Algorithmen für Routenplanung

3. Vorlesung, Sommersemester 2017

Moritz Baum | 10. Mai 2017

Institut für Theoretische Informatik · Algorithmik · Prof. Dr. Dorothea Wagner

Kürzeste Wege in Straßennetzwerken

Beschleunigungstechniken

- A*
- ALT
- CALT

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Beschleunigungstechniken

Beobachtung:

- Viele Anfragen in (statischem) Netzwerk
- „Unnötige“ Berechnungen

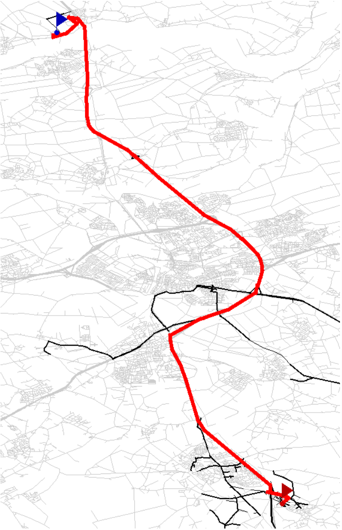
Beschleunigungstechniken

Beobachtung:

- Viele Anfragen in (statischem) Netzwerk
- „Unnötige“ Berechnungen

Idee:

- Zwei Phasen
  - Offline: Generiere Zusatzinformation in Vorberechnung
  - Online: Beschleunige Anfrage mit dieser Information
- Drei Kriterien: Vorberechnungsplatz, Vorberechnungszeit, Beschleunigung



Ideensammlung

Karlsruher Institut für Technologie

Wie Suche zielgerichtet machen?

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Ideensammlung

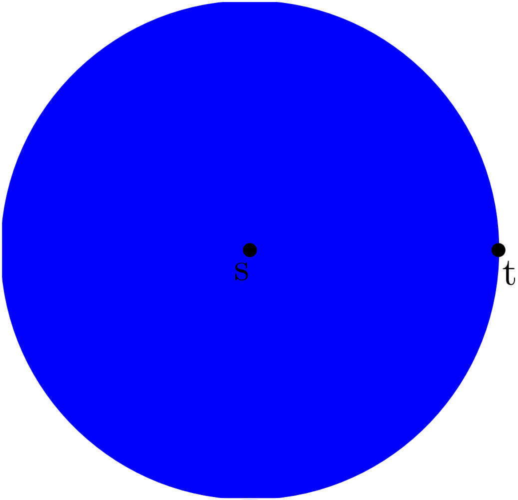
Wie Suche zielgerichtet machen?

- Reihenfolge in der Knoten besucht werden ändern
- Nichtbeachten von Kanten, die in die “falsche” Richtung führen

Jetzt: ersteres

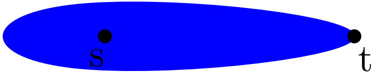
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Schematischer Suchraum



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Schematischer Suchraum



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

ALT

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

A*

Dijkstra's Algorithmus

- benutzt $d[v]$, um zu entscheiden, welcher Knoten als nächstes abgearbeitet wird

A*

- auch zielgerichtete Suche (goal-directed search) genannt

Idee:

- Benutze Potential $\pi_t : V \to \mathbb{R}$
- $\pi_t(v)$ ist ein Schätzwert für Entfernung $d(v, t)$ zum Ziel t
- Benutze $d[v] + \pi_t(v)$ als Key in Q

$\Rightarrow$ Knoten mit geschätzt geringerer Entfernung zu t zuerst besucht

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Pseudocode A*

A*($G = (V, E), s, t$ **)**

```
d[s] = 0
Q.clear(), Q.add(s, π_t(s))
while !Q.empty() do
    u ← Q.deleteMin()
    break if u = t
    forall the edges e = (u, v) ∈ E do
        if d[u] + len(e) < d[v] then
            d[v] ← d[u] + len(e)
            if v ∈ Q then Q.decreaseKey(v, d[v] + π_t(v))

            else Q.insert(v, d[v] + π_t(v))
```

Frage

Können beliebige Potentialfunktionen benutzt werden?

- (Fast) beliebige Abarbeitungsreihenfolgen lassen sich erzeugen
- Dies kann offensichtlich zu falschen Ergebnissen führen.
  Warum sonst benutzen wir überhaupt Dijkstra und nicht Breitensuche?

Wie kommen wir also zu zulässigen Potentialfunktionen?

A* – Äquivalente Formulierung

A* auf Graph $G = (V, E, \text{len})$, Knoten $s, t \in V$, mit Potential π_t

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

A* – Äquivalente Formulierung

A* auf Graph $G = (V, E, \text{len})$, Knoten $s, t \in V$, mit Potential π_t

Dijkstra auf $\overline{G} = (V, E, \overline{\text{len}})$ mit $\overline{\text{len}}(u, v) = \text{len}(u, v) - \pi_t(u) + \pi_t(v)$

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

A* – Äquivalente Formulierung

A* auf Graph $G = (V, E, \text{len})$, Knoten $s, t \in V$, mit Potential π_t

Dijkstra auf $\overline{G} = (V, E, \overline{\text{len}})$ mit $\overline{\text{len}}(u, v) = \text{len}(u, v) - \pi_t(u) + \pi_t(v)$

Beweis: Folgt direkt aus Pseudocode

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

A* – Äquivalente Formulierung

A* auf Graph $G = (V, E, \text{len})$, Knoten $s, t \in V$, mit Potential π_t

Dijkstra auf $\overline{G} = (V, E, \overline{\text{len}})$ mit $\overline{\text{len}}(u, v) = \text{len}(u, v) - \pi_t(u) + \pi_t(v)$

Beweis: Folgt direkt aus Pseudocode

Zulässiges Potential (feasible potential)

Eine Potentialfunktion $\pi_t : V \to \mathbb{R}$ heißt *zulässig*, falls
$\text{len}(u, v) - \pi_t(u) + \pi_t(v) \geq 0$ für alle Kanten $(u, v) \in E$.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

A* – Äquivalente Formulierung

A* auf Graph $G = (V, E, \text{len})$, Knoten $s, t \in V$, mit Potential π_t

Dijkstra auf $\overline{G} = (V, E, \overline{\text{len}})$ mit $\overline{\text{len}}(u, v) = \text{len}(u, v) - \pi_t(u) + \pi_t(v)$

Beweis: Folgt direkt aus Pseudocode

Zulässiges Potential (feasible potential)

Eine Potentialfunktion $\pi_t : V \to \mathbb{R}$ heißt *zulässig*, falls
$\text{len}(u, v) - \pi_t(u) + \pi_t(v) \geq 0$ für alle Kanten $(u, v) \in E$.

Es gilt für jeden Pfad $P = (s = v_1, \ldots, v_k = t)$

$$\begin{aligned}
\overline{\text{len}}(P) &= \sum_{i=1}^{k} \overline{\text{len}}(v_i, v_{i+1}) = \sum_{i=1}^{k} (\text{len}(v_i, v_{i+1}) - \pi_t(v_i) + \pi_t(v_{i+1})) \\
&= \left(\sum_{i=1}^{k} \text{len}(v_i, v_{i+1}) \right) - \pi_t(v_1) + \pi_t(v_k) \\
&= \text{len}(P) - \pi_t(s) + \pi_t(t).
\end{aligned}$$

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Ein Beispiel - Euklidische Ebene

- Knoten sind Punkte in der (euklidischen) Ebene
- Kantenlängen sind euklidische Abstände (d.h. $\text{len}(u, v) = \|u - v\|_2$).
- $\pi_t(v) = \|v - t\|_2$

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Ein Beispiel - Euklidische Ebene

- Knoten sind Punkte in der (euklidischen) Ebene
- Kantenlängen sind euklidische Abstände (d.h. $\text{len}(u, v) = \|u - v\|_2$).
- $\pi_t(v) = \|v - t\|_2$

π_t ist zulässiges Potential, denn

$$\text{len}(u, v) - \pi_t(u) + \pi_t(v) = \|u - v\|_2 - \|u - t\|_2 + \|v - t\|_2 \geq 0$$

wegen Dreiecksungleichung ($\triangle$-UGL)

$$\|u - v\|_2 + \|v - t\|_2 \geq \|u - t\|_2$$

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Ein Beispiel II

Idee:

- Knoten besitzen Geokoordinaten
- Kanten besitzen Fahrtgeschwindigkeiten
- es gibt eine Maximalgeschwindigkeit $v_{\max}$ in G
- nimm $\|u - t\|_2 / v_{\max}$ als Potential

Ein Beispiel II

Idee:

- Knoten besitzen Geokoordinaten
- Kanten besitzen Fahrtgeschwindigkeiten
- es gibt eine Maximalgeschwindigkeit $v_{\max}$ in G
- nimm $\|u - t\|_2 / v_{\max}$ als Potential

Ist zulässiges Potential, denn

$$\operatorname{len}(u, v) - \pi_t(u) + \pi_t(v) = \frac{\|u - v\|_2}{v_{(u,v)}} - \frac{\|u - t\|_2}{v_{\max}} + \frac{\|v - t\|_2}{v_{\max}} \geq 0$$

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Ein Beispiel II

Idee:

- Knoten besitzen Geokoordinaten
- Kanten besitzen Fahrtgeschwindigkeiten
- es gibt eine Maximalgeschwindigkeit $v_{\max}$ in G
- nimm $\|u - t\|_2 / v_{\max}$ als Potential

Ist zulässiges Potential, denn

$$\mathrm{len}(u, v) - \pi_t(u) + \pi_t(v) = \frac{\|u - v\|_2}{v_{(u,v)}} - \frac{\|u - t\|_2}{v_{\max}} + \frac{\|v - t\|_2}{v_{\max}} \geq 0$$

Probleme (bei Straßengraphen):

- Overhead zur Berechnung des Potentials $\|u - t\|_2 / v_{\max}$
- Schlechte untere Schranke an die tatsächliche Reisezeit
- deswegen praktisch keine Beschleunigung in Transportnetzen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Zulässige Potentiale – Eigenschaften

- Intuition: $\pi_t(v)$ ist ein Schätzwert für $d(v, t)$
- Falls π_t zulässig und $\pi_t(t) = 0$, so ist das Potential $\pi_t(v)$ eine untere Schranke für $d(v, t)$.
- Bessere untere Schranken ergeben kleinere Suchräume
- Ist $\pi_t(v) = d(v, t)$ für alle $v \in V$, so werden nur Knoten auf kürzesten s-t-Wegen abgearbeitet.

Kombinierbarkeit von Potentialen

Kombinierbarkeit von Potentialen

Seien π_1 und π_2 zulässige Potentiale. Dann ist $p = \max\{\pi_1, \pi_2\}$ (komponentenweises Maximum) auch ein zulässiges Potential.

Kombinierbarkeit von Potentialen

Kombinierbarkeit von Potentialen

Seien π_1 und π_2 zulässige Potentiale. Dann ist $p = \max\{\pi_1, \pi_2\}$ (komponentenweises Maximum) auch ein zulässiges Potential.

Herleitung

$$\begin{aligned}
\text{len}(u, v) - \pi_t^1(u) + \pi_t^1(v) &\geq 0 \\
\text{len}(u, v) - \pi_t^2(u) + \pi_t^2(v) &\geq 0
\end{aligned}$$

$$\begin{aligned}
\text{len}(u, v) - \pi_t^1(u) + \max\{\pi_t^1(v), \pi_t^2(v)\} &\geq 0 \\
\text{len}(u, v) - \pi_t^2(u) + \max\{\pi_t^1(v), \pi_t^2(v)\} &\geq 0
\end{aligned}$$

$$\text{len}(u, v) - \max\{\pi_t^1(u), \pi_t^2(u)\} + \max\{\pi_t^1(v), \pi_t^2(v)\} \geq 0$$

Dreiecksungleichung für Graphen

Für alle Knoten $s, u, t \in V$ gilt

$$d(s,u) + d(u,t) \geq d(s,t)$$

Beweis: $d(s,t)$ laut Definition kürzeste-Weg-Distanz. Weg über u muss also mindestens genauso lang sein.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Der ALT-Algorithmus

- Der ALT-Algorithmus ist A* mit einer speziellen vorberechneten Potentialfunktion
- ALT steht für A*, landmarks, triangle-inequality

Vorberechnung

- Dazu wird eine kleine Menge L (≈ 16) an Knoten ausgewählt (Landmarken)
- Für jede Landmarke ℓ und jeden Knoten $v \in V$ werden die Distanzen $d(v, \ell)$ und $d(\ell, v)$ vorberechnet (da Graph gerichtet)

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

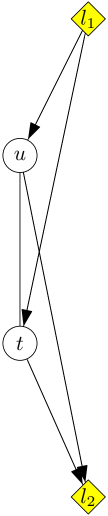
Potentiale durch Landmarken

$\triangle$-UGL

$$d(s,u) + d(u,t) \geq d(s,t)$$

also gilt im Beispiel für ℓ_1, ℓ_2:

$$\begin{aligned} d(\ell_1, u) + d(u,t) &\geq d(\ell_1, t) \\ d(u,t) + d(t, \ell_2) &\geq d(u, \ell_2) \end{aligned}$$



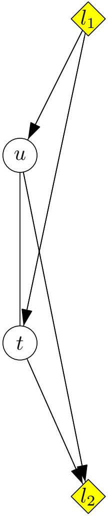
Potentiale durch Landmarken

$\triangle$-UGL

$$d(s,u) + d(u,t) \geq d(s,t)$$

also gilt im Beispiel für ℓ_1, ℓ_2:

$$\begin{aligned} d(u,t) &\geq d(\ell_1,t) - d(\ell_1,u) \\ d(u,t) &\geq d(u,\ell_2) - d(t,\ell_2) \end{aligned}$$



Potentiale durch Landmarken

$\triangle$-UGL

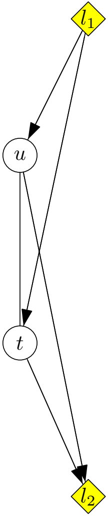
$$d(s,u) + d(u,t) \geq d(s,t)$$

also gilt im Beispiel für ℓ_1, ℓ_2:

$$\begin{aligned} d(u,t) &\geq d(\ell_1,t) - d(\ell_1,u) \\ d(u,t) &\geq d(u,\ell_2) - d(t,\ell_2) \end{aligned}$$

Benutze für alle Knoten $u, t, \ell \in V$ die Potentiale

$$\begin{aligned} \pi_t^+(u) &= d(\ell,t) - d(\ell,u) \quad \leq d(u,t) \\ \pi_t^-(u) &= d(u,\ell) - d(t,\ell) \quad \leq d(u,t) \end{aligned}$$



Potentiale durch Landmarken

Satz

Die Potentiale

$$\pi_t^{\ell+}(u) = d(\ell, t) - d(\ell, u)$$
$$\pi_t^{\ell-}(u) = d(u, \ell) - d(t, \ell)$$

sind zulässig für jede Landmarke $\ell \in L$.

Beweis: Mit $\triangle$-UGL für Graphen.

Korollar

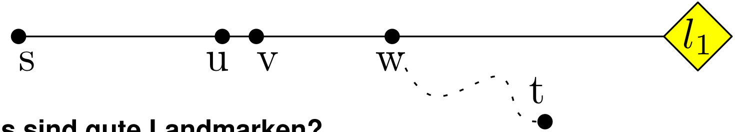
Für eine Menge L von Landmarken ist das Potential

$$\pi_t^{L}(u) = \max_{\ell \in L}\{\max\{\pi_t^{\ell+}(u), \pi_t^{\ell-}(u)\}\}$$

zulässig.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Eigenschaften Landmarken



Was sind gute Landmarken?

- $\pi_t^{\ell_1 -}(u) = d(u, \ell_1) - d(t, \ell_1)$, $\pi_t^{\ell_1 -}(v) = d(v, \ell_1) - d(t, \ell_1)$
- also $\text{len}_\pi(u, v) = \text{len}(u, v) - d(u, \ell_1) + d(v, \ell_1) = 0$

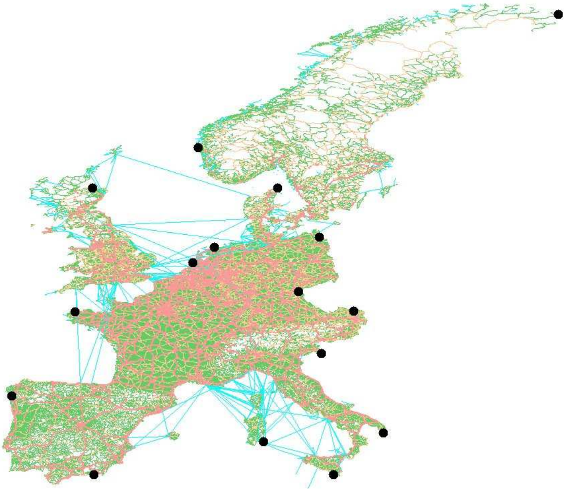
$\Rightarrow$ gemeinsame Kanten (kürzester Weg und Weg zur Landmarke) haben reduzierte Kosten von 0

Also:

- "gute" Landmarken überdecken viele Kanten für viele s, t-Paare
- trifft unter anderem zu, wenn "hinter" vielen Knoten
- am Rand des Graphen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Beispiel gute Landmarken



Berechnung Landmarken

mehrere Ansätze:

- brute force: teste alle Knotenteilmengen der Größe $|L|$:
  $\mathcal{O}(n^{|L|} \cdot \underbrace{n(m + n \log n)}_{\text{all pair shortest path}})$
  - $+$ höchste Beschleunigung
  - $-$ zu lange Vorberechnung

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Berechnung Landmarken

mehrere Ansätze:

- brute force: teste alle Knotenteilmengen der Größe $|L|$:
  $\mathcal{O}(n^{|L|} \cdot \underbrace{n(m + n \log n)}_{\text{all pair shortest path}})$
  - $+$ höchste Beschleunigung
  - $-$ zu lange Vorberechnung
- wähle zufällig
  - $+$ schnellste Vorberechnung
  - $-$ schlechte Beschleunigung

Berechnung Landmarken

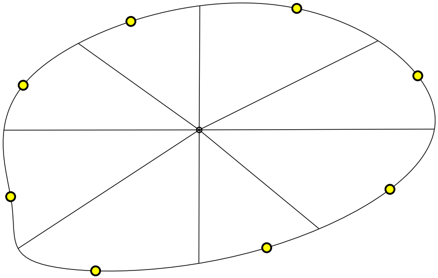
mehrere Ansätze:

- brute force: teste alle Knotenteilmengen der Größe $|L|$:
  $\mathcal{O}(n^{|L|} \cdot \underbrace{n(m + n \log n)}_{\text{all pair shortest path}})$
  - $+$ höchste Beschleunigung
  - $-$ zu lange Vorberechnung
- wähle zufällig
  - $+$ schnellste Vorberechnung
  - $-$ schlechte Beschleunigung
- mehrere Heuristiken, die versuchen den Rand zu finden
  - planar
  - farthest
  - avoid
  - lokale Optimierung (maxCover)

Planar-Landmarken

Vorgehen:

- suche Mittelpunkt c des Graphen
- teile Graphen in k Teile
- in jedem Teil wähle Knoten mit maximalen Abstand zu c als Landmarke



Anmerkungen:

- (benötigt planare Einbettung)
- liefert erstaunlich schlechte Ergebnisse

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Farthest-Landmarken

Karlsruher Institut für Technologie

Farthest-Landmarks(G,k)

$L \leftarrow \emptyset$
while $|L| < k$ **do**
    if $|L| = 0$ **then** DIJKSTRA(G, RANDOMNODE)
    else DIJKSTRA(G, L)
    $u \leftarrow$ last settled node
    $L \leftarrow L \cup \{u\}$

Anmerkungen:

- Multi-Startknoten Dijkstra
- schlecht für kleine k
- erste Landmarke schlecht
- weitere Landmarken massiv abhängig von erster

Avoid-Landmarken

Idee:

- Identifiziere rekursiv Teile des Graphen, für die die bisherigen Landmarken L keine guten Schranken liefern

Vorgehen:

- Berechne kürzeste-Wege-Baum T_r von (zufälligem) Knoten r
- Für jeden Knoten u: $\mathit{weight}(u) := d(r,u) - \pi_u^L(r)$
  Je größer das Gewicht desto schlechter die aktuelle Abschätzung für u
- Betrachte Teilbaum T_u:
  $$\mathit{size}(u) := \begin{cases} 0 & L \cap T_u \neq \varnothing \\ \sum_{v \in T_u} \mathit{weight}(v) & \mathit{sonst} \end{cases}$$
  Summe der Gewichte des Teilbaums von u
- Sei $w := \arg\max_{u \in T_r} \mathit{size}(u)$
  Teilbaum T_w hat (von r aus betrachtet) keine Landmarke “hinter” sich und in Summe die schlechteste Abschätzung
- Traversiere T_w, folge jeweils dem Kind maximaler Größe
- das erreichte Blatt wird zu L hinzugefügt

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Optimierung von Landmarken

Problem:

- konstruktive Heuristik
- anfangs gewählte Landmarken eventuell suboptimal

Idee:

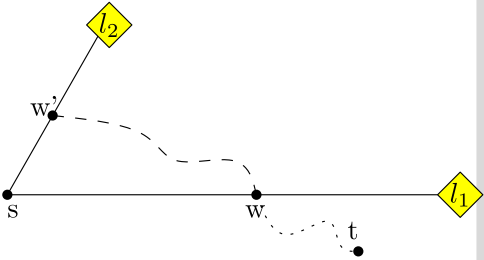
- lokale Optimierung
- berechne mehr landmarken als nötig ($\approx 4k$)
- wähle beste durch Optimierungsfunktion, z.B.
  1. maximiere Anzahl überdeckter Kanten, d.h. $\text{len}(u, v) - \pi(u) + \pi(v) = 0$
  2. maximiere $\pi(s)$ für 1 Mio. s-t Paare (simuliert Anfragen)
- Avoid + Funktion 1 wird maxCover genannt

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Anfragen: Aktive Landmarken

Problem:

- Landmarken können Suche in die falsche Richtung ziehen
- Auswertung von vielen Landmarken erzeugt großen overhead



Lösung:

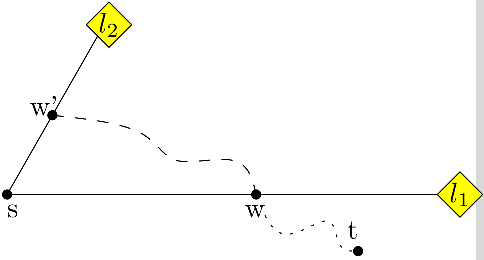
- wähle während Initialisierung eine Teilmenge von L als aktiv
- diese, für die $\pi_t^\ell(s)$ maximal sind
- Führe die Suche nur mit aktiven Landmarken aus

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Anfragen: Aktive Landmarken

Problem:

- Landmarken können Suche in die falsche Richtung ziehen
- Auswertung von vielen Landmarken erzeugt großen overhead



Verbesserungen:

- Benutze Heuristik um während der Suche die Wahl der aktiven Landmarken anzupassen.
- Starte mit zwei Landmarken, füge während der Suche weitere hinzu.
- Schlechte Landmarken können auch wieder inaktiviert werden.

Eine Intuition für den ALT-Suchraum

Der Suchraum von Dijkstra's Algorithmus

$$V(s,t) = \{v \in V \mid d(s,v) \leq d(s,t)\}$$

ist ein graphentheoretischer Kreis.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Eine Intuition für den ALT-Suchraum

Der Suchraum von ALT ist (vgl. Pseudocode)

$$V^L(s,t) \subseteq \{v \in V \mid d(s,v) + \pi^L(v) \leq d(s,t)\}$$

mit Potentialen

$$\begin{aligned}
\pi_t^L(v) &= \max_{\ell \in L}\{\max\{\pi_t^{\ell+}(v), \pi_t^{\ell-}(v)\}\} \\
\pi_t^{\ell+}(v) &= d(\ell,t) - d(\ell,v) \\
\pi_t^{\ell-}(v) &= d(v,\ell) - d(t,\ell)
\end{aligned}$$

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Eine Intuition für den ALT-Suchraum

Der Suchraum von ALT ist (vgl. Pseudocode)

$$V^L(s,t) \subseteq \{v \in V \mid d(s,v) + \pi^L(v) \leq d(s,t)\}$$

mit Potentialen

$$\begin{aligned}
\pi_t^L(v) &= \max_{\ell \in L}\{\max\{\pi_t^{\ell+}(v), \pi_t^{\ell-}(v)\}\} \\
\pi_t^{\ell+}(v) &= d(\ell,t) - d(\ell,v) \\
\pi_t^{\ell-}(v) &= d(v,\ell) - d(t,\ell)
\end{aligned}$$

Wir betrachten jedes $\ell \in L$ einzeln, einsetzen oben:

$$V_{\ell+}(s,t) = \{v \in V \mid d(s,v) + \pi_t^{\ell+}(v) \leq d(s,t)\}$$

$$V_{\ell-}(s,t) = \{v \in V \mid d(s,v) + \pi_t^{\ell-}(v) \leq d(s,t)\}$$

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Eine Intuition für den ALT-Suchraum

Der Suchraum von ALT ist (vgl. Pseudocode)

$$V^L(s,t) \subseteq \{v \in V \mid d(s,v) + \pi^L(v) \leq d(s,t)\}$$

mit Potentialen

$$\begin{aligned}
\pi_t^L(v) &= \max_{\ell \in L}\{\max\{\pi_t^{\ell+}(v), \pi_t^{\ell-}(v)\}\} \\
\pi_t^{\ell+}(v) &= d(\ell,t) - d(\ell,v) \\
\pi_t^{\ell-}(v) &= d(v,\ell) - d(t,\ell)
\end{aligned}$$

Wir betrachten jedes $\ell \in L$ einzeln, einsetzen oben:

$$V_{\ell^+}(s,t) = \{v \in V \mid d(s,v) + d(\ell,t) - d(\ell,v) \leq d(s,t)\}$$

$$V_{\ell^-}(s,t) = \{v \in V \mid d(s,v) + d(v,\ell) - d(t,\ell) \leq d(s,t)\}$$

Eine Intuition für den ALT-Suchraum

Der Suchraum von ALT ist (vgl. Pseudocode)

$$V^L(s,t) \subseteq \{v \in V \mid d(s,v) + \pi^L(v) \leq d(s,t)\}$$

mit Potentialen

$$\begin{aligned}
\pi_t^L(v) &= \max_{\ell \in L}\{\max\{\pi_t^{\ell+}(v), \pi_t^{\ell-}(v)\}\} \\
\pi_t^{\ell+}(v) &= d(\ell,t) - d(\ell,v) \\
\pi_t^{\ell-}(v) &= d(v,\ell) - d(t,\ell)
\end{aligned}$$

Wir betrachten jedes $\ell \in L$ einzeln, einsetzen oben:

$$V_{\ell^+}(s,t) = \{v \in V \mid d(s,v) - d(\ell,v) \leq d(s,t) - d(\ell,t)\}$$

$$V_{\ell^-}(s,t) = \{v \in V \mid d(s,v) + d(v,\ell) \leq d(s,t) + d(t,\ell)\}$$

Eine Intuition für den ALT-Suchraum

$$V_{\ell^-}(s,t) = \{v \in V \mid d(s,v) + d(v,\ell) \leq d(s,t) + d(t,\ell)\}$$

ist eine graphentheoretische Ellipse. (Mit Brennpunkten s und ℓ)

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Eine Intuition für den ALT-Suchraum

$$V_{\ell^-}(s,t) = \{v \in V \mid d(s,v) + d(v,\ell) \leq d(s,t) + d(t,\ell)\}$$

ist eine graphentheoretische Ellipse. (Mit Brennpunkten s und ℓ)

$$V_{\ell^+}(s,t) = \{v \in V \mid d(s,v) - d(\ell,v) \leq d(s,t) - d(\ell,t)\}$$

ist eine graphentheoretische Hyperbel. (Mit Brennpunkten s und ℓ)

Eine Intuition für den ALT-Suchraum

$$V_{\ell^-}(s,t) = \{v \in V \mid d(s,v) + d(v,\ell) \leq d(s,t) + d(t,\ell)\}$$

ist eine graphentheoretische Ellipse. (Mit Brennpunkten s und ℓ)

$$V_{\ell^+}(s,t) = \{v \in V \mid d(s,v) - d(\ell,v) \leq d(s,t) - d(\ell,t)\}$$

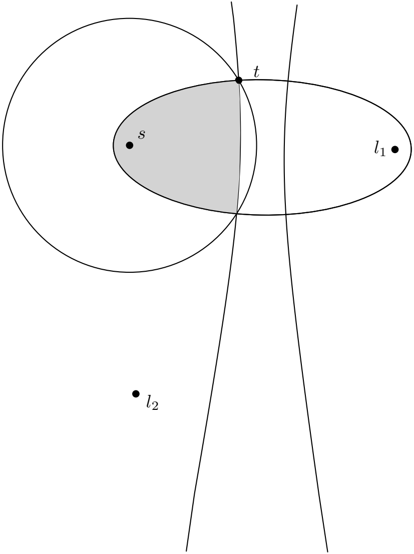
ist eine graphentheoretische Hyperbel. (Mit Brennpunkten s und ℓ)

Der Suchraum von ALT

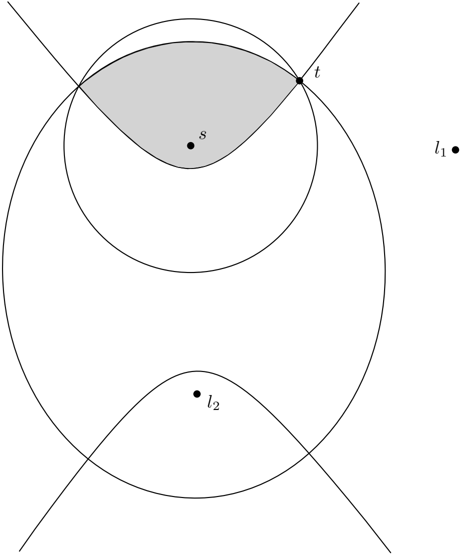
$$V^L(s,t) \subseteq \{v \in V \mid d(s,v) + \pi^L(v) \leq d(s,t)\}$$

ist also die Schnittmenge aus den Ellipsen und Hyperbeln über alle Landmarken in L.

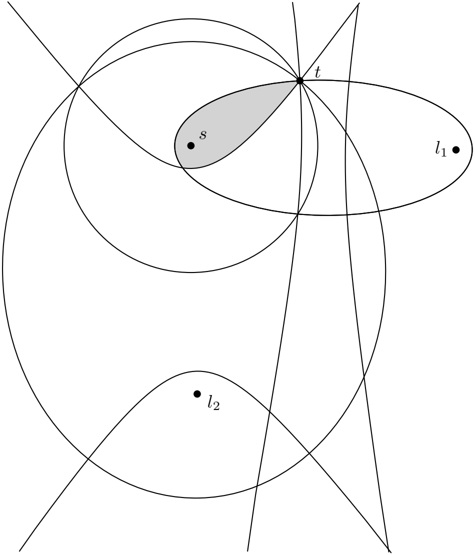
Eine Intuition für den ALT-Suchraum



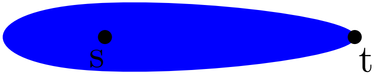
Eine Intuition für den ALT-Suchraum



Eine Intuition für den ALT-Suchraum

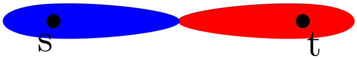


Bidirektionaler A*

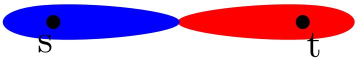


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Bidirektionaler A*



Bidirektionaler A*



Zur Erinnerung Bidirektionaler Dijkstra:

- alternierende/parallele Vorwärts- und Rückwärtssuche
- Vorwärts- $\overrightarrow{Q}$ und Rückwärtsqueue $\overleftarrow{Q}$
- Tentative Distanz μ
- Abbruchkriterium: $\mu \leq \mathit{minKey}(\overrightarrow{Q}) + \text{minKey}(\overleftarrow{Q})$?

Bidirektionaler A*

Erster Ansatz:

- benutze Vorwärtspot. π_f und Rückwärtspot. π_b

$$\pi_f(u) = \max_{\ell \in L}\{\max\{d(\ell,t) - d(\ell,u), d(u,\ell) - d(t,\ell)\}\}$$

$$\pi_b(u) = \max_{\ell \in L}\{\max\{d(\ell,u) - d(\ell,s), d(s,\ell) - d(u,\ell)\}\}$$

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Bidirektionaler A*

Erster Ansatz:

- benutze Vorwärtspot. π_f und Rückwärtspot. π_b

$$\pi_f(u) = \max_{\ell \in L}\{\max\{d(\ell,t) - d(\ell,u), d(u,\ell) - d(t,\ell)\}\}$$

$$\pi_b(u) = \max_{\ell \in L}\{\max\{d(\ell,u) - d(\ell,s), d(s,\ell) - d(u,\ell)\}\}$$

Problem:

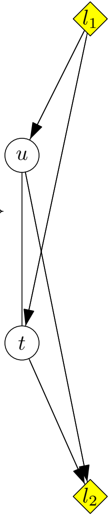
Bidirektionaler A*

Erster Ansatz:

- benutze Vorwärtspot. π_f und Rückwärtspot. π_b

$$\pi_f(u) = \max_{\ell \in L}\{\max\{d(\ell,t)-d(\ell,u), d(u,\ell)-d(t,\ell)\}\}$$

$$\pi_b(u) = \max_{\ell \in L}\{\max\{d(\ell,u)-d(\ell,s), d(s,\ell)-d(u,\ell)\}\}$$



Problem:

- Suchen operieren auf unterschiedlichen Längenfunktionen

$\Rightarrow$ Abbruchkriterium Bidirektionaler Dijkstra funktioniert nicht mehr

- konservatives Abbruchkriterium: stoppe erst, wenn $minKey(\overrightarrow{Q}) > \mu$ oder $minKey(\overleftarrow{Q}) > \mu$

Bidirektionaler A*

Zweiter Ansatz:

- Wann operieren Suchen auf dem gleichen Graphen?
- wenn reduzierte Kosten gleich:

$$\begin{aligned}
\mathrm{len}_{\pi_b}(v,u) &= \mathrm{len}_{\pi_f}(u,v) \\
\mathrm{len}(u,v) - \pi_b(v) + \pi_b(u) &= \mathrm{len}(u,v) - \pi_f(u) + \pi_f(v) \\
-\pi_b(v) + \pi_b(u) &= -\pi_f(u) + \pi_f(v) \\
\pi_b(u) + \pi_f(u) &= \pi_b(v) + \pi_f(v)
\end{aligned}$$

$\Rightarrow$ Also muss gelten: $\pi_b + \pi_f \equiv \text{const}$.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Bidirektionaler A*

Idee:

- Kombiniere π_f und π_b zu $p_f + p_b \equiv 0$:

$$p_f = \frac{\pi_f - \pi_b}{2} \qquad p_b = \frac{\pi_b - \pi_f}{2} = -p_f$$

Somit

- wie bidirektionaler Dijkstra
- aber mit $d(s, u) + p_f(u)$ und $d(v, t) + p_b(v)$ als keys
- stoppe wenn $minKey(\overrightarrow{Q}) + minKey(\overleftarrow{Q}) > \mu_{p_f} = \mu + p_f(s) - p_f(t)$
- dadurch bidirektional zielgerichtet

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Experimente

Eingaben:

- Straßennetzwerke
  - Europa: 18 Mio. Knoten, 42 Mio. Kanten
  - USA: 22 Mio. Knoten, 56 Mio. Kanten

Evaluation:

- Vorberechnung in Minuten und zusätzliche Bytes pro Knoten
- durchschnittlicher Suchraum (# abgearbeitete Knoten) und Suchzeiten (in *ms*) von 10 000 Zufallsanfragen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Zufallsanfragen ALT

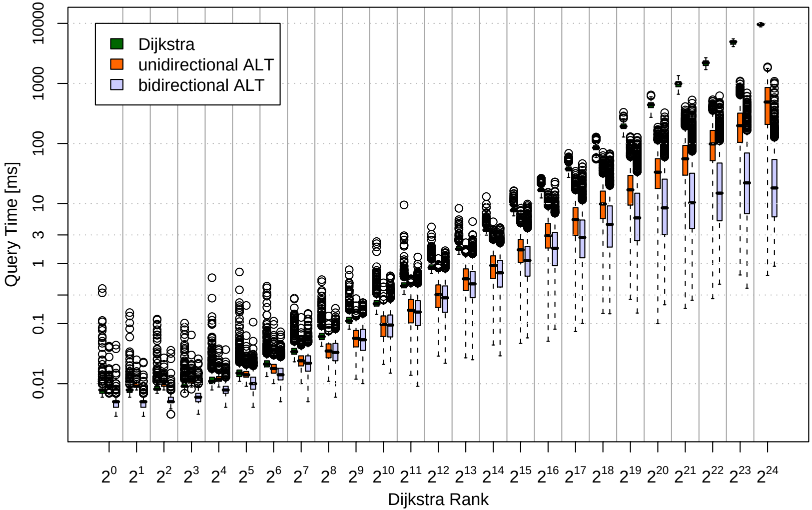
Vorberechnung: Bis 24 Landmarken maxCover, sonst avoid.

		Prepro		Query Unidir.			Query Bidir.		
	algorithm	time [min]	space [B/n]	# settled nodes	time [ms]	spd up	# settled nodes	time [ms]	spd up
EUR	Dijkstra	0	0	9 114 385	5 591.6	1.0	4 764 110	2 713.2	2.1
	ALT-4	12.1	32	1 289 070	469.1	11.9	355 442	254.1	22.0
	ALT-8	26.1	64	1 019 843	391.6	14.3	163 776	127.8	43.8
	ALT-16	85.1	128	815 639	327.6	17.1	74 669	53.6	104.3
	ALT-24	145.2	192	742 958	303.7	18.4	56 338	44.2	126.5
	ALT-32	27.1	256	683 566	301.4	18.6	40 945	29.4	190.2
	ALT-64	68.2	512	604 968	288.5	19.4	25 324	19.6	285.3
USA	Dijkstra	0	0	11 847 523	6 780.7	1.0	7 345 846	3 751.4	1.8
	ALT-8	44.5	64	922 897	329.8	20.6	328 140	219.6	30.9
	ALT-16	103.2	128	762 390	308.6	22.0	180 804	129.3	52.4
	ALT-32	35.8	256	628 841	291.6	23.3	109 727	79.5	85.3
	ALT-64	92.9	512	520 710	268.8	25.2	68 861	48.9	138.7

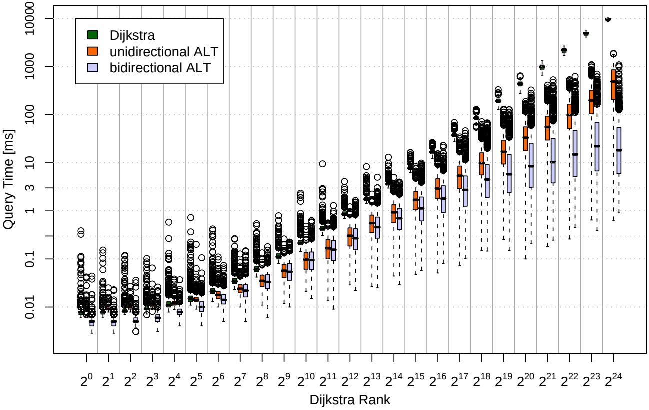
Beobachtungen

- bidirektionale Suche: Beschleunigung von 2
- unidirektionaler ALT: mehr als 16 Landmarken nicht sinnvoll
- bidirektionaler ALT: verdoppelung der Landmarken halbiert den Suchraum (ungefähr)
- 16 Landmarken: Beschleunigung $\approx$ 100 für Europa
- 64 Landmarken: Beschleunigung $\approx$ 300 für Europa
- hoher Speicherverbrauch (Graph-DS: 424 MB, pro Landmarke: 144 MB, Europa)
- USA schlechter als Europa (Vermutung: schlechtere Hierarchie)

Dijkstra-Rank – Bidirektionale Suche



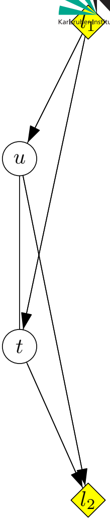
Dijkstra-Rank – Bidirektionale Suche



- Beschleunigung steigt mit Rang, gering für nahe Anfragen
- hohe Varianz, Ausreißer bis zu Faktor 100 langsamer als Median

Zusammenfassung ALT

- Zielgerichtet durch geänderte Reihenfolge, wie Knoten abgearbeitet werden
- wird erreicht durch hinzufügen eines Knotenpotentials π
- korrekt wenn $\text{len}(u, v) - \pi(u) + \pi(v) \geq 0$ für alle Kanten
- Potentiale durch Landmarken
- kann bidirektional gemacht werden
- Beschleunigung von bis zu 300 gegenüber Dijkstra
- aber hoher Platzverbrauch (Reduktion später)
- hohe Varianz (manche Anfragen haben keine guten Landmarken)

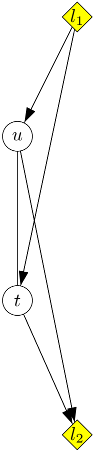


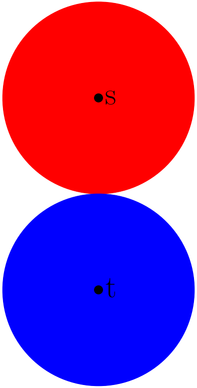
CALT

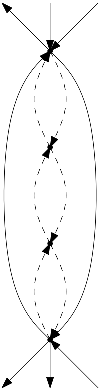
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Core-ALT

Landmarken	Bidirektionale Suche	Kontraktion	Arc-Flags	Table-Lookups

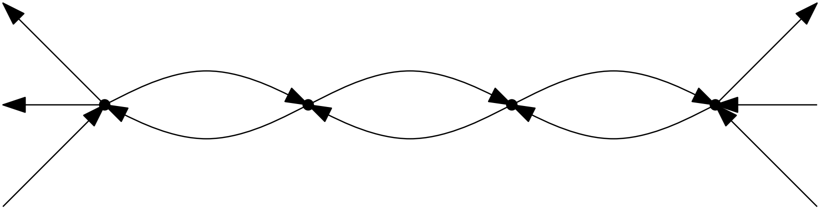






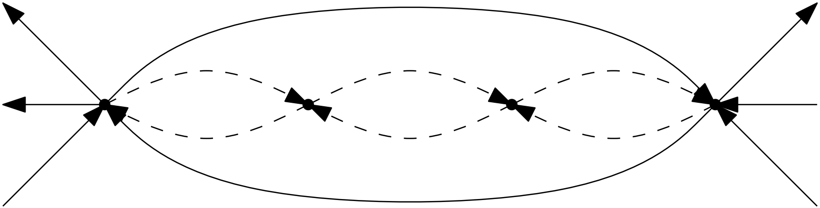
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Wdh: Kontraktion



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Wdh: Kontraktion



- Identifiziere “unwichtige” Knoten
- Menge “unwichtige” Knoten: Component, verbliebene: Core
- Füge Shortcuts zwischen Core-Knoten hinzu, um Distanz innerhalb Core zu erhalten
- Generelles Query-Framework: Betrachte ggfs. Component um s und t, nutze sonst nur Core und Shortcuts

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Core-ALT

Motivation:

- ALT ist robust gegenüber der Eingabe
- aber hoher Speicherverbrauch

Hauptidee:

- berechne ALT nur auf kleinem Subgraphen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Core-ALT

(Landmarken, bidirektionale Suche, Kontraktion)

Idee

- begrenze Beschleunigungstechnik auf kleinen Subgraphen (Kern)

Vorberechnung

- kontrahiere Graphen zu Kern
- Landmarken nur im Kern

Core-ALT

(Landmarken, bidirektionale Suche, Kontraktion)

Idee

- begrenze Beschleunigungstechnik auf kleinen Subgraphen (Kern)

Vorberechnung

- kontrahiere Graphen zu Kern
- Landmarken nur im Kern

Anfrage

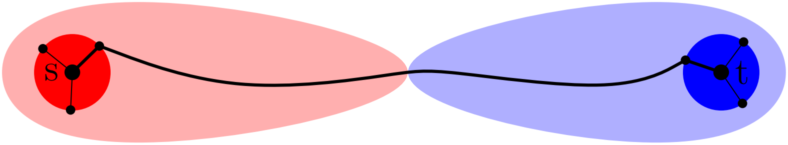
- Initialphase: normaler Dijkstra

Core-ALT

(Landmarken, bidirektionale Suche, Kontraktion)

Idee

- begrenze Beschleunigungstechnik auf kleinen Subgraphen (Kern)



Vorberechnung

- kontrahiere Graphen zu Kern
- Landmarken nur im Kern

Anfrage

- Initialphase: normaler Dijkstra
- Im Kern: benutze Landmarken und Shortcuts(!)

Proxy Knoten

Problem:

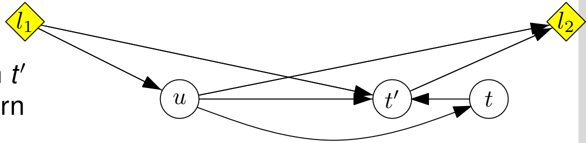
- ALT braucht Potential von jedem Knoten zu t und s
- s und/oder t könnten außerhalb des Kerns liegen
- somit keine Abstandswerte von den Landmarken zu s und t

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Proxy Knoten

Problem:

- ALT braucht Potential von jedem Knoten zu t und s
- s und/oder t könnten außerhalb des Kerns liegen
- somit keine Abstandswerte von den Landmarken zu s und t



Lösung:

- bestimme Proxy-Knoten t' für t (s analog), t' im Kern
- neue Ungleichungen:

$$
\begin{aligned}
d(u,t) &\geq d(u,l_2) - d(t',l_2) - d(t,t') \\
d(u,t) &\geq d(l_1,t') - d(l_1,u) - d(t,t')
\end{aligned}
$$

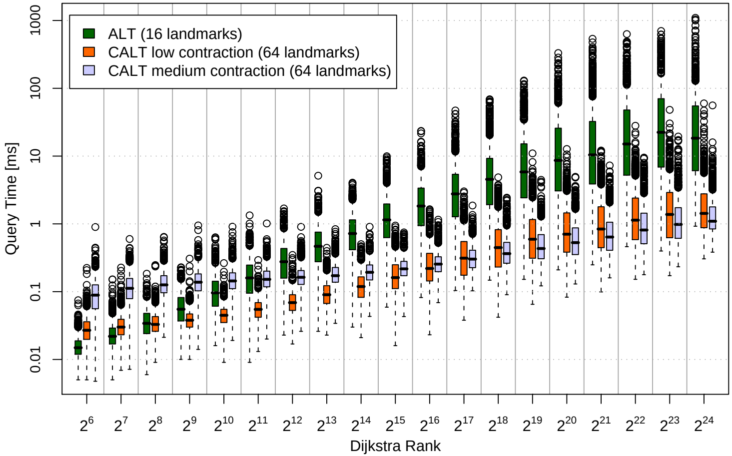
Einfluss Anzahl Landmarken

$\|L\|$	no cont. (c=0.0, h=0) PREPRO. time [min]	PREPRO. space [B/n]	QUERY #settled nodes	QUERY time [ms]	low cont. (c=1.0, h=20) PREPRO. time [min]	PREPRO. space [B/n]	QUERY #settled nodes	QUERY time [ms]
8	26.1	64	163 776	127.8	7.1	10.9	12 529	10.25
16	85.2	128	74 669	53.6	9.4	14.9	5 672	5.77
32	27.1	256	40 945	29.4	6.8	23.0	3 268	2.97
64	68.2	512	25 324	19.6	8.5	36.2	2 233	2.16

$\|L\|$	med: c=2.5, h=50 PREPRO. time [min]	PREPRO. space [B/n]	QUERY #settled nodes	QUERY time [ms]	high: c=5.0, h=100 PREPRO. time [min]	PREPRO. space [B/n]	QUERY #settled nodes	QUERY time [ms]
8	10.1	7.0	4 431	3.98	17.8	5.9	4 106	2.51
16	11.0	8.2	2 456	2.33	18.3	6.5	3 500	2.23
32	10.0	10.6	1 704	1.66	17.7	7.6	3 264	2.01
64	10.5	15.4	1 394	1.34	18.0	9.8	3 126	1.67

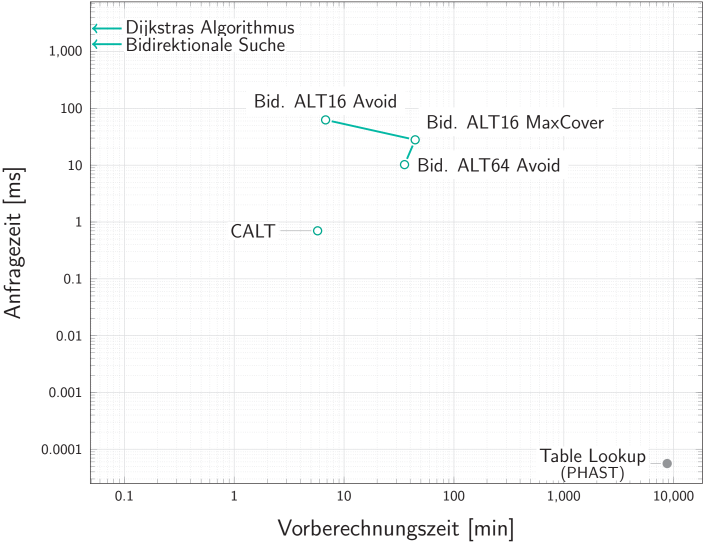
$|L| <= 16$: maxcover, $|L| >= 32$: avoid

Dijkstra Rank ALT vs CALT



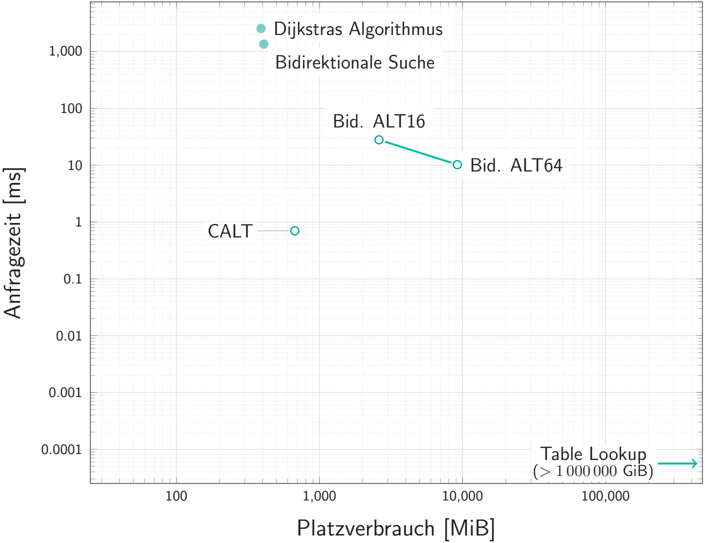
- Weit: CALT mehr als eine Größenordnung schneller als ALT, wobei kleinerer Kern ("medium contraction") besser
- Nah: je kleiner Kern, desto langsamer (da Initialphase unbeschl.)

Übersicht bisherige Techniken



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Übersicht bisherige Techniken



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Nächster Termin

Montag, 15.5.2016

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Literatur I

Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, and Dorothea Wagner.
Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm.
ACM Journal of Experimental Algorithmics, 15(2.3):1–31, January 2010.
Special Section devoted to WEA'08.

Andrew V. Goldberg and Chris Harrelson.
Computing the Shortest Path: A* Search Meets Graph Theory.
In *Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'05)*, pages 156–165. SIAM, 2005.

Andrew V. Goldberg and Renato F. Werneck.
Computing Point-to-Point Shortest Paths from External Memory.
In *Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX'05)*, pages 26–40. SIAM, 2005.