Algorithmen für Routenplanung – Vorlesung 2

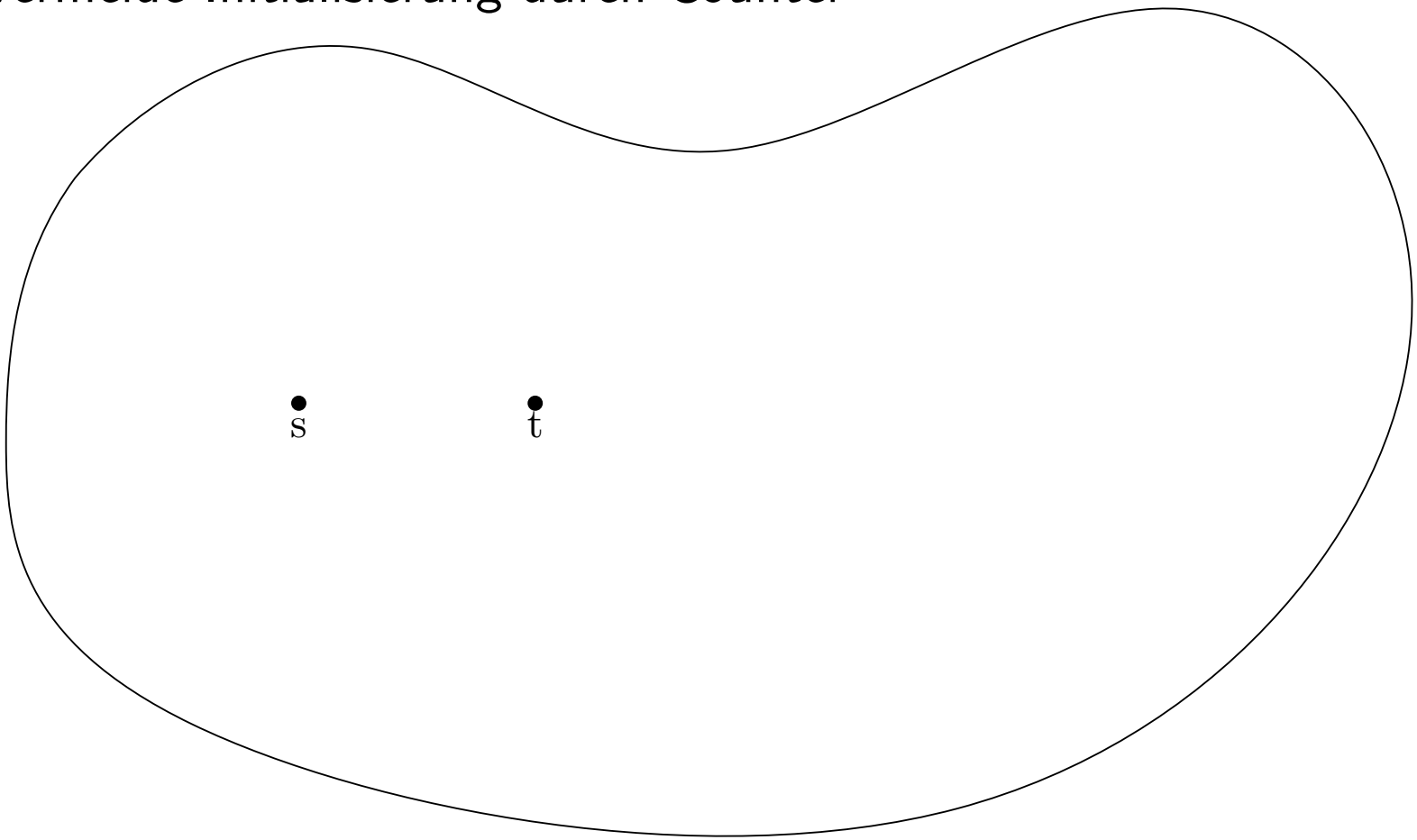
Daniel Delling

Lehrstuhl für Algorithmik I
Institut für theoretische Informatik
Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825

2. Beschleunigungstechniken

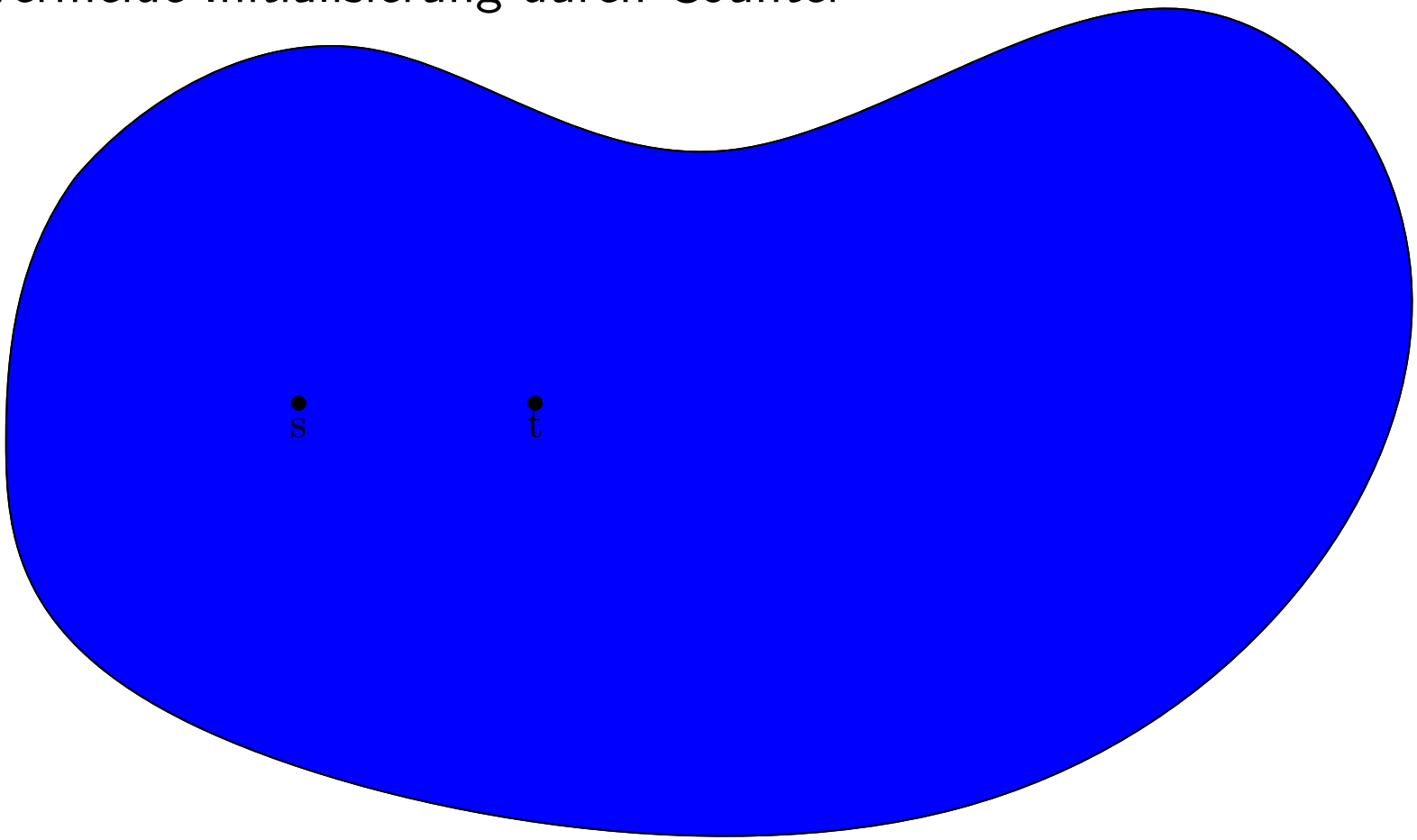
Letztes Mal

- breche Suche ab, sobald Ziel abgearbeitet worden ist
- vermeide Initialisierung durch Counter



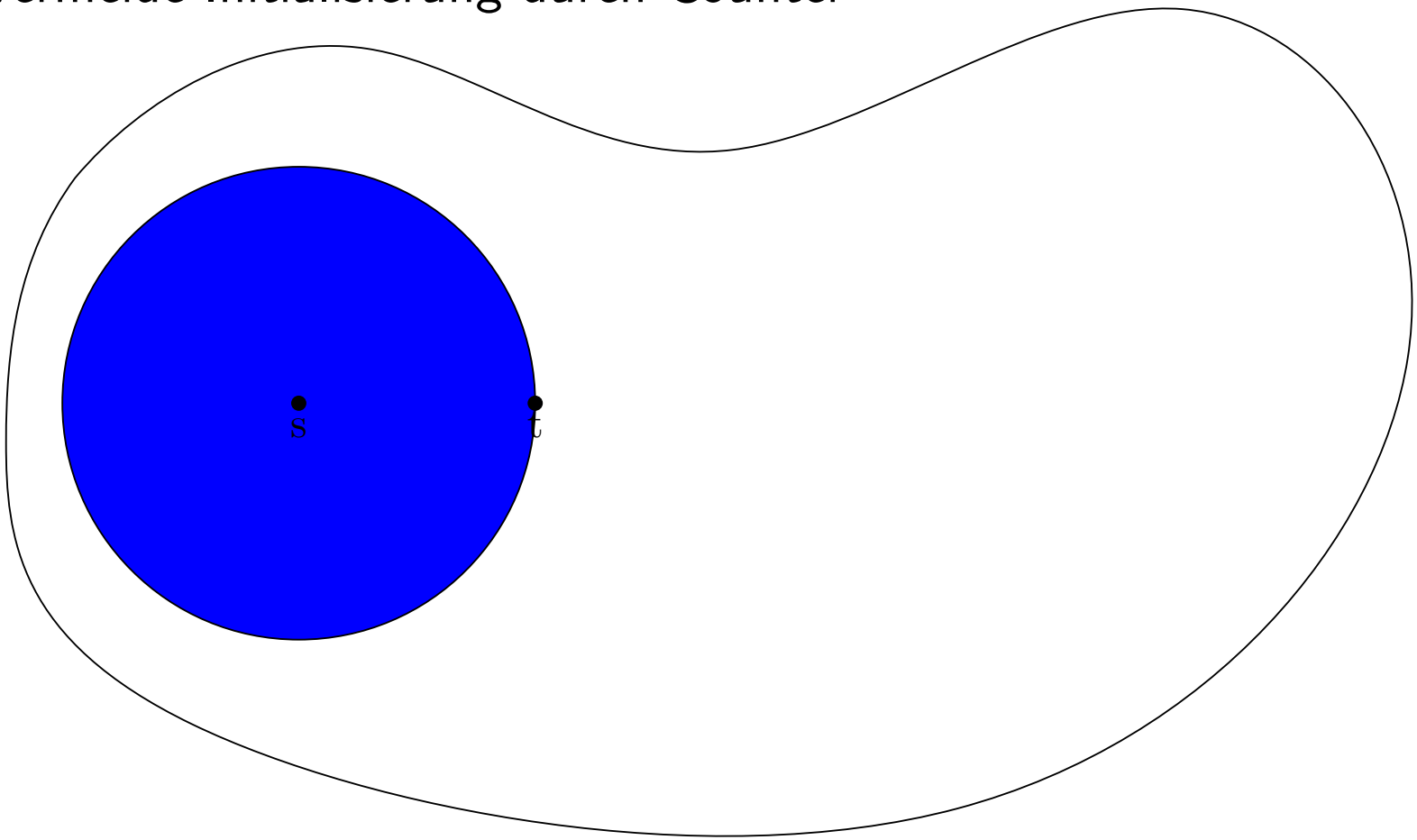
Letztes Mal

- breche Suche ab, sobald Ziel abgearbeitet worden ist
- vermeide Initialisierung durch Counter



Letztes Mal

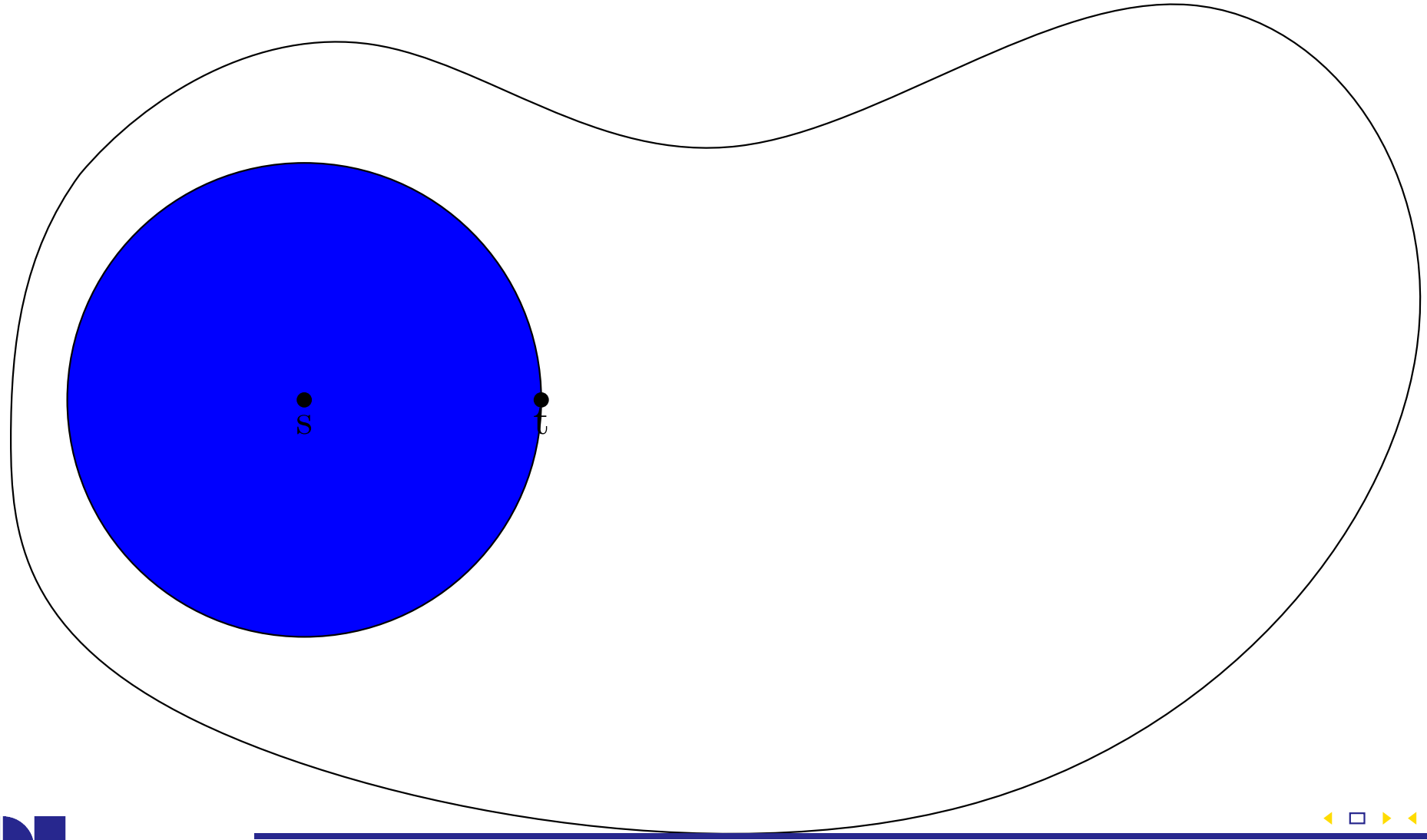
- breche Suche ab, sobald Ziel abgearbeitet worden ist
- vermeide Initialisierung durch Counter



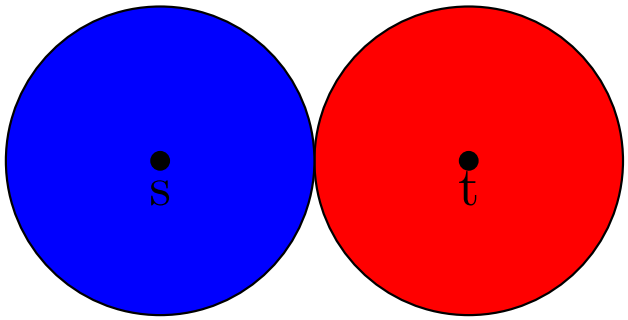
Themen Heute

- bidirektionale Suche
- A* Suche
- A* mit Landmarken
- bidirektionale A* Suche

Bidirektionale Suche



Bidirektionale Suche



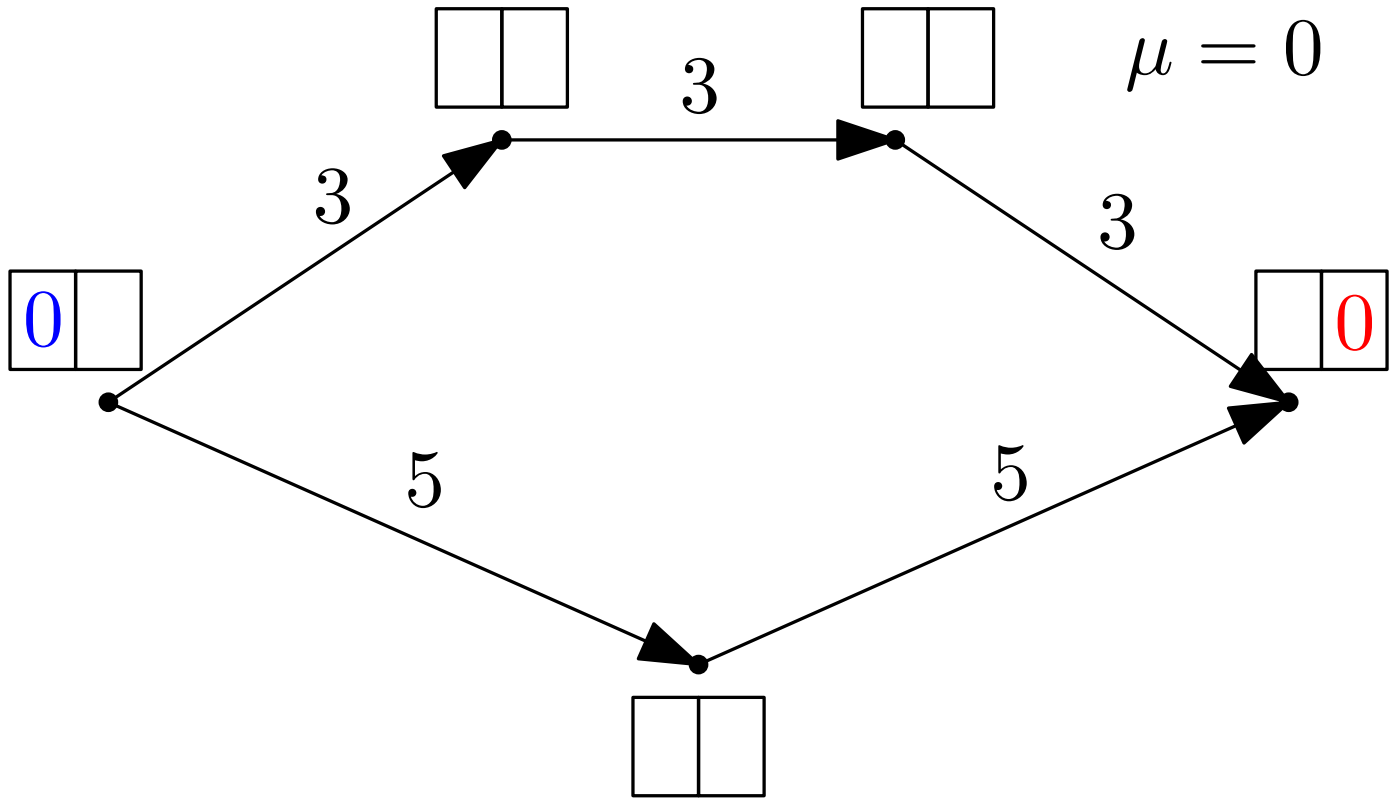
Idee:

- starte zweite Suche von t
- relaxiere rückwärtse nur eingehende Kanten
- stoppe die Suche, wenn beide Suchräume sich treffen

Bidirektionale Suche

Anfrage:

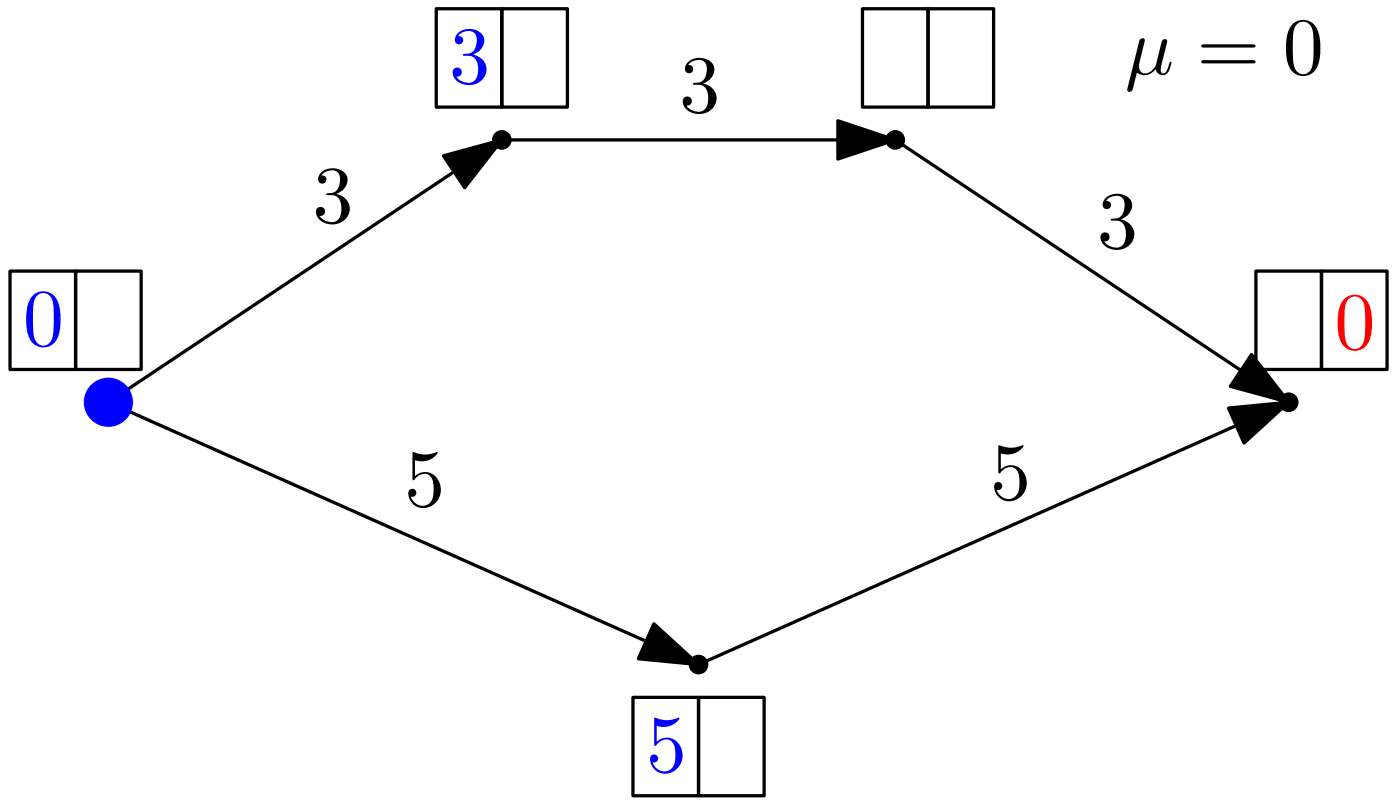
- alterniere Vorwärts- und Rückwärtsuche
  - vorwärts: relaxiere ausgehende Kanten
  - rückwärts: relaxiere eingehende Kanten



Bidirektionale Suche

Anfrage:

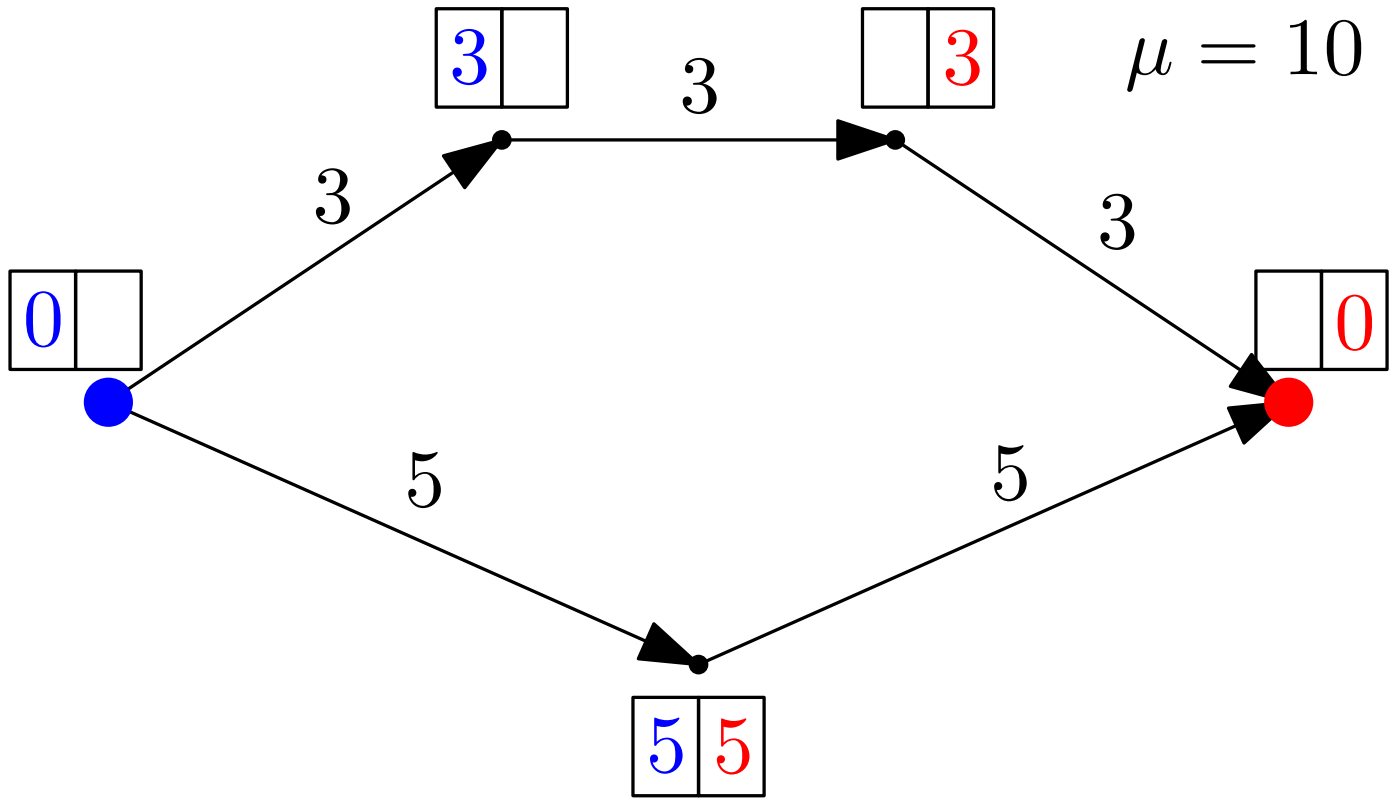
- alterniere Vorwärts- und Rückwärtsuche
  - vorwärts: relaxiere ausgehende Kanten
  - rückwärts: relaxiere eingehende Kanten



Bidirektionale Suche

Anfrage:

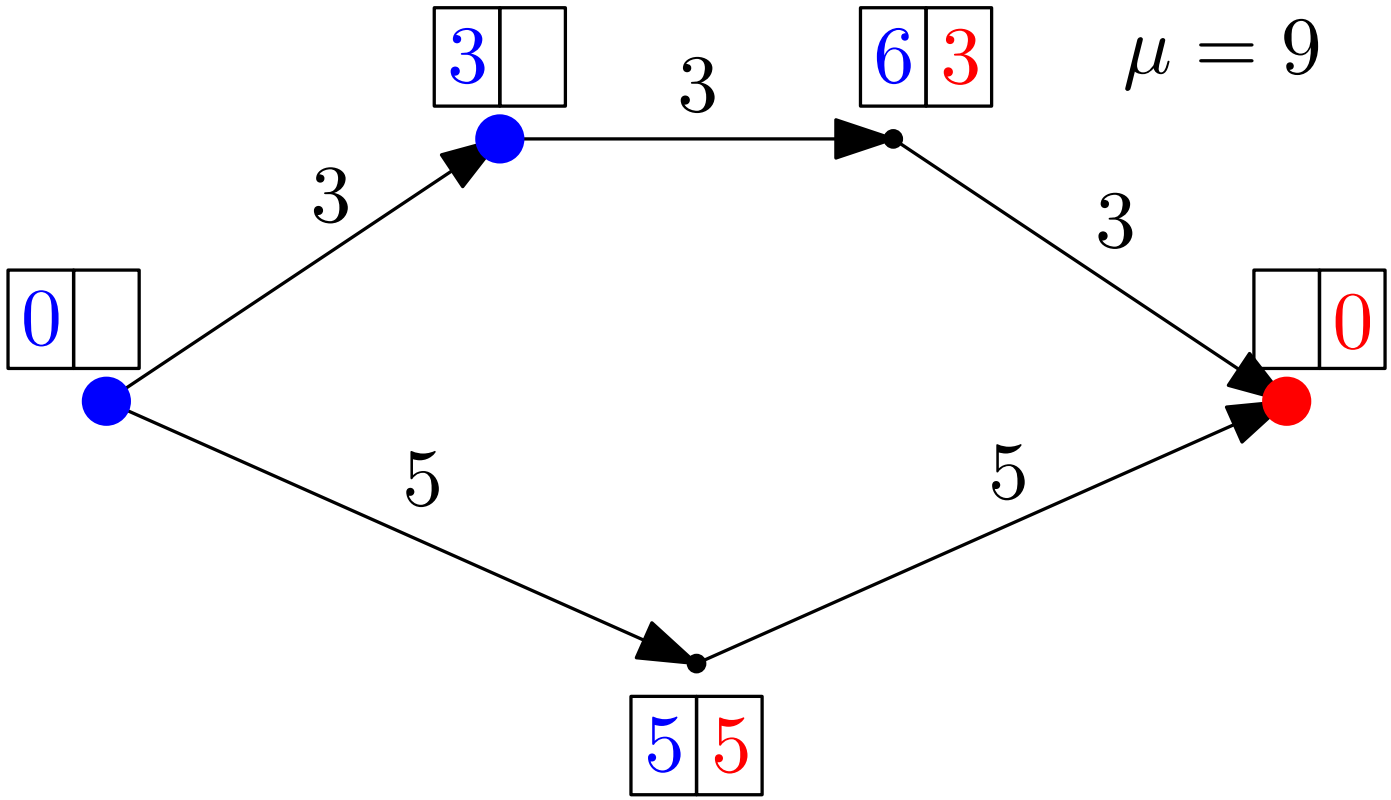
- alterniere Vorwärts- und Rückwärtsuche
  - vorwärts: relaxiere ausgehende Kanten
  - rückwärts: relaxiere eingehende Kanten



Bidirektionale Suche

Anfrage:

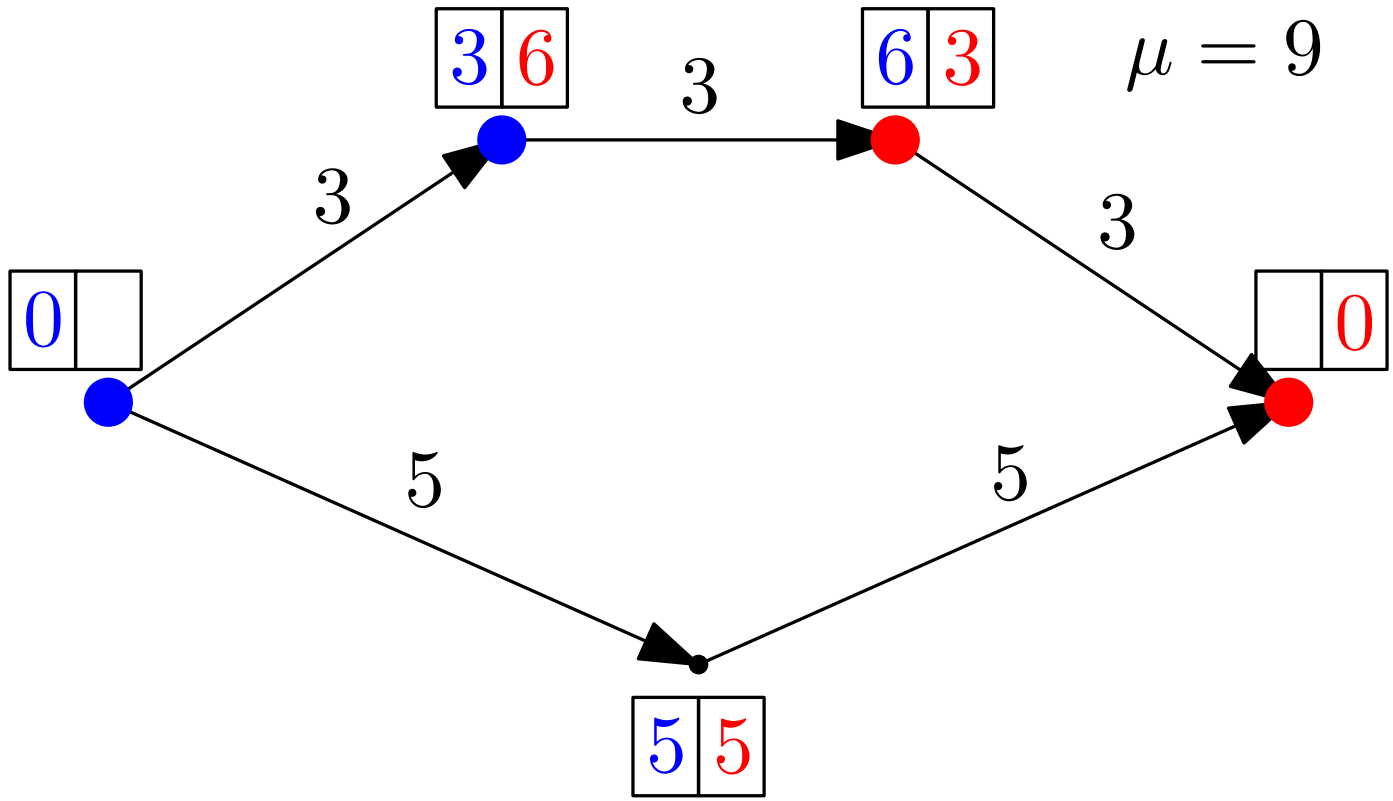
- alterniere Vorwärts- und Rückwärtsuche
  - vorwärts: relaxiere ausgehende Kanten
  - rückwärts: relaxiere eingehende Kanten



Bidirektionale Suche

Anfrage:

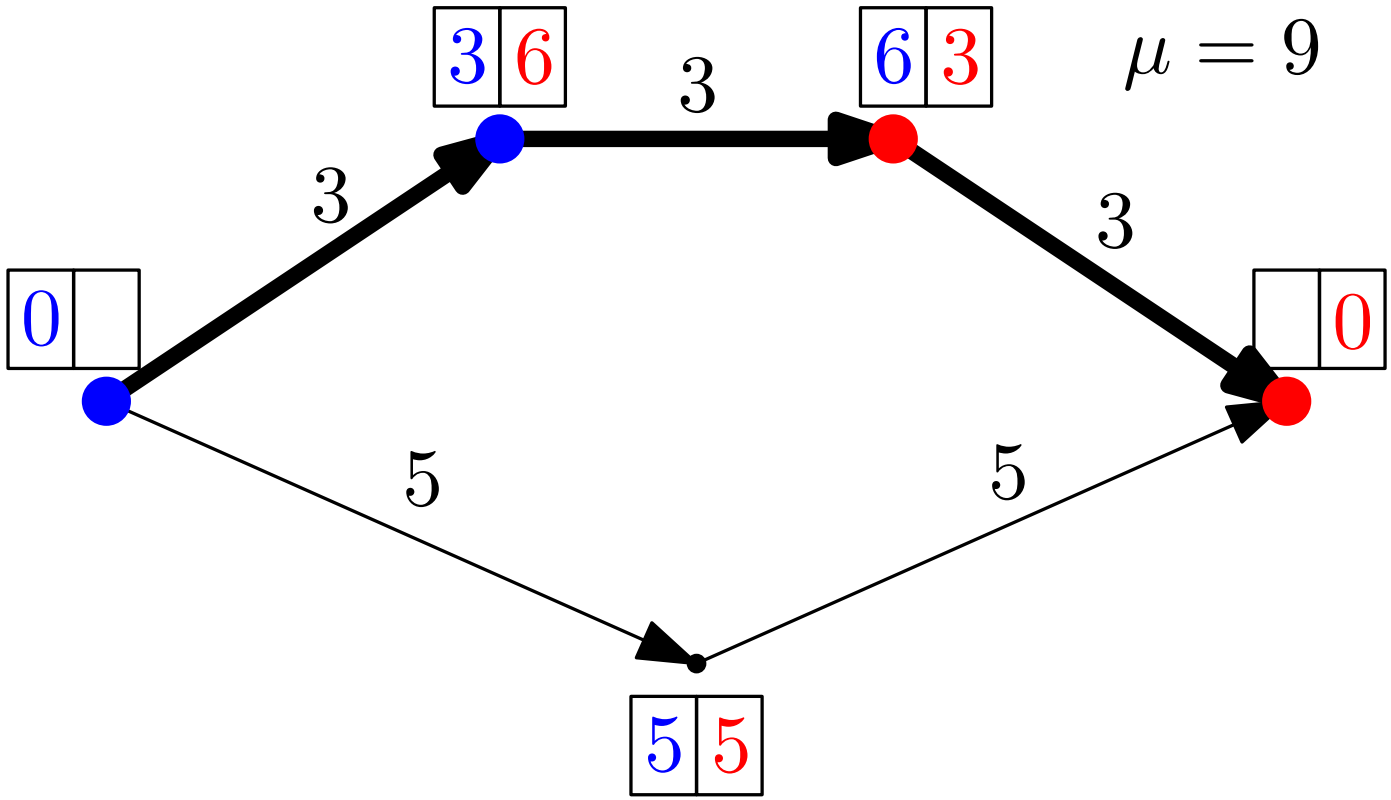
- alterniere Vorwärts- und Rückwärtsuche
  - vorwärts: relaxiere ausgehende Kanten
  - rückwärts: relaxiere eingehende Kanten



Bidirektionale Suche

Anfrage:

- alterniere Vorwärts- und Rückwärtsuche
  - vorwärts: relaxiere ausgehende Kanten
  - rückwärts: relaxiere eingehende Kanten



Bidirektionale Suche

Anfrage:

- alterniere Vorwärts- und Rückwärtsuche
  - vorwärts: relaxiere ausgehende Kanten
  - rückwärts: relaxiere eingehende Kanten
- μ: vorläufiges $d(s,t)$
- bei relax checken, ob Knoten von anderer Suche schon besucht
  - wenn ja: $d(s,t) = \mu$, $M = v$
- stoppe wenn $\mu < minKey(\overrightarrow{Q}) + minKey(\overleftarrow{Q})$
- kürzester Weg: $(s, \ldots, M, \ldots, t)$

Bidirektionale Suche: Korrektheit

Korrektheit

Bidirektionale Suche (BS) ist korrekt

- BS terminiert spätestens wenn beide Queues leer sind
- wenn $\mu > d(s,t)$ nach Terminierung dann
  - gibt es einen s-t Pfad P der kürzer als μ ist
  - auf P gibt es eine Kante (u,v) mit $d(s,u) < minKey(\overrightarrow{Q})$ und $d(v,t) < minKey(\overleftarrow{Q})$.
  - also müssen u und v schon abgearbeitet worden sein (o.B.d.A. u vor v
  - beim relaxieren von (u,v) wäre P entdeckt worden und μ aktuasisiert
- $\mu < d(s,t)$: Widerspruch
- somit korrekt

Strategie zum Wechsel der Suchen

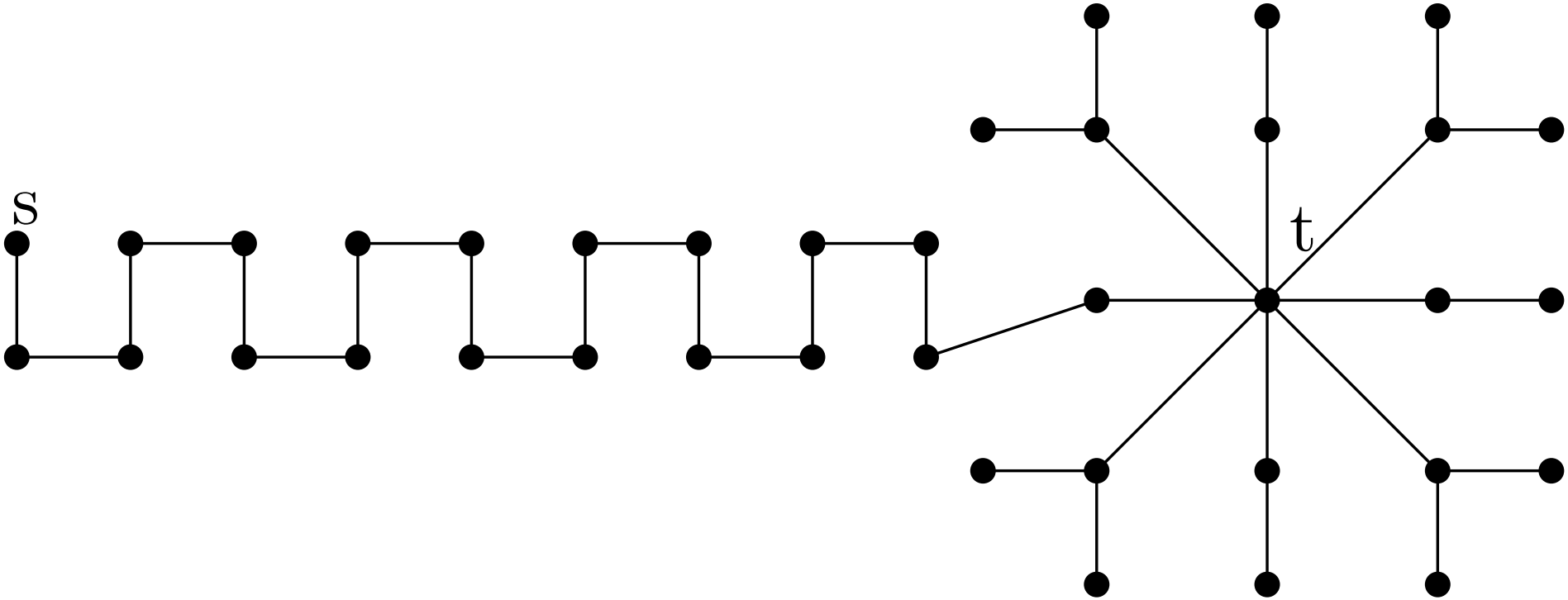
Beobachtung:

- jede Art der Abwechslung klappt

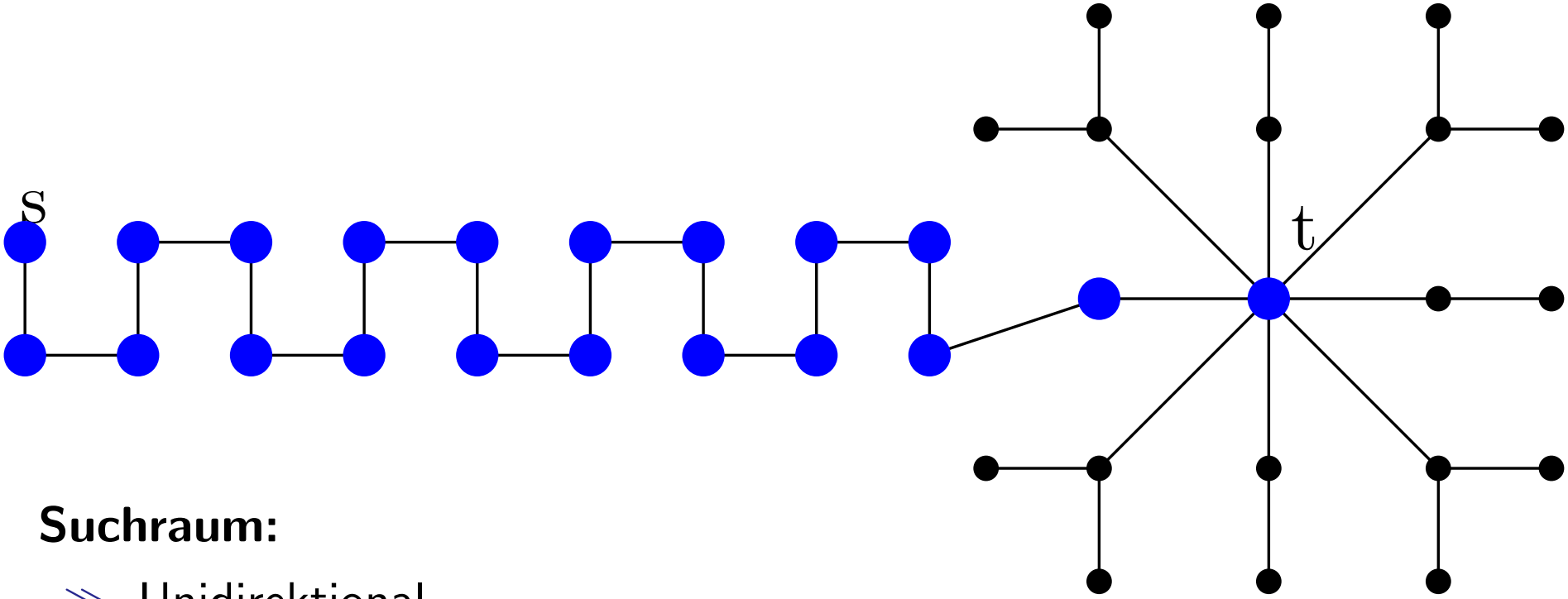
“sinnvolle” Strategien:

- streng alternierend
- wähle $\min\{minKey(\overrightarrow{Q}), minKey(\overleftarrow{Q})\}$
- wähle $\min\{|\overrightarrow{Q}|, |\overleftarrow{Q}|\}$

Worst-Case



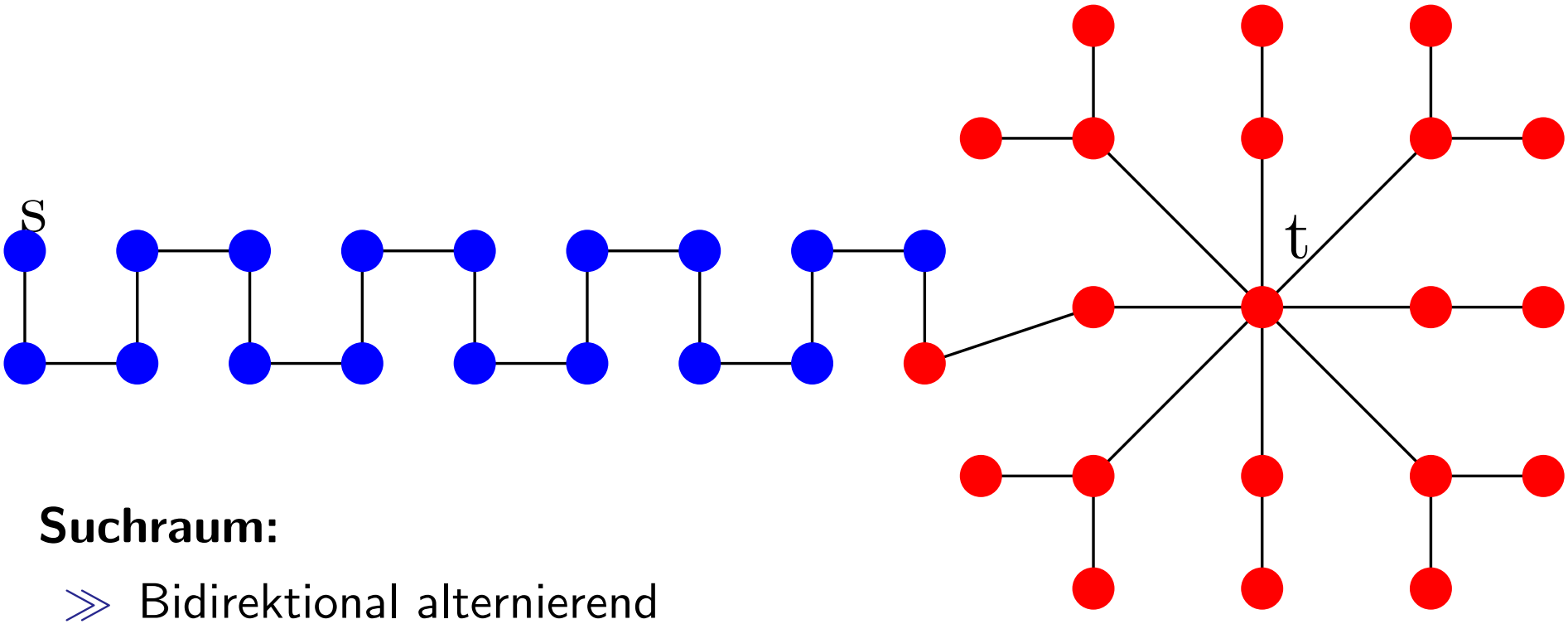
Worst-Case



Suchraum:

- Unidirektional

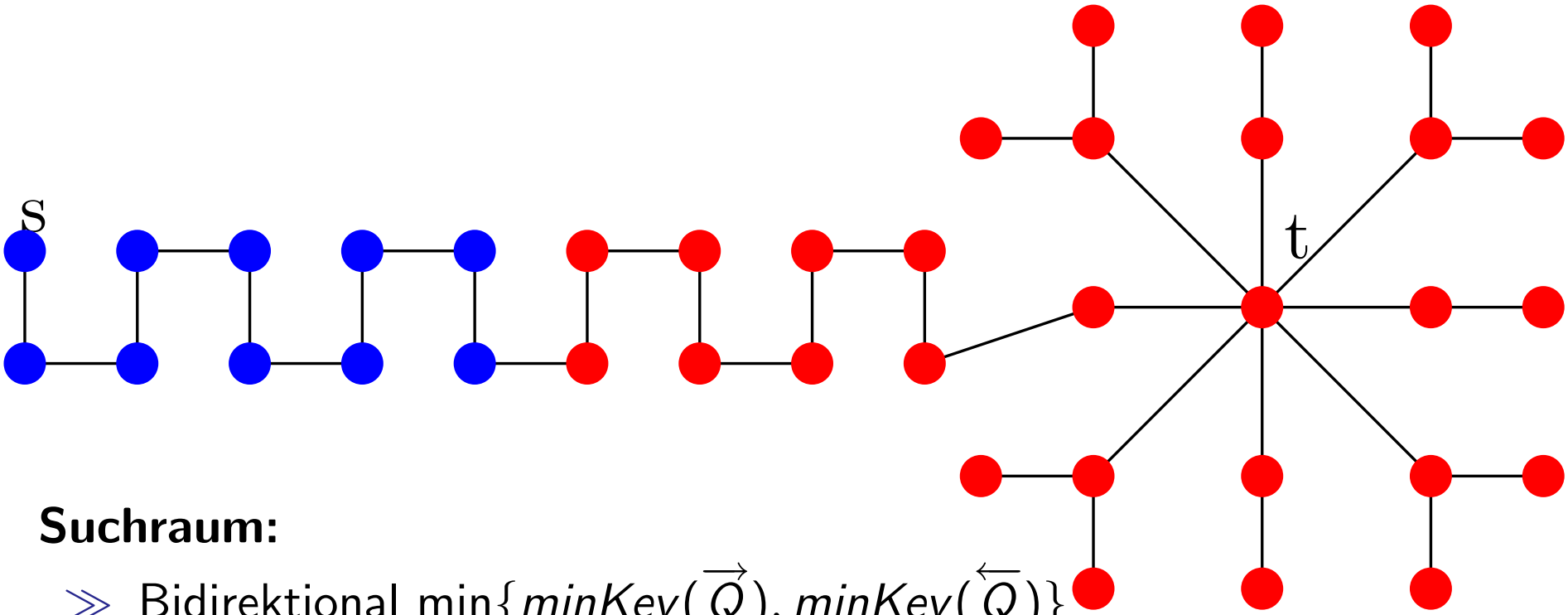
Worst-Case



Suchraum:

- Bidirektional alternierend
- Suchraum vergrößert sich

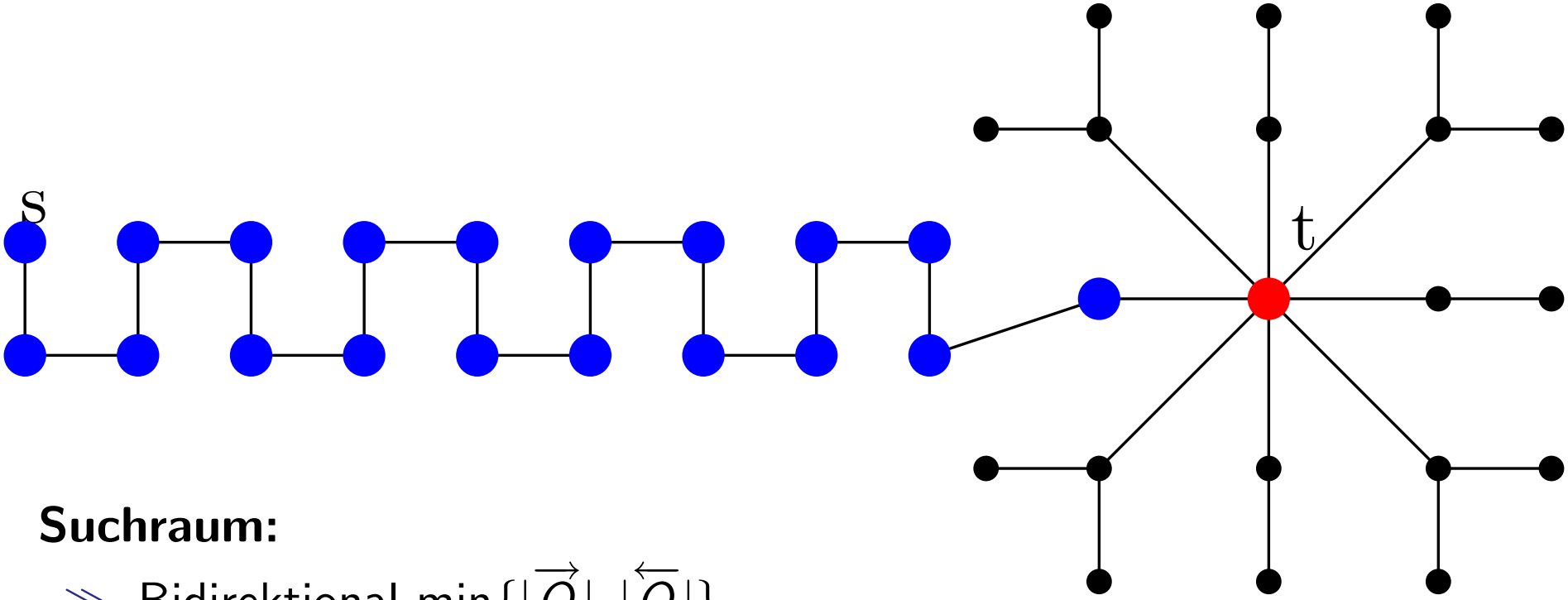
Worst-Case



Suchraum:

- Bidirektional $\min\{minKey(\overrightarrow{Q}), minKey(\overleftarrow{Q})\}$
- Suchraum vergrößert sich

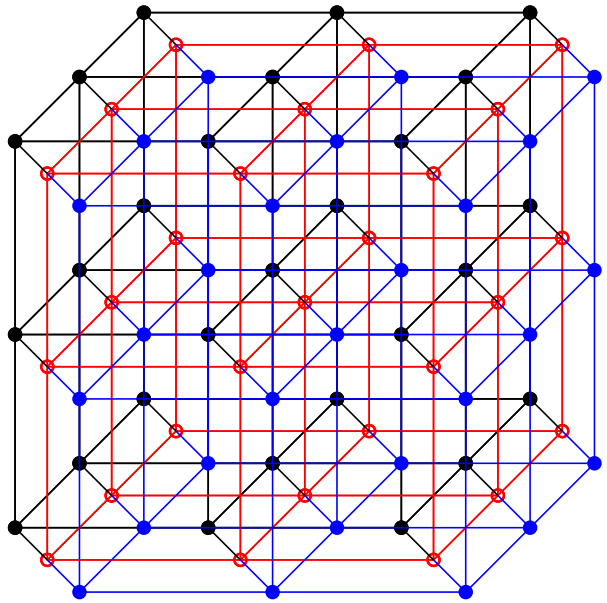
Worst-Case



Suchraum:

- Bidirektional $\min\{|\overrightarrow{Q}|, |\overleftarrow{Q}|\}$
- Suchraum bleibt gleich

Best-Case



Eingabe:

- k-dimensionales Gitter
- Suchraum unidirektional: $(2 \cdot d(s,t))^k$
- Suchraum bidirektional: $2 \cdot d(s,t)^k$

Somit Beschleunigung:

$$\frac{(2 \cdot d(s,t))^k}{2 \cdot d(s,t)^k} = 2^{k-1}$$

also exponentielle Beschleunigung!

Straßennetzwerke

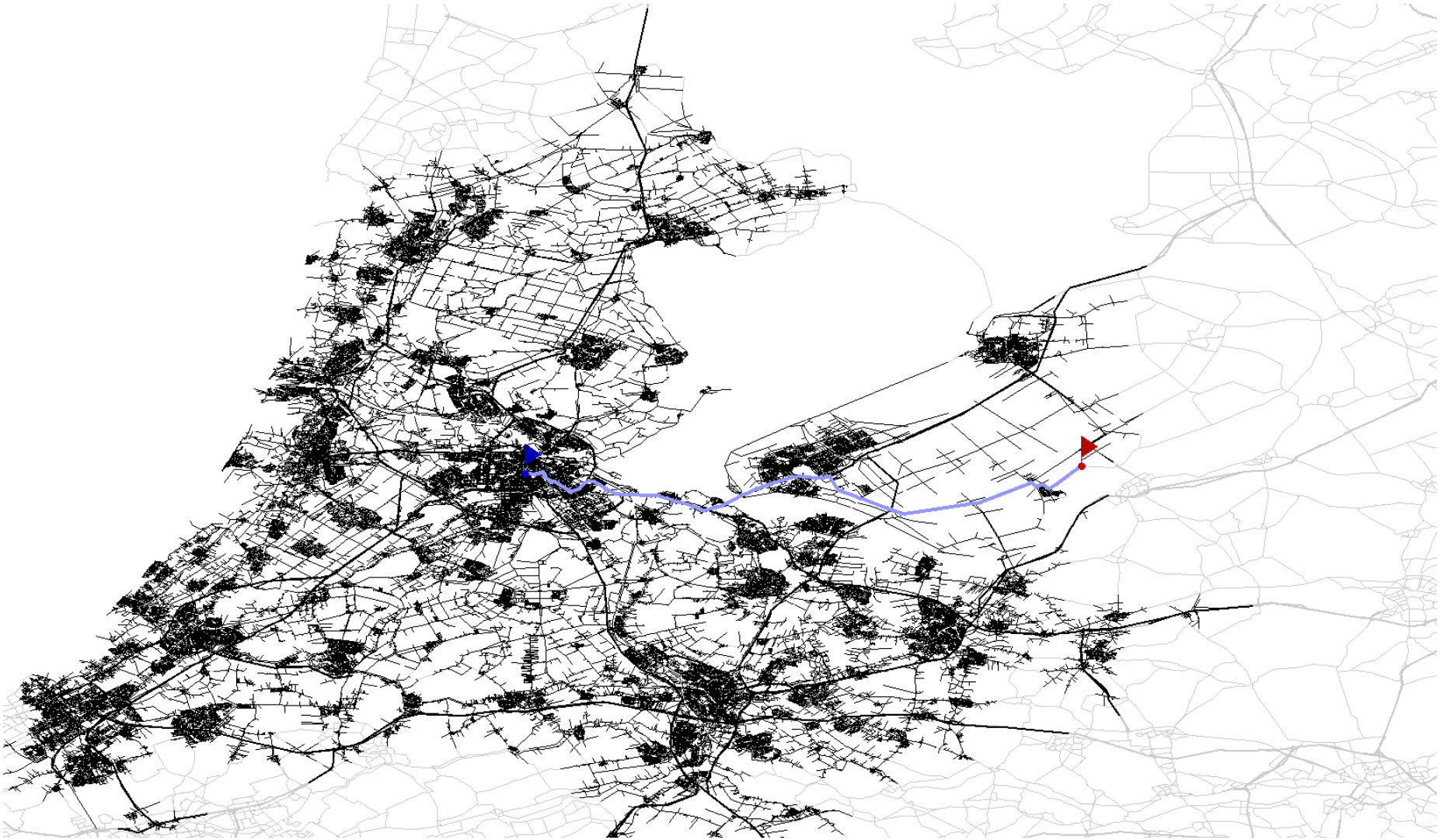
In Straßennetzen:

- Zur Erinnerung: fast planar
- Suchraum unidirektional: $(\pi \cdot d(s,t))^2$
- Suchraum bidirektional: $2 \cdot \left(\pi/2 \cdot d(s,t)\right)^2$

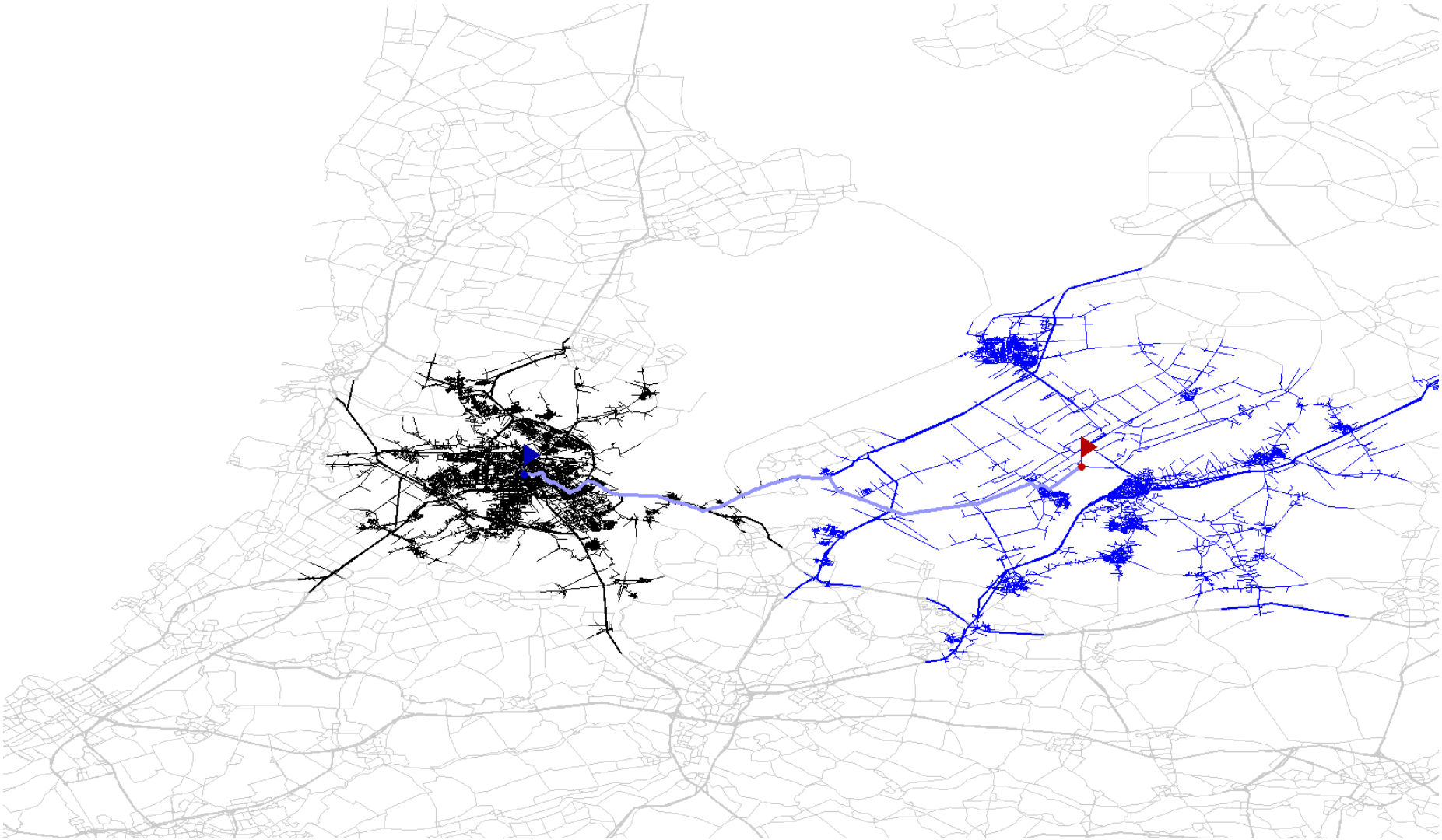
Somit Beschleunigung:

$$\frac{(\pi \cdot d(s,t))^2}{2 \cdot \left(\frac{\pi}{2} \cdot d(s,t)\right)^2} = 2$$

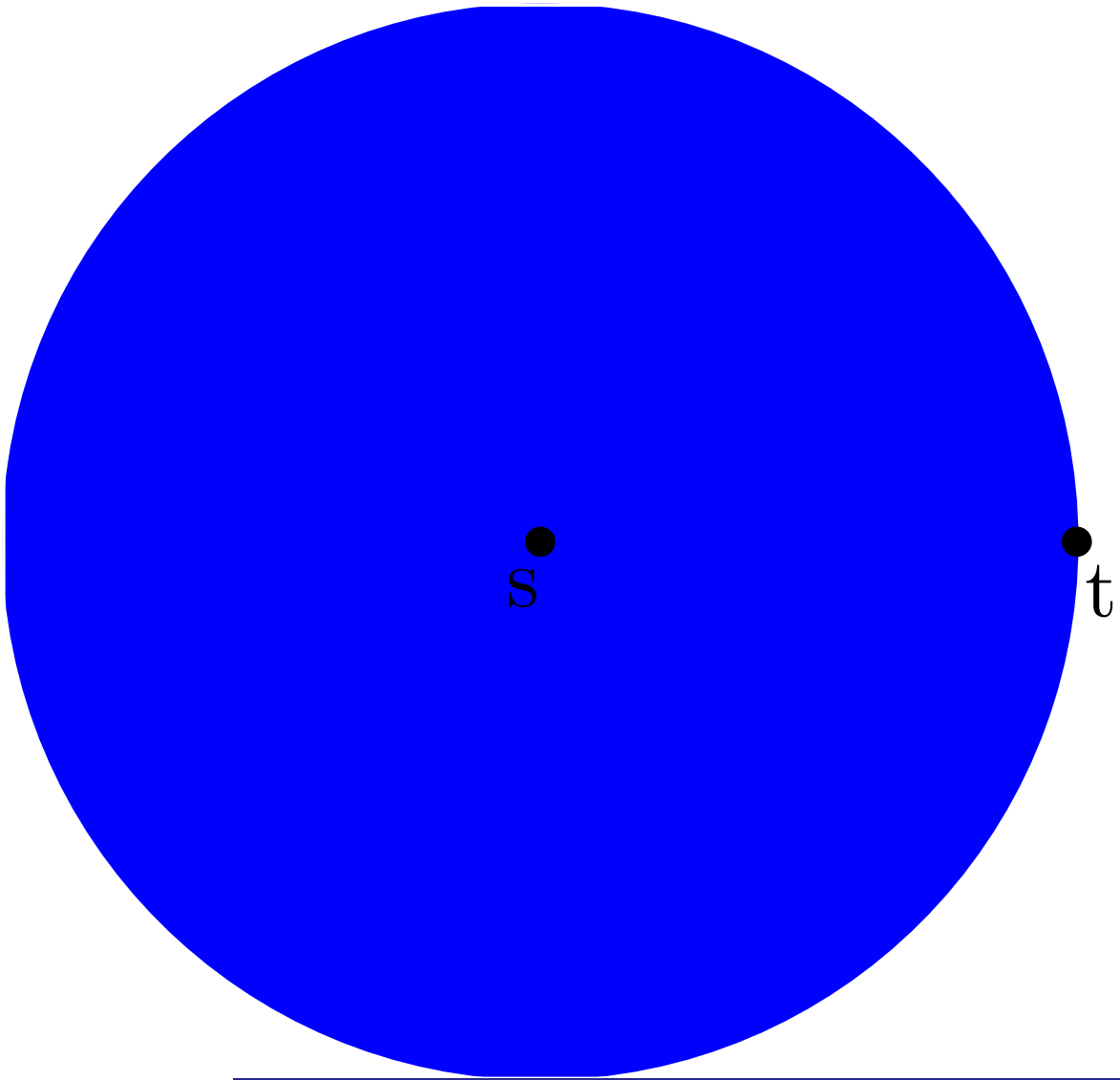
Beispiel



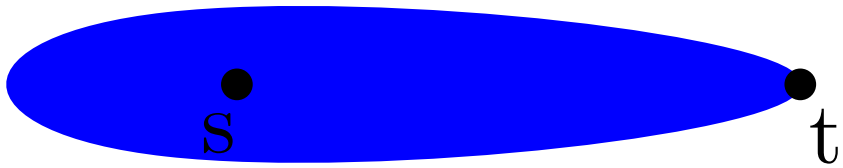
Beispiel



Schematischer Suchraum



Schematischer Suchraum



Ideensammlung

Wie Suche zielgerichtet machen?

- ≫ prunen von Kanten, Knoten die in die “falsche” Richtung liegen
- ≫ Reihenfolge in der Knoten besucht werden ändern

heute letzteres

A^*

Idee:

- berechne Potential $\pi : V \to \mathbb{R}$ mit
  - $\pi(u) \leq len(u, v) + \pi(v)$ für alle $(u, v) \in E$
- während der Anfrage: nutze $d[u] + \pi(u)$ als key

Dijkstra - PseudoCode

DIJKSTRA($G = (V, E)$,s)

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

A^* - Pseudocode

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

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

A^*: Beobachtungen

- Knoten werden in anderer Reihenfolge abgearbeitet
  - Dijkstra: u vor v wenn $d[u] < d[v]$
  - A^*: u vor v wenn $d[u] + \pi(u) < d[v] + \pi(v)$
  - $\pi \equiv 0$ dann gleicher Suchraum
- Abbruchkriterium bleibt erhalten

A^*: Korrektheit

Korrektheit

A^* ist korrekt

- neue Längenfunktion $len_\pi(u,v) = len(u,v) - \pi(u) + \pi(v)$
- wenn wir len durch len_π ersetzen
  - die Länge aller Pfade zwischen s und t ändern sich um den gleichen Wert $\pi(t) - \pi(s)$
  - also bleiben kürzeste Wege erhalten
  - $d_\pi(s,u) = d(s,u) - \pi(s) + \pi(u)$
  - Reihenfolge der besuchten Knoten von Dijkstra in G_π entspricht der von A^* in G, da $\pi(s)$ gleich für alle Knoten
  - A^* korrekt wenn $len_\pi(u,v) \geq 0$ für alle $(u,v) \in E$
    - folgt aus $\pi(u) \leq len(u,v) + \pi(v)$
  - nennt man ein gültiges Potential

Gültige Potentiale: Eigenschaften I

wenn $\pi(u) = d(u, t)$, dann

- gilt für alle Kanten (u_i, u_{i+1}) auf einem kürzestem Weg von s nach t: $len_\pi(u_i, u_{i+1}) = 0$ und $len_\pi(u, v) > 0$ für alle andere Kanten (u, v)
- werden also nur Knoten auf dem kürzestem Weg angeschaut (sofern Wege eindeutig)
- also: je besser die untere Schranke, desto höher die Beschleunigung

Gültige Potentiale: Eigenschaften II

Kombinierbarkeit von Potentialen

Seien π_1 und π_2 gültige Potentiale. Dann ist $p = \max\{\pi_1, \pi_2\}$ auch ein gültiges Potential.

- $len(u,v) - \pi_1(u) + \pi_1(v) \geq 0$ und $len(u,v) - \pi_1(u) + \pi_1(v) \geq 0$
- Sei $\pi_1(u) \geq \pi_2(v)$, anderer Fall symmetrisch
  - wenn $\pi_1(v) \geq \pi_2(v)$, dann
    $len(u,v) - p(u) + p(v) = len(u,v) - \pi_1(u) + \pi_1(v) \geq 0$
  - sonst $len(u,v) - p(u) + p(v) = len(u,v) - \pi_1(u) + \pi_2(v) \geq$
    $len(u,v) - \pi_1(u) + \pi_1(v) \geq 0$

Anmerkungen:

- gilt auch für Minimum und jede beliebige (konvexe) Linearkombination
- wir können also für jede s–t Anfrage ein eigenes Potential wählen

Gültige Potentiale: Eigenschaften III

wenn $\pi(t) = 0$, dann

- $\pi(u) > d(u,t)$ für alle $u \in V$
- ist $\pi(u)$ eine untere Schranke für $d(u,t)$
- $\pi(u)$ schätzt den Abstand von u nach t
- A^* besucht Knoten in aufsteigendem "erwarteten Abstand"
- jede Abschätzung $\underline{d(u,t)}$ ist ein gültiges Potential

Euklidisches Potential

Idee:

- Knoten haben natürliche x und y Koordinaten
- es gibt eine Maximalgeschwindigtkeit $v_{\max}$ in G
- nimm $\sqrt{(x(t)-x(u))^2-(y(t)-y(u))^2}/v_{\max}$ als Potential

Probleme:

- bei jedem Abarbeiten muss zusätzlich potenziert werden $\Rightarrow$ Overhead zur Berechnung recht groß
- Abschätzung sehr konservativ
  - nimmt an, dass es eine Autobahn zum Ziel gibt
  - die zudem noch der Fluglinie folgt

$\Rightarrow$ praktisch keine Beschleunigung in Transportnetzen

Potentiale durch Landmarken

Idee:

- wähle Landmarken L aus V (≈ 16)
- berechne Distanzen von und zu allen Landmarken
- dann gilt:

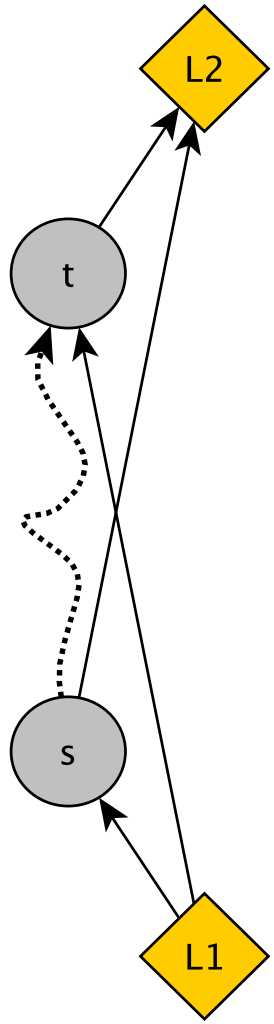
$$\begin{aligned} d(u,t) &\geq d(L_1,t) - d(L_1,u) \\ d(u,t) &\geq d(u,L_2) - d(t,L_2) \end{aligned}$$

für alle $u \in V$.

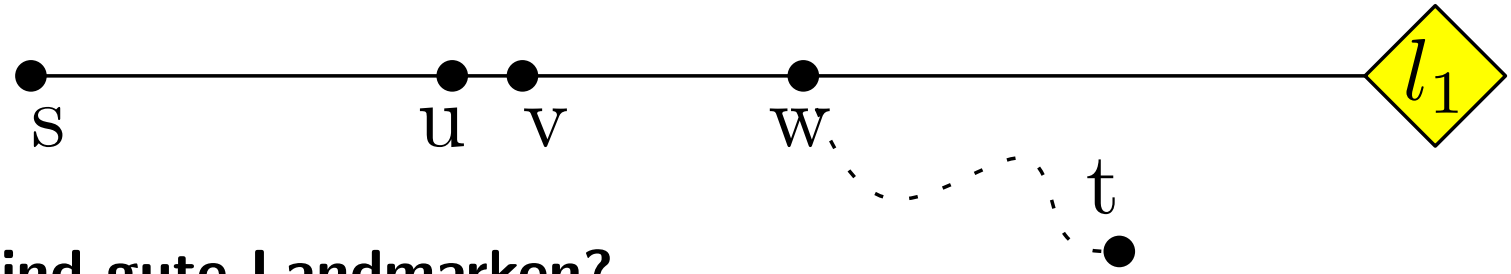
- somit ist

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

ein gültiges Potential



Eigenschaften Landmarken



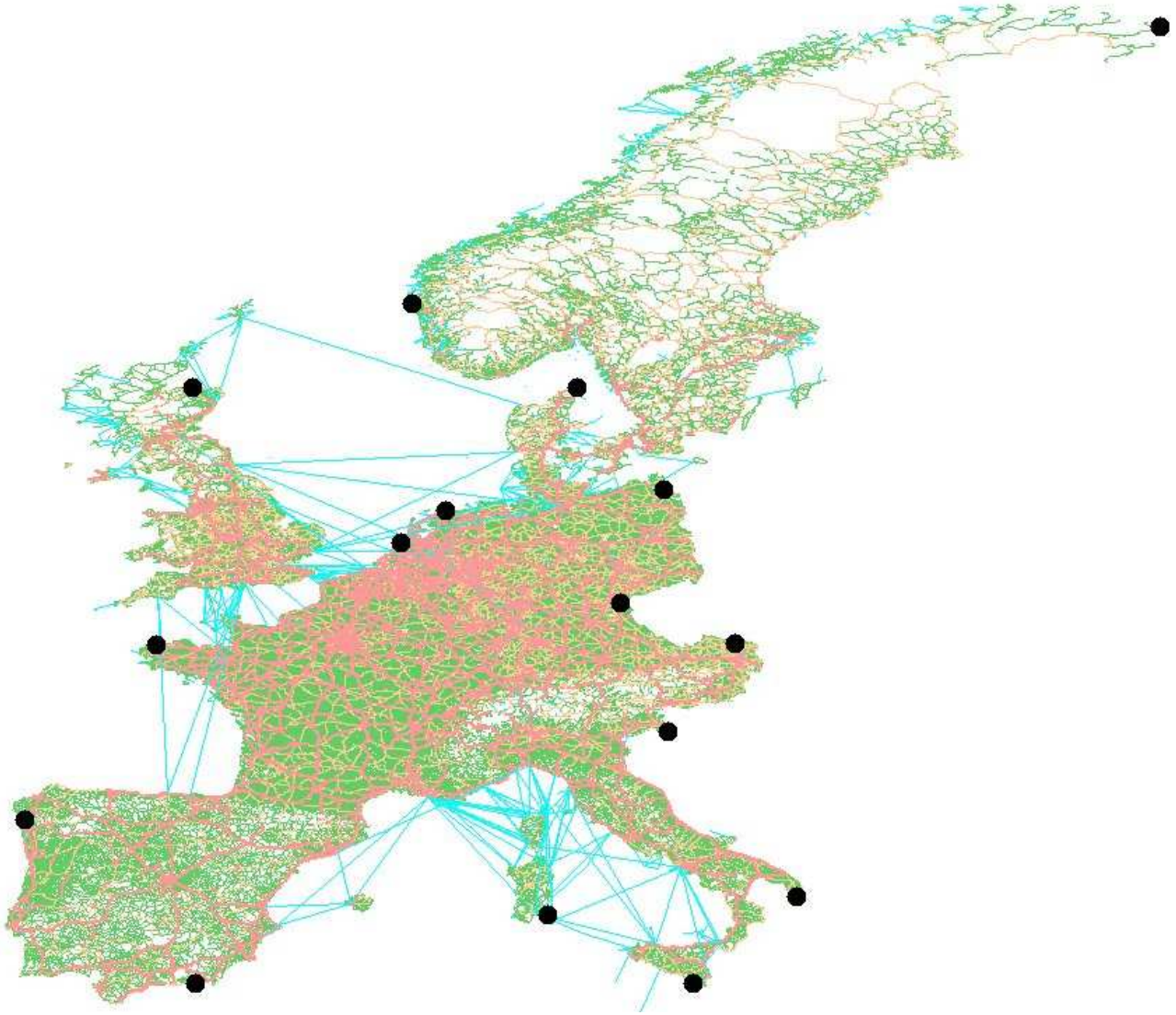
Was sind gute Landmarken?

- $\pi(u) = d(u, l_1) - d(t, l_1)$, $\pi(v) = d(v, l_1) - d(t, l_1)$
- also $len_\pi(u, v) = len(u, v) - d(u, l_1) + d(v, l_1) = 0$
- gemeinsame Kanten (kürzester Weg und Weg zur Landmarke) haben reduzierte Kosten von 0

Also:

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

Beispiel gute Landmarken



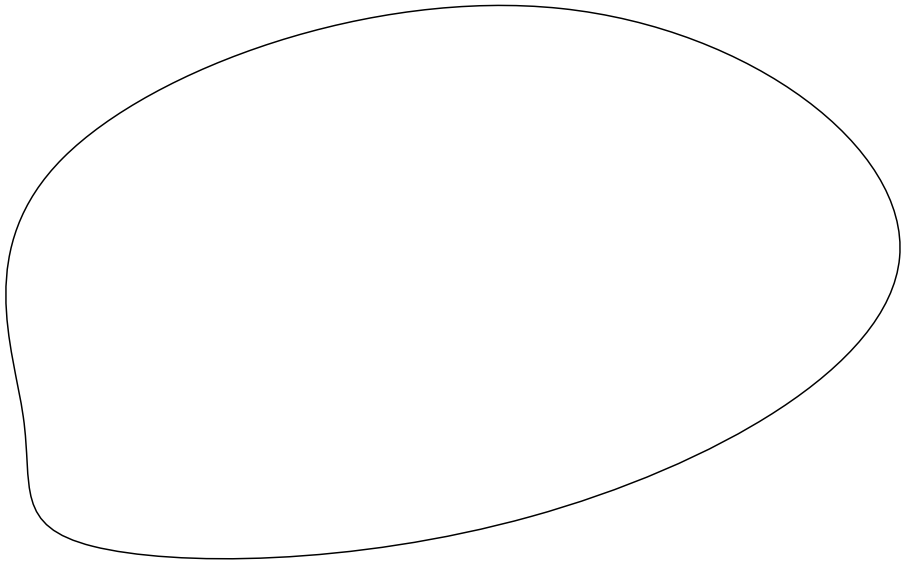
Berechnung Landmarken

mehrere Ansätze:

- brute force: $O(n^{|L|} \cdot \underbrace{n(m + n \log n)}_{\text{all pair shortest path}})$
  - \+ 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

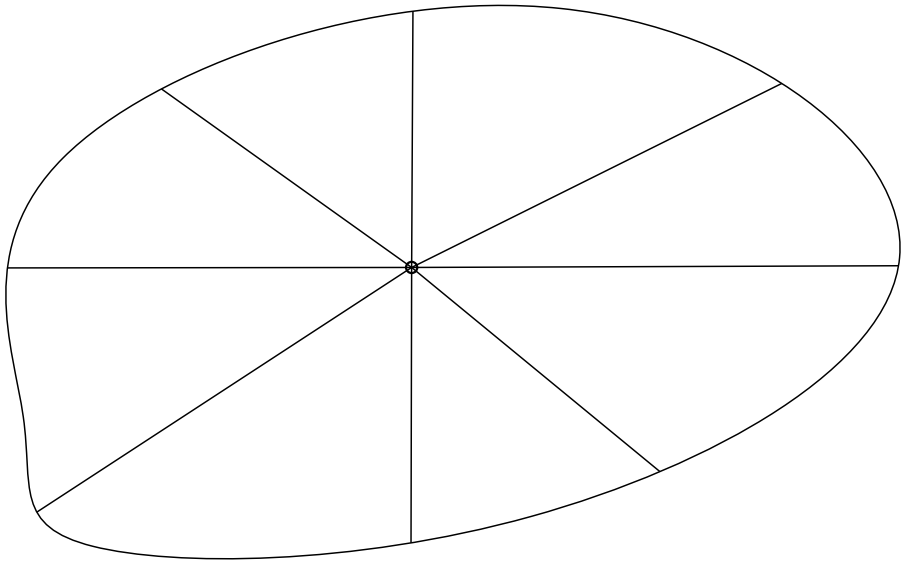
Vorgehen:



Planar-Landmarken

Vorgehen:

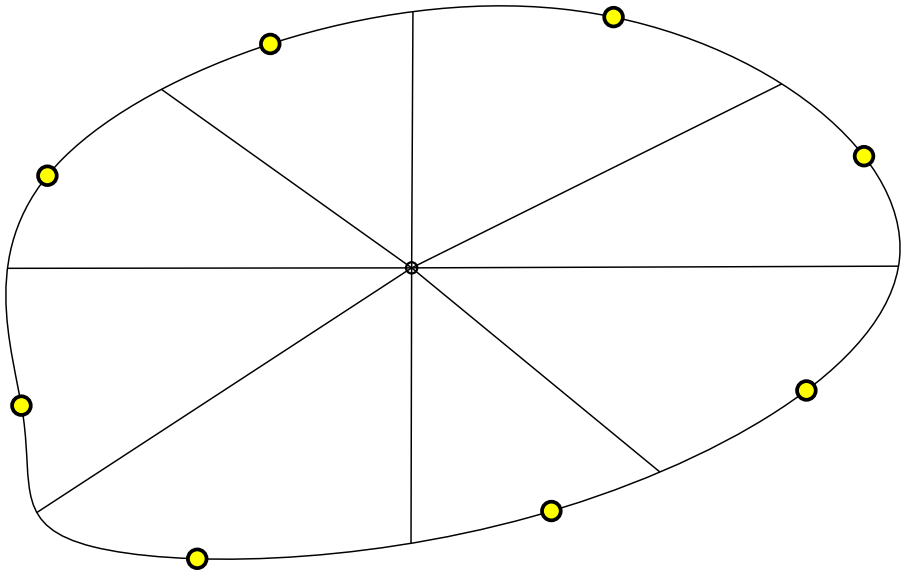
- suche Mittelpunkt c des Graphen
- teile Graphen in k Teile



Planar-Landmarken

Vorgehen:

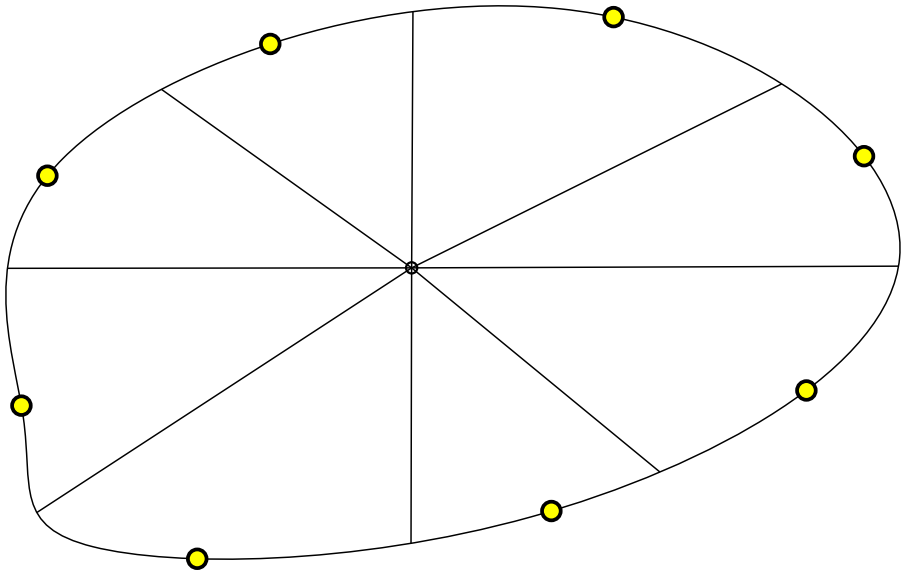
- suche Mittelpunkt c des Graphen
- teile Graphen in k Teile
- in jedem Teil wähle Knoten mit maximalen Abstand zu c als Landmarke



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:

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

Avoid-Landmarken

Vorgehen:

- berechne kürzeste Wege Baum T_r von einem 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

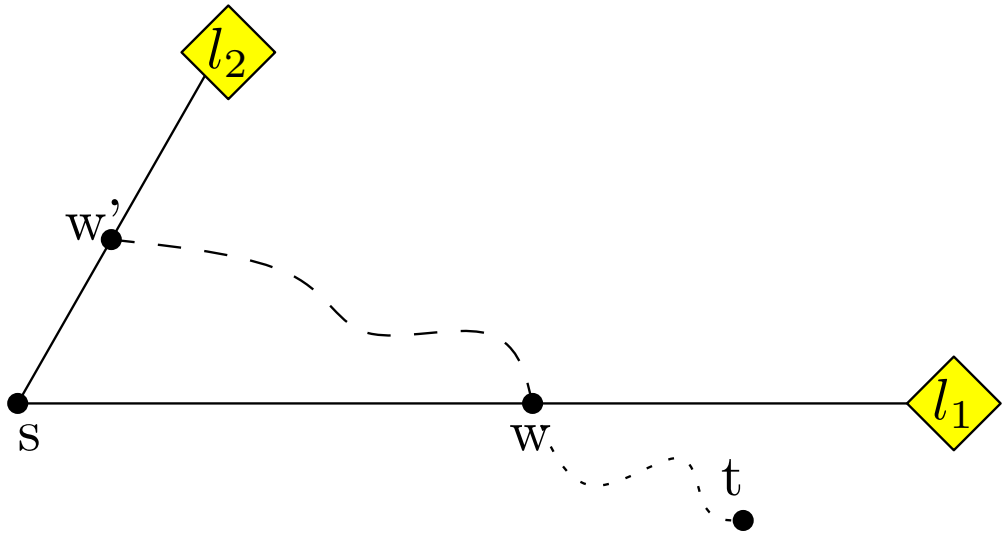
Idee:

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

Aktive Landmarken

Problem:

- viele Kanten die zu einer Landmarke führen haben reduziertes Gewicht 0
- daher: Landmarken können Suche in die falsche Richtung ziehen



Lösung:

- wähle während Initialisierung eine Untermenge
- diese, für die $p_f(s)$ und $p_r(t)$ maximal sind
- die Landmarken liefern die besten Schranken und sind (hoffentlich) die besten
- Aktualisiere die aktiven Landmarken während der Suche

Datenstruktur

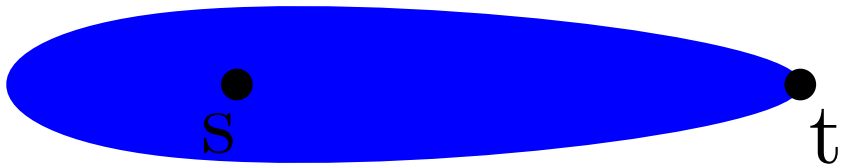
Erster Ansatz:

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

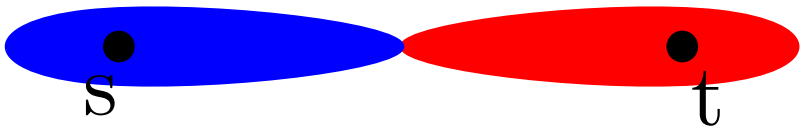
Besser:

- 1 64-bit Vektor der Größe $|L| \cdot n$
- speicher 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 4

Bidirektionaler A^*



Bidirektionaler A^*



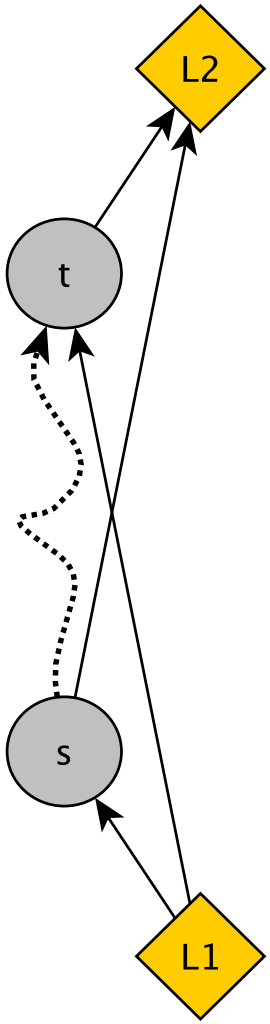
Bidirektionaler A^*

Erster Ansatz:

- benutze Vorwärtspot π_f und Rückwärtspot π_b

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

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



Bidirektionaler A^*

Erster Ansatz:

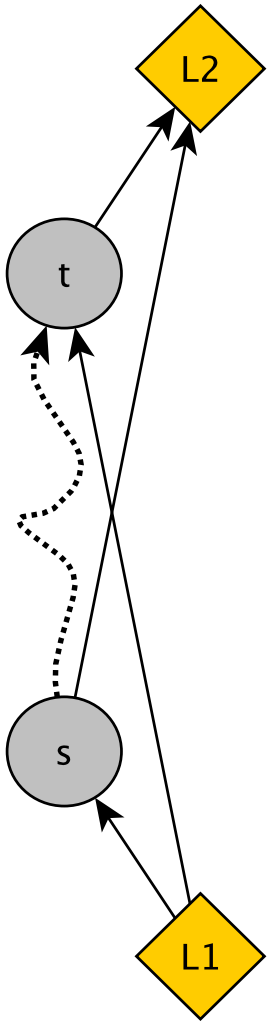
- benutze Vorwärtspot π_f und Rückwärtspot π_b

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

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

Problem:

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



Bidirektionaler A^*

Zweiter Ansatz:

- Wann operieren Suchen auf dem gleichen Graphen?
- wenn

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

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

ALT [Goldberg et al. 04,05]

- **A***, **L**andmarken, **T**riangle inequality (Dreiecksungleich)
- 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

		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

- 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 ≈ 100 für Europa
- 64 Landmarken: Beschleunigung ≈ 300 für Europa
- hoher Speicherverbrauch (Graph-DS: 424 MB, pro Landmarke: 144 MB, Europa)
- USA schlechter als Europa (Vermutung: schlechtere Hierarchie)

Dijkstra-Rank

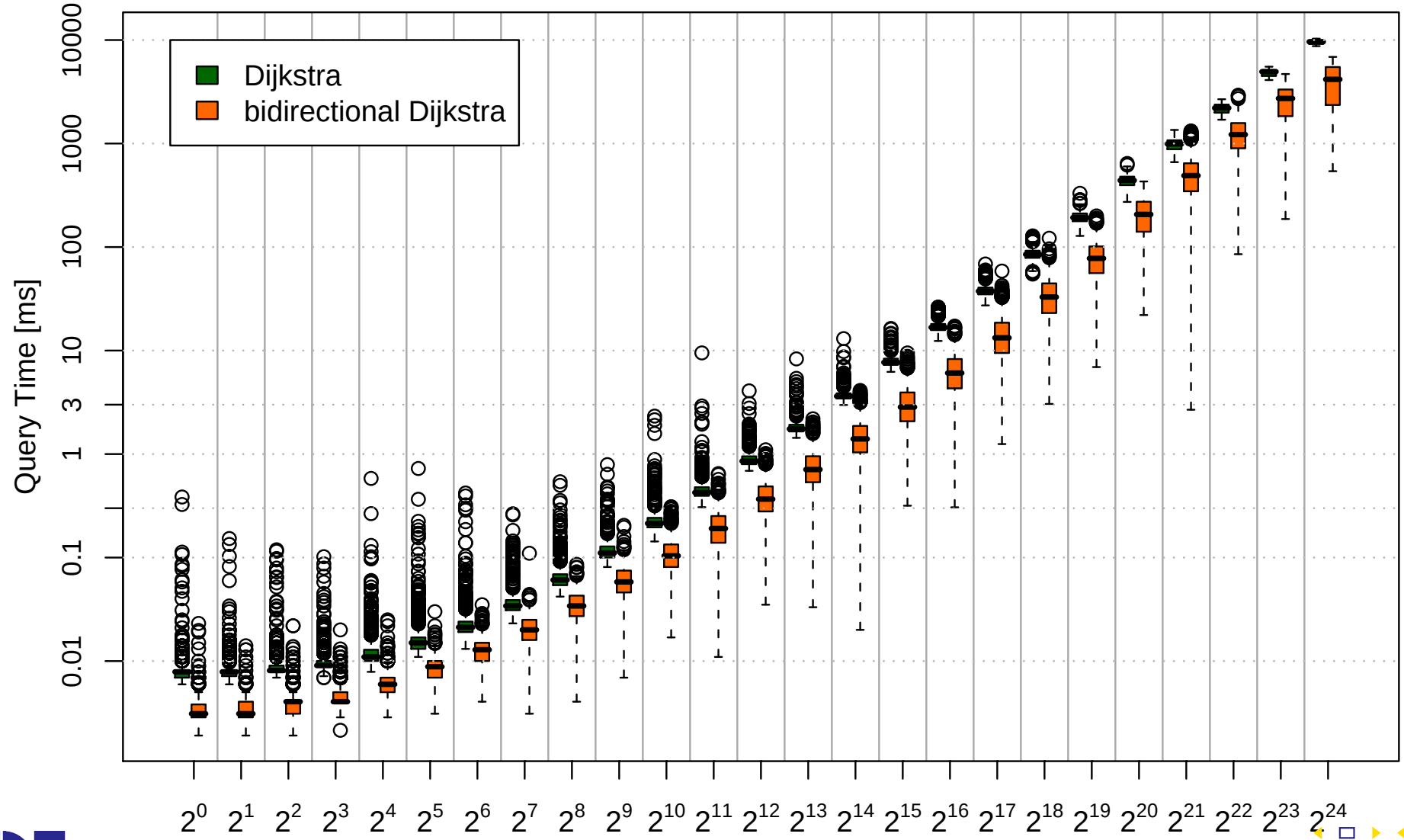
Problem:

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

Idee:

- Dijkstra definiert für gegebenen Startknoten Ordnung für auf den Knoten
- Dijkstra-Rang $r_s(u)$ eines Knoten u für gegebenes s
- wähle 1000 Startknoten und analysiere jeweils die Suchzeiten um die Knoten mit Rang $2^1, \ldots, 2^{\log n}$ zu finden
- zeichne Plot

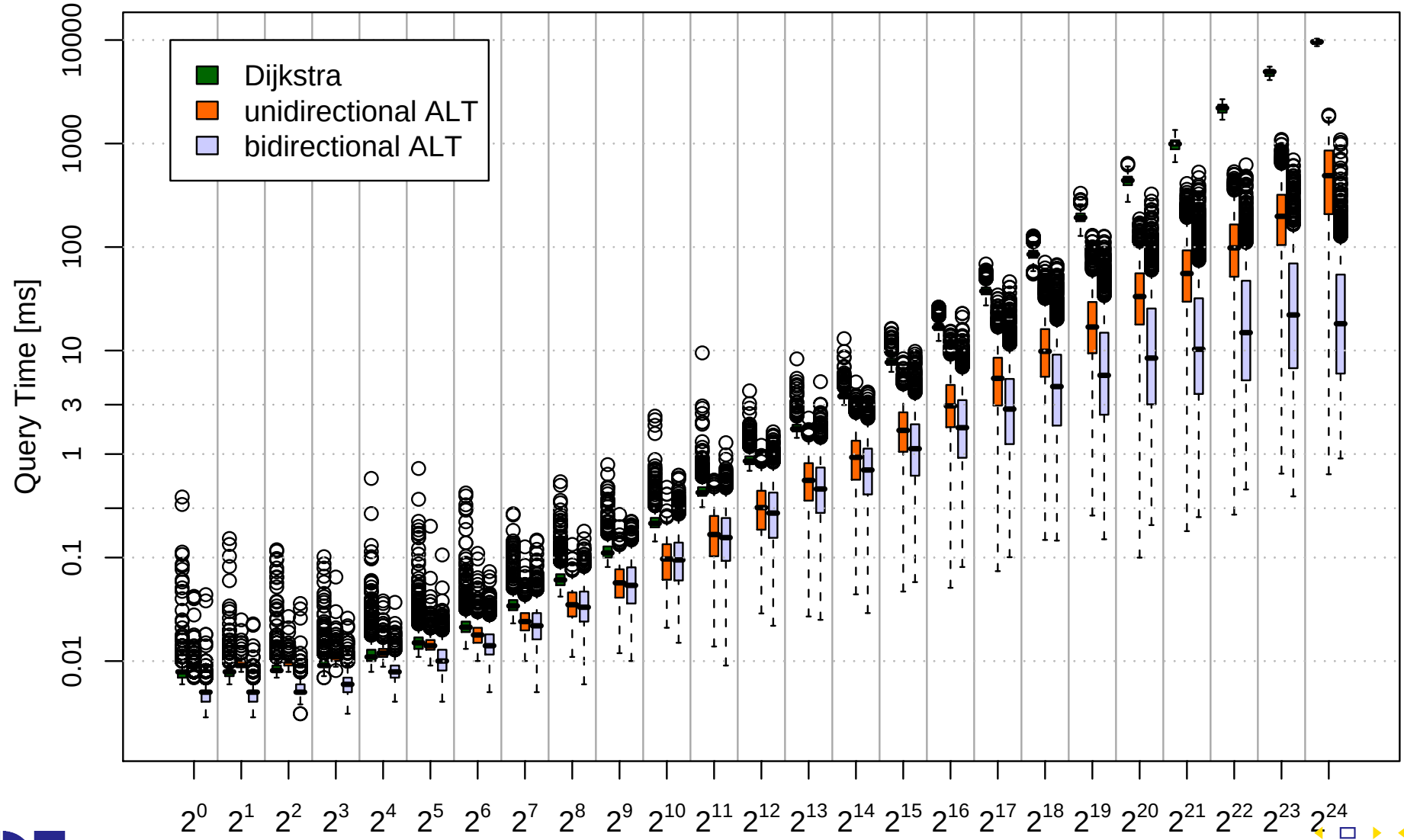
Lokale Anfragen Bidirektionale Suche



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 sehr schnell

Lokale Anfragen ALT



Beobachtungen

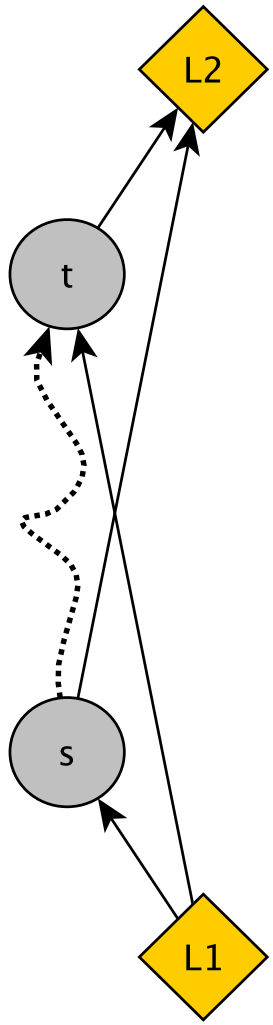
- Beschleunigung steigt mit Rang
- kaum Beschleunigung für nahe Anfragen
- hohe Varianz für ALT
- Ausreißer bis zu Faktor 100 langsamer als Median

Zusammenfassung Bidirektionale Suche

- 2. Suche von t auf eingehenden Kanten
- Abbruch wenn beide Suchen sich treffen
- verschiedene Strategien, welche Suche gewählt wird
- Suchraumreduktion abhängig von der Eingabe
  - worst-case Suchraumvergößerung
  - exponentielles Wachstum in hochdimensionalen Gittern
  - Faktor 2 in Straßennetzen

Zusammenfassung ALT

- Zielgerichtet durch geänderte Reihenfolge, wie Knoten abgearbeitet werden
- wird erreicht durch hinzufügen eines Knotenpotentials π
- korrekt wenn $len(u, v) - \pi(u) + \pi(v) \geq 0$ 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)



Literatur

Bidirektionale Suche, A^* und ALT:

- Andrew Goldberg, Georg Harrelson, SODA 2005
- Andrew Goldberg, Renato Werneck, ALENEX 2005

Anmerkung:

- wird auf der Homepage verlinkt