Karlsruher Institut für Technologie

Algorithmen für Routenplanung

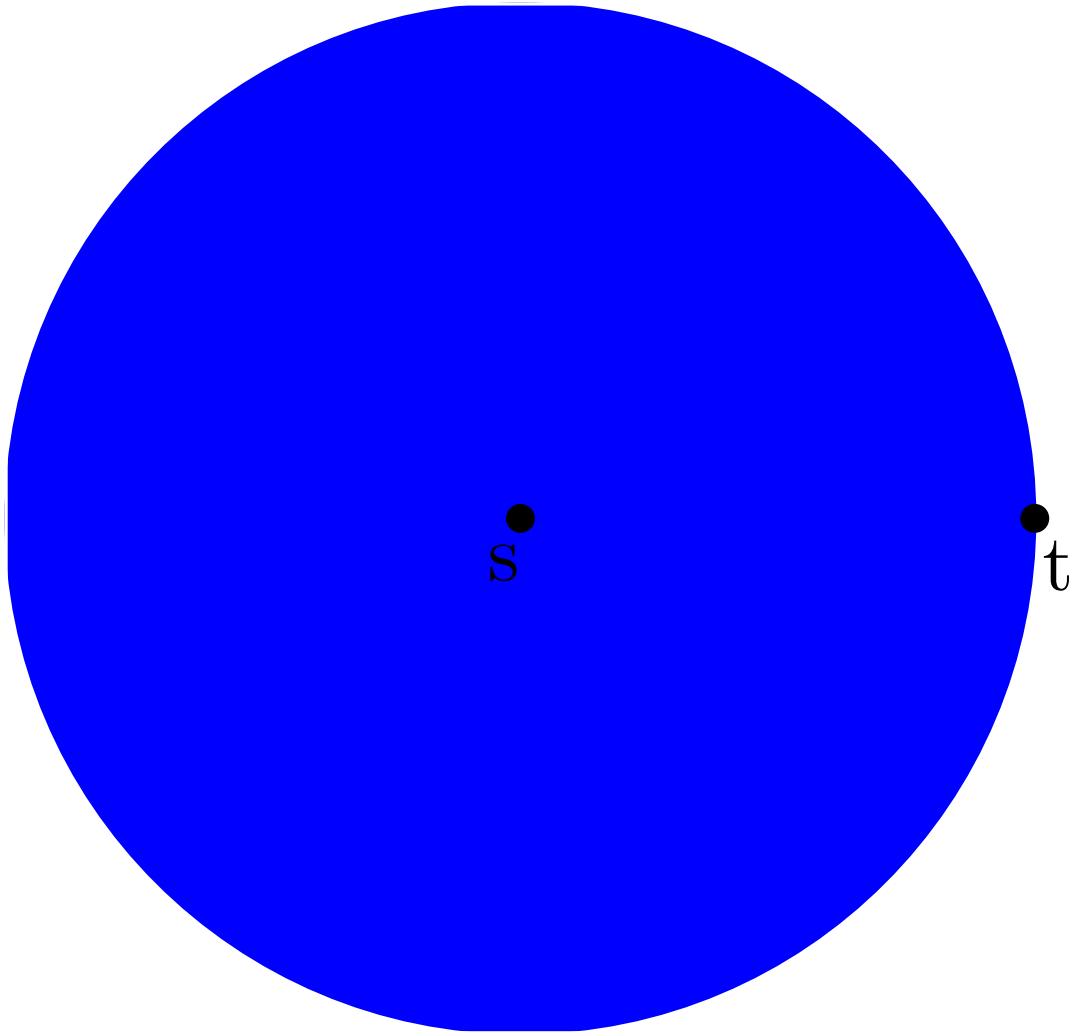
3. Termin, Sommersemester 2011

Reinhard Bauer | 21.11.2011

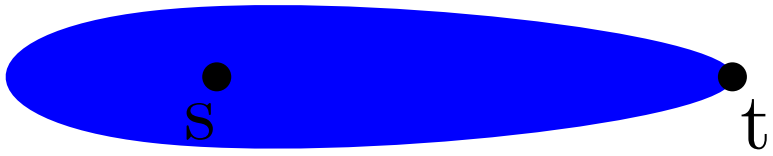
Institut für Theoretische Informatik · Algorithmik I · Prof. Dr. Dorothea Wagner

Zielgerichtete Suche

Schematischer Suchraum



Schematischer Suchraum



Ideensammlung

Wie Suche zielgerichtet machen?

Ideensammlung

Wie Suche zielgerichtet machen?

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

heute letzteres

A*

A*

- auch zielgerichtete Suche (goal-directed search) genannt

Idee:

- Dijkstra's Algorithmus benutzt $d[u]$ um zu entscheiden, welcher Knoten als nächstes abgearbeitet wird
- Benutze Potential $\pi_t : V \to \mathbb{R}$
- $\pi_t[v]$ ist ein Schätzwert für Entfernung $dist(v, t)$ zum Ziel t
- Benutze $d[u] + \pi_t(u)$ als Key in Q

Pseudocode A*

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

```
1  d[s] = 0
2  Q.clear(), Q.add(s, π_t(s))
3  while !Q.empty() do
4      u ← Q.deleteMin()
5      break if u = t
6      forall the edges e = (u, v) ∈ E do
7          if d[u] + len(e) < d[v] then
8              d[v] ← d[u] + len(e)
9              if v ∈ Q then Q.decreaseKey(v, d[v] + π_t(v))
10             else Q.insert(v, d[v] + π_t(v))
```

Frage

Können beliebige Potentialfunktionen benutzt werden?

Frage

Können beliebige Potentialfunktionen benutzt werden?

- Damit könnten (fast) beliebige Abarbeitungsreihenfolgen erzeugt werden
- Dies kann offensichtlich zu falschen Ergebnissen führen.

Frage

Können beliebige Potentialfunktionen benutzt werden?

- Damit könnten (fast) beliebige Abarbeitungsreihenfolgen erzeugt werden
- Dies kann offensichtlich zu falschen Ergebnissen führen.

Wie kommen wir also zu gültigen Potentialfunktionen?

A* – Äquivalente Formulierung

Gegeben: Graph $G = (V, E, \text{len})$, Knoten $s, t \in V$, Potential π_t

Betrachte: Graph $\overline{G} = (V, E, \overline{\text{len}})$ mit

$$\overline{\text{len}}(u, v) = \text{len}(u, v) - \pi_t(u) + \pi_t(v)$$

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

$$\begin{aligned}
\overline{\text{len}}(P) &= \sum_{i=1}^{k} \overline{\text{len}}(v_i, v_{i+1}) = \sum_{i=1}^{k} \text{len}(v_i, v_{i+1}) - \Pi_t(v_i) + \Pi_t(v_{i+1}) \\
&= -\Pi_t(s) + \Pi_t(v) + \sum_{i=1}^{k} \text{len}(v_i, v_{i+1}) = -\Pi_t(s) + \Pi_t(v) + \text{len}(P).
\end{aligned}$$

A* – Äquivalente Formulierung

Gegeben: Graph $G = (V, E, \text{len})$, Knoten $s, t \in V$, Potential π_t

Betrachte: Graph $\overline{G} = (V, E, \overline{\text{len}})$ mit

$$\overline{\text{len}}(u, v) = \text{len}(u, v) - \pi_t(u) + \pi_t(v)$$

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

$$\begin{aligned}
\overline{\text{len}}(P) &= \sum_{i=1}^{k} \overline{\text{len}}(v_i, v_{i+1}) = \sum_{i=1}^{k} \text{len}(v_i, v_{i+1}) - \Pi_t(v_i) + \Pi_t(v_{i+1}) \\
&= -\Pi_t(s) + \Pi_t(v) + \sum_{i=1}^{k} \text{len}(v_i, v_{i+1}) = -\Pi_t(s) + \Pi_t(v) + \text{len}(P).
\end{aligned}$$

Was können wir daraus folgern?

A* – Äquivalente Formulierung

Gegeben: Graph $G = (V, E, \text{len})$, Knoten $s, t \in V$, Potential π_t

Betrachte: Graph $\overline{G} = (V, E, \overline{\text{len}})$ mit

$$\overline{\text{len}}(u, v) = \text{len}(u, v) - \pi_t(u) + \pi_t(v)$$

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

$$\begin{aligned}
\overline{\text{len}}(P) &= \sum_{i=1}^{k} \overline{\text{len}}(v_i, v_{i+1}) = \sum_{i=1}^{k} \text{len}(v_i, v_{i+1}) - \Pi_t(v_i) + \Pi_t(v_{i+1}) \\
&= -\Pi_t(s) + \Pi_t(v) + \sum_{i=1}^{k} \text{len}(v_i, v_{i+1}) = -\Pi_t(s) + \Pi_t(v) + \text{len}(P).
\end{aligned}$$

Was können wir daraus folgern?
A* auf G ist gleich Dijkstra's Algorithmus auf $\overline{G}$

A* – Äquivalente Formulierung

Gegeben: Graph $G = (V, E, \text{len})$, Knoten $s, t \in V$, Potential π_t

Betrachte: Graph $\overline{G} = (V, E, \overline{\text{len}})$ mit

$$\overline{\text{len}}(u, v) = \text{len}(u, v) - \pi_t(u) + \pi_t(v)$$

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

$$\begin{aligned}
\overline{\text{len}}(P) &= \sum_{i=1}^{k} \overline{\text{len}}(v_i, v_{i+1}) = \sum_{i=1}^{k} \text{len}(v_i, v_{i+1}) - \Pi_t(v_i) + \Pi_t(v_{i+1}) \\
&= -\Pi_t(s) + \Pi_t(v) + \sum_{i=1}^{k} \text{len}(v_i, v_{i+1}) = -\Pi_t(s) + \Pi_t(v) + \text{len}(P).
\end{aligned}$$

Wie bekommen wir dann gültige Potentiale?

A* – Äquivalente Formulierung

Gegeben: Graph $G = (V, E, \text{len})$, Knoten $s, t \in V$, Potential π_t

Betrachte: Graph $\overline{G} = (V, E, \overline{\text{len}})$ mit

$$\overline{\text{len}}(u, v) = \text{len}(u, v) - \pi_t(u) + \pi_t(v)$$

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

$$\begin{aligned}\overline{\text{len}}(P) &= \sum_{i=1}^{k} \overline{\text{len}}(v_i, v_{i+1}) = \sum_{i=1}^{k} \text{len}(v_i, v_{i+1}) - \Pi_t(v_i) + \Pi_t(v_{i+1}) \\ &= -\Pi_t(s) + \Pi_t(v) + \sum_{i=1}^{k} \text{len}(v_i, v_{i+1}) = -\Pi_t(s) + \Pi_t(v) + \text{len}(P).\end{aligned}$$

Zulässiges Potential (feasible potential)

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

Ein Beispiel - Euklidische Ebene

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

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 ist zulässiges Potential, den

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

wegen Dreiecksungleichung ($\triangle$-UGL)

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

Ein Beispiel - Daten mit geographischer Herkunft

Idee:

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

Ein Beispiel - Daten mit geographischer Herkunft

Idee:

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

Ist gültiges Potential, denn

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

Ein Beispiel - Daten mit geographischer Herkunft

Idee:

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

Probleme (bei Straßengraphen):

- Overhead zur Berechnung des Potentials $||u - t||/v_{\max}$ groß
- Abschätzung eher schlecht
- deswegen praktisch keine Beschleunigung in Transportnetzen

Zulässige Potentiale – Eigenschaften

- Intuition: $\pi_t(v)$ ist ein Schätzwert für $\mathrm{dist}(v, t)$
- Falls $\pi_t(t) = 0$, so ist das Potential $p_t(v)$ eine untere Schranke für $\mathrm{dist}(v, t)$.
- **Faustregel:** (stimmt nur am Rand des Suchraums nicht) Bessere untere Schranken ergeben kleinere Suchräume
- Ist $\pi_t(v) = \mathrm{dist}(v, t)$ 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 gültiges Potential.

Herleitung

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

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

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

Die Dreiecksungleichung für Graphen

Für alle Knoten $s, u, t \in V$ gilt

$$\text{dist}(s, u) + \text{dist}(u, t) \geq \text{dist}(s, t)$$

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 (sog. Landmarken) ausgewählt
- Für jede Landmarke l und jeden Knoten $v \in V$ werden die Distanzen $d(v, l)$ und $d(l, v)$ vorberechnet

Potentiale durch Landmarken

$\triangle$-UGL

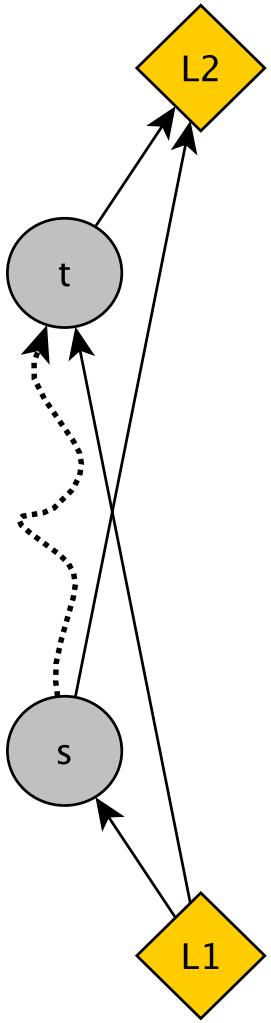
$$\text{dist}(s,u) + \text{dist}(u,t) \geq \text{dist}(s,t)$$

also gilt für alle Knoten $u, t, l \in V$

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

Benutze die Potentiale

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



Potentiale durch Landmarken

Satz

Die Potentiale

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

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

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

Korollar

Für eine Menge L von Landmarken ist das Potential

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

zulässig.

Eine Intuition für den ALT-Suchraum

Der Suchraum von Dijkstra's Algorithmus

$$V_d = \{v \in V \mid \text{dist}(s, v) \leq \text{dist}(s, t)\}$$

ist ein graphentheoretischer Kreis. Der Suchraum von ALT ist

$$V^L(s, t) \subseteq \{v \in V \mid \text{dist}(s, v) + \Pi^L(v) \leq \text{dist}(s, t)\}$$

mit Potentialen

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

Kann dies anschaulich dargestellt werden?

Eine Intuition für den ALT-Suchraum

Der Suchraum von ALT ist

$$V^L(s,t) \subseteq \{v \in V \mid \text{dist}(s,v) + \Pi^L(v) \leq \text{dist}(s,t)\}$$

mit Potentialen

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

Wir betrachten jedes $l \in L$ einzeln

$$V_{l+}(s,t) = \{v \in V \mid \text{dist}(s,v) + \text{dist}(v,l) \leq \text{dist}(s,t) + \text{dist}(t,l)\}$$

$$V_{l-}(s,t) = \{v \in V \mid \text{dist}(s,v) - \text{dist}(l,v) \leq \text{dist}(s,t) - \text{dist}(l,t)\}$$

Eine Intuition für den ALT-Suchraum

$$V_{l+}(s,t) = \{v \in V \mid \text{dist}(s,v) + \text{dist}(v,l) \leq \text{dist}(s,t) + \text{dist}(t,l)\}$$

ist eine graphentheoretische Ellipse.

$$V_{l-}(s,t) = \{v \in V \mid \text{dist}(s,v) - \text{dist}(l,v) \leq \text{dist}(s,t) - \text{dist}(l,t)\}$$

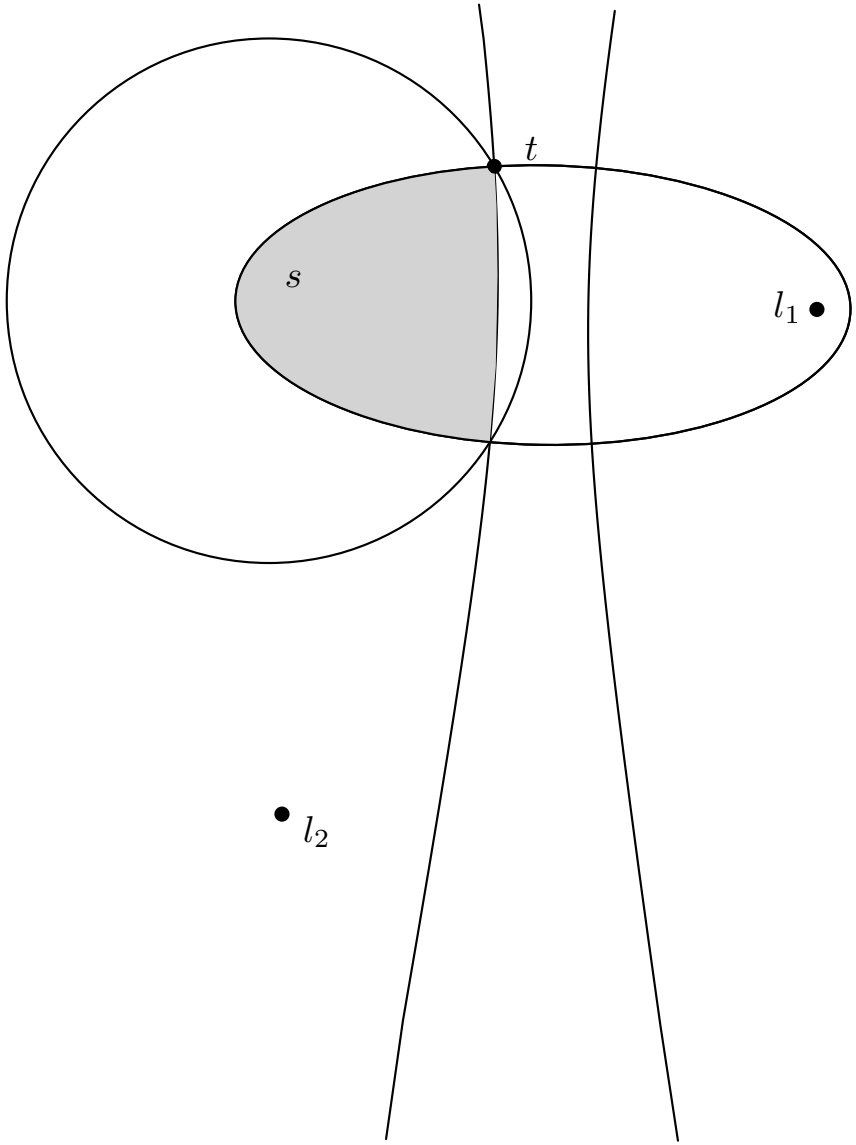
ist eine graphentheoretische Hyperbel.

Der Suchraum

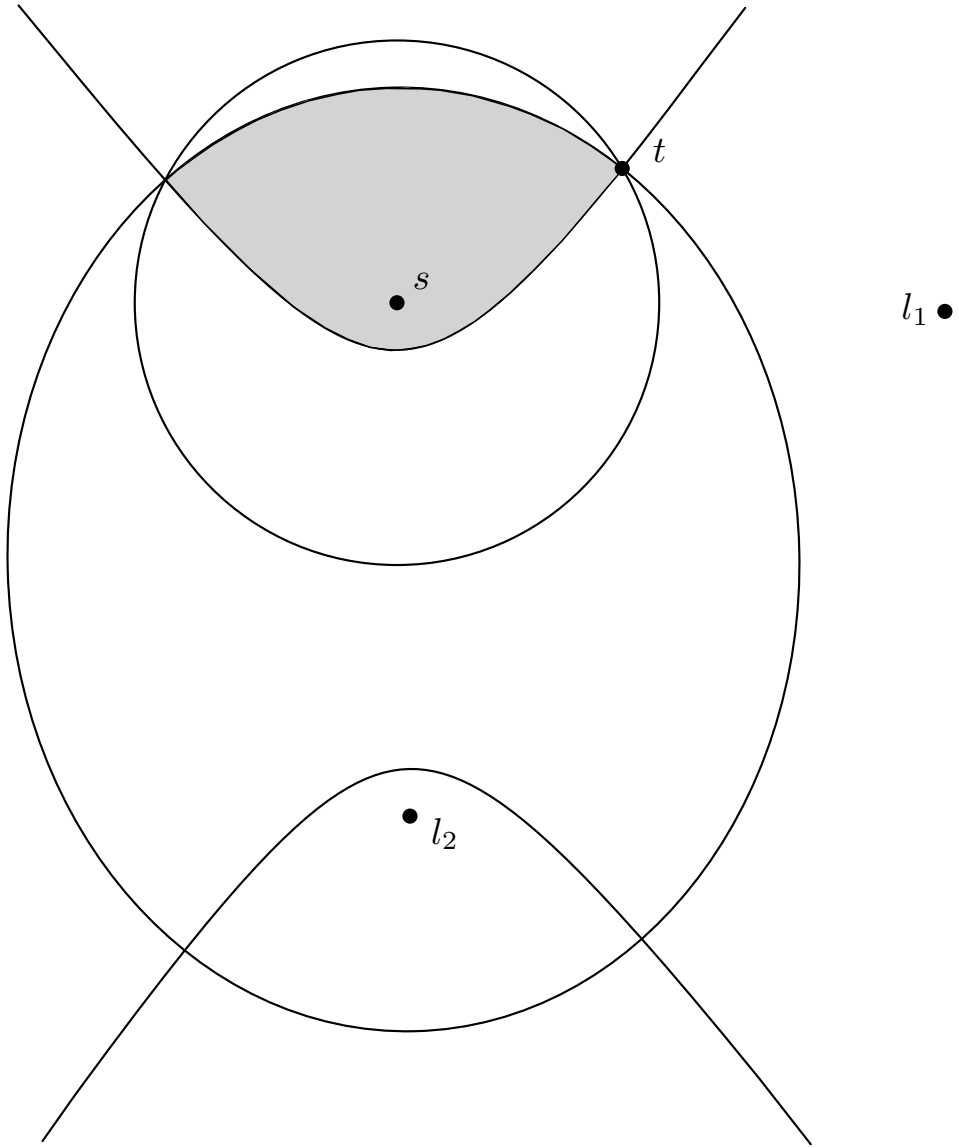
$$V^L(s,t) \subseteq \{v \in V \mid \text{dist}(s,v) + \Pi^L(v) \leq \text{dist}(s,t)\}$$

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

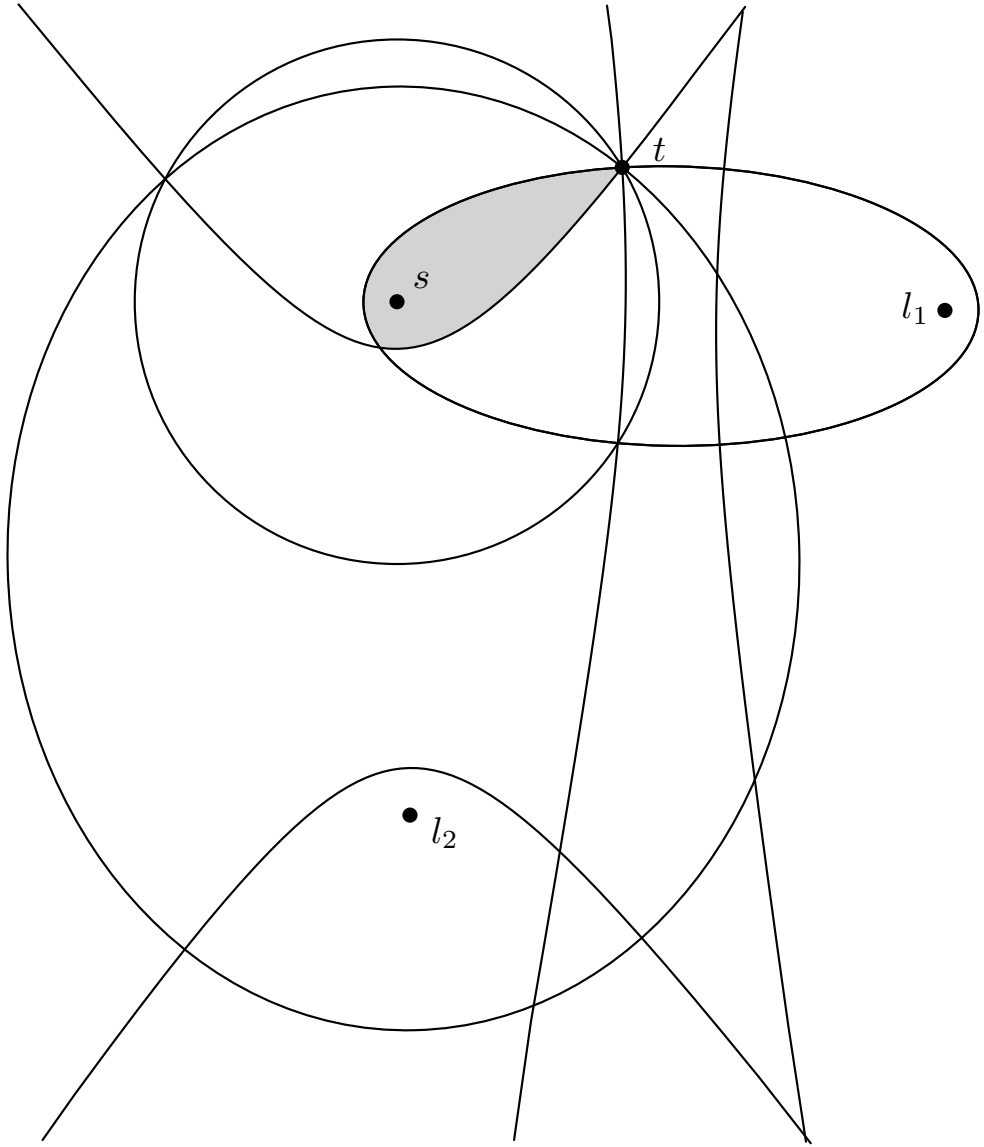
Eine Intuition für den ALT-Suchraum



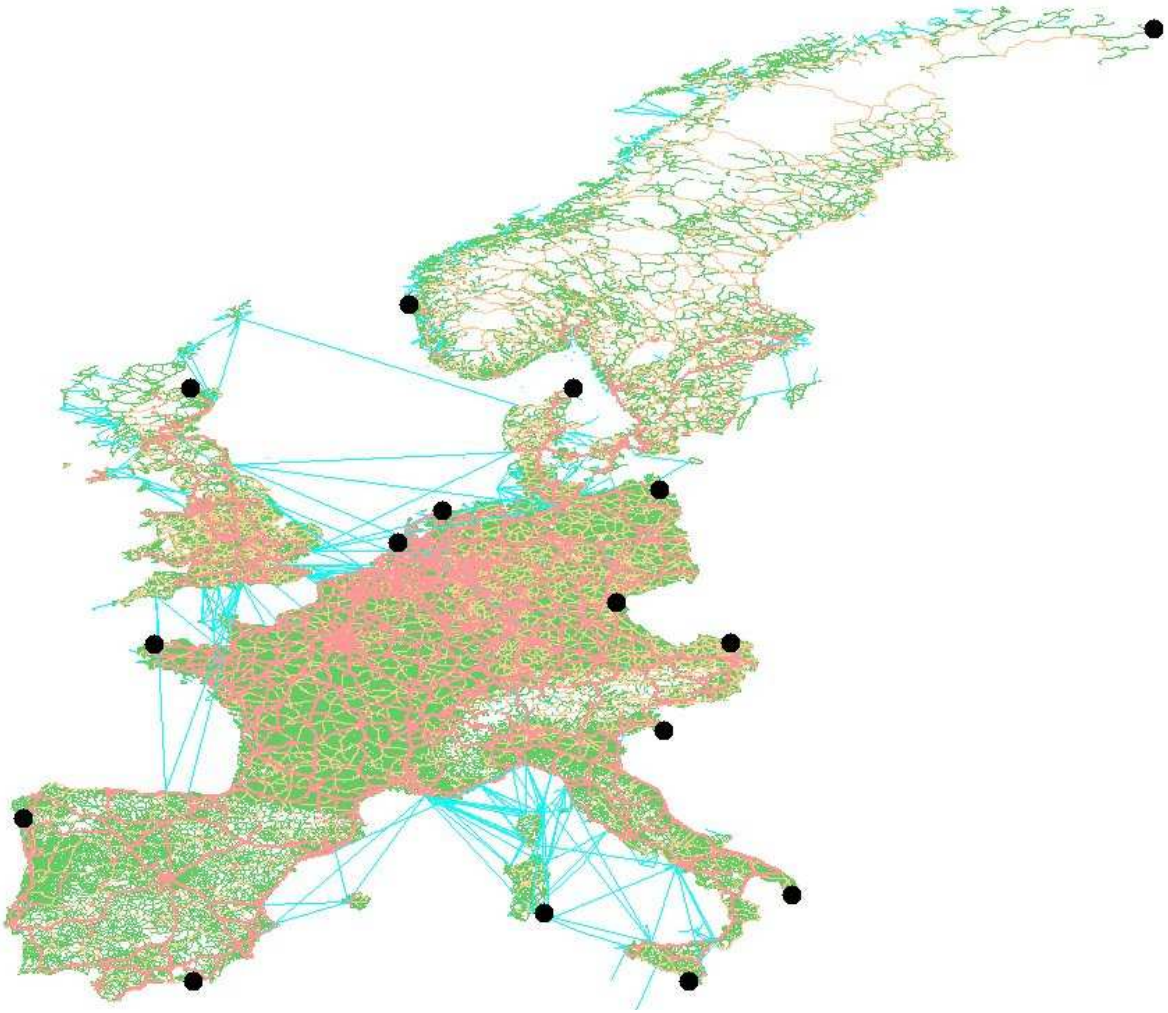
Eine Intuition für den ALT-Suchraum



Eine Intuition für den ALT-Suchraum



Beispiel gute Landmarken



Berechnung Landmarken

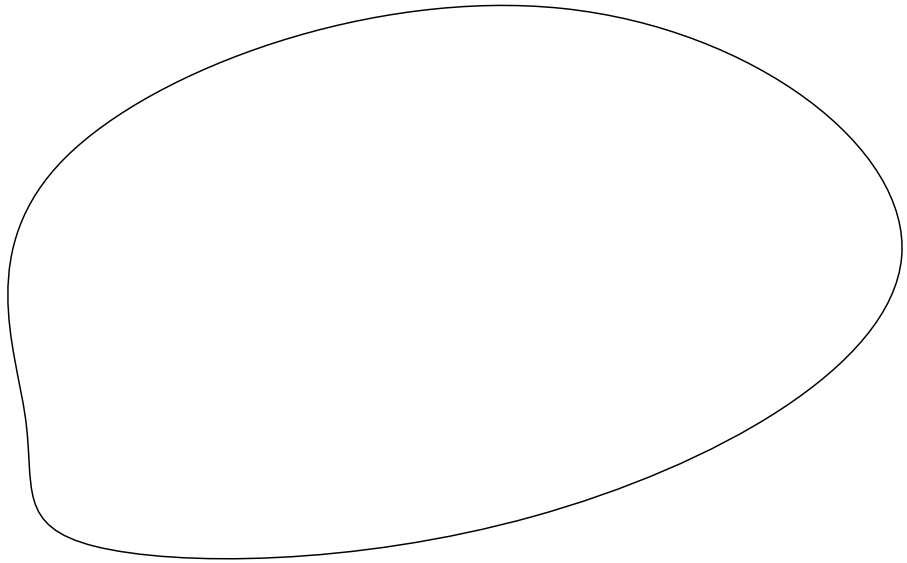
mehrere Ansätze:

- brute force
  - \+ 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



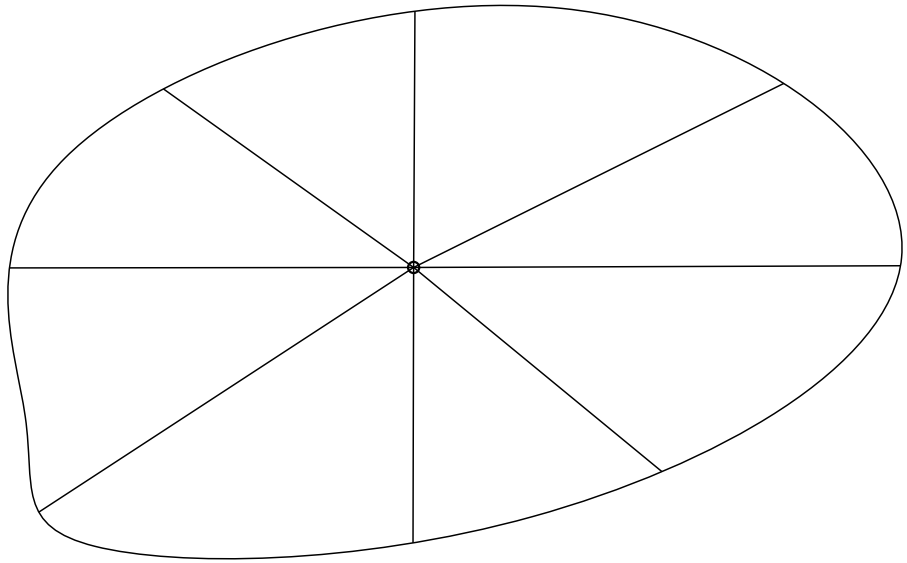
Anmerkungen:

- benötigt planare Einbettung
- liefert erstaunlich schlechte Ergebnisse

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



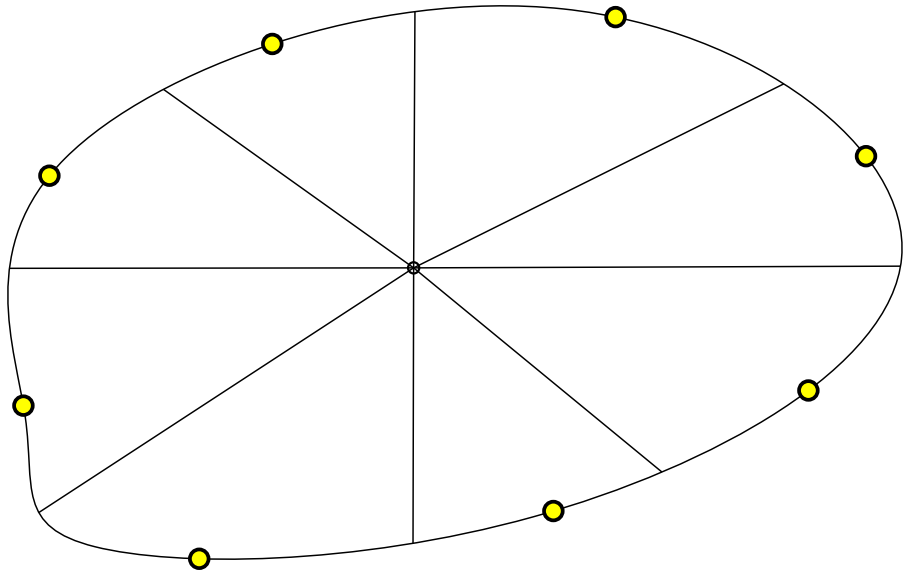
Anmerkungen:

- benötigt planare Einbettung
- liefert erstaunlich schlechte Ergebnisse

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



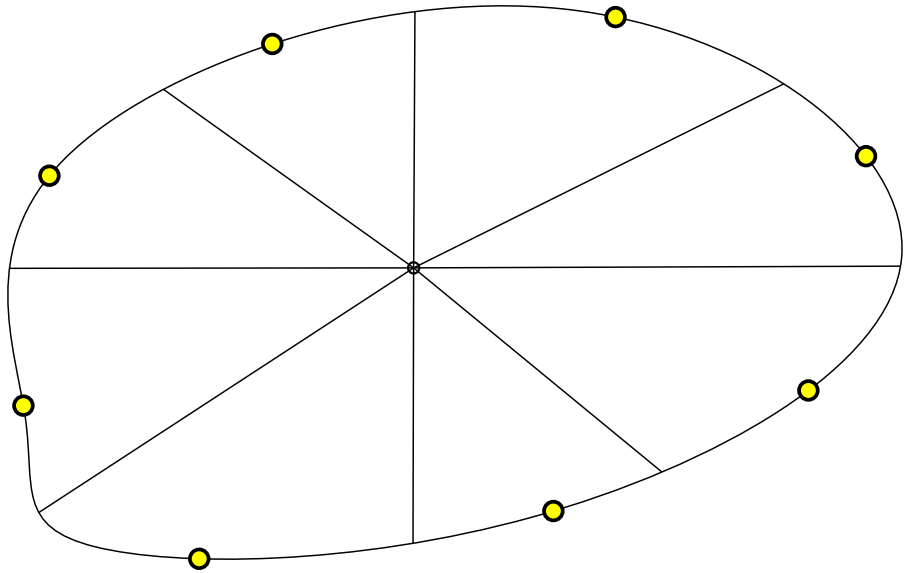
Anmerkungen:

- benötigt planare Einbettung
- liefert erstaunlich schlechte Ergebnisse

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

Farthest-Landmarken

FARTHEST-LANDMARKS(G,k)

```
1 L ← ∅
2 while |L| < k do
3     if |L| = 0 then DIJKSTRA(G,RANDOMNODE)
4     else DIJKSTRA(G,L)
5     u ← last settled node
6     L ← L ∪ {u}
```

Anmerkungen:

- DIJKSTRA(G,L) : Multi-Startknoten Dijkstra:
  Wie Dijkstra, aber mit mehreren Startknoten gleichzeitig
- schlecht für kleine k
- erste Landmarke schlecht
- weitere Landmarken massiv abhängig von erster

Avoid-Landmarken

Vorgehen:

- berechne kürzeste Wege Baum T_r von einem zufälligen Knoten r
- weight$(u) = d(u, r) - \underline{d(u, r)}$
- size(u) Summe der Gewichte (weight) seiner Nachfolger in T_r
- size$(u) = 0$ wenn mindestens ein Nachfolger in T_r eine Landmarke ist
- w sei der Knoten mit maximaler Größe (size)
- traversiere T_r startend von w, folge immer dem Knoten mit maximaler Größe
- das erreichte Blatt wird zu L hinzugefügt

Anmerkungen:

- Verfeinerung von Farthest Strategie

Optimierung von Landmarken

Problem:

- konstruktive Heuristik
- anfangs gewähle Landmarken eventuell suboptimal

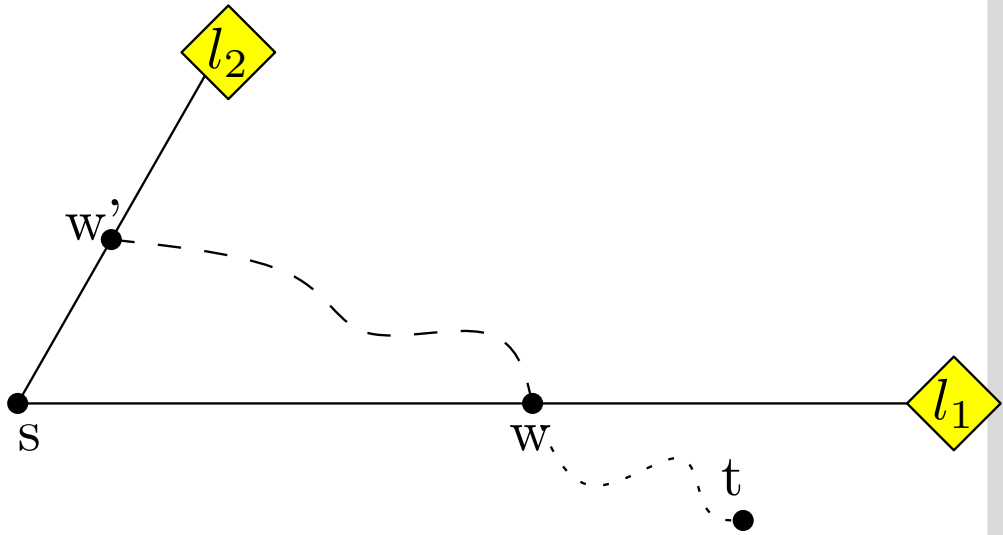
Ideen:

- lokale Suche
- Benutze Heuristik die iterativ Landmarken löscht und wieder auffüllt
- Generiere dabei zuviele Landmarken und benutze heuristische Bewertung der Güte von Landmarken
- Selektiere damit schlechte Landmarken aus

Aktive Landmarken

Problem:

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



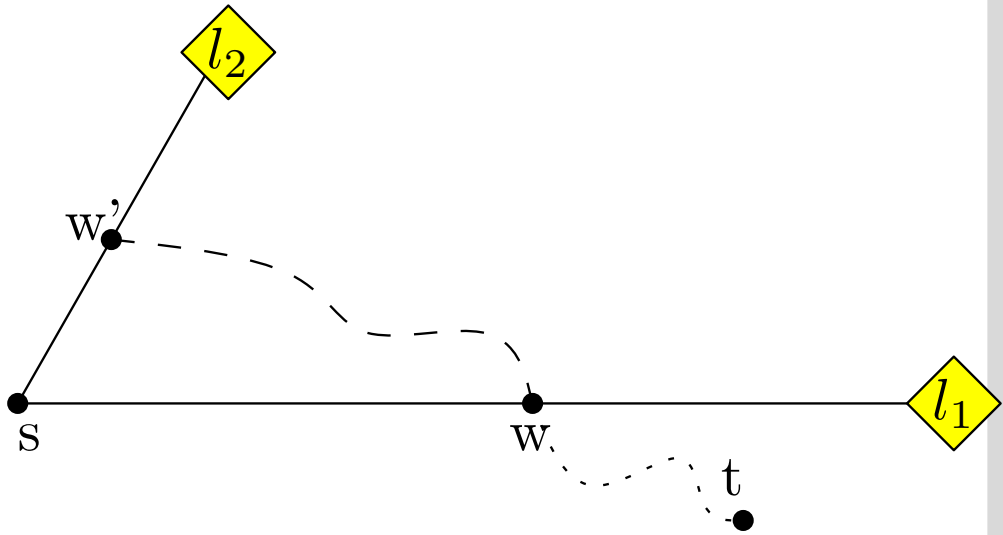
Lösung:

- wähle während Initialisierung eine Teilmenge von L als aktiv
- diese, für die $\pi_t^l(s)$ maximal sind
- solche Landmarken liefern die besten Schranken und sind (hoffentlich) die besten
- Führe die Suche nur mit aktiven Landmarken aus

Aktive Landmarken

Problem:

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



Verbesserung:

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

Dynamische Strategie aus [GW05]

„Update attempts happen whenever a search scans a vertex v whose distance estimate to the destination, as determined by the current lower bound function, is smaller than a certain value (called checkpoint).

At this point, the algorithm verifies if the best lower bound on the distance from v to the destination (using all landmarks) is at least a factor $1 + \epsilon$ better than the current lower bound (using only active landmarks). If so, the landmark yielding the improved lower bound is activated. In our experiments, we used $\epsilon = 0.01$.

Checkpoint are determined by the original lower bound b on the distance between s and t, calculated before the computation starts. The i-th checkpoint for each search has value $b(10 - i)/10$.

Dynamische Strategie aus [GW05]

After every landmark update, the potential function changes, and we must update the priority queues as described in the previous section.

This rule works well when *s* and *t* are reasonably far apart; when they are close, update attempts would happen too often, thus dominating the running time of the algorithm.

Therefore, we also require the algorithm to scan at least 100 vertices between two consecutive update attempts in the same direction “

Datenstruktur

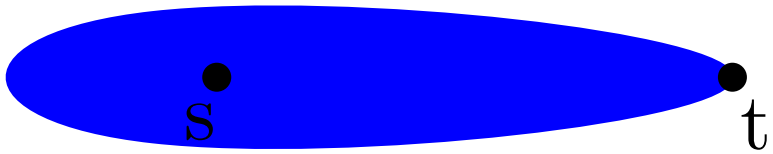
Erster Ansatz:

- $2 \cdot |L|$ 32-bit Vektoren der Größe n
- Problem: viele Cache-Misses

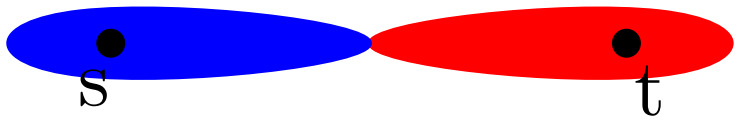
Besser:

- 1 64-bit Vektor der Größe $|L| \cdot n$
- speichere Distanz von und zu Landmarke in einem 64-bit Integer
- Zugriff auf Knoten mit id k und Landmarken-Nummer l in Segment $l \cdot k + l$
- dadurch deutlich erhöhte Lokalität
- beschleunigt die Anfragezeit um ca. einen Faktor von 4

Bidirektionaler A*



Bidirektionaler A*



Bidirektionaler A*

Erster Ansatz:

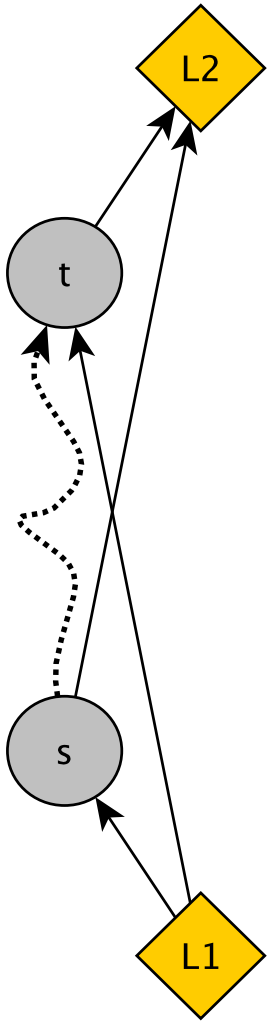
- benutze Vorwärtspot. π_f und Rückwärtspot. π_b

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

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

Problem:

- Suchen operieren auf unterschiedlichen Längenfunktionen
- konservatives Abbruchkriterium: Stoppe erst, wenn $\text{minKey}(\overrightarrow{Q}) > \mu$ oder $\text{minKey}(\overleftarrow{Q}) > \mu$



Bidirektionaler A*

Erster Ansatz:

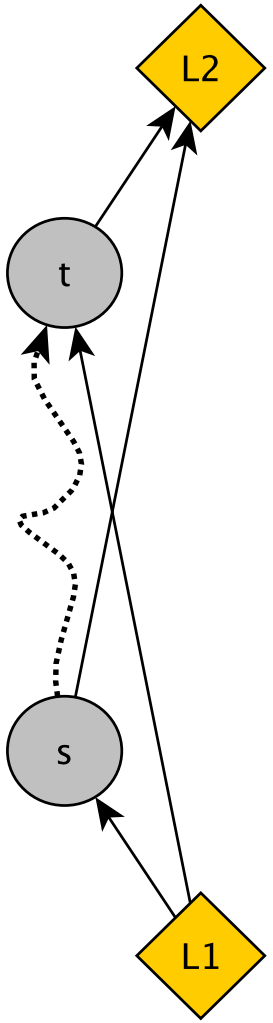
- benutze Vorwärtspot. π_f und Rückwärtspot. π_b

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

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

Problem:

- Suchen operieren auf unterschiedlichen Längenfunktionen
- konservatives Abbruchkriterium: Stoppe erst, wenn $\text{minKey}(\overrightarrow{Q}) > \mu$ oder $\text{minKey}(\overleftarrow{Q}) > \mu$



Bidirektionaler A*

Zweiter Ansatz:

- Wann operieren Suchen auf dem gleichen Graphen?
- wenn

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

$\pi_r + \pi_f \equiv \text{const.}$

Bidirektionaler A*

Idee:

- nehme Kombination aus π_f und π_r

$$p_f = \frac{\pi_f - \pi_r}{2} \qquad p_r = \frac{\pi_r - \pi_f}{2} = -p_f$$

Somit

- wie bidirektionaler Dijkstra
- aber mit $d(s,u) + p_f(u)$ und $d(v,t) + p_r(v)$ als Keys
- stoppe wenn

$$\text{minKey}(\overrightarrow{Q}) + \text{minKey}(\overleftarrow{Q}) > \mu_{p_f} = \mu + p_f(s) - p_f(t)$$

- dadurch bidirektional zielgerichtet

ALT [Goldberg et al. 04,05]

- **A***, **L**andmarken, **T**riangle inequality (Dreiecksungleichung)
- bidirektionaler Landmarken-A* (2. Ansatz)
- aktive Landmarken
- Pruning
- meist 16 Landmarken
- meist avoid oder maxCover Landmarken

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

Zufallsanfragen ALT

		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	851	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

- 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-Datenstruktur: 424 MB
  - pro Landmarke: 144 MB in Europa
- USA schlechter als Europa (Vermutung: schlechtere Hierarchie)

Dijkstra-Rank

Problem:

- Zufallsanfragen geben wenig Informationen
- Wie ist die Varianz?
- Werden nahe oder ferne Anfragen beschleunigt?

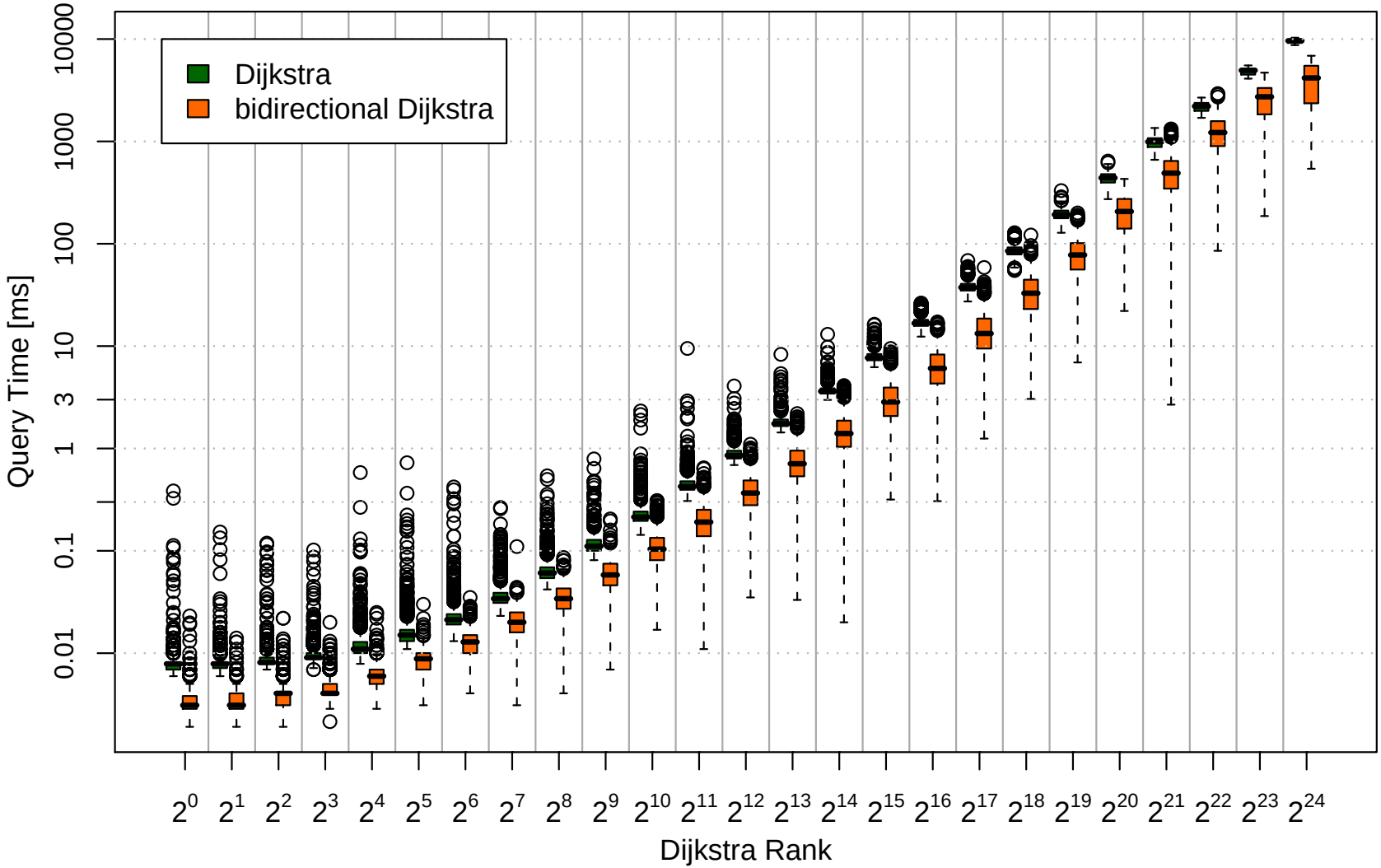
Idee:

- DIJKSTRA definiert für gegebenen Startknoten s Ordnung $\prec_s$ auf den Knoten durch

$$u \prec_s v :\Leftrightarrow d[u] \leq d[v]$$

- DIJKSTRA-Rang $r_s(u)$ eines Knoten u für gegebenes s: Index von u bezüglich $\prec_s$
- wähle 1000 Startknoten und analysiere jeweils die Suchzeiten um die Knoten mit Rang $2^1, \ldots, 2^{\log n}$ zu finden

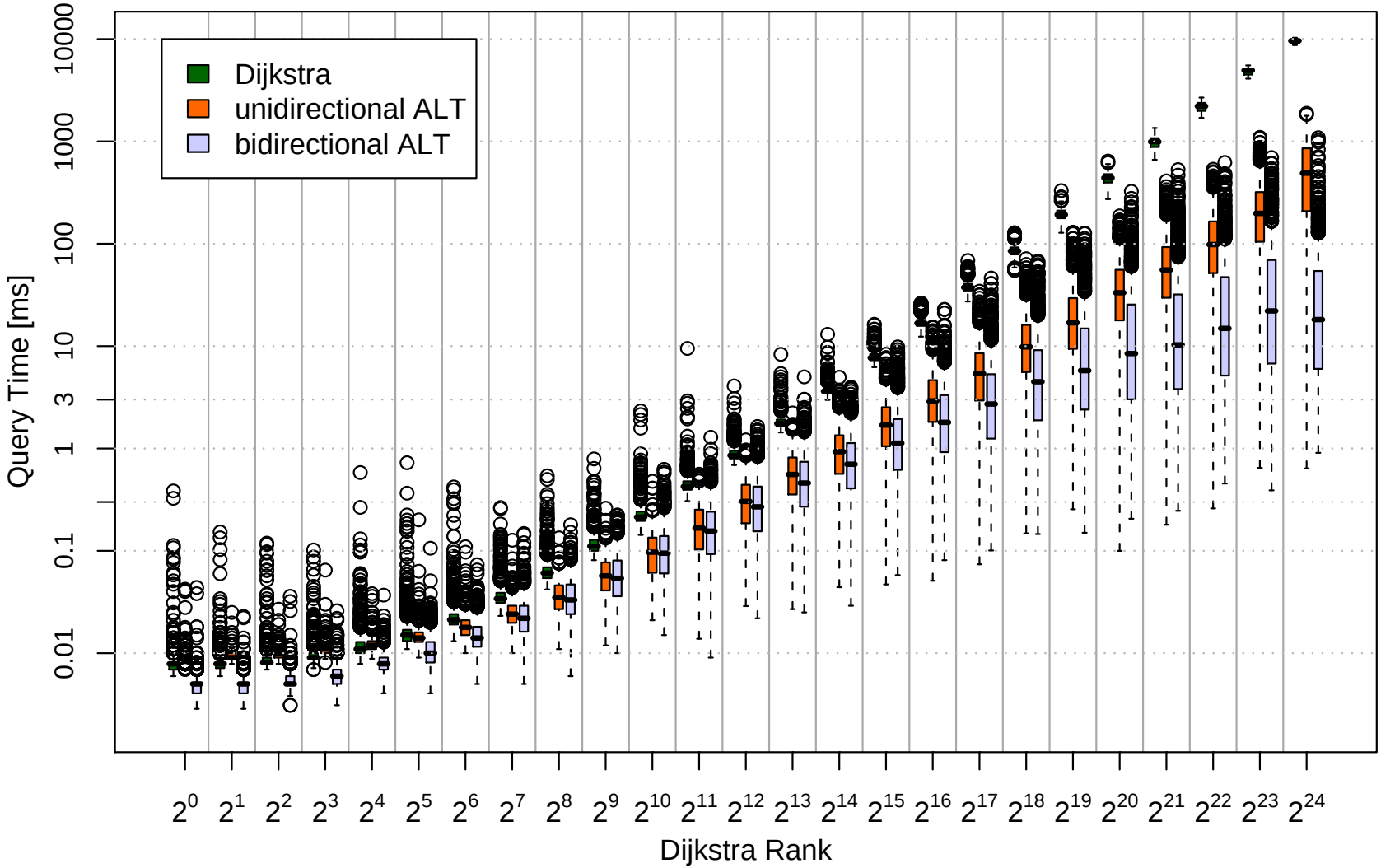
Lokale Anfragen: Bidir. Suche



Beobachtungen

- Ausreißer bei nahen Anfragen (vor allem unidirektional)
- Beschleunigung unabhängig vom Rang (immer ca. Faktor 2)
- Varianz etwas höher als bei unidirektionaler Suche
- manche Anfragen sind sehr schnell

Lokale Anfragen: ALT



Beobachtungen

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

Ende

Literatur:

- 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)*, 2005 pages 156–165.
- 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)*, 2005 pages 26–40.