Universität Stuttgart

fmi
alg

Lukas Epple

Fachpraktikum Algorithmik für OSM-Daten

Contents

Introduction

Introduction

- Implemented Contraction Hierarchies

Introduction

- Implemented Contraction Hierarchies
- Levels for each node $l : V \to \mathbb{N}$

Introduction

- Implemented Contraction Hierarchies
- Levels for each node $l : V \to \mathbb{N}$
- If a path exists between $u, v \in V$ then it will be found by two upward searches from the u and v

Introduction

- Implemented Contraction Hierarchies
- Levels for each node $l : V \to \mathbb{N}$
- If a path exists between $u, v \in V$ then it will be found by two upward searches from the u and v
- only using upwards reduces the searchspace

CH-Query

2

CH-Query

- given $s, t \in V$, find path from s to t and distance $d(s, t)$

CH-Query

- given $s, t \in V$, find path from s to t and distance $d(s, t)$
- perform dijkstra only following edges to nodes with higher levels from s and from t

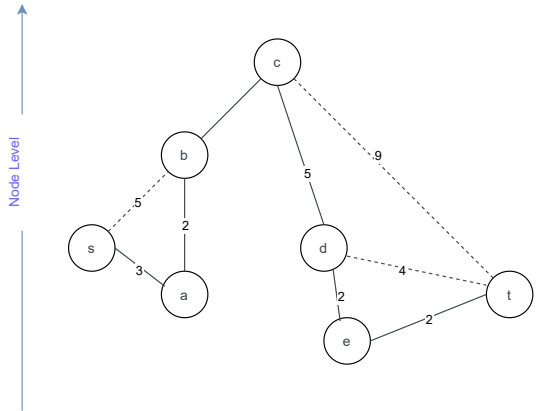
CH-Query

- given $s, t \in V$, find path from s to t and distance $d(s, t)$
- perform dijkstra only following edges to nodes with higher levels from s and from t
- S_s, S_t are the settled nodes from the two dijkstras

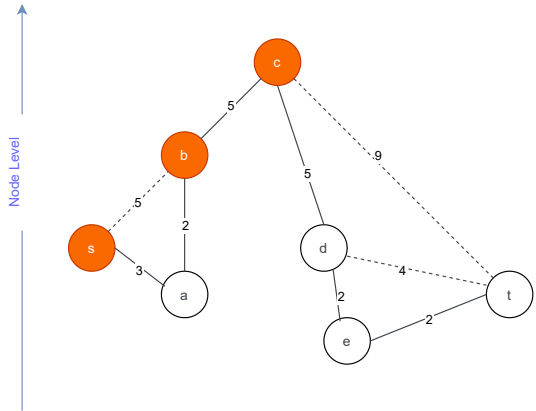
CH-Query

- given $s, t \in V$, find path from s to t and distance $d(s, t)$
- perform dijkstra only following edges to nodes with higher levels from s and from t
- S_s, S_t are the settled nodes from the two dijkstras
- find node $n \in S_s \cap S_t$ where $d(s, n) + d(n, t)$ is minimal

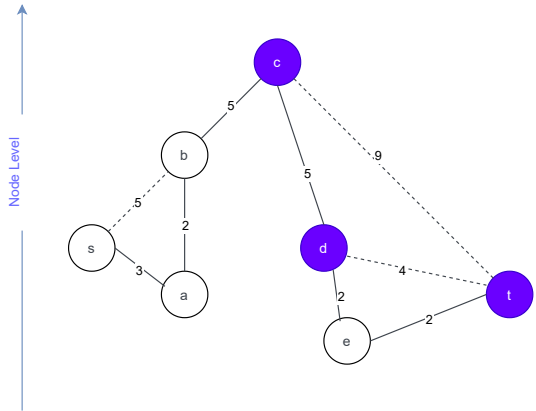
CH-Query



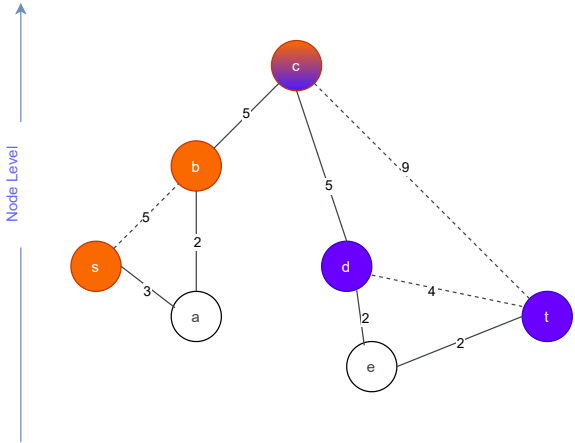
CH-Query



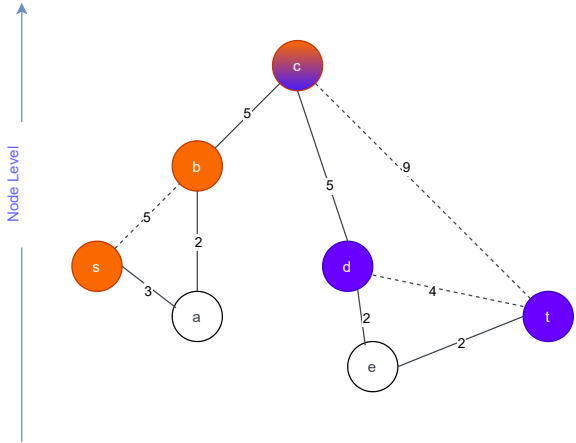
CH-Query



CH-Query

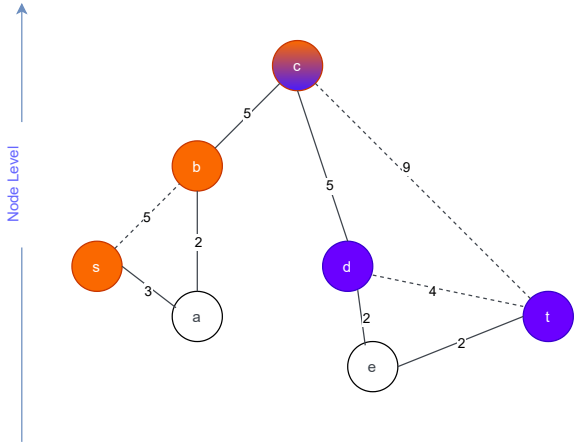


CH-Query



- $S_s = \{s, b, c\}$
- $S_t = \{t, d, c\}$

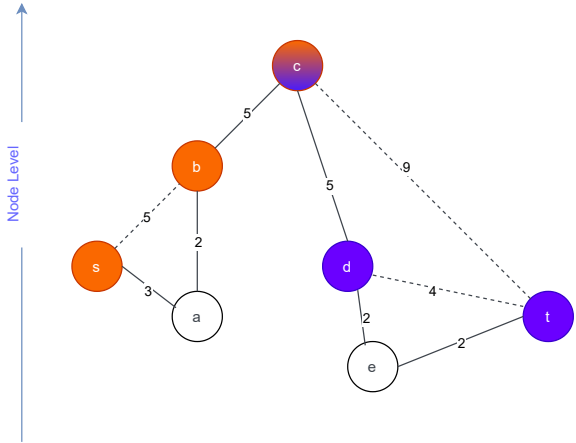
CH-Query



- $S_s = \{s, b, c\}$
- $S_t = \{t, d, c\}$

$\Rightarrow S_s \cap S_t = \{c\}$

CH-Query

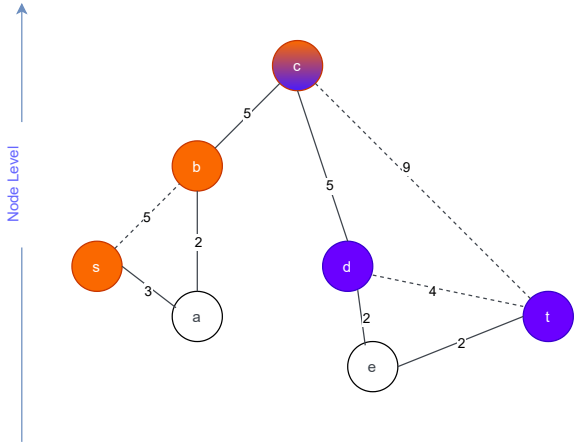


- $S_s = \{s, b, c\}$
- $S_t = \{t, d, c\}$

$\Rightarrow S_s \cap S_t = \{c\}$

$\Rightarrow d(s,t) = 19$

CH-Query



- $S_s = \{s, b, c\}$
- $S_t = \{t, d, c\}$

$\Rightarrow S_s \cap S_t = \{c\}$

$\Rightarrow d(s,t) = 19$

$\Rightarrow$ recursive unpack the path

Optimizations

3

Caching Settled Nodes Lists

Caching Settled Nodes Lists

currently S_s, S_t are cleaned up after a query.

Caching Settled Nodes Lists

currently S_s, S_t are cleaned up after a query.
Idea: keep S_s, S_t and only clean them up before the next query if needed.

Caching Settled Nodes Lists

currently S_s, S_t are cleaned up after a query.
Idea: keep S_s, S_t and only clean them up before the next query if needed.

But...

This optimization cannot be observed well when benchmarking random $s - t$ queries.

Independent Sets

Currently every node has a different level.

Independent Sets

Currently every node has a different level.
Idea: Nodes $x, y \in V$ can have the same level if $xy = e \notin E$

Independent Sets

Independent Sets

- Select $U \subseteq V$ such that $\forall x, y \in U : xy \notin E$

Independent Sets

- Select $U \subseteq V$ such that $\forall x, y \in U : xy \notin E$
- Contract all nodes $x \in U$

Independent Sets

- Select $U \subseteq V$ such that $\forall x, y \in U : xy \notin E$
- Contract all nodes $x \in U$
- Calculate average edge difference ($|shortcutsadded| - |edgesdeleted|$)

Independent Sets

- Select $U \subseteq V$ such that $\forall x, y \in U : xy \notin E$
- Contract all nodes $x \in U$
- Calculate average edge difference ($|shortcutsadded| - |edgesdeleted|$)
- add contractions where the $edgedifference \leq avgedgedifference$

Independent Sets

- Select $U \subseteq V$ such that $\forall x, y \in U : xy \notin E$
- Contract all nodes $x \in U$
- Calculate average edge difference ($|shortcutsadded| - |edgesdeleted|$)
- add contractions where the $edgedifference \leq avgedgedifference$

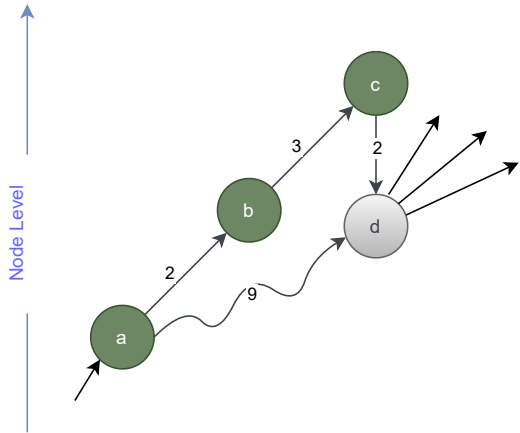
$\Rightarrow$Reduces the preprocessing time

Stall On Demand

Stall On Demand

Some Nodes are settled with wrong distances and can never be part of a shortest path.

Stall On Demand



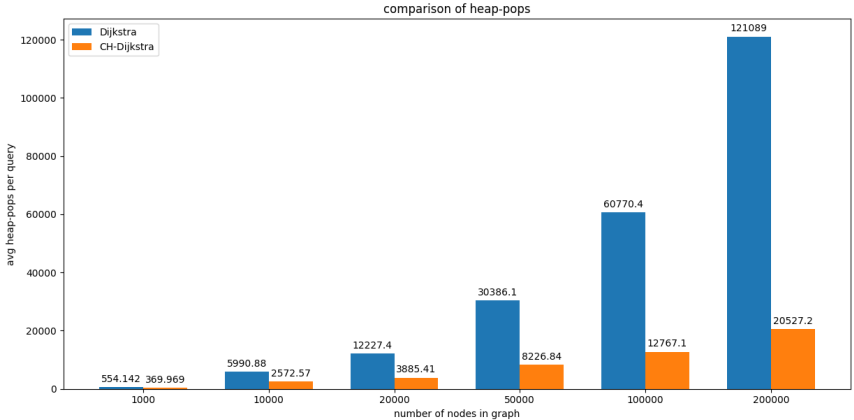
Benchmarks

4

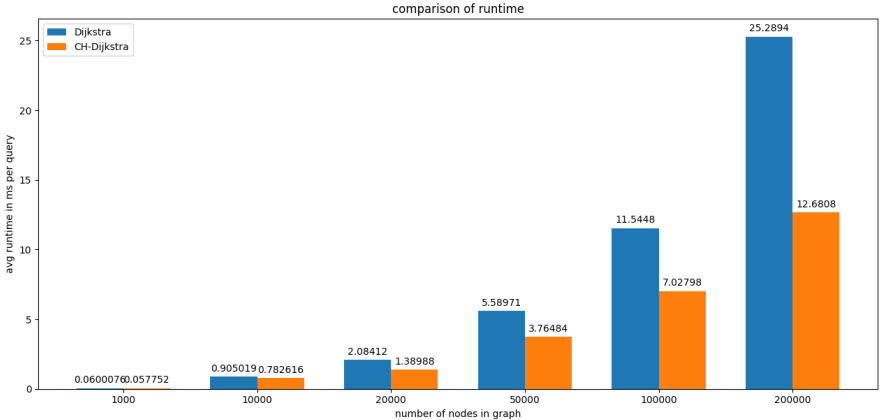
Benchmarks

The benchmarks compare a normal Dijkstra with an optimized CH-Dijkstra

Heap-Pops


comparison of heap-pops
Dijkstra
CH-Dijkstra
avg heap-pops per query
number of nodes in graph
0
20000
40000
60000
80000
100000
120000
1000
10000
20000
50000
100000
200000
554.142
369.969
5990.88
2572.57
12227.4
3885.41
30386.1
8226.84
60770.4
12767.1
121089
20527.2

Querytime


comparison of runtime
Dijkstra
CH-Dijkstra
avg runtime in ms per query
number of nodes in graph
0.0600076
0.057752
0.905019
0.782616
2.08412
1.38988
5.58971
3.76484
11.5448
7.02798
25.2894
12.6808
1000
10000
20000
50000
100000
200000

THE END

Thanks for listening!