Karlsruher Institut für Technologie

Algorithmen für Routenplanung

2. Sitzung, Sommersemester 2010

Thomas Pajor | 26. April 2010

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

2. Beschleunigungstechniken

Bewertungskriterien

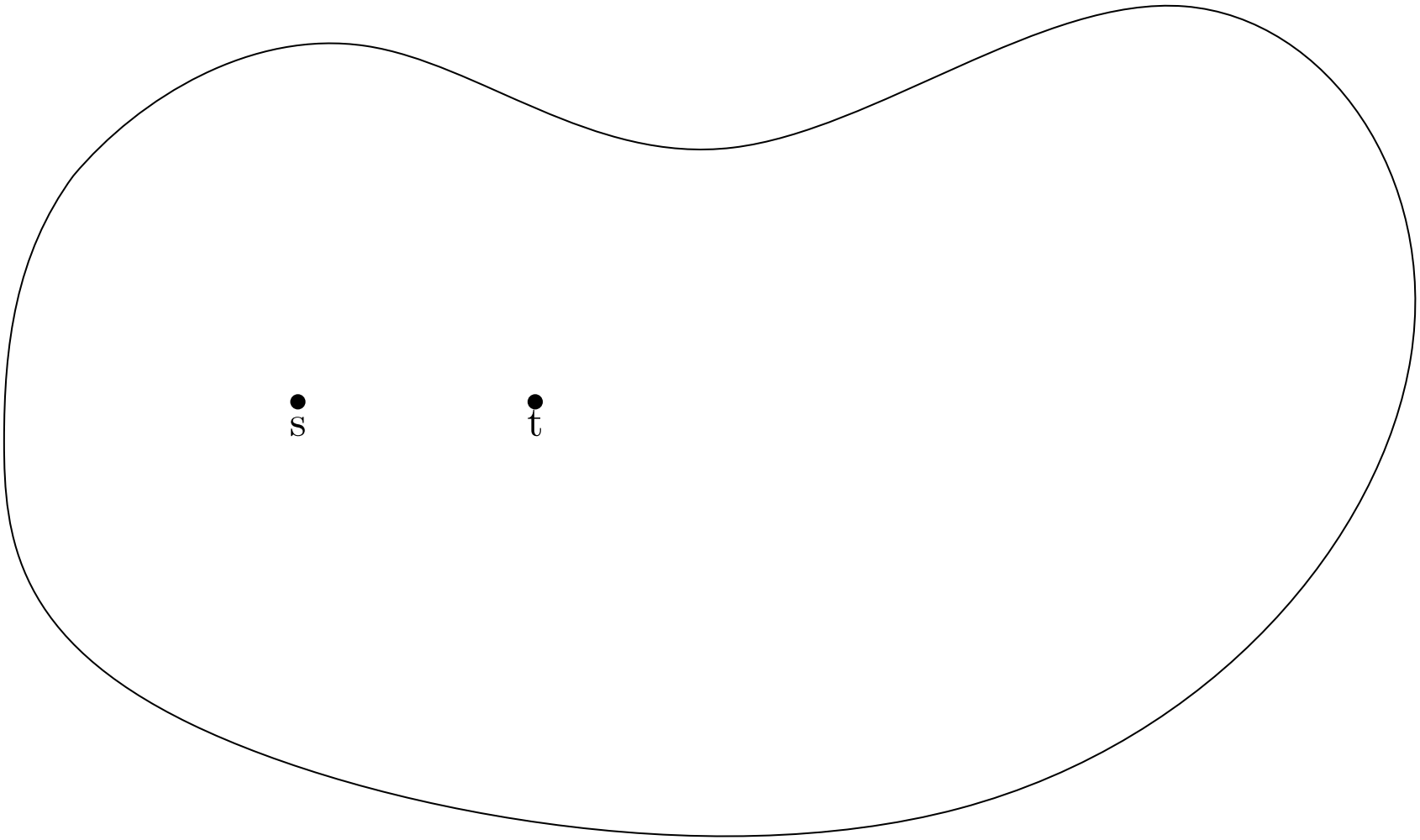
Vorberechnung:

- Platz
- Zeit

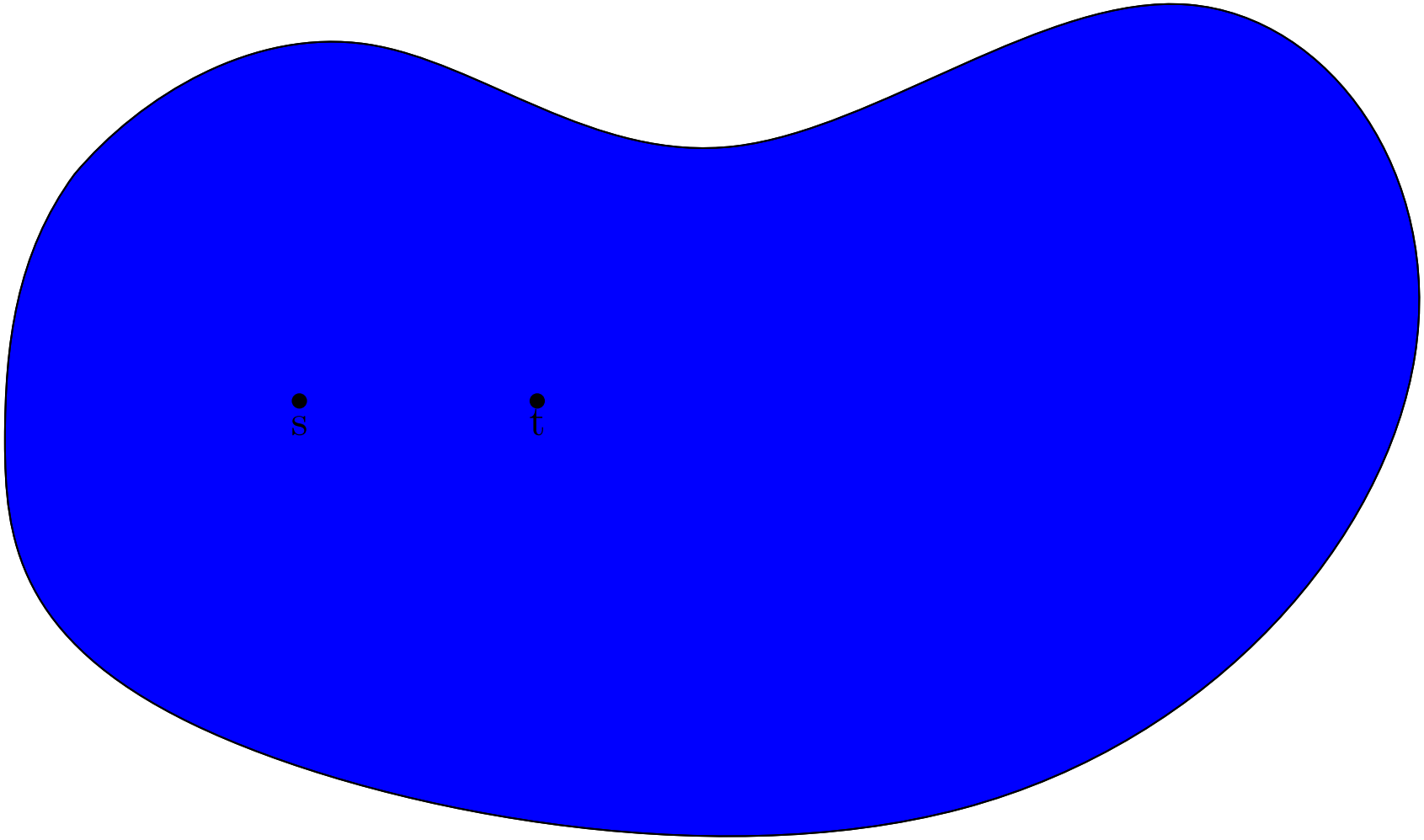
Beschleunigung der Anfragen:

- Suchraum
  - Anzahl abgearbeiteter Knoten
  - Anzahl relaxierter Kanten
  - Problem: Overhead durch Beschleunigung?
- Anfragezeit (Vergleichbarkeit?)
- auf Basis von $k \geq 1\,000$ Zufallsanfragen, d. h. wähle s und t zufällig (gleichverteilt)

Schematischer Suchraum, Dijkstra



Schematischer Suchraum, Dijkstra



Abbruch

Beobachtung:

- durchsuchen des ganzen Graphen wenig “sinnvoll”
- vor allem wenn s und t nah beinander sind

Idee:

- stop die Anfrage, sobald t aus der Queue entfernt wurde
- Korrektheit analog zu Dijkstra
- im Schnitt: Beschleunigungsfaktor 2

Abbruch

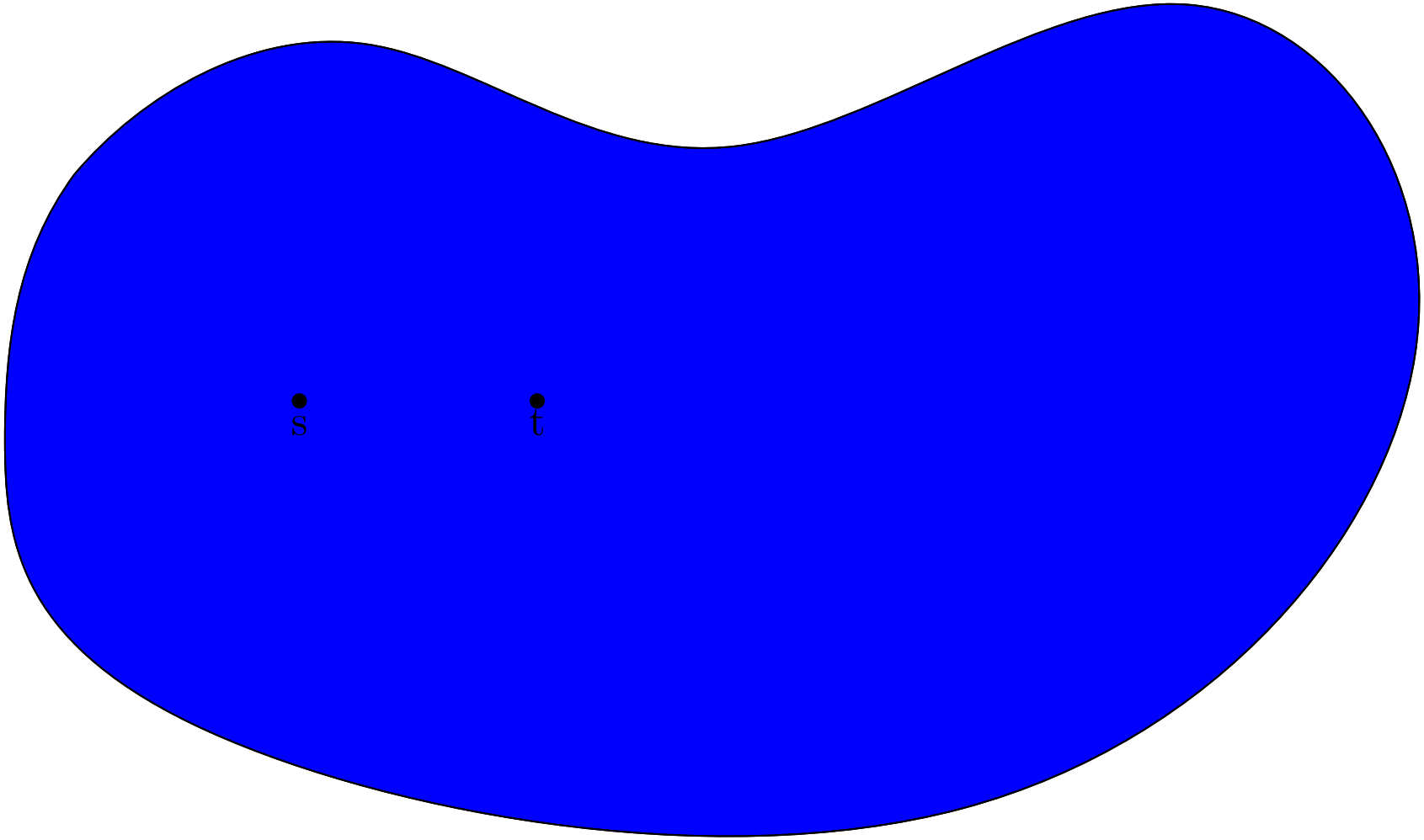
Beobachtung:

- durchsuchen des ganzen Graphen wenig “sinnvoll”
- vor allem wenn s und t nah beinander sind

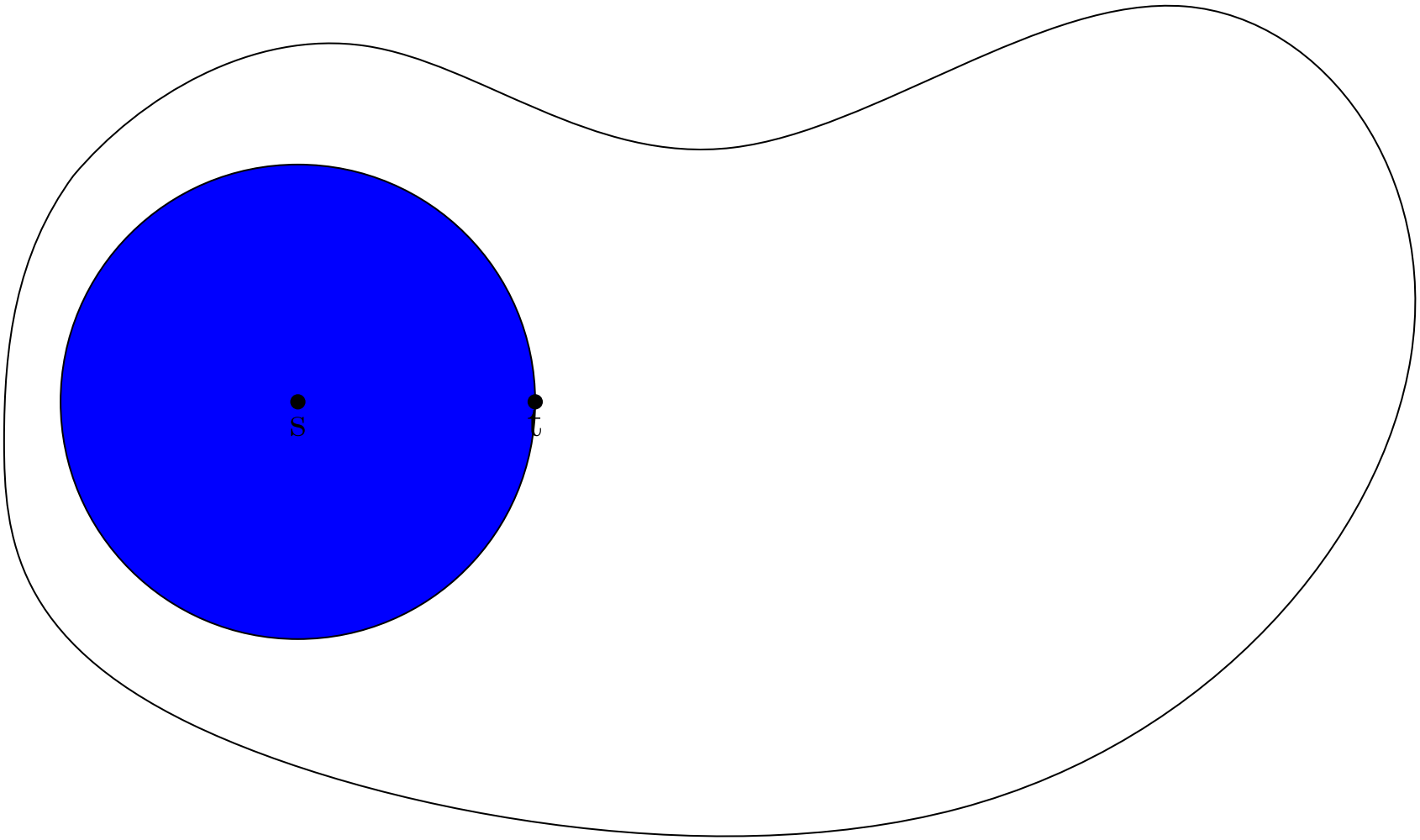
Idee:

- stop die Anfrage, sobald t aus der Queue entfernt wurde
- Korrektheit analog zu Dijkstra
- im Schnitt: Beschleunigungsfaktor 2

Schematischer Suchraum, Dijkstra



Schematischer Suchraum, Dijkstra



Initialisierung

1 **for** *all nodes* $v \in V$ **do**
2 $\quad d[v] = \infty$, $p[v] = \texttt{NULL}$

Problem:

- $\mathcal{O}(n)$
- viele Anfragen auf gleichem Netzwerk
- wird auch ausgeführt, wenn s und t im vorigen Lauf nah beinander waren

Idee:

- lasse Counter mitlaufen

Initialisierung

1 **for** *all nodes* $v \in V$ **do**
2 $\quad d[v] = \infty$, $p[v] = \texttt{NULL}$

Problem:

- $\mathcal{O}(n)$
- viele Anfragen auf gleichem Netzwerk
- wird auch ausgeführt, wenn s und t im vorigen Lauf nah beinander waren

Idee:

- lasse Counter mitlaufen

Initialisierung

1 **for** *all nodes* $v \in V$ **do**
2 $\quad d[v] = \infty$, $p[v] = \texttt{NULL}$

Problem:

- $\mathcal{O}(n)$
- viele Anfragen auf gleichem Netzwerk
- wird auch ausgeführt, wenn s und t im vorigen Lauf nah beinander waren

Idee:

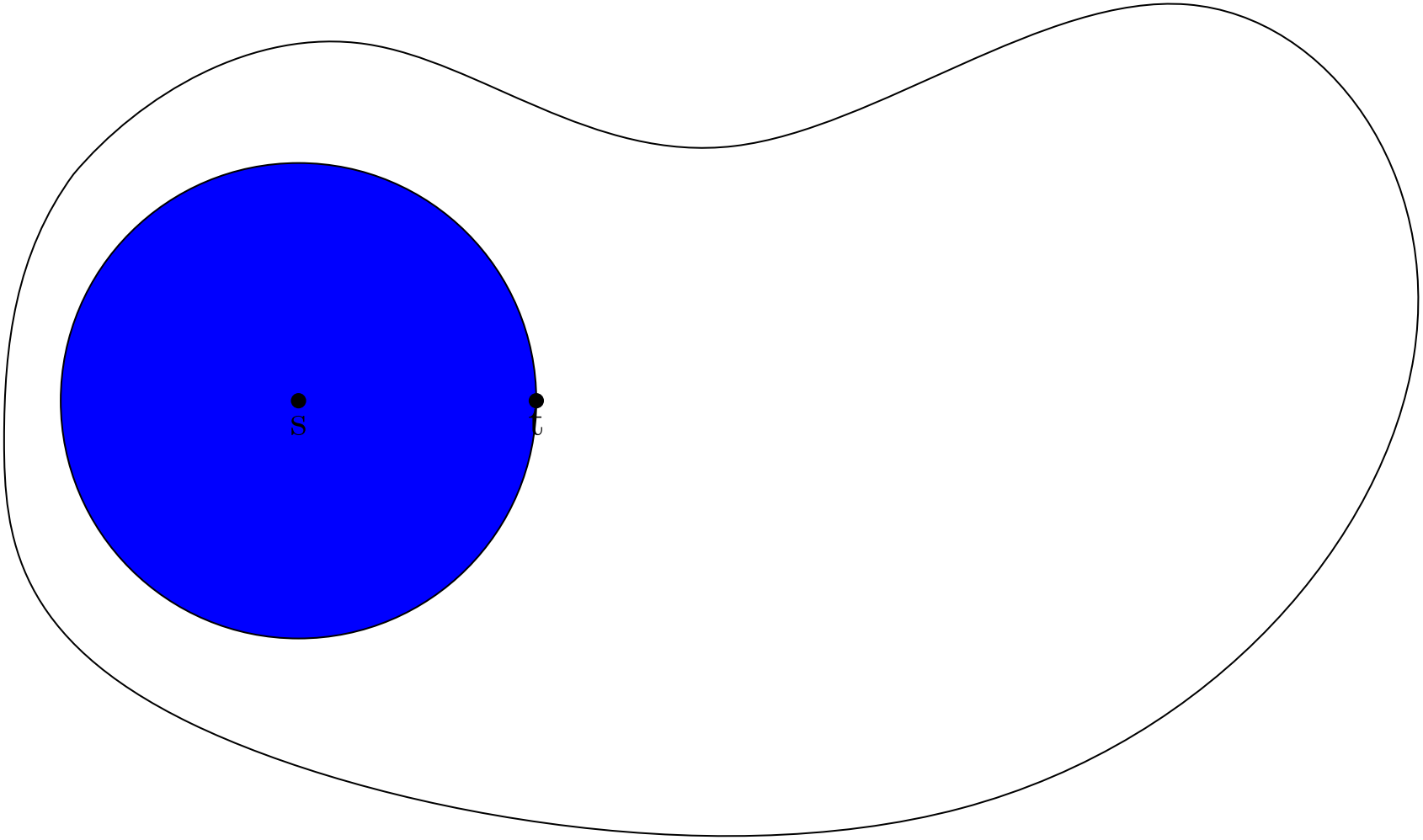
- lasse Counter mitlaufen

Dijkstra

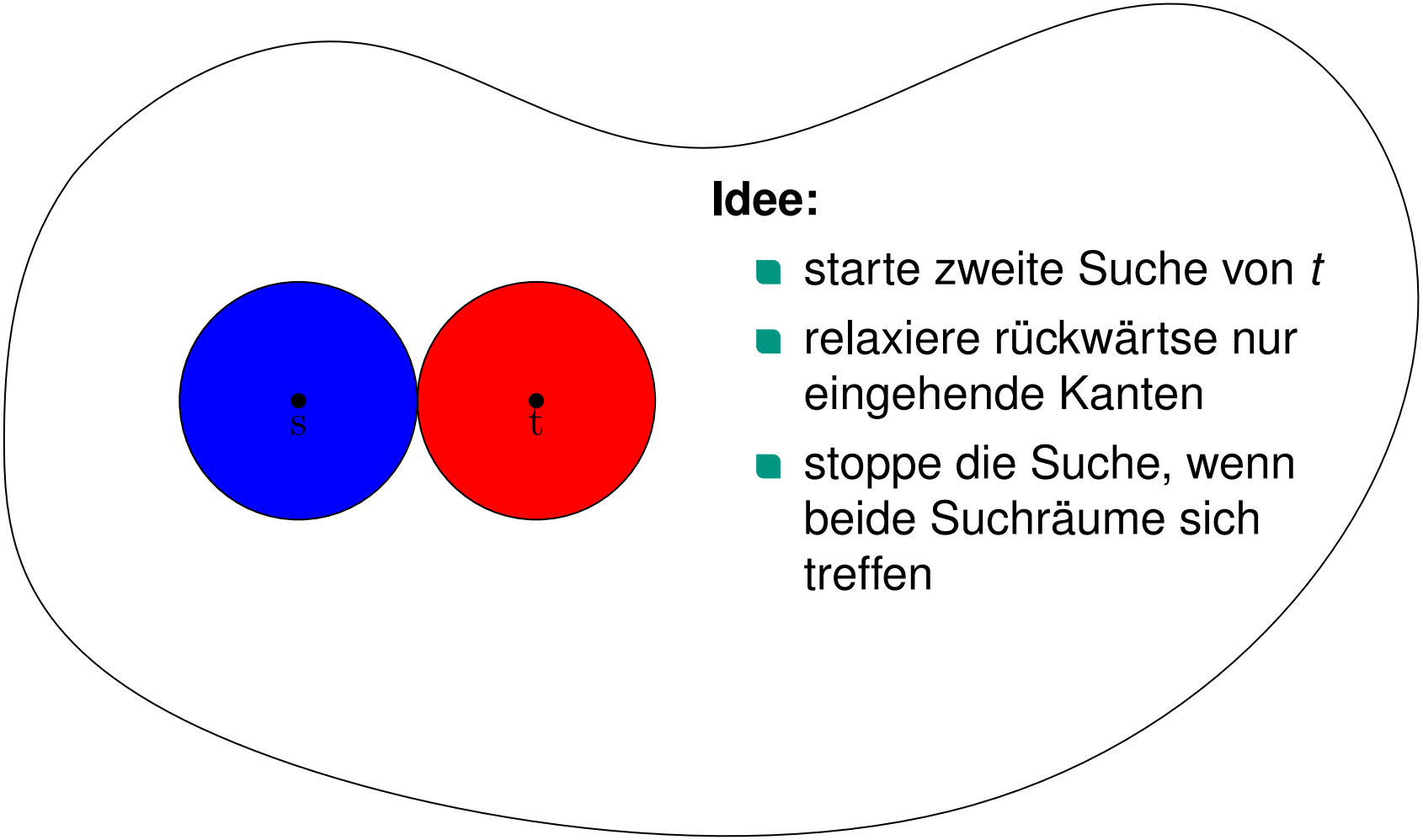
```
1  ++count
2  d[s] = 0
3  Q.clear(), Q.add(s, 0)
4  while !Q.empty() do
5      u ← Q.deleteMin()
6      if u = t then return
7      forall edges e = (u, v) ∈ E do
8          if run[v] ≠ count then
9              d[v] ← d[u] + len(e)
10             Q.insert(v, d[v])
11             run[v] ← count
12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

Bidirektionale Suche

Bidirektionale Suche



Bidirektionale Suche



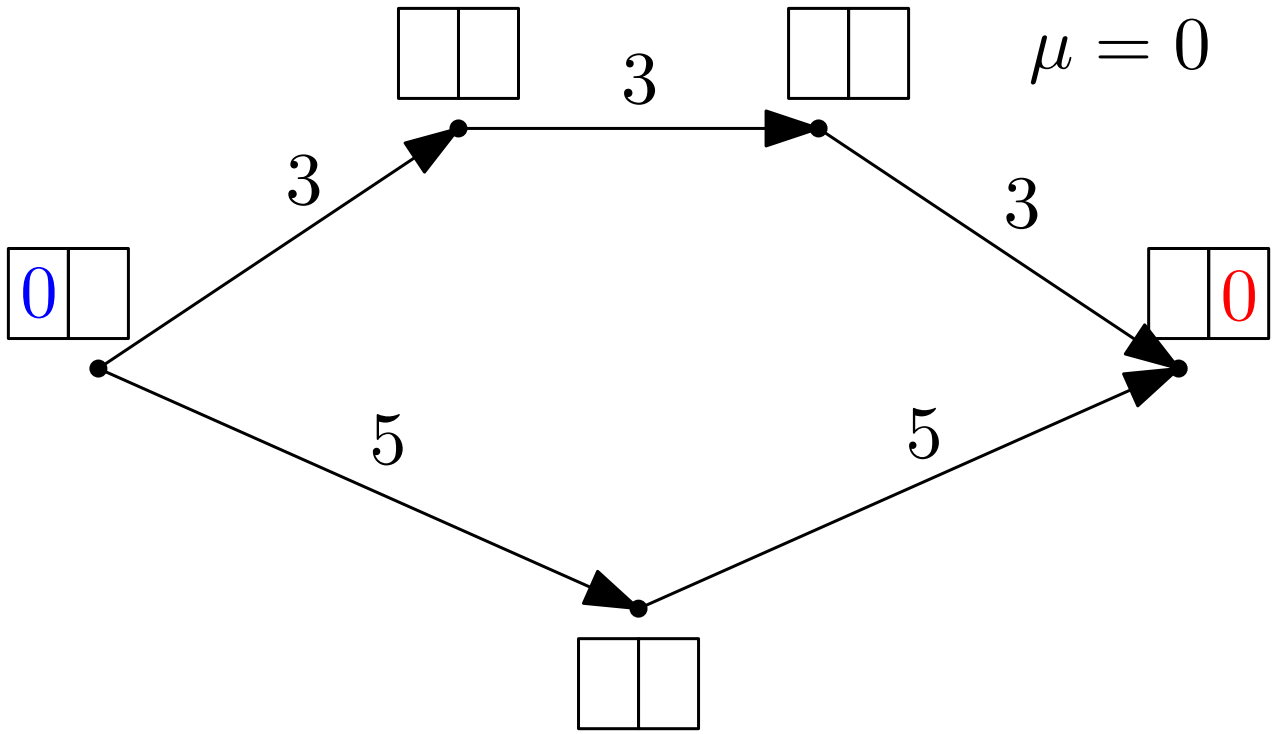
Idee:

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

Bidirektionale Suche

Anfrage:

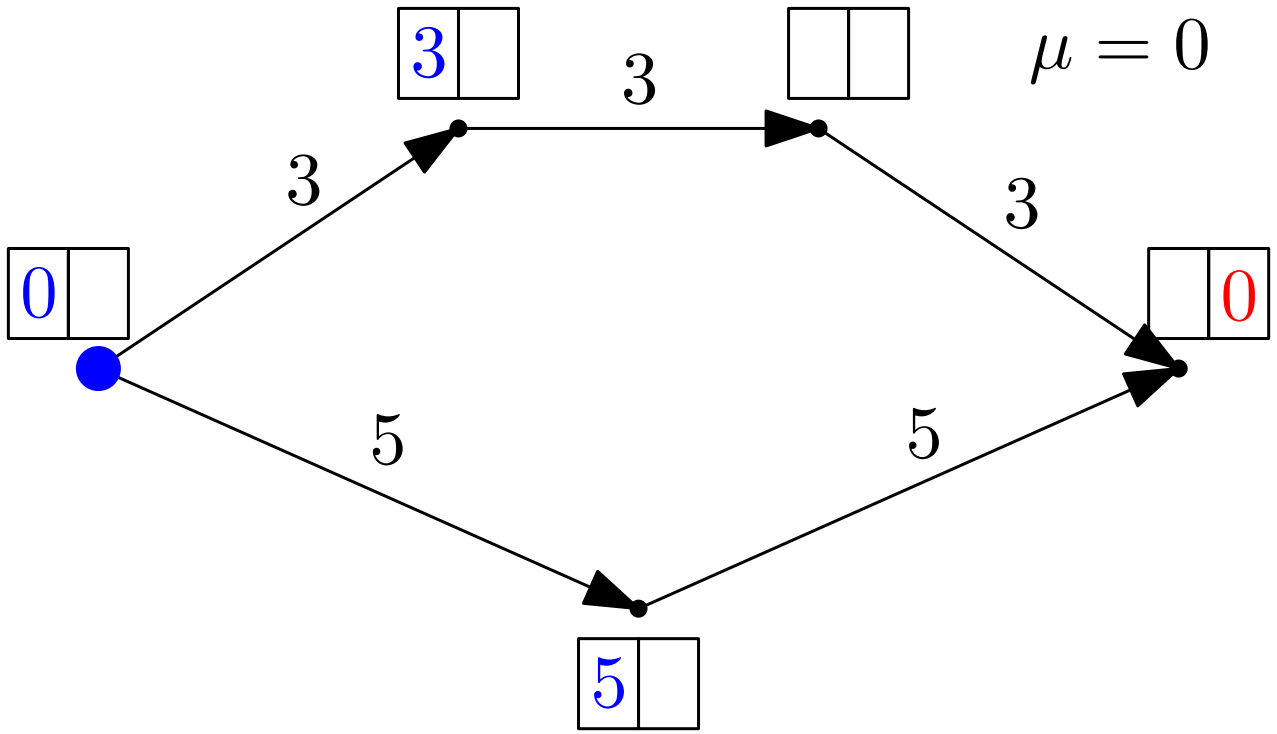
- alterniere Vorwärts- und Rückwärtsuche
  - vorwärts: relaxiere ausgehende Kanten
  - rückwärts: relaxiere eingehende Kanten



Bidirektionale Suche

Anfrage:

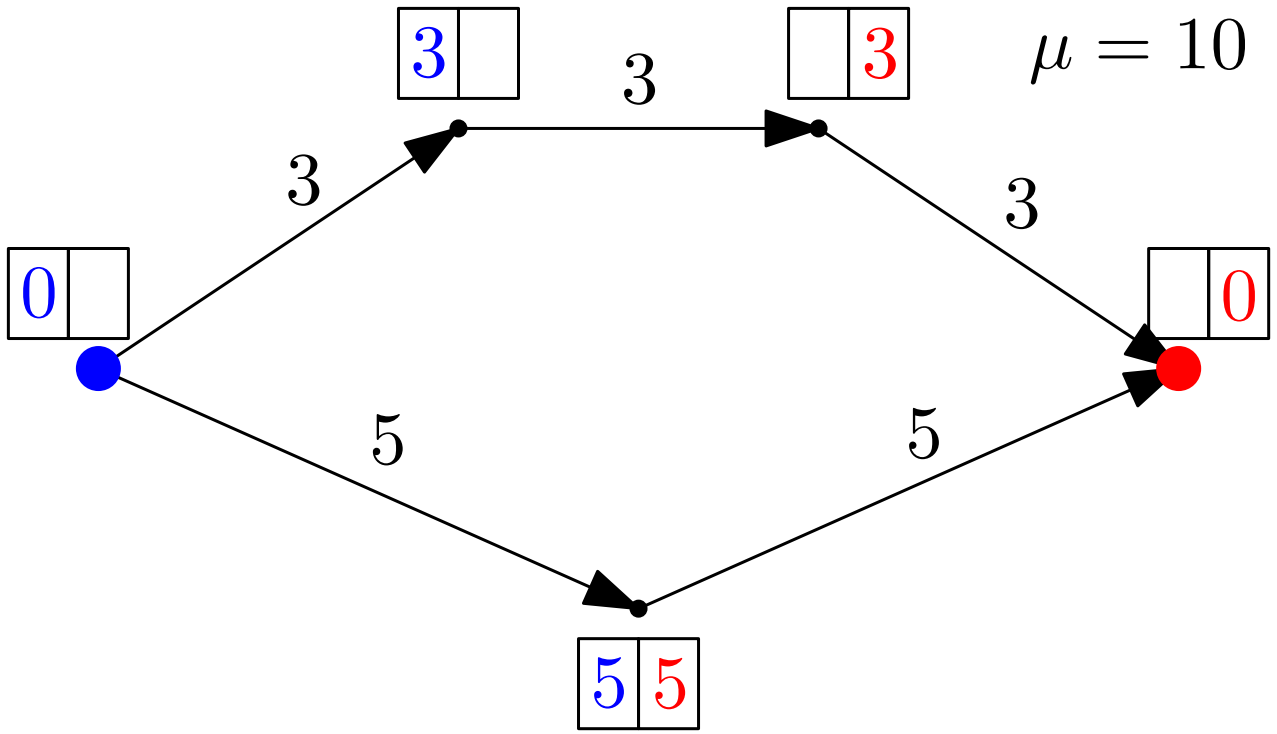
- alterniere Vorwärts- und Rückwärtsuche
  - vorwärts: relaxiere ausgehende Kanten
  - rückwärts: relaxiere eingehende Kanten



Bidirektionale Suche

Anfrage:

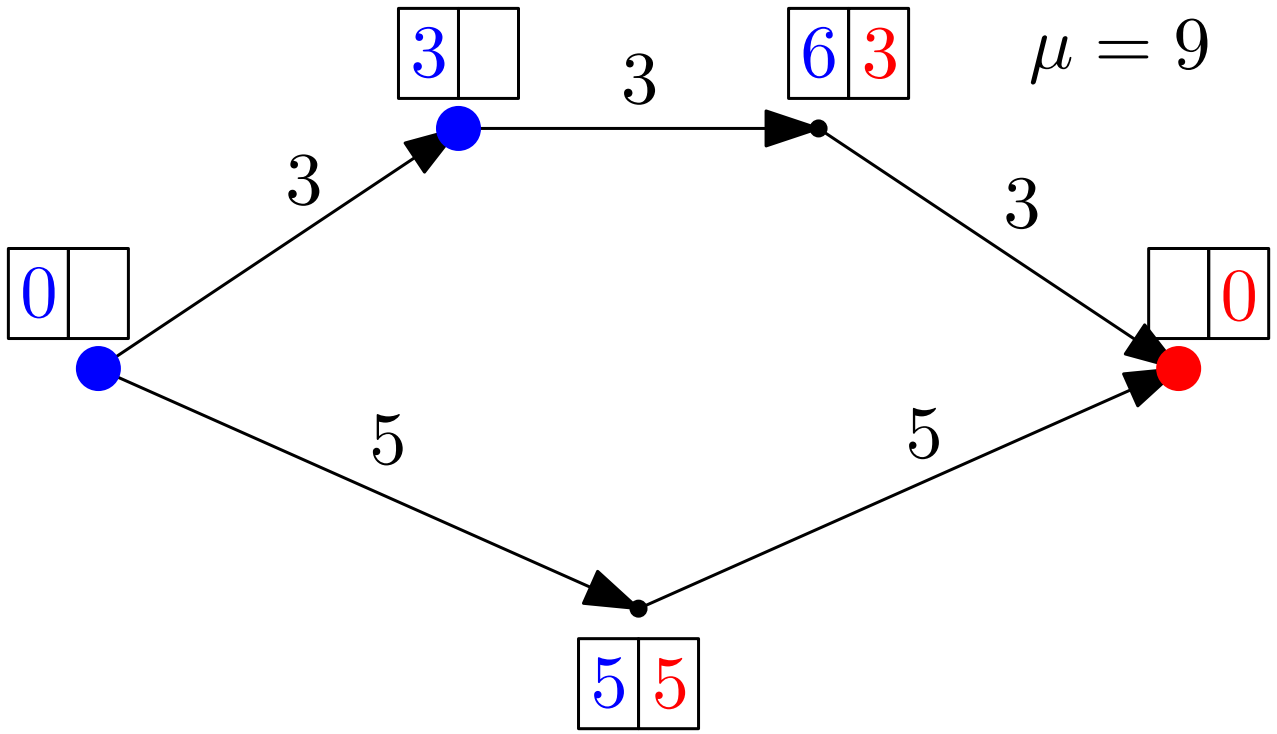
- alterniere Vorwärts- und Rückwärtsuche
  - vorwärts: relaxiere ausgehende Kanten
  - rückwärts: relaxiere eingehende Kanten



Bidirektionale Suche

Anfrage:

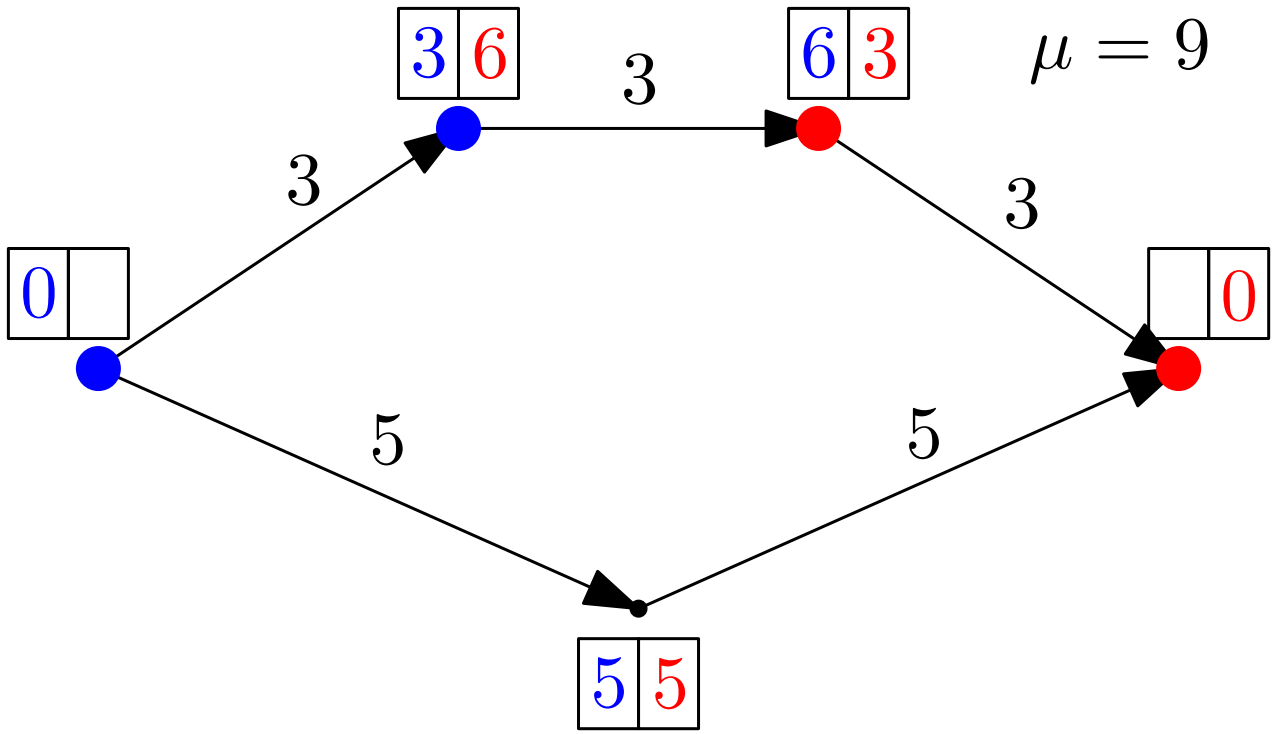
- alterniere Vorwärts- und Rückwärtsuche
  - vorwärts: relaxiere ausgehende Kanten
  - rückwärts: relaxiere eingehende Kanten



Bidirektionale Suche

Anfrage:

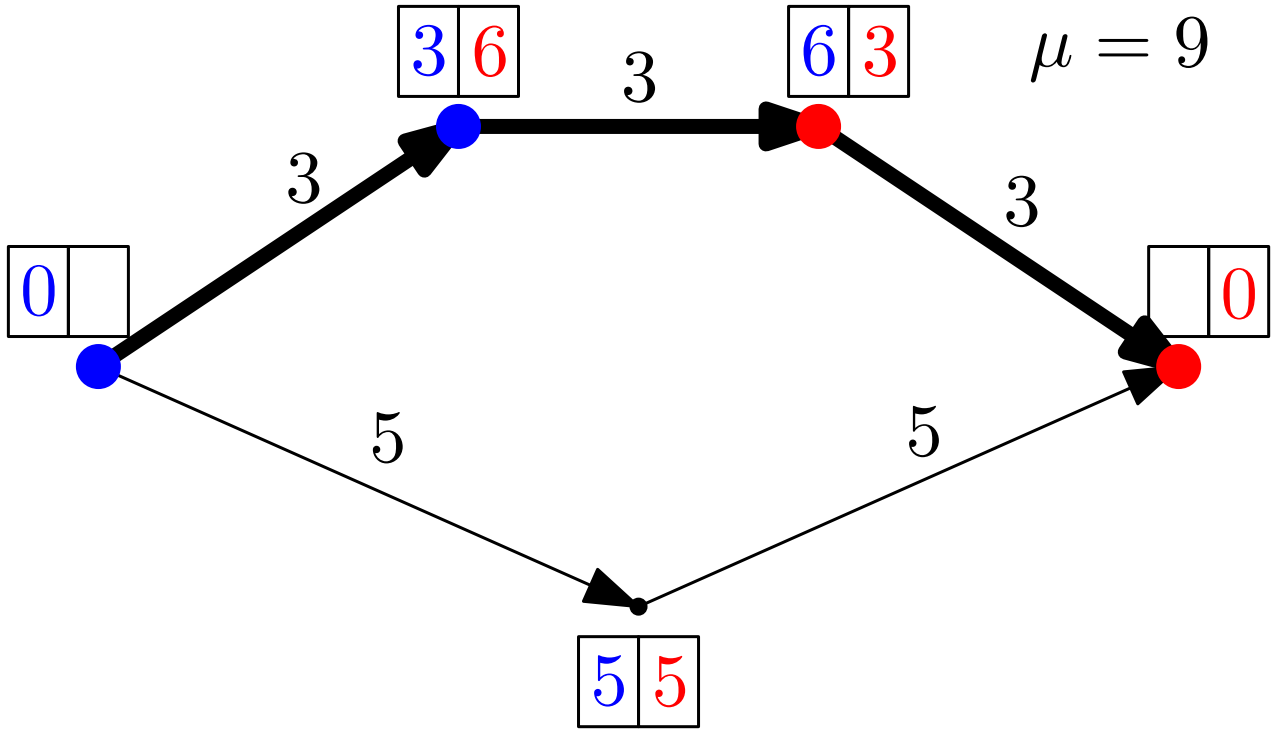
- alterniere Vorwärts- und Rückwärtsuche
  - vorwärts: relaxiere ausgehende Kanten
  - rückwärts: relaxiere eingehende Kanten



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

Definition.
Sei μ die *vorläufige Distanz* von s zu t, und $M \in V$ der Knoten, an dem sich die Suchräume treffen.

Vorgehen:

- bei Relax checken, ob Zielknoten v der Kante von anderer Suche schon besucht
  - wenn ja: $\mu := \min\{\mu, \overrightarrow{d}[v] + \overleftarrow{d}[v]\}$, ggf. $M := v$
- stoppe wenn $\mu < \text{minKey}(\overrightarrow{Q}) + \text{minKey}(\overleftarrow{Q})$

$\rightsquigarrow$ kürzester Weg: $(s, \ldots, M, \ldots, t)$ und $d(s,t) = \mu$

Bidirektionale Suche: Korrektheit

Korrektheit

Bidirektionale Suche (BS) ist korrekt.

Beweis.

Übung. □

Strategie zum Wechsel der Suchen

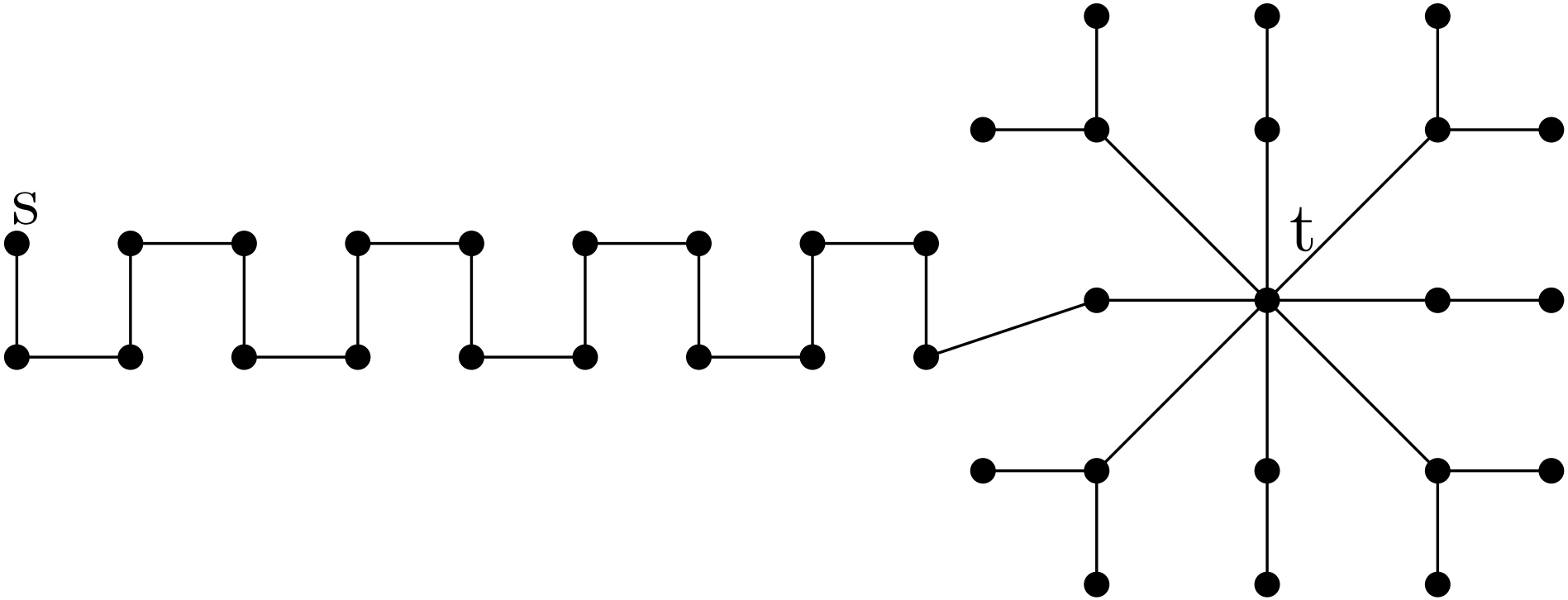
Beobachtung:

- jede Art der Abwechslung klappt

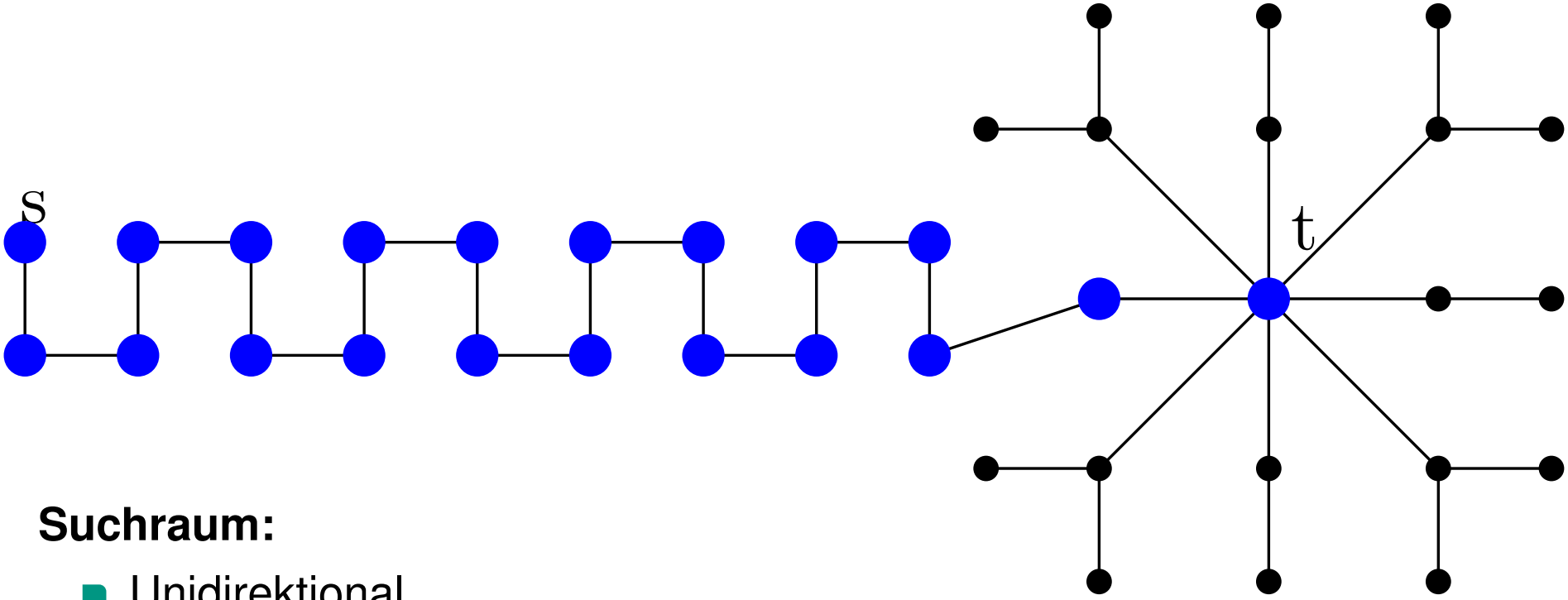
sinnvolle Strategien:

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

Worst-Case



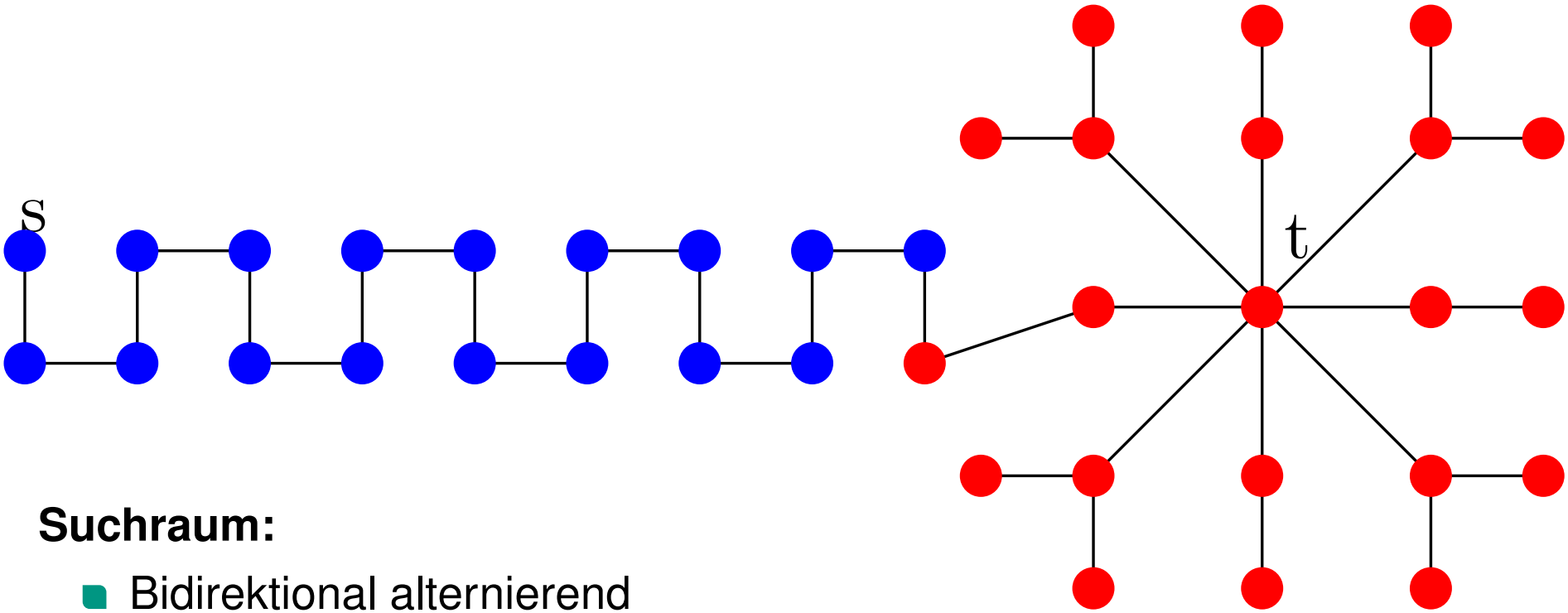
Worst-Case



Suchraum:

- Unidirektional

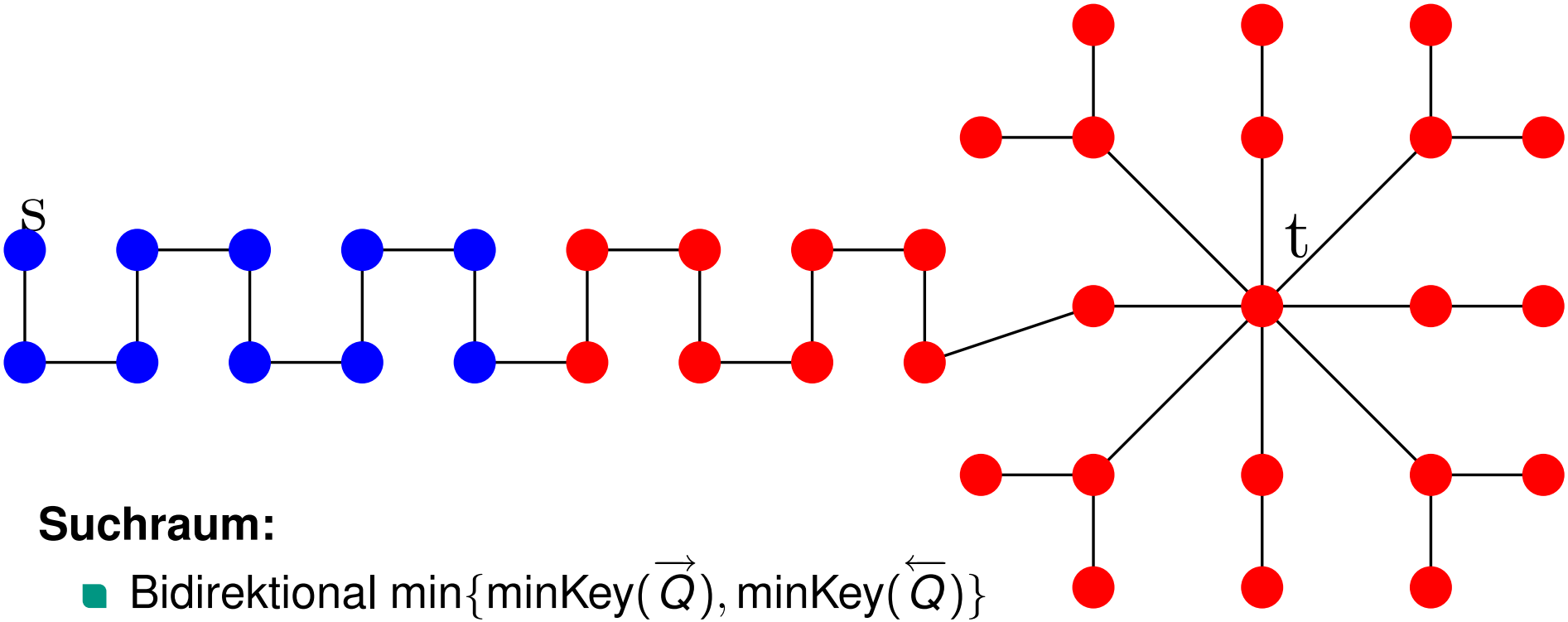
Worst-Case



Suchraum:

- Bidirektional alternierend
- Suchraum vergrößert sich

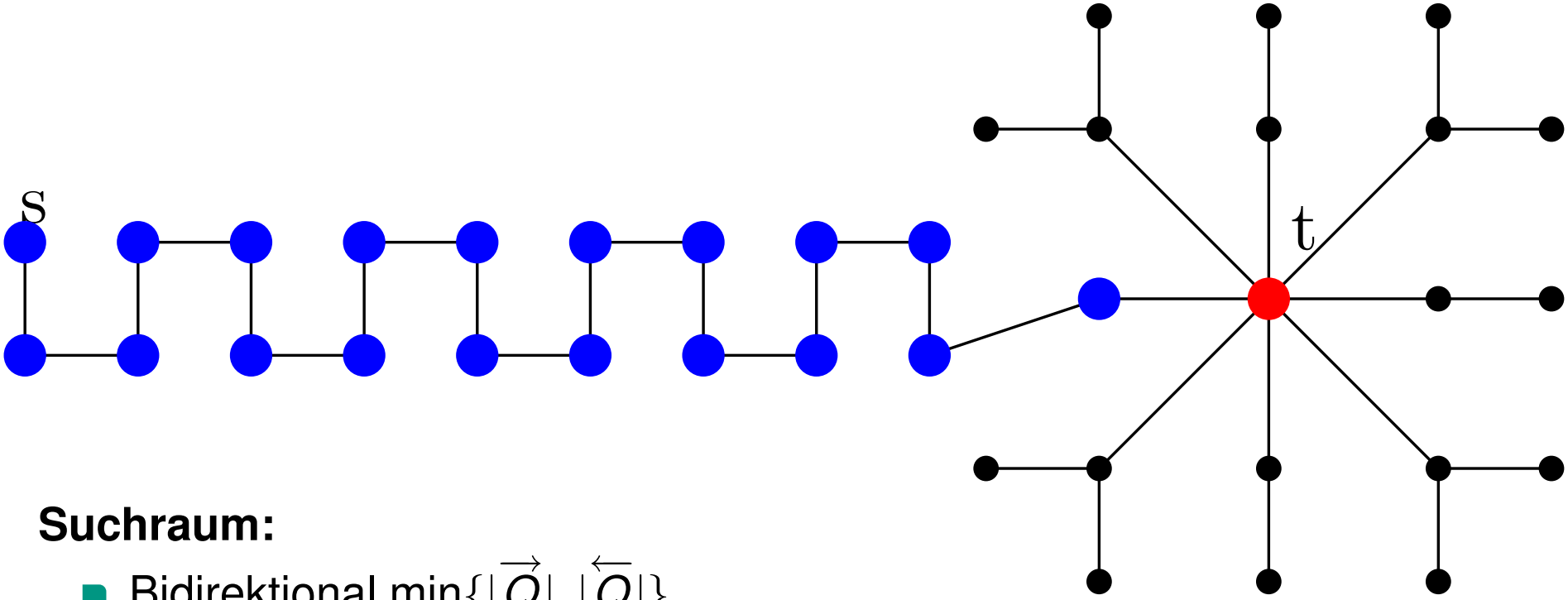
Worst-Case



Suchraum:

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

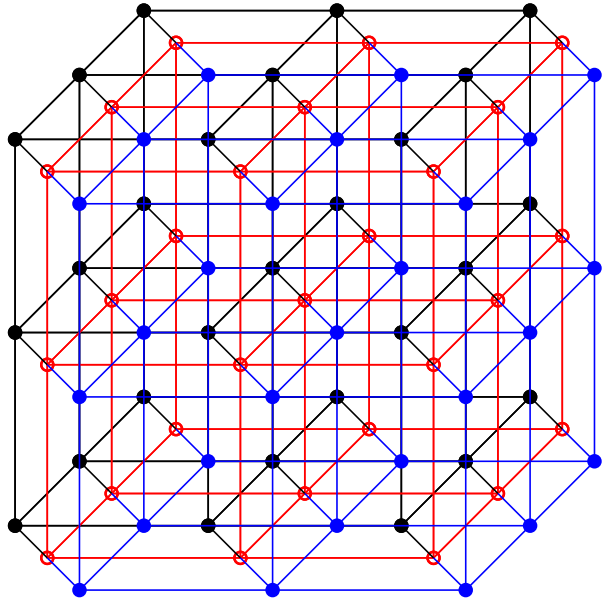
Worst-Case



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

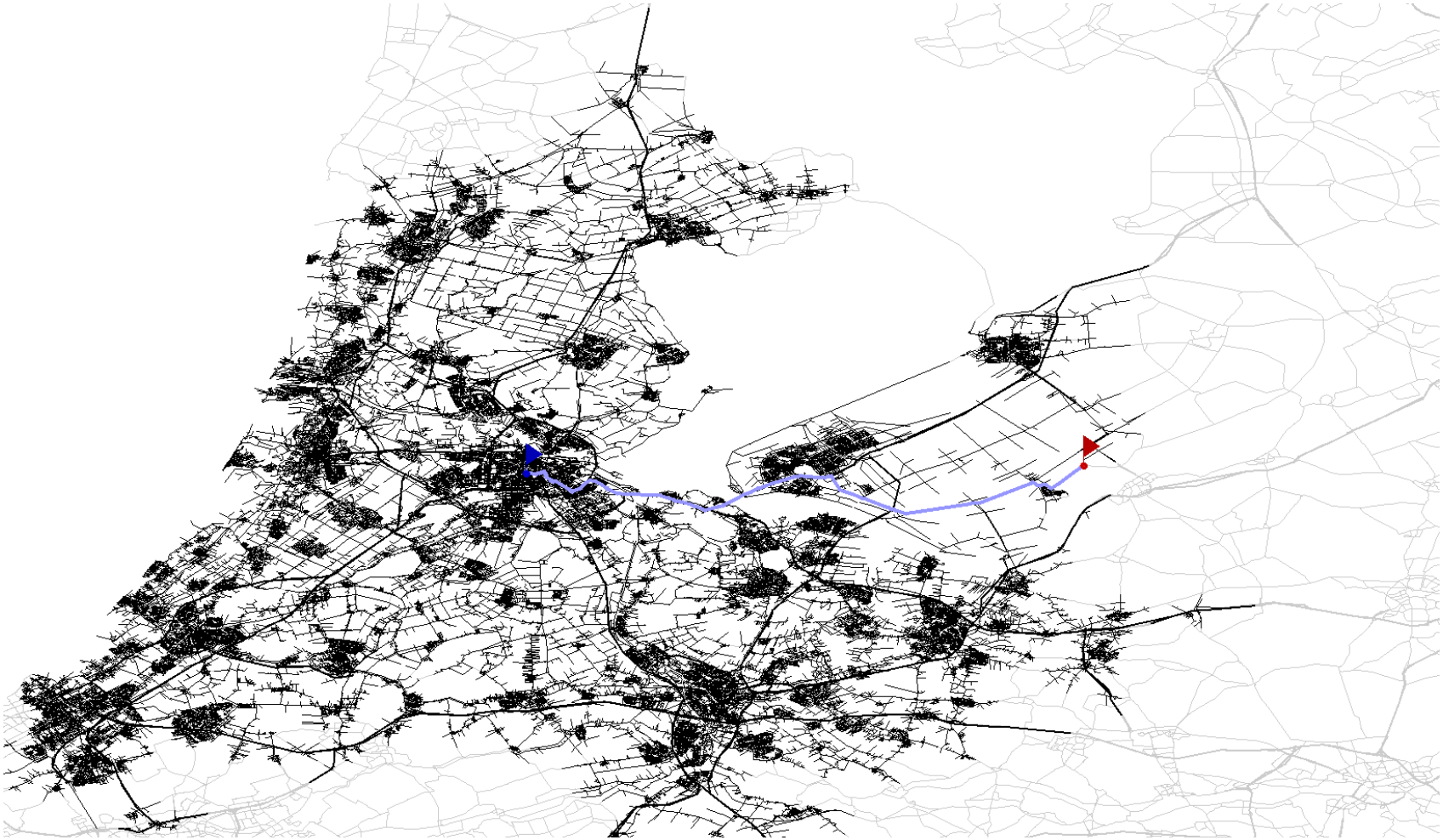
In Straßennetzen:

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

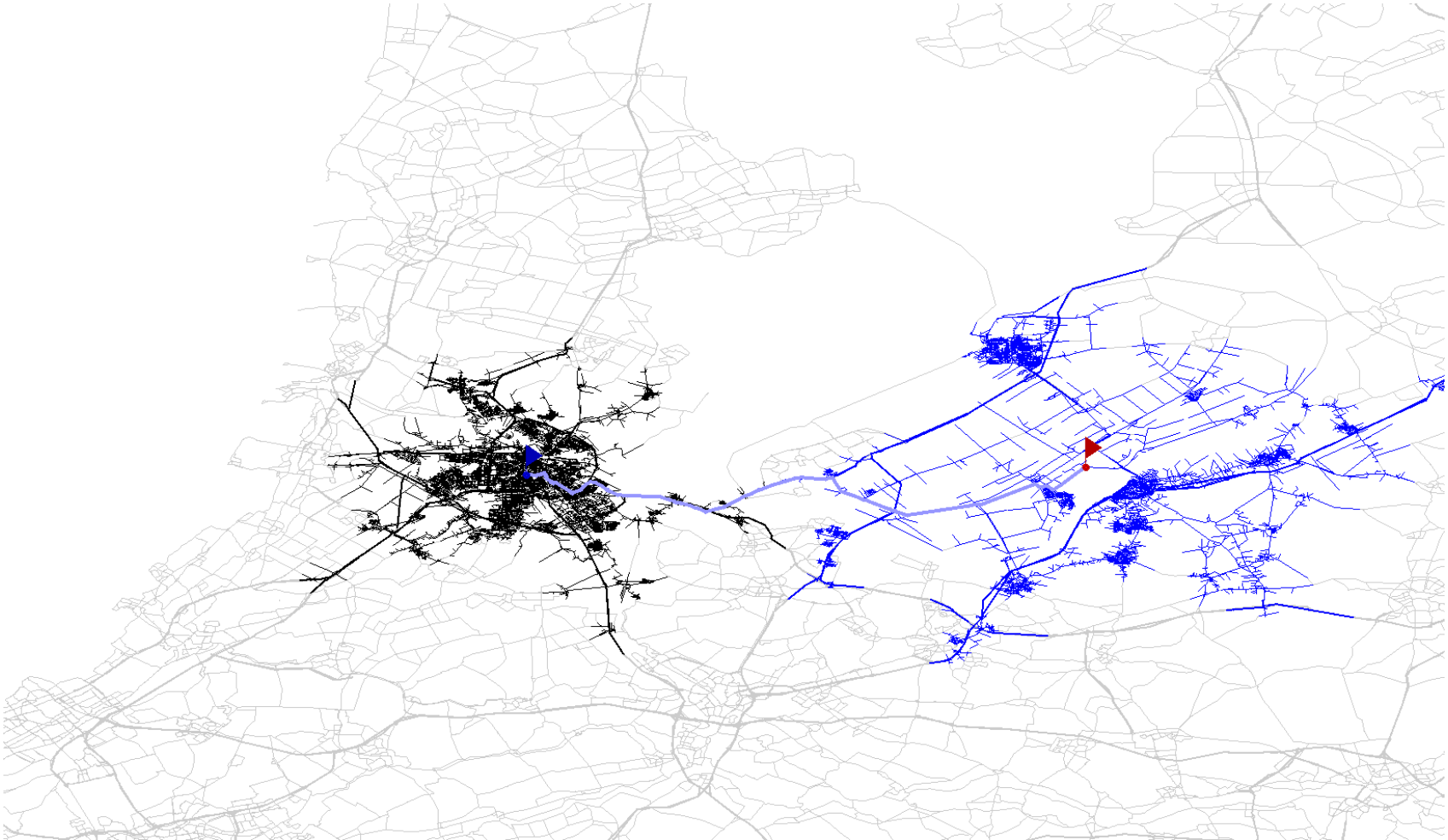
Somit Beschleunigung:

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

Beispiel

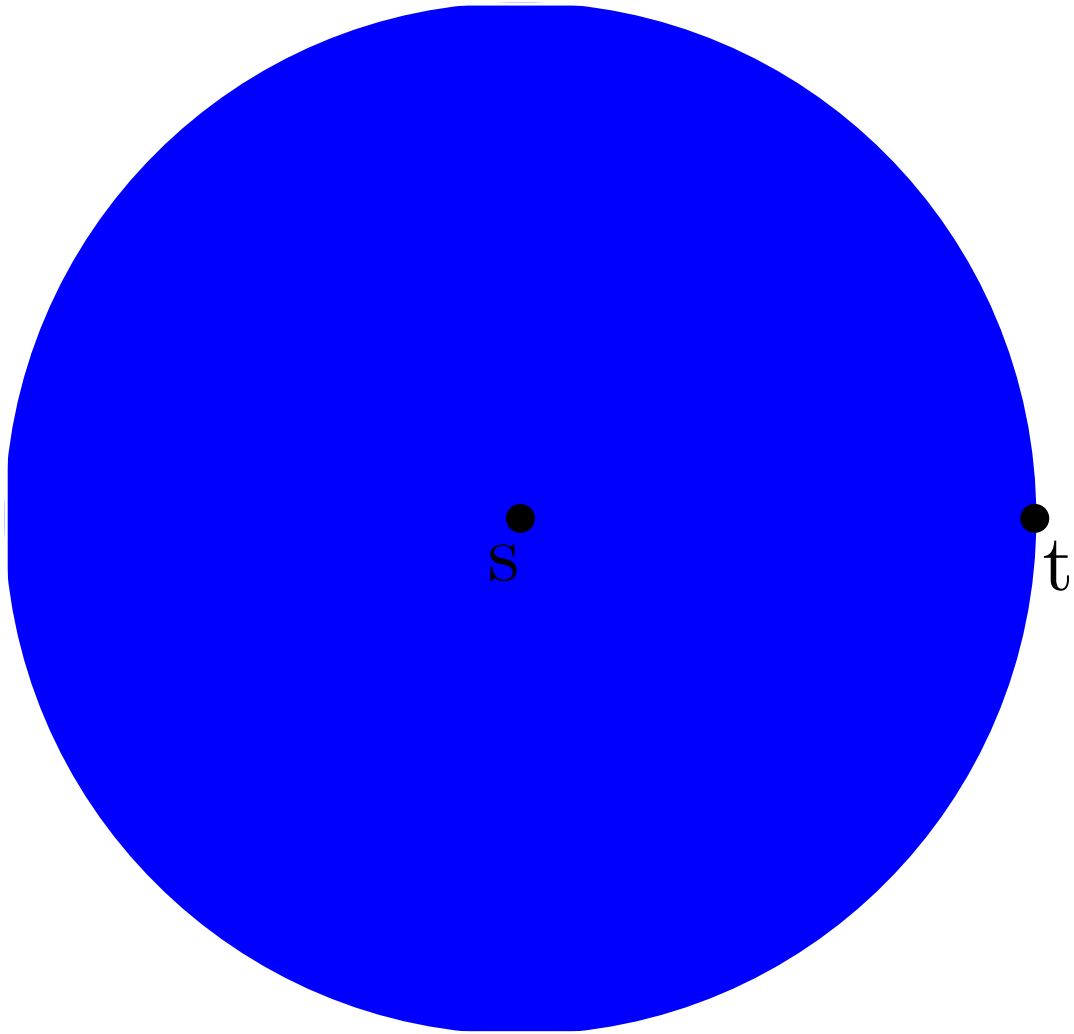


Beispiel

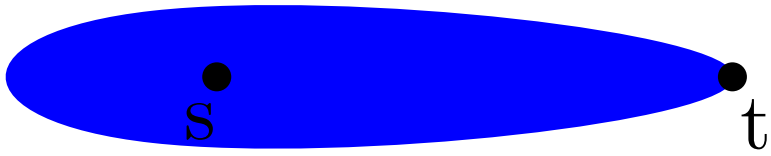


Zielgerichtete Suche

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

Idee:

- berechne Potential $\pi : V \to \mathbb{R}$ mit

$$\pi(u) \leq \text{len}(u, v) + \pi(v) \text{ für alle } (u, v) \in E$$

- während der Anfrage: nutze $d[u] + \pi(u)$ als Key in Q

Dijkstra

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     forall 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^*

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

```
1 d[s] = 0
2 Q.clear(), Q.add(s, π(s))
3 while !Q.empty() do
4     u ← Q.deleteMin()
5     forall 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 $d[u] + \pi(u) < d[v] + \pi(v)$
  - wenn $\pi \equiv 0$ dann gleicher Suchraum
- Abbruchkriterium bleibt erhalten

A^*: Korrektheit

Korrektheit

A^* ist korrekt.

Beweis.

Übung. □

Gültige Potentiale: Eigenschaften I

Sei $\pi(u) := d(u, t)$ und $\text{len}_\pi(u, v) := \text{len}(u, v) - \pi(u) + \pi(v)$, dann gilt

(i) $\text{len}_\pi(u, v) = 0$ für alle Kanten (u, v) auf einem kürzesten Weg von s nach t,

(ii) $\text{len}_\pi(u, v) > 0$ für alle restlichen Kanten (u, v).

(iii) Es werden nur Knoten auf kürzesten Wegen angeschaut.

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.

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

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

- $\pi(u) \leq d(u,t)$ 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 $L \subset V$ (≈ 16)
- berechne Distanzen von und zu allen Landmarken
- dann gilt:

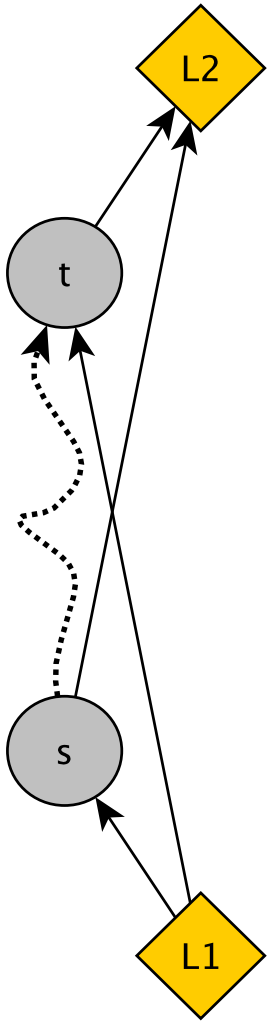
$$\begin{aligned} d(u,t) &\geq d(L_1,t) - d(L_1,u) \\ d(u,t) &\geq d(u,L_2) - d(t,L_2) \end{aligned}$$

  für alle $u \in V$.

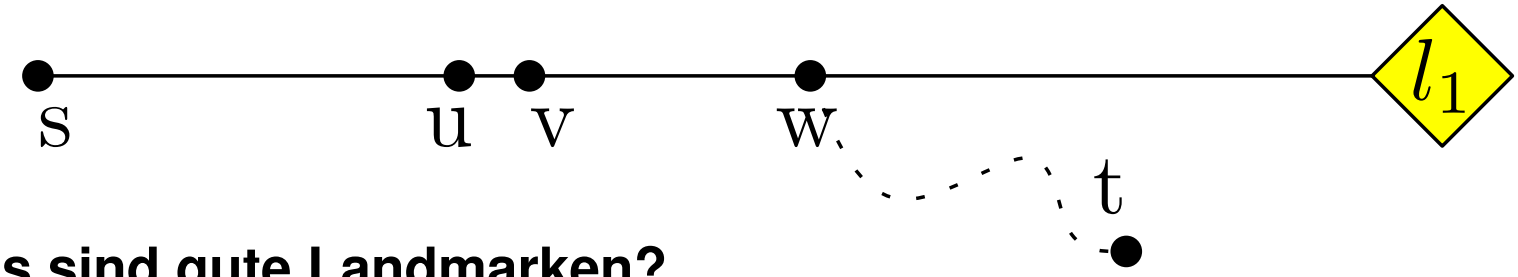
- somit ist

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

  ein gültiges Potential



Eigenschaften Landmarken



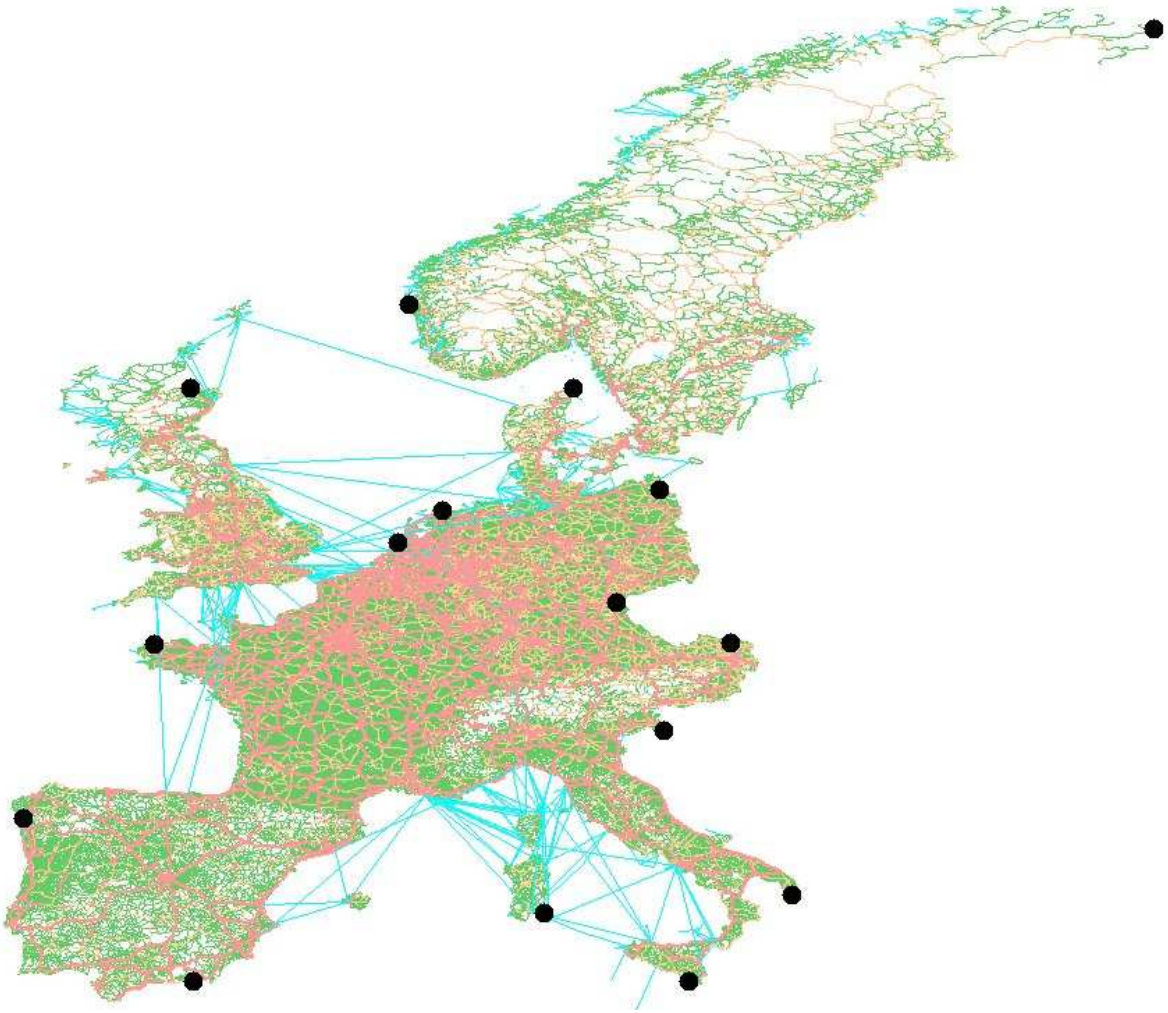
Was sind gute Landmarken?

- $\pi(u) = d(u, l_1) - d(t, l_1)$, $\pi(v) = d(v, l_1) - d(t, l_1)$
- also $\text{len}_\pi(u, v) = \text{len}(u, v) - d(u, l_1) + d(v, l_1) = 0$
- 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



Ende

Nächste Vorlesung:
Freitag, 30. April, 09:45 Uhr

Ende

Karlsruher Institut für Technologie

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.

Username: `routePlanning` Passwort: `ss10`