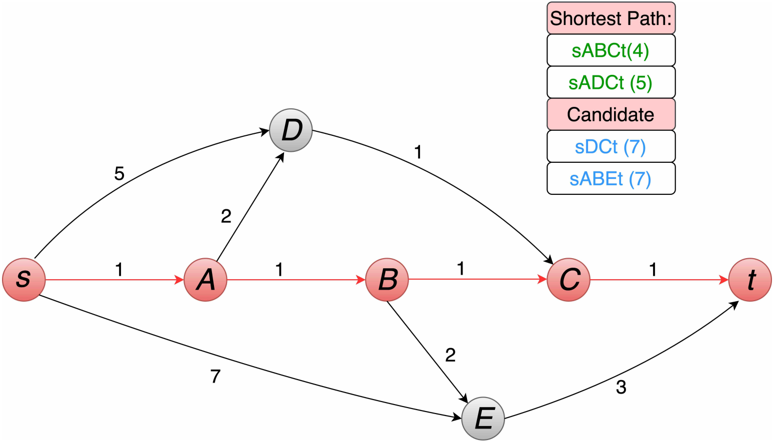


Shortest Path:
sABCt(4)
Candidate
sDCt (7)
sADCt (5)
sABEt (7)

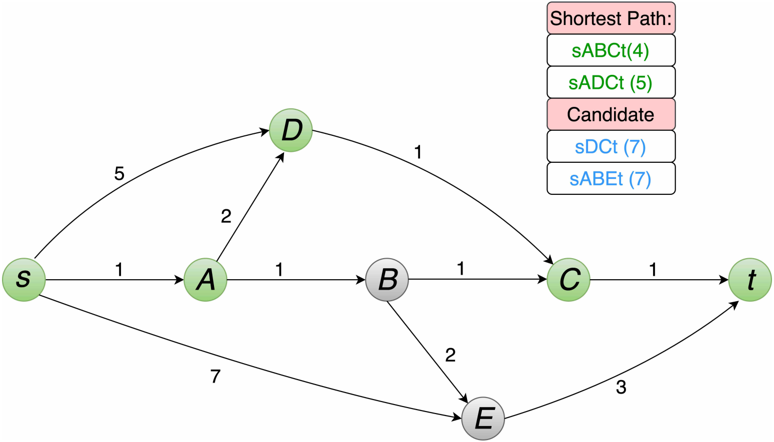
Yen's algorithm (example)



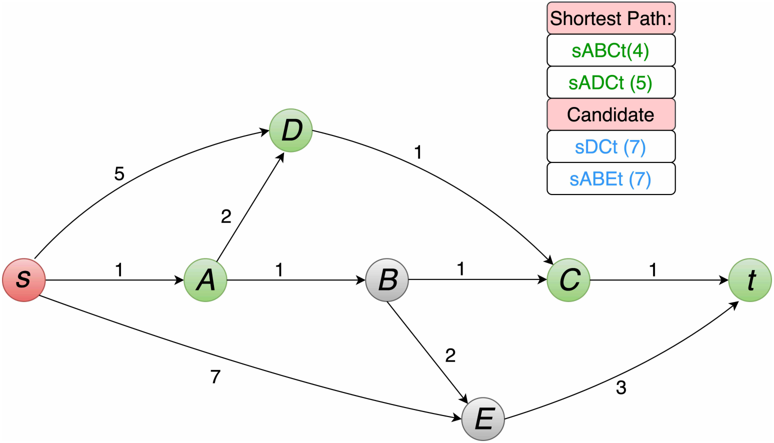
Yen's algorithm (example)



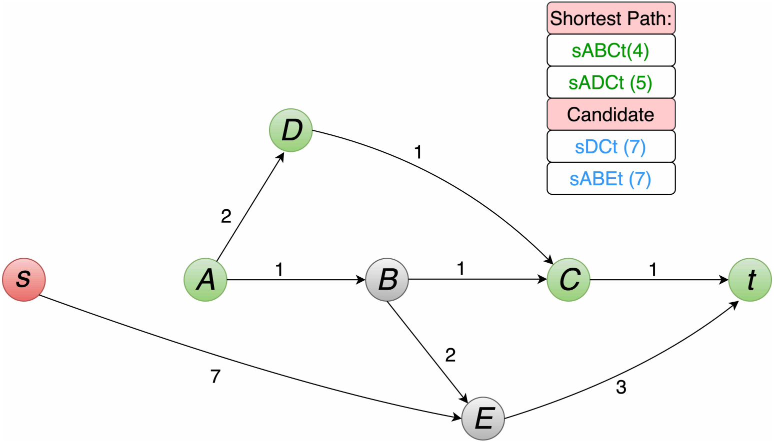
Yen's algorithm (example)



Yen's algorithm (example)

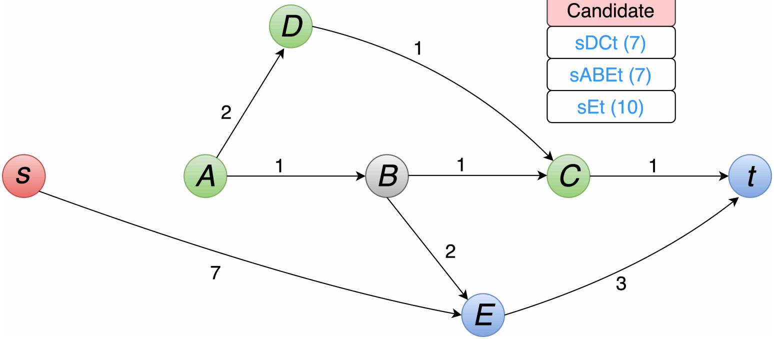


Yen's algorithm (example)

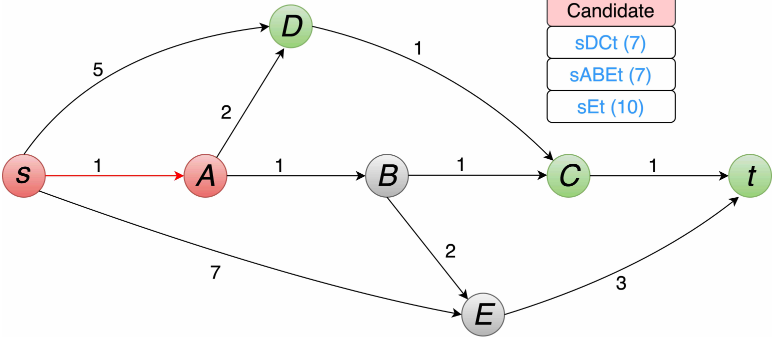


Yen's algorithm (example)

Shortest Path:
sABCt(4)
sADCt (5)
Candidate
sDCt (7)
sABEt (7)
sEt (10)

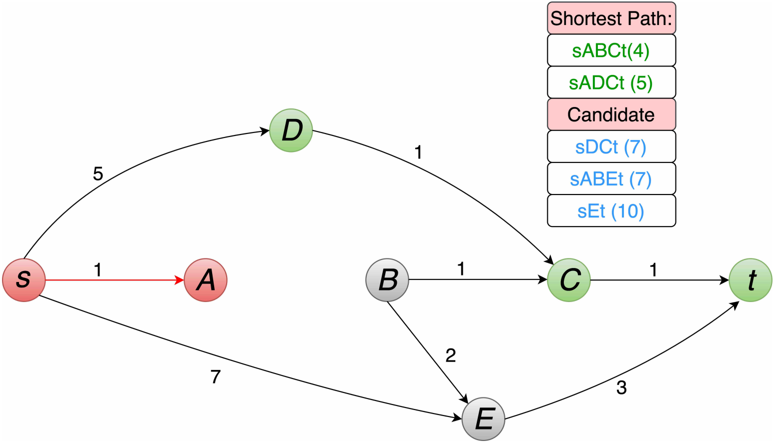


Yen's algorithm (example)

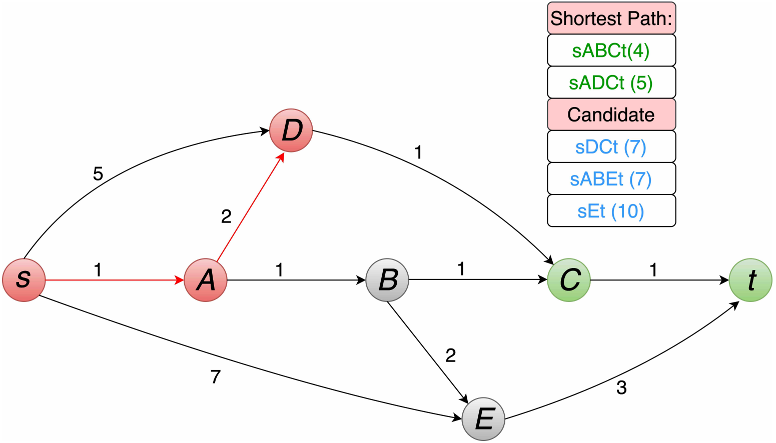


Shortest Path:
sABCt(4)
sADCt (5)
Candidate
sDCt (7)
sABEt (7)
sEt (10)

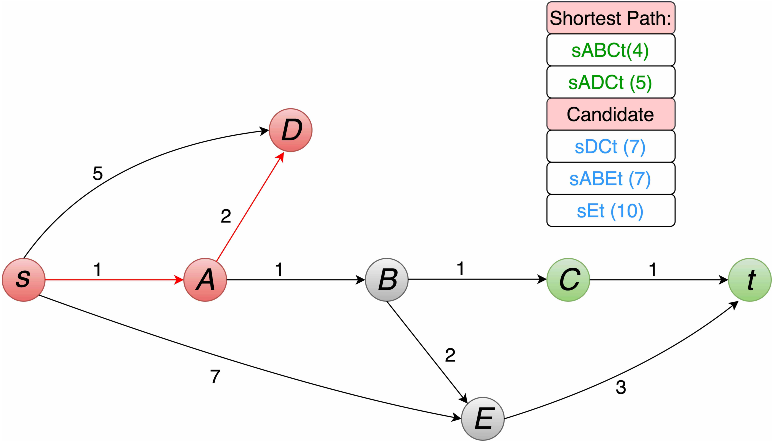
Yen's algorithm (example)



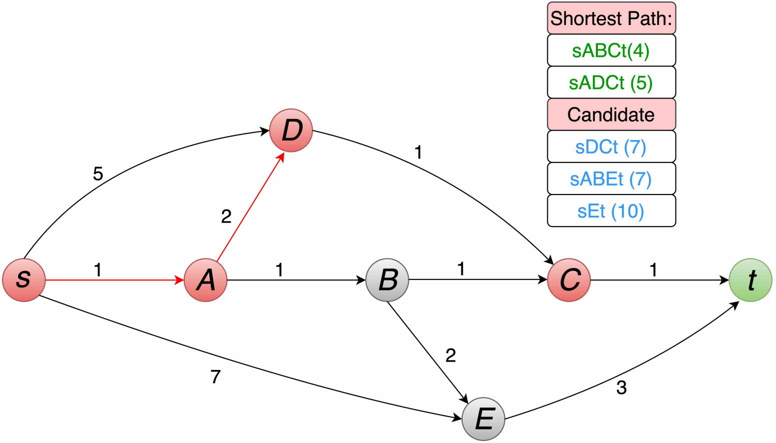
Yen's algorithm (example)



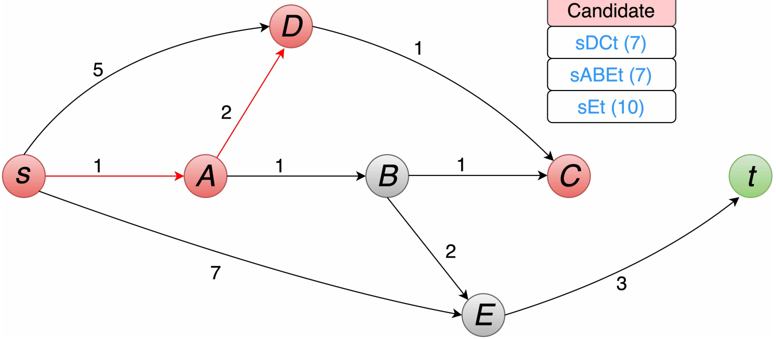
Yen's algorithm (example)



Yen's algorithm (example)



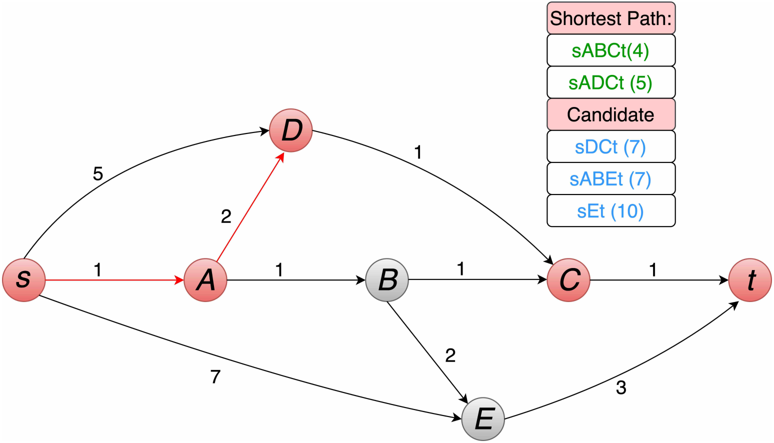
Yen's algorithm (example)



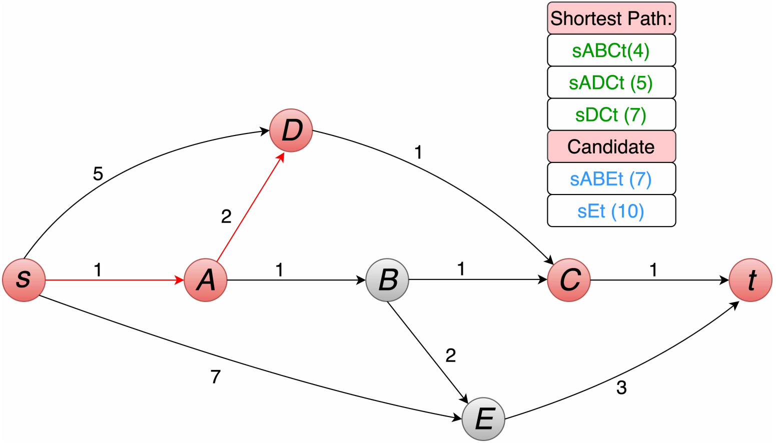
Shortest Path:
sABCt(4)
sADCt (5)

Candidate
sDCt (7)
sABEt (7)
sEt (10)

Yen's algorithm (example)



Yen's algorithm (example)



Algorithm engineering

On Road networks:

- 9th DIMAC'S implementation challenge followed by a set of improvements
- (NC) by Feng 2014 (speed up the detours computation)
- (SB*) by Kurz and Mutzel 2016, followed by Al Zoobi et al. 2019 (larger memory consumption)

Algorithm	time (ms)
Yen	11,316
NC	823
SB* (large memory)	117

Table: DC network ($n \approx 10,000$; $m \approx 15,000$ and $k = 1,000$)

- We proposed the PNC
  - The fastest algorithm with low memory consumption

Postponed Node Classification (PNC) aglorithm

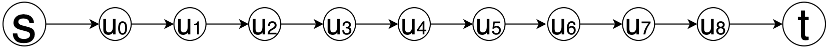
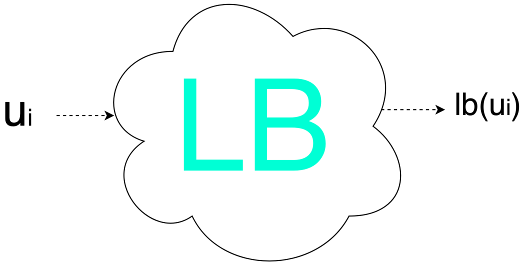
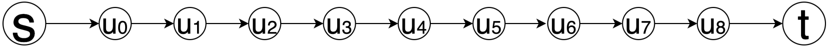


Figure: A shortest path P_0 from s to t

- So many calls of SP algorithm
- Can we skip some ?

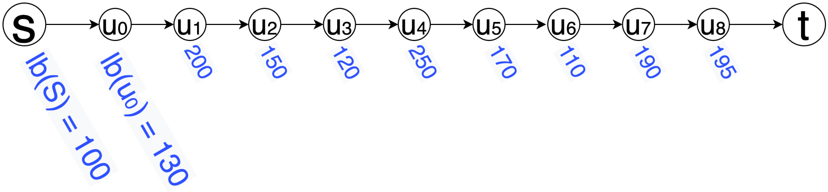
PNC: skipping SP calls





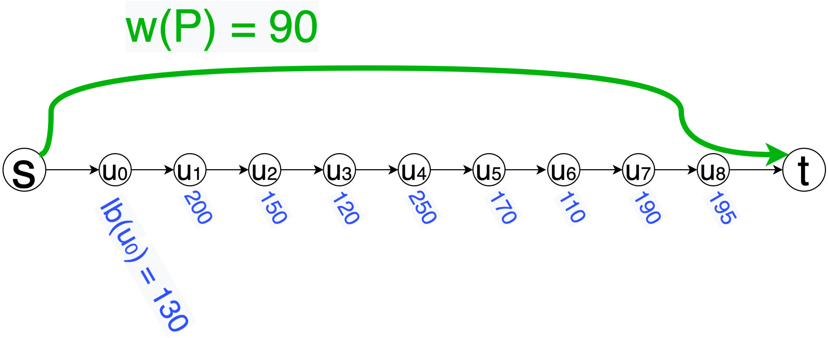
- $lb(u_i) = c$ s.t. a shortest simple detour of P_0 at u_i is bigger than c
- LB answers in a pivot step

PNC: skipping SP calls



- After applying LB on each vertex of P_0

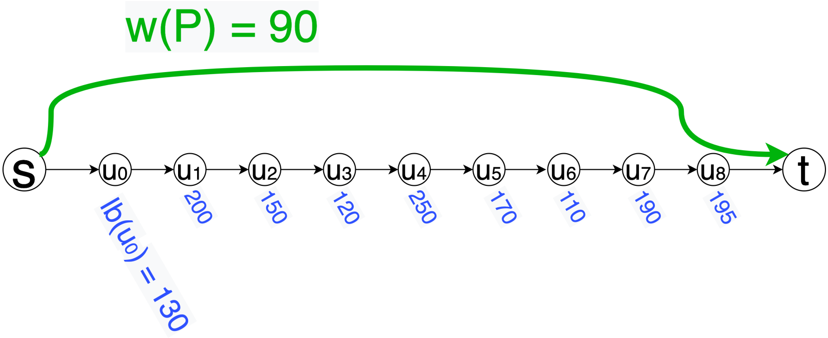
PNC: skipping SP calls



If a shortest simple detour P of P_0 has length less $lb(u_i)$ for each $u_i \in P_0$

- What can we deduce?

PNC: skipping SP calls



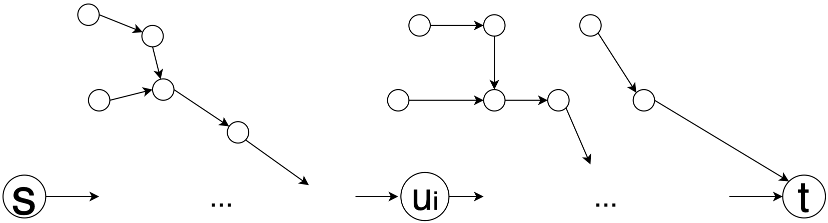
If $w(P) \leqslant lb(u_i)$ for each $u_i \in P_0$

- Then P is a second shortest path

Otherwise

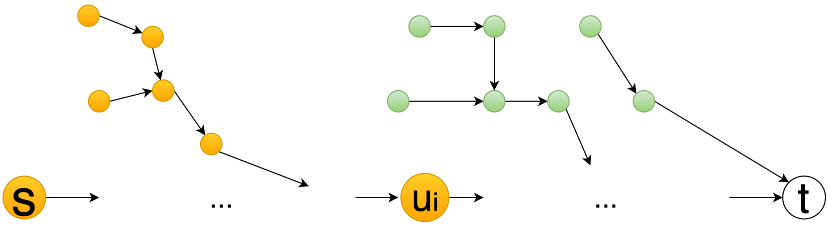
- continue until P' (with $w(P') \leqslant lb(u_i)$ for each $u_i \in P_0$) is found

PNC: describing *LB*



While computing P_0, the reversed shortest path tree T rooted at t is kept

PNC : describing LB

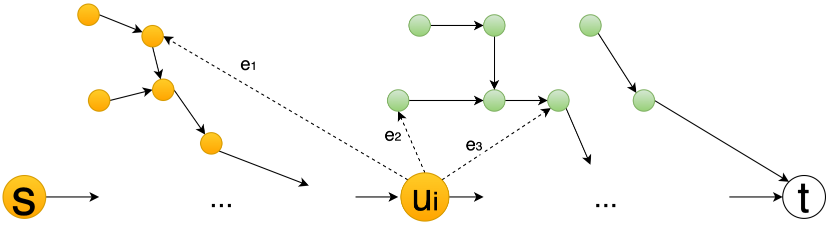


For each vertex u_i, each vertex v in T is colored:

- Yellow if the path from v to t in T cross u_i
- Green Otherwise

Classification of Feng 2014

PNC : describing *LB*



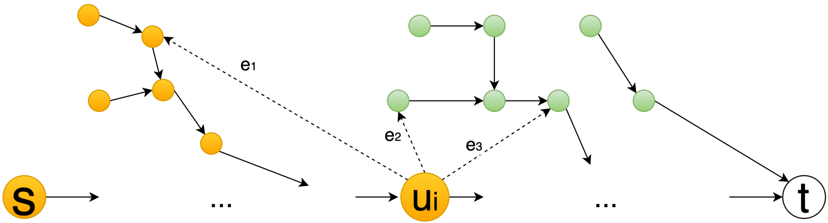
For each arc $e_j = (u_i, v)$ tailing at u_i:

- $\delta(e_i) = w(s, \cdots, u_i) + w(e_j) + w(P^T_{v-t})$: the cost of the shortest detour at e_j

Let $e_{min} = (u_i, v_{min})$ be the arc with minimum δ (δ_{min})

inspired by Kurz and Mutzel (2016)

PNC : describing LB

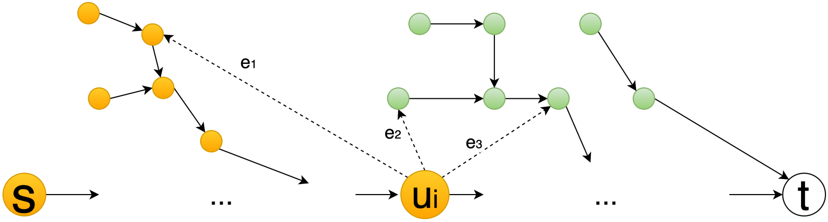


Let $e_{min} = (u_i, v_{min})$ be the arc with minimum δ (δ_{min})

Claim: $\delta_{min} = lb(u_i)$

δ_{min} is the length of a shortest detour (not necessarily simple) at u_i

PNC : describing *LB*



NC algorithm calls an SP algorithm at u_i

- (PNC) postpone such call so it may be skipped

PNC : The algorithm

Algorithm 1 $PNC(G, s, t)$

$C \leftarrow \{P_0\}$
while C is not empty **do**
    $P \leftarrow extractMin(C)$
    if P is simple **then**
        add P to the output
        using the LB procedure, add the shortest detours of P to C
    else
        repair P into a simple path and add it to C

PNC : The algorithm

Algorithm 2 $PNC(G, s, t)$

```
C ← {P_0}
while C is not empty do
    P ← extractMin(C)
    if P is simple then
        add P to the output
        using the LB procedure, add the shortest detours of P to C
    else
        repair P into a simple path and add it to C
```

Remarque:

- Low memory consumption: only one shortest path is kept in the memroy (+ candidate paths)

PNC : Evalutation

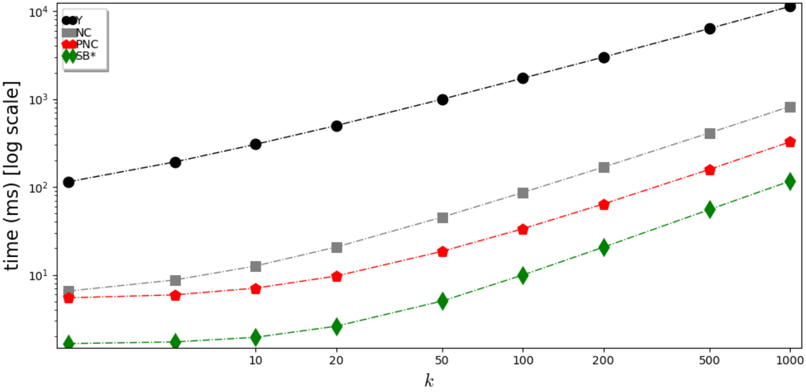


Figure: The average running time of the *kSSP* algorithms with respect to k on DC road networks ($n \approx 10,000$; $m \approx 15,000$)

PNC : Evalutation

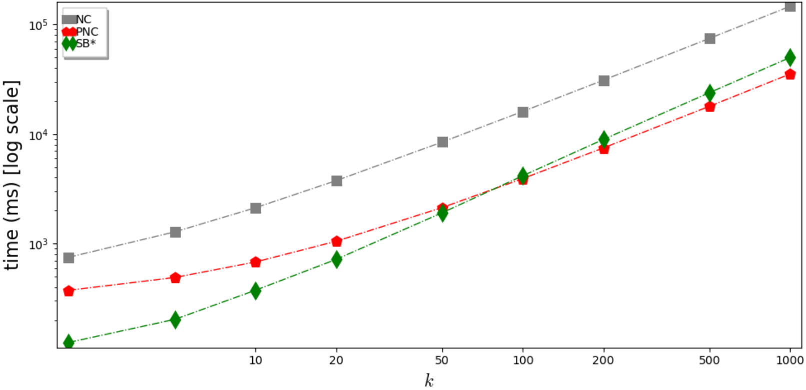


Figure: The average running time of the *kSSP* algorithms with respect to k on COL road networks ($n \approx 500,000$; $m \approx 1,000,000$)

Ongoing - future work

- Evaluate these algorithms on complex networks
- Study the impact of these improvements on others problems
  - Path with resource constraints, bioinformatics problems ...
- Study the problem when the arcs may have negative weights

Questions ?