Karlsruher Institut für Technologie

Algorithmen für Routenplanung

15. Vorlesung, Sommersemester 2018

Tobias Zündorf | 27. Juli 2018

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

Elektromobilität

Elektrofahrzeuge (EVs):

- Transportmittel der Zukunft
- Emissionsfreie Mobilität

Aber:

- Akkukapazität eingeschränkt (und damit Reichweite)
- Lange Ladezeiten, wenig öffentliche Ladestationen
- “Reichweitenangst”

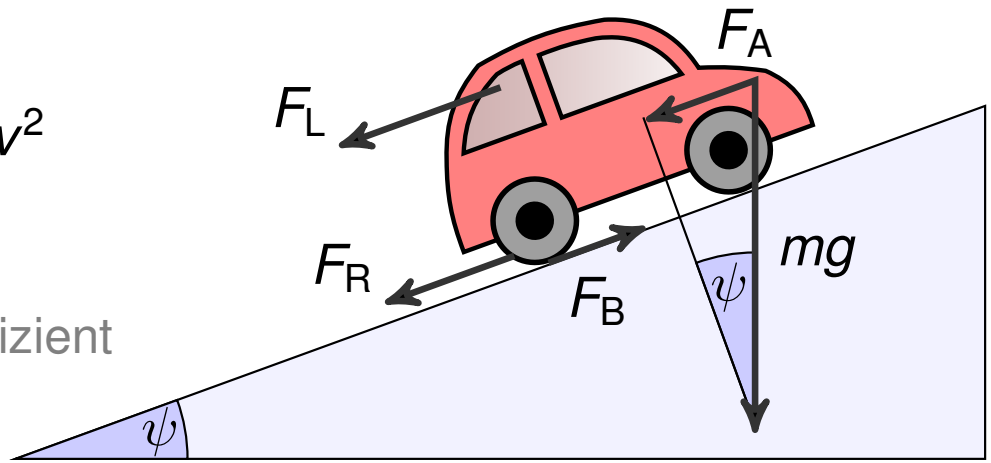
$\Rightarrow$ Berücksichtigung von Energieverbrauch bei der Routenplanung

Verbrauchsmodell

Wie kommen wir an Energieverbrauch?

Typisches (vereinfachtes) Modell

- Rollwiderstand: $F_R = \mu_R mg$
  μ_R: Rollwiderstandskoeffizient
  m: Fahrzeugmasse
  g: Erdbeschleunigung
- Luftwiderstand: $F_L = \frac{1}{2}\rho A C_W v^2$
  ρ: Luftdichte
  A: Stirnfläche
  C_W: Strömungswiderstandskoeffizient
  v: Geschwindigkeit
- Anstieg: $F_A = mg \sin \psi$
- Beschleunigung: $F_B = ma$
  a: Beschleunigung



Insgesamt benötigte Kraft: $F = F_R + F_L + F_A + F_B$

Verbrauchsmodell

Wie kommen wir an Energieverbrauch?

Typisches (vereinfachtes) Modell

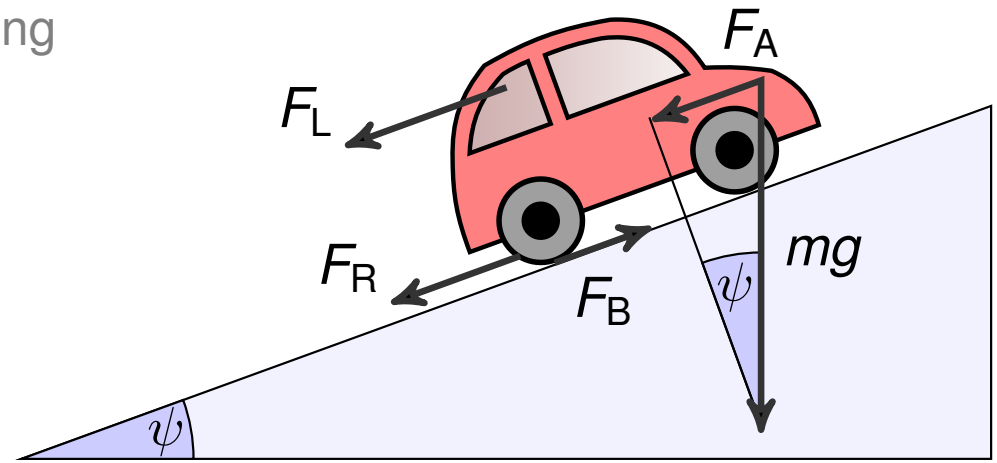
Nötige Leistung: $P = \frac{F \cdot v}{\eta} + P_0$

v: Geschwindigkeit

η: Effizienzkoeffizient der Übersetzung

P_0: Leistung für Nebenverbraucher (Klimaanlage,. . .)

Energieverbrauch über einen bestimmten Zeitraum:
$E = \int_0^T P \, dt$



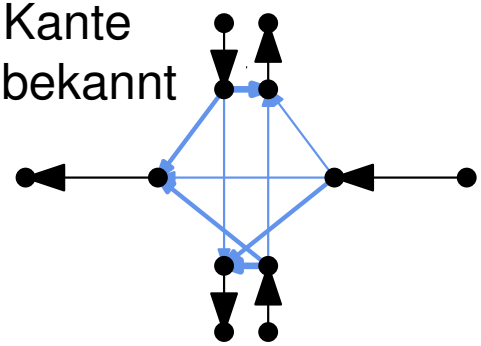
Beobachtung: Nur Beschleunigung, Geschwindigkeit und Steigung hängen vom Straßensegment ab

Alle anderen Parameter sind konstant oder vom Fahrzeugzustand abhängig (Masse, Nebenverbraucher)

Verbräuche auf Kanten

Wie bekommen wir Verbräuche auf den Kanten im Straßengraphen?

- Nur Beschleunigung, Geschwindigkeit und Steigung von Straßensegment abhängig
- Annahme:
  - Geschwindigkeit und Steigung ist konstant auf jeder Kante
  - Andere Parameter (Masse, Nebenverbraucher) sind bekannt
- Zwischenknoten, wenn sich Steigung oder Geschwindigkeitsbegrenzung ändern
- Extra Kanten für Brems-/Beschleunigungskosten

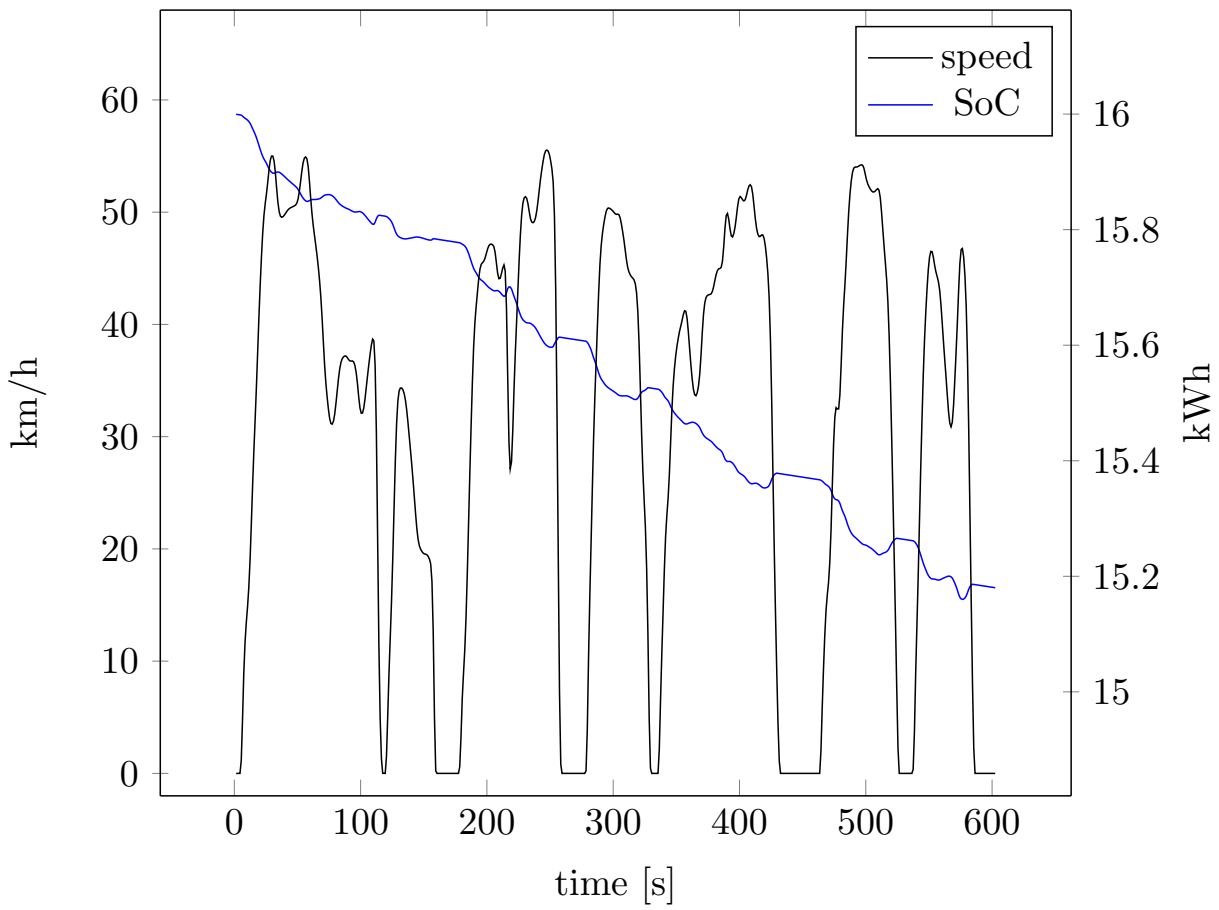


$\Rightarrow$ Verbrauch auf Segment vereinfacht (für realistische Steigung) als

$$\lambda_1 v^2 + \lambda_2 s + \lambda_3$$

v: Geschwindigkeit
s: Steigung
λ_1, λ_2, λ_3: (nichtnegative) Konstanten

Alternative Datenquellen



Ermittle Kantengewichte mit Hilfe von Messungen aus Realwelt-Tests, Fahrzeugsimulation, . . .

Rekuperation

Besonderheiten von Elektrofahrzeugen:

- Verbrauch kann auch negativ werden
  - F_A ist negativ beim Bergabfahren
  - F_B ist negative beim Bremsen
- Elektromotor fungiert als Generator
- Rückgewinnung von Energie (Akku wird aufgeladen)

Aber: keine negativen Zyklen (physikalische Gesetze)

Energieverbrauch als Metrik

Besonderheiten:

- Rekuperation
  - Rückgewinnung von Energie möglich (bergab fahren, bremsen)
  - Negative Kantengewichte

  Aber: keine negativen Zyklen (physikalische Gesetze)

- Akkukapazität (Battery Constraints)
  - Akku darf nicht leer laufen
    (Andernfalls Kante nicht benutzbar)
  - Akku kann nicht überschritten werden
    (Kanten können genutzt werden, aber Energie verfällt ggf.)
  - Muss für jeden Knoten eines Pfades gelten

$\Rightarrow$ Ladestand (state of charge, SoC) während Query berücksichtigen

Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

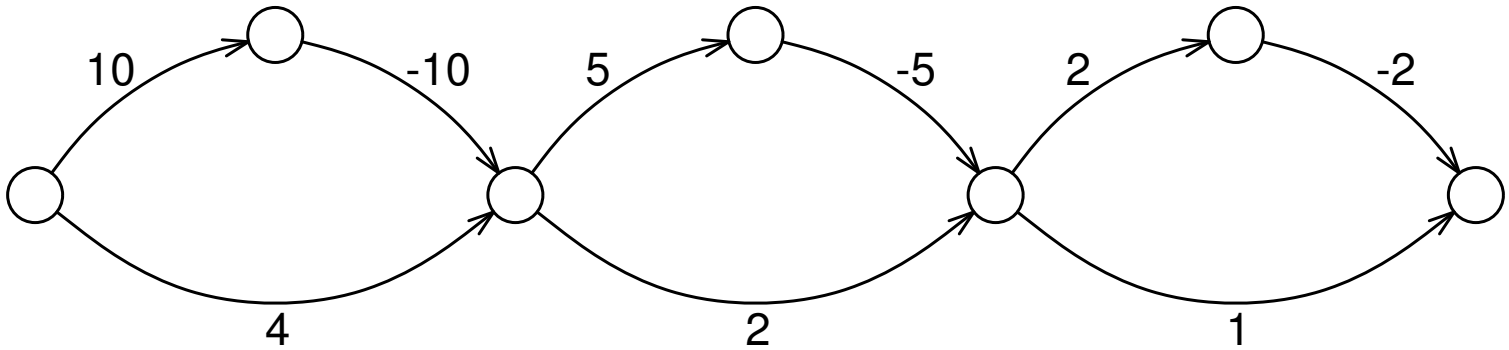
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

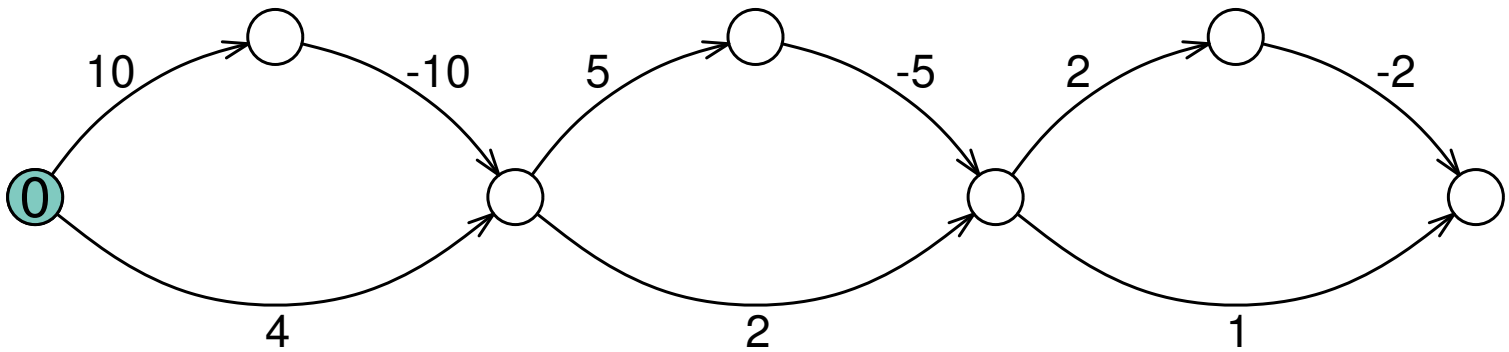


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

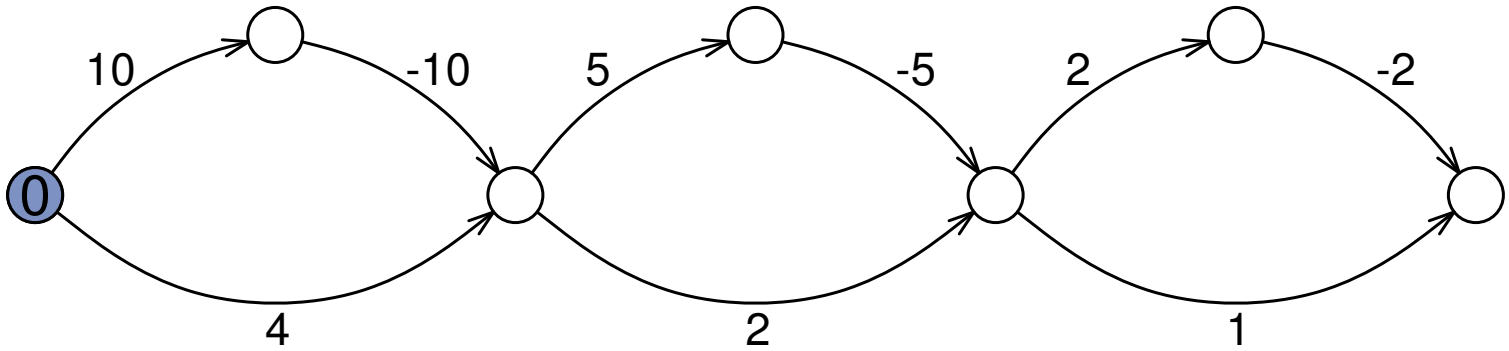


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

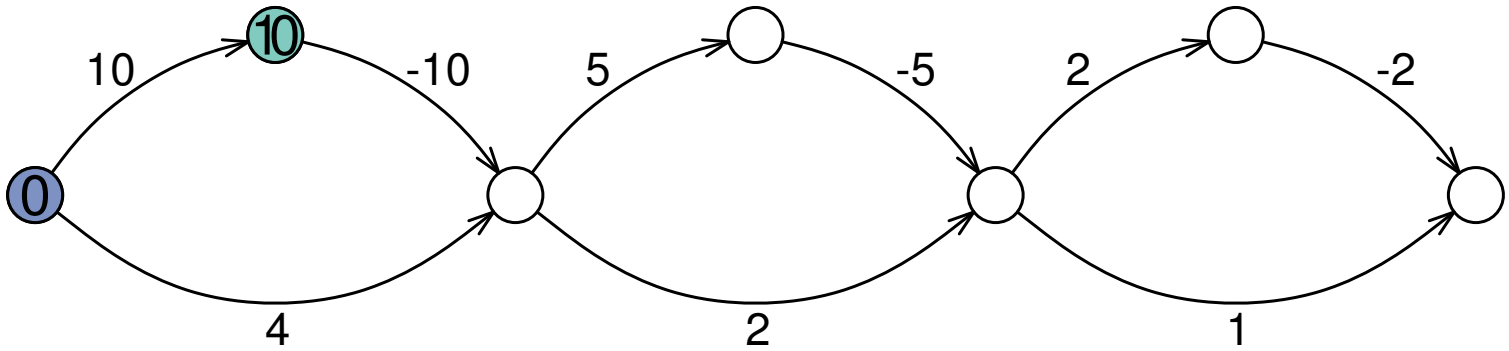


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

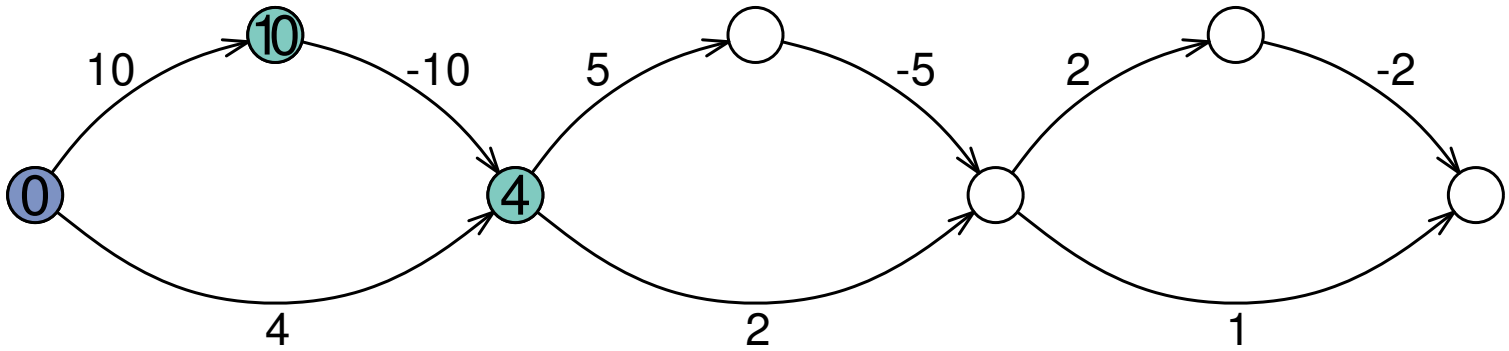


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

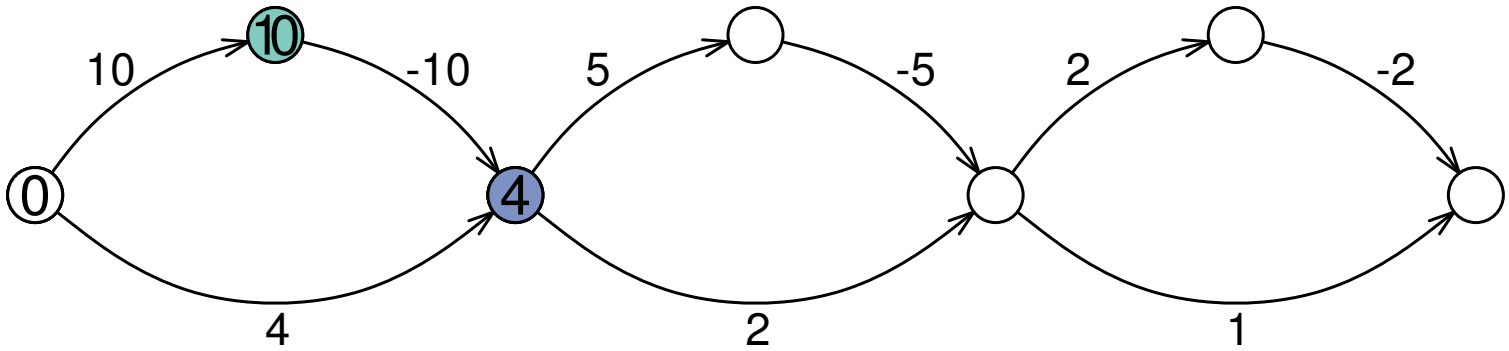


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

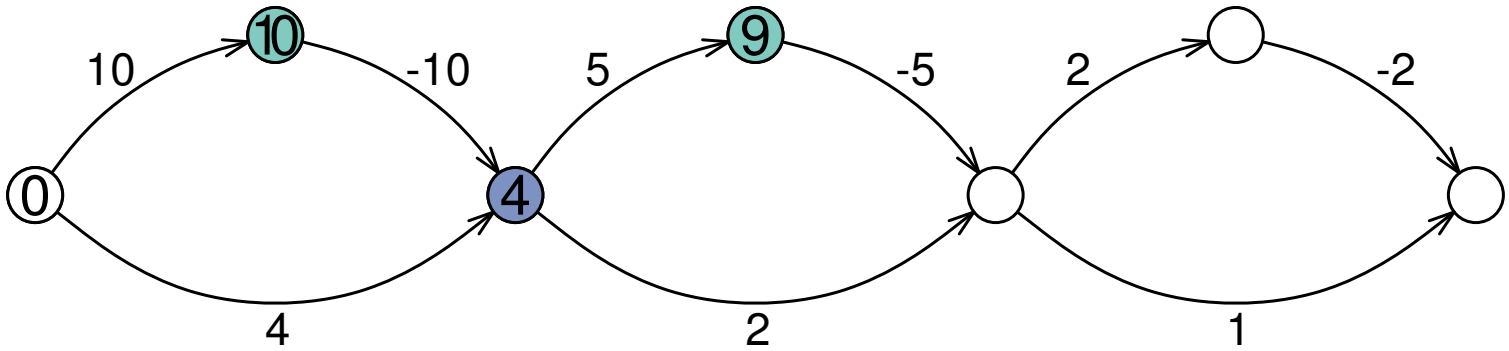


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

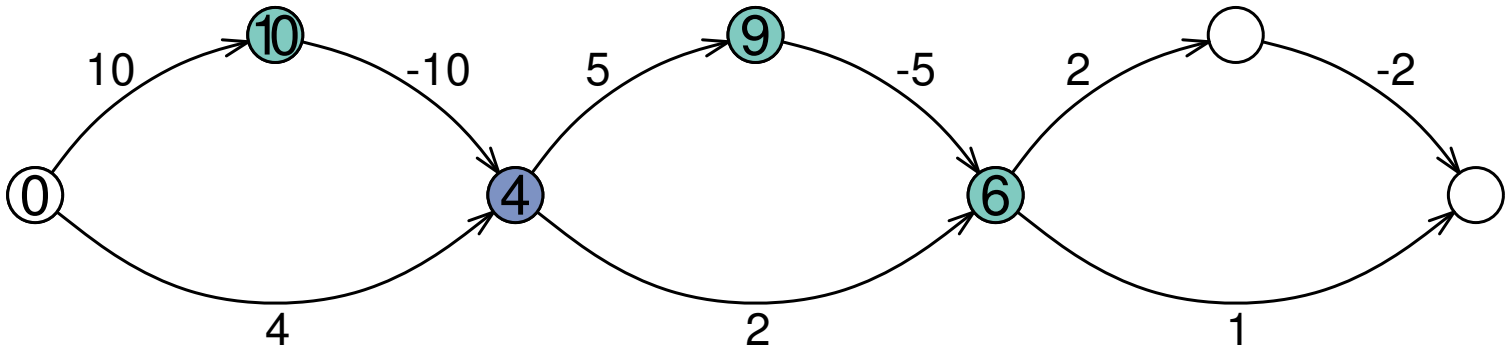


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

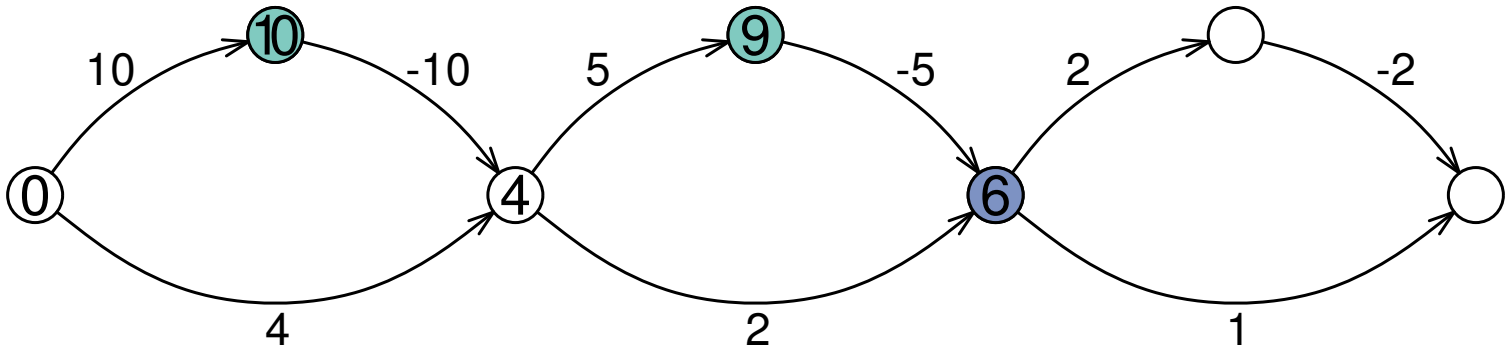


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

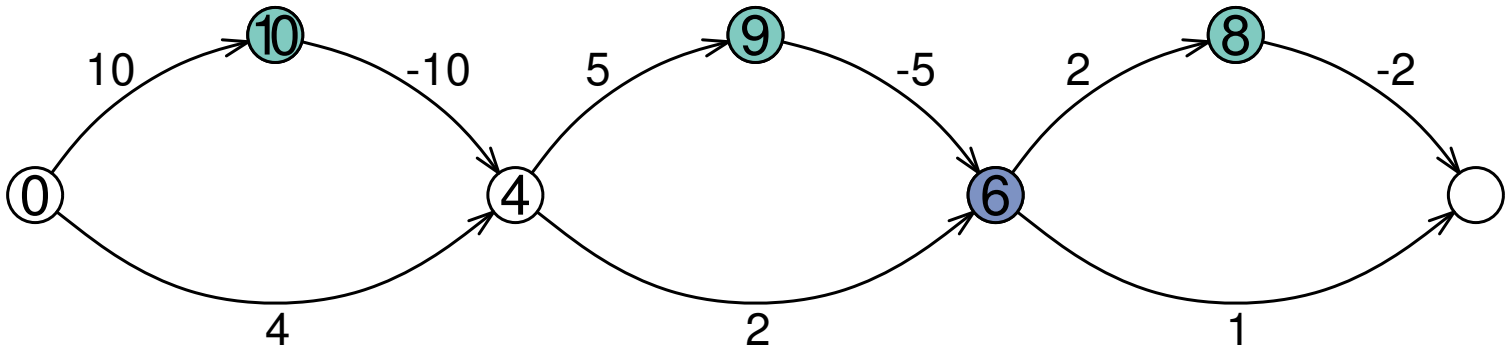


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

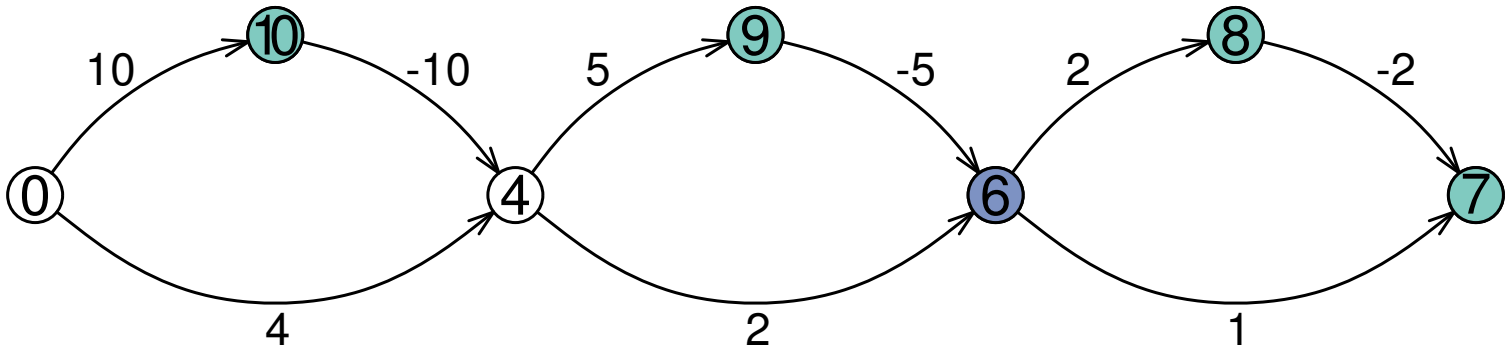


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

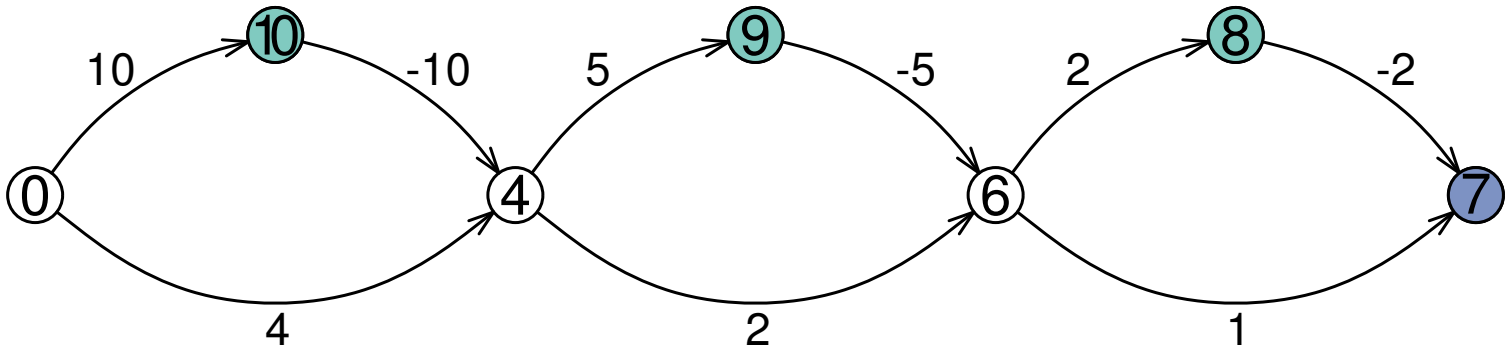


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

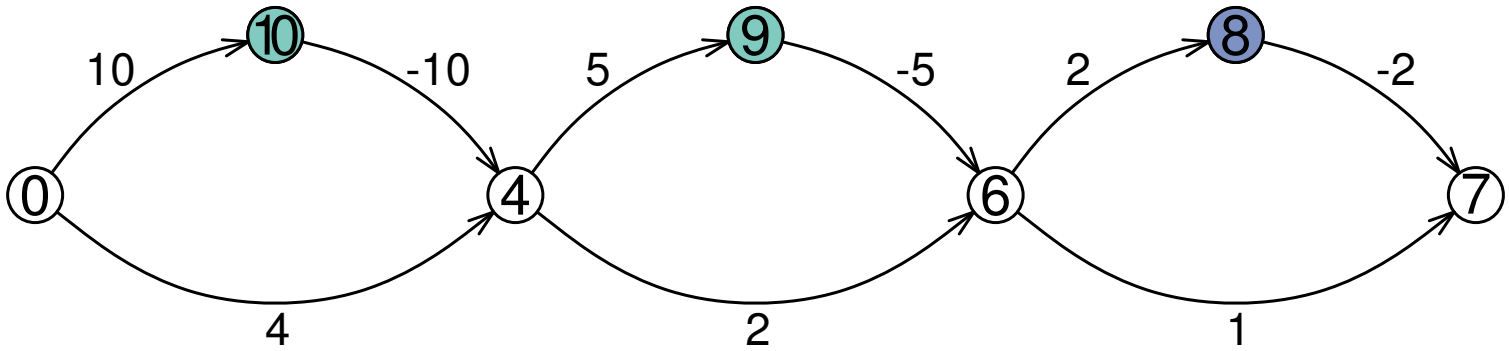


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

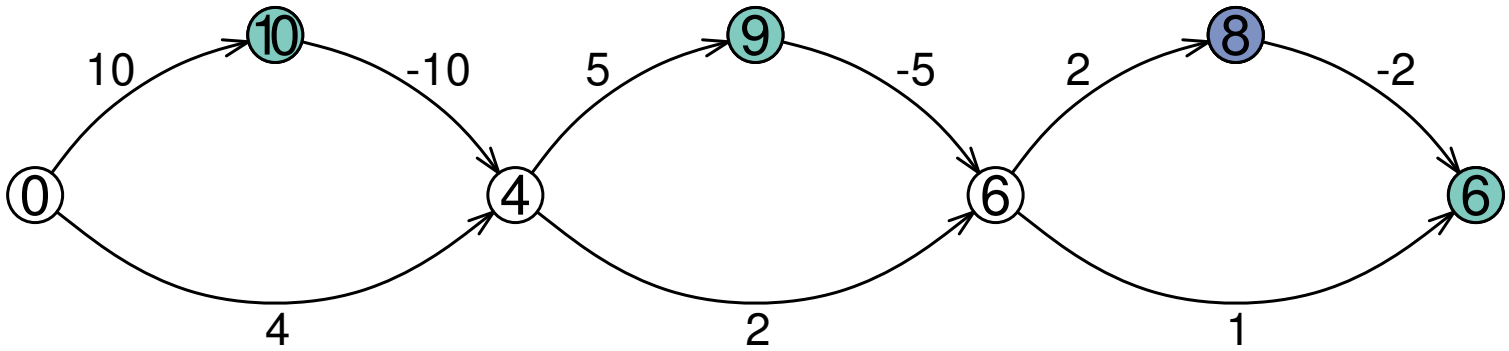


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

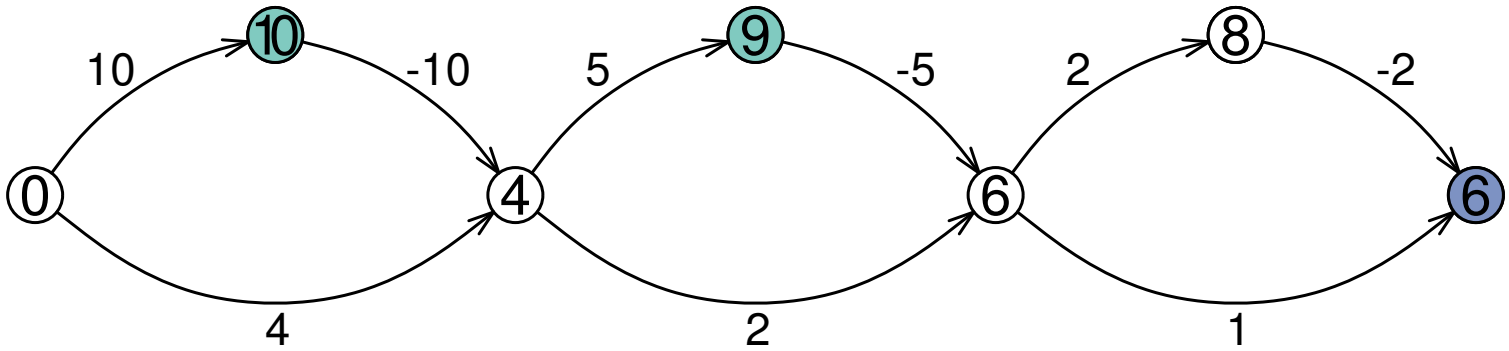


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

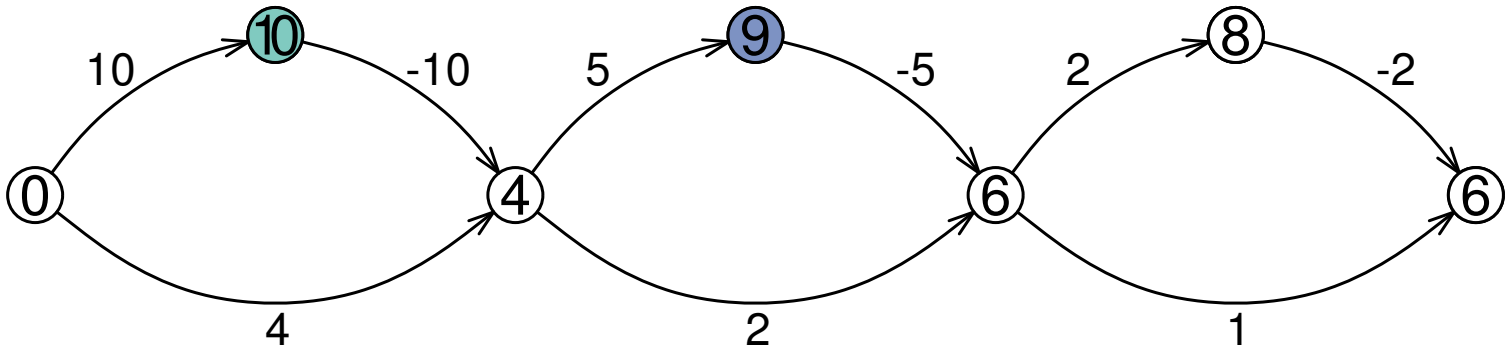


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

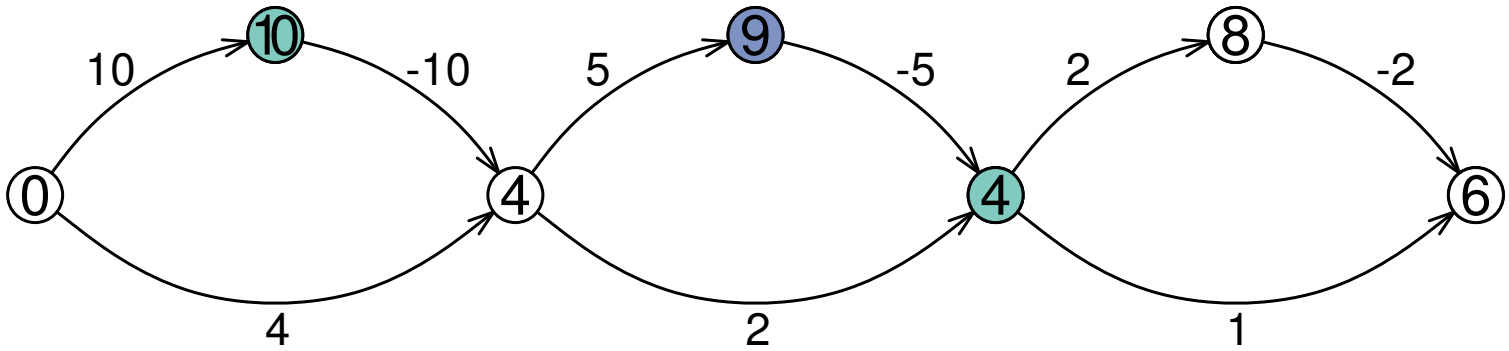


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

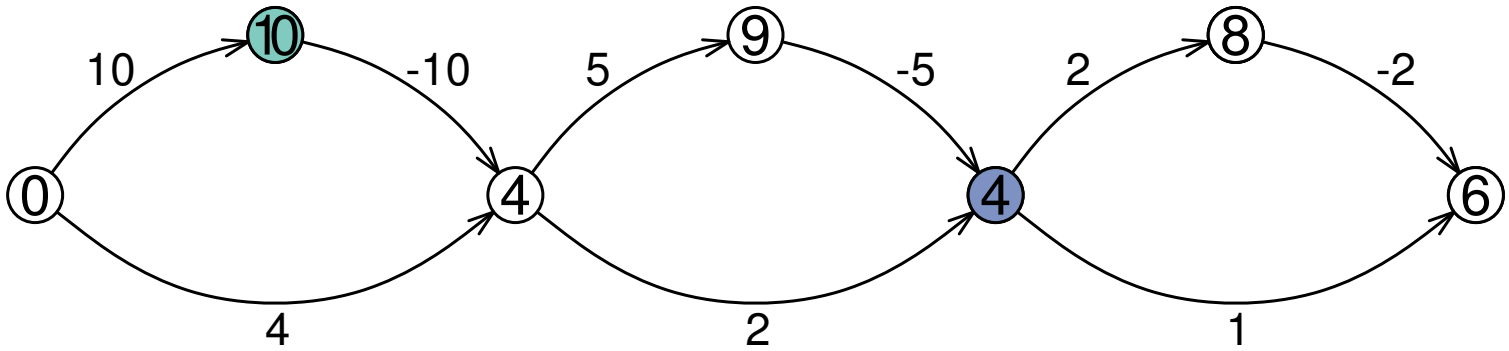


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

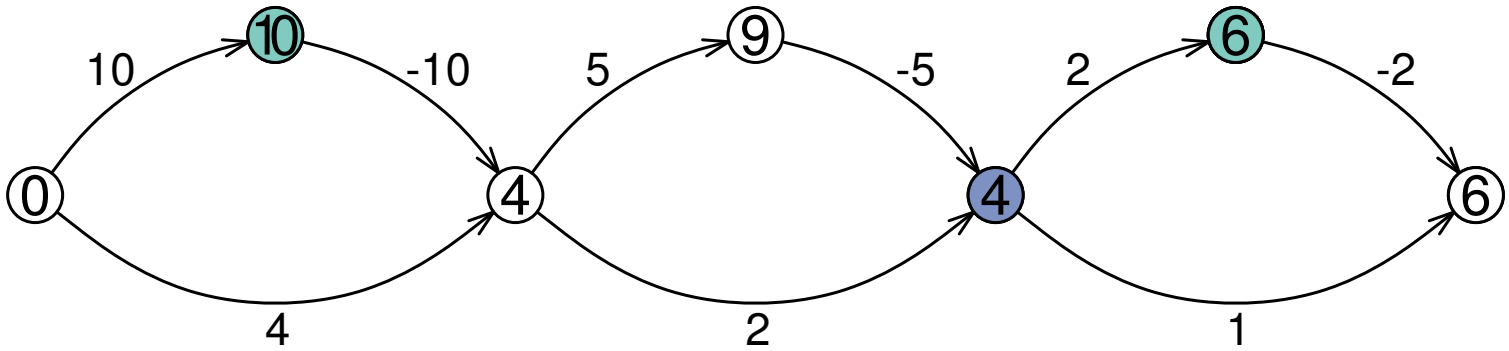


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

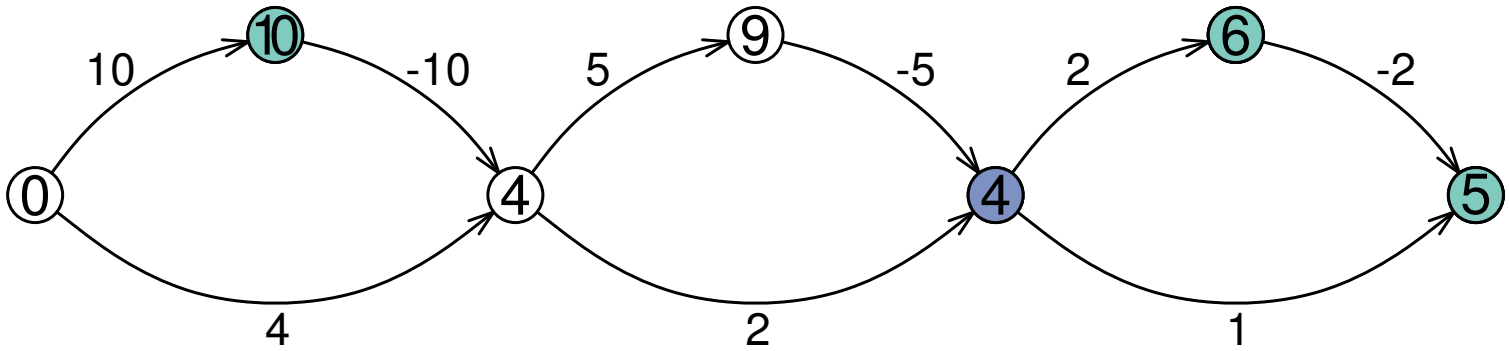


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

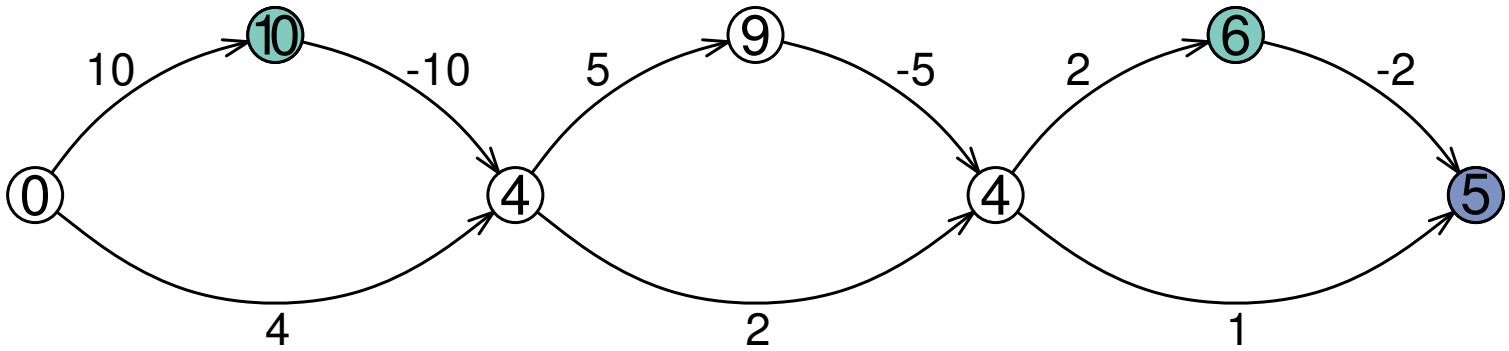


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

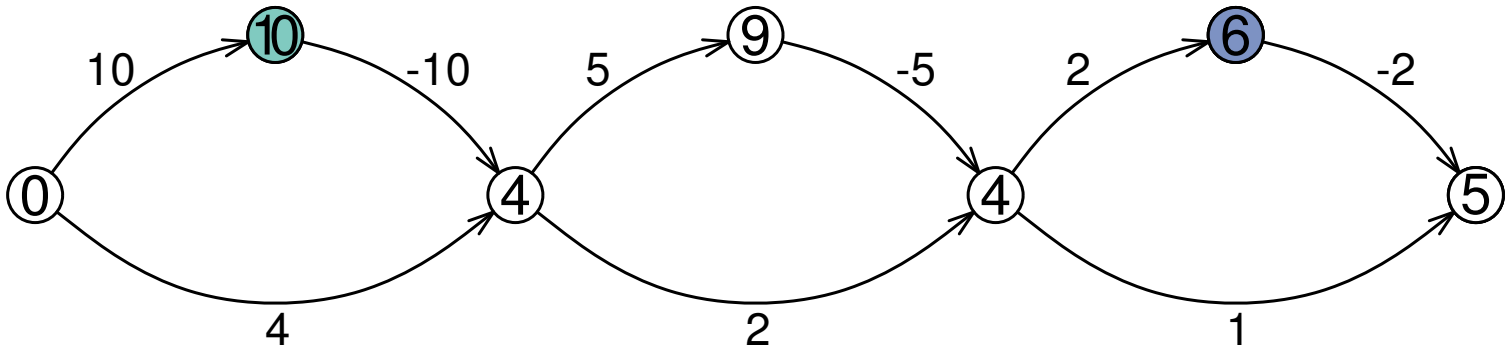


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

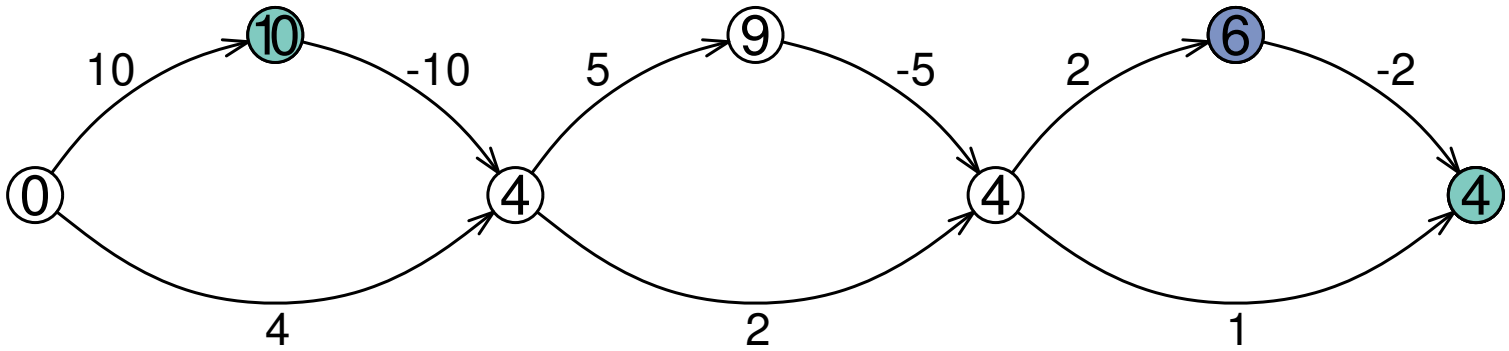


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

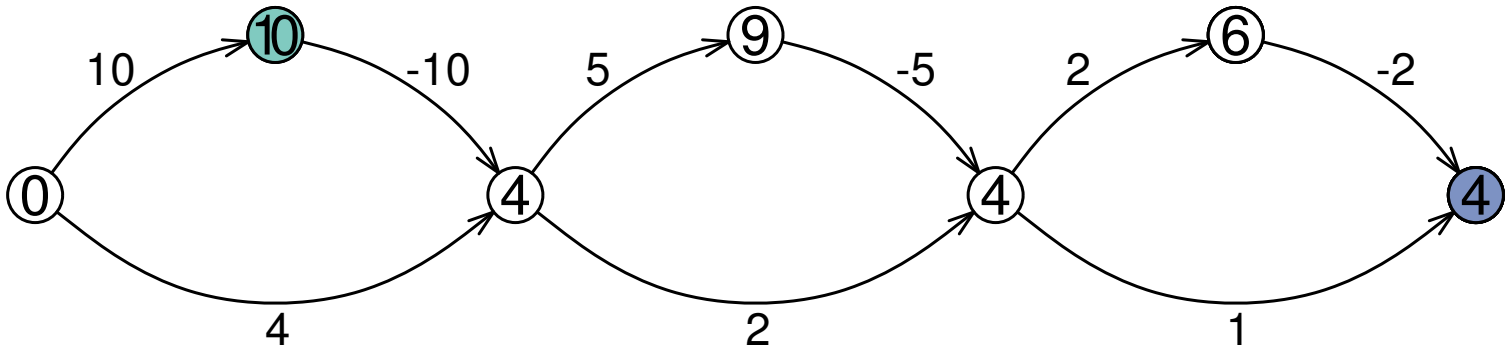


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

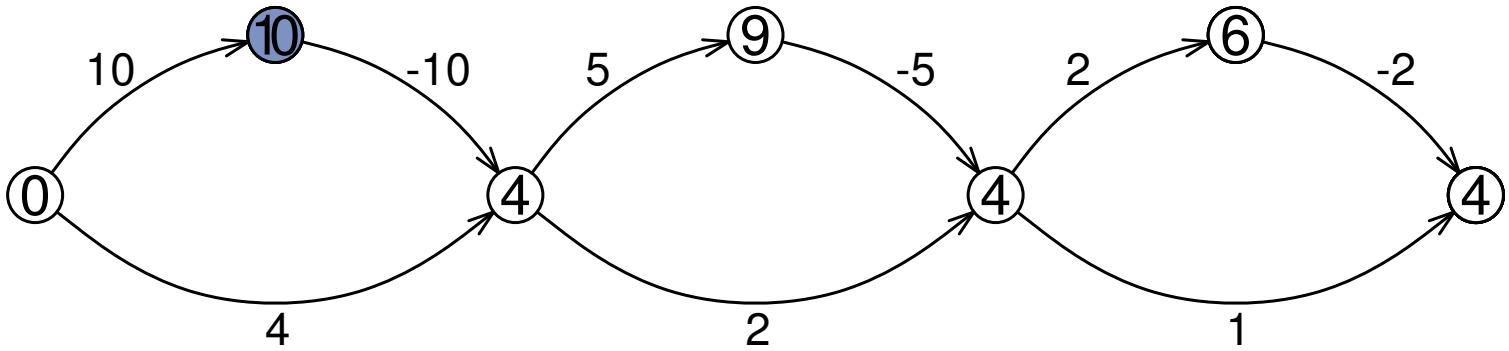


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

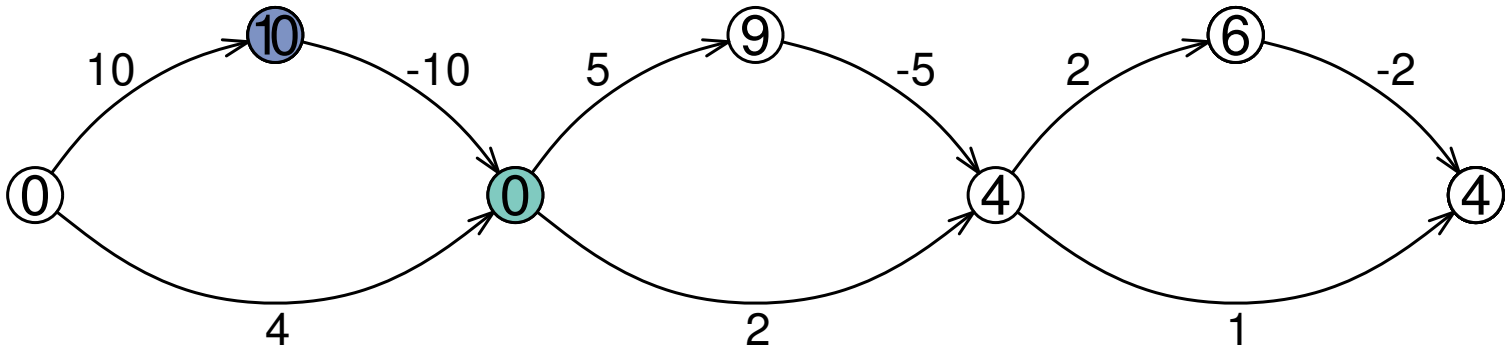


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

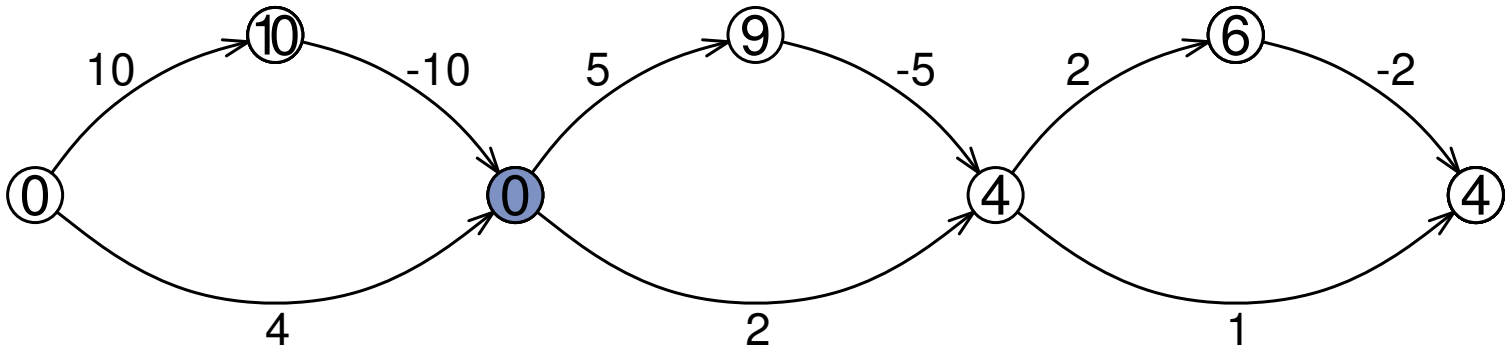


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

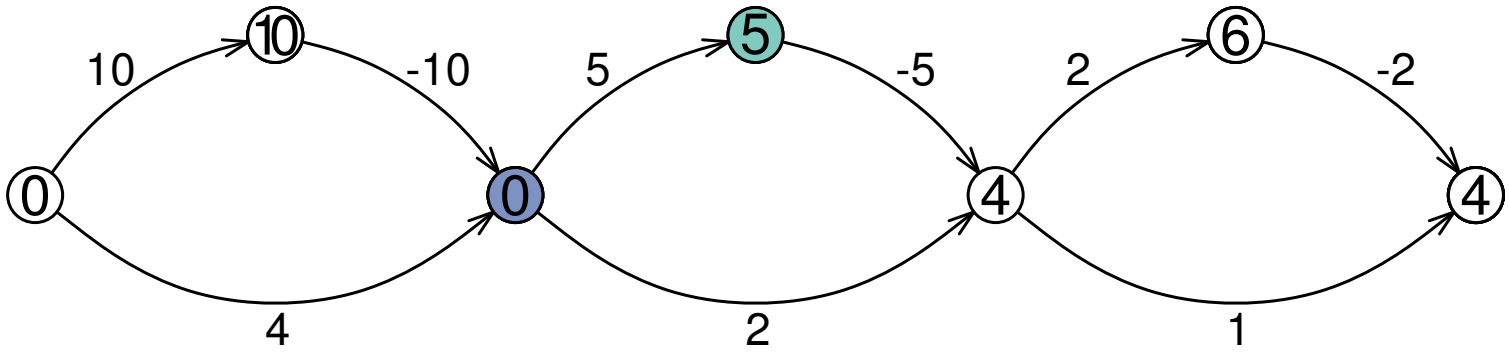


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

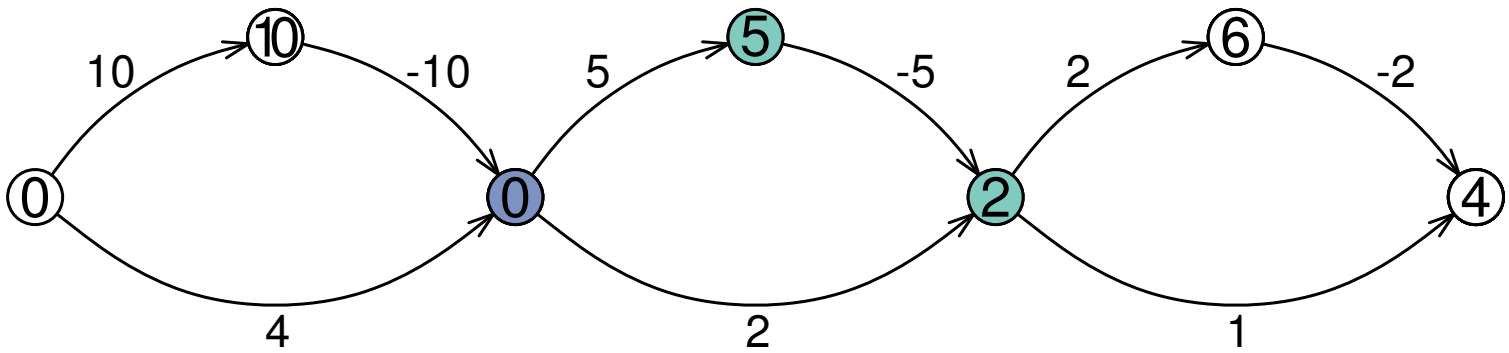


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

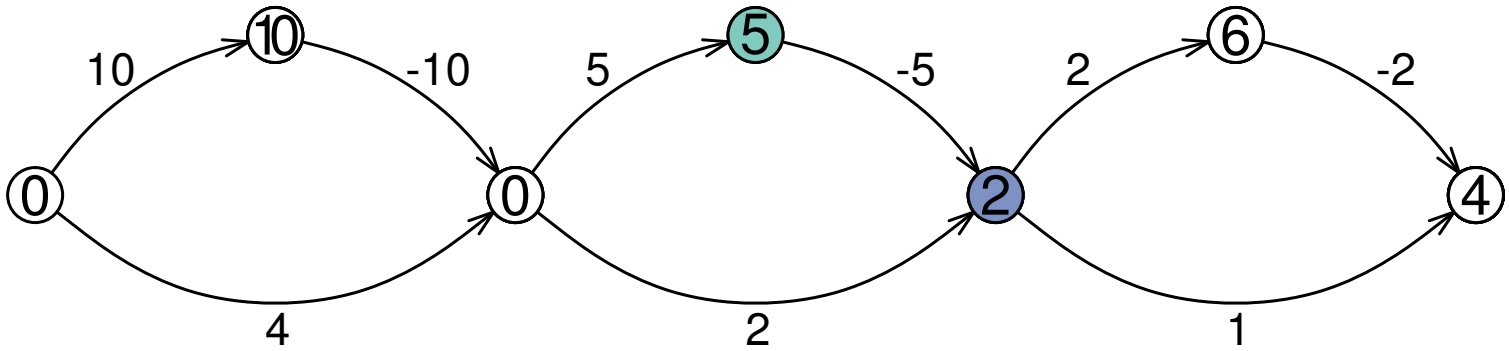


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

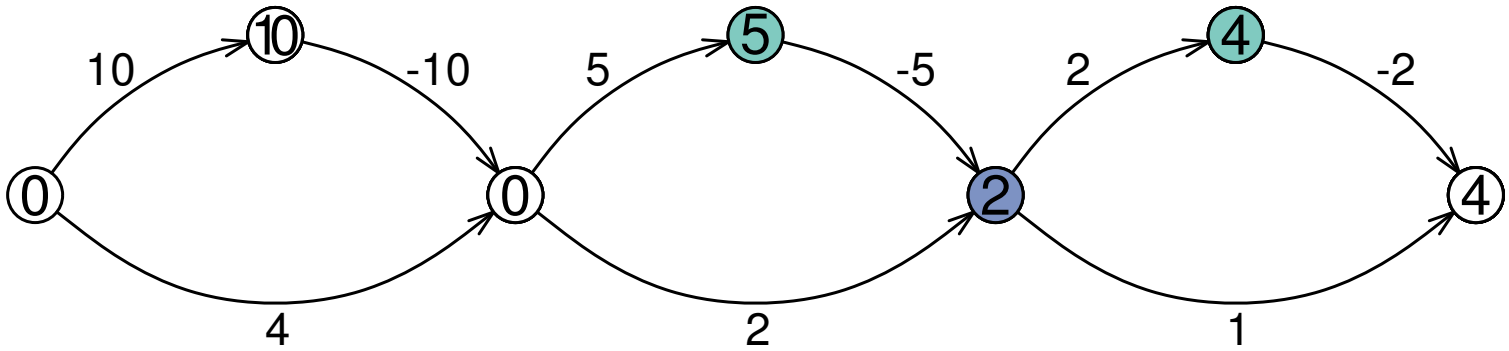


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

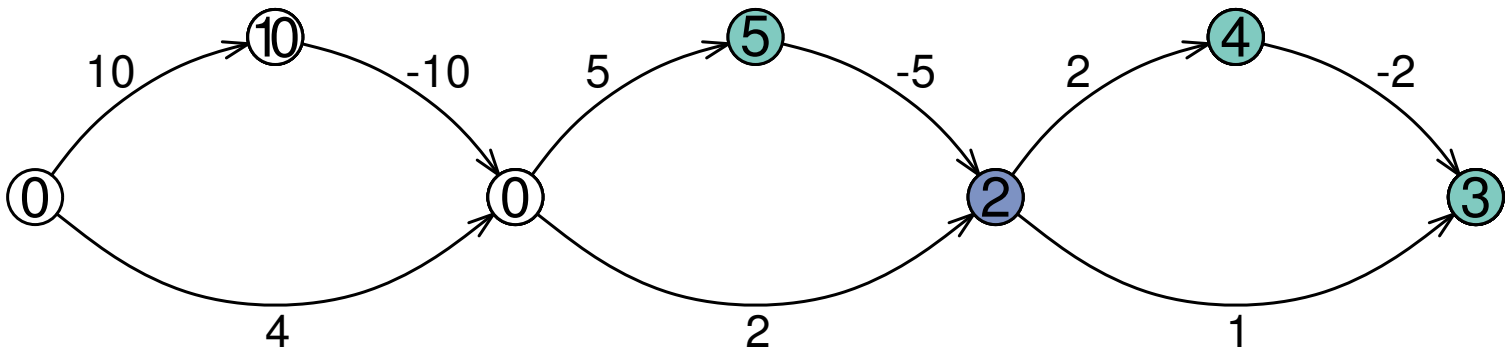


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

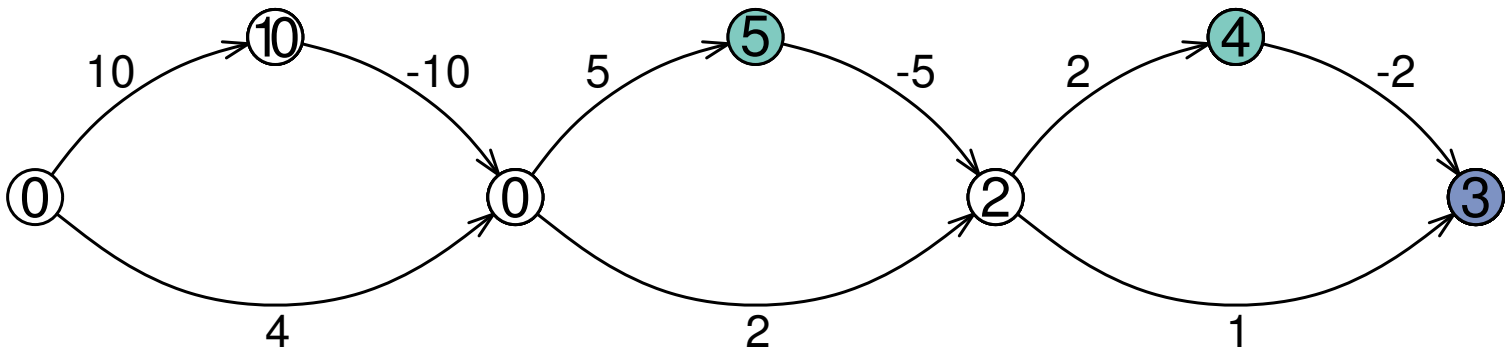


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

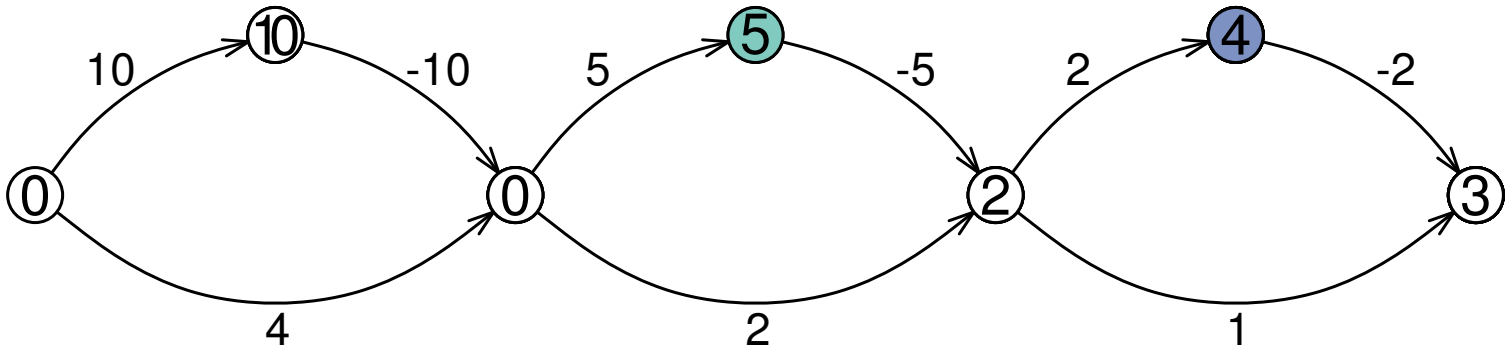


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

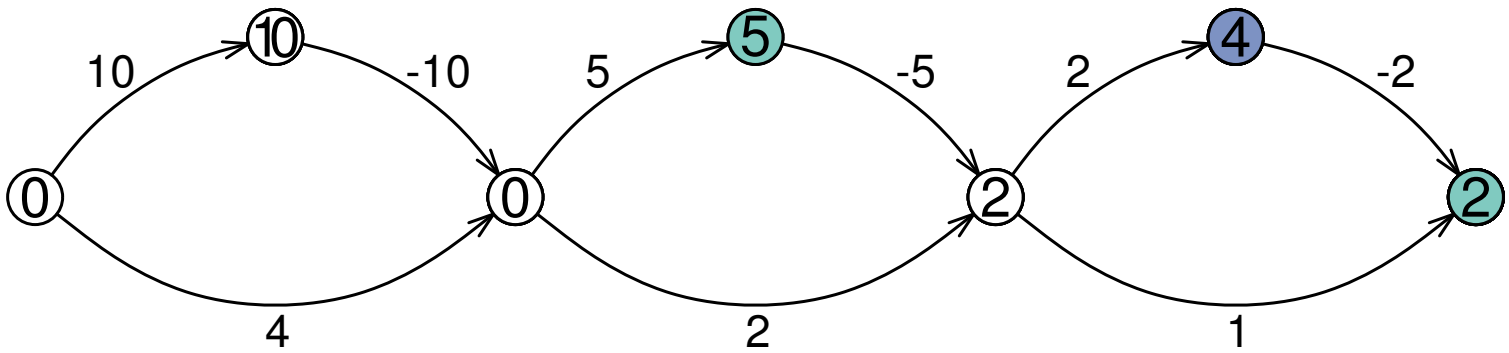


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

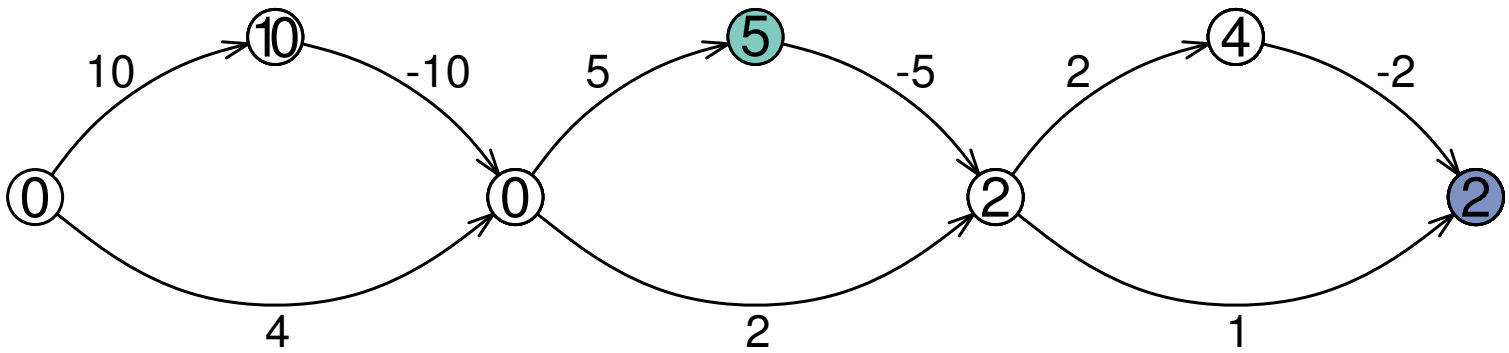


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

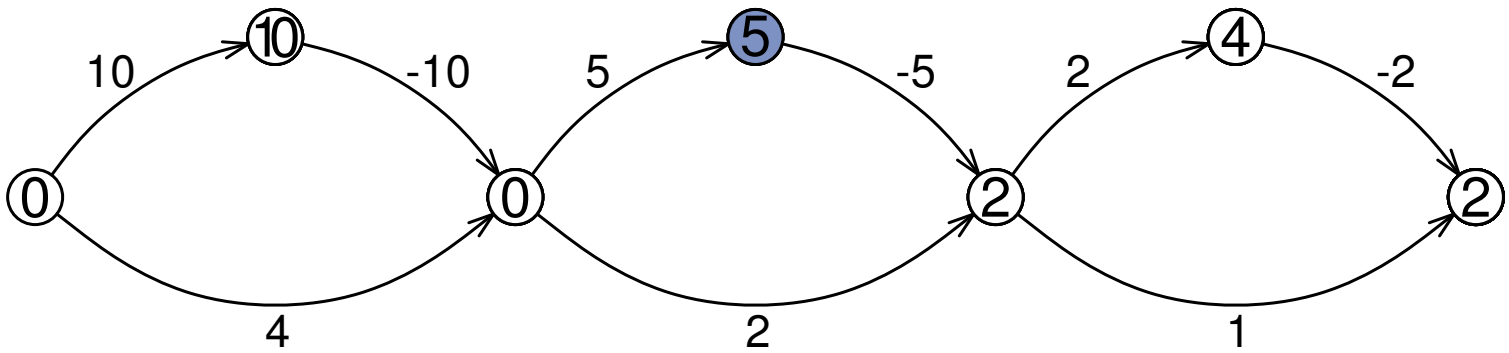


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

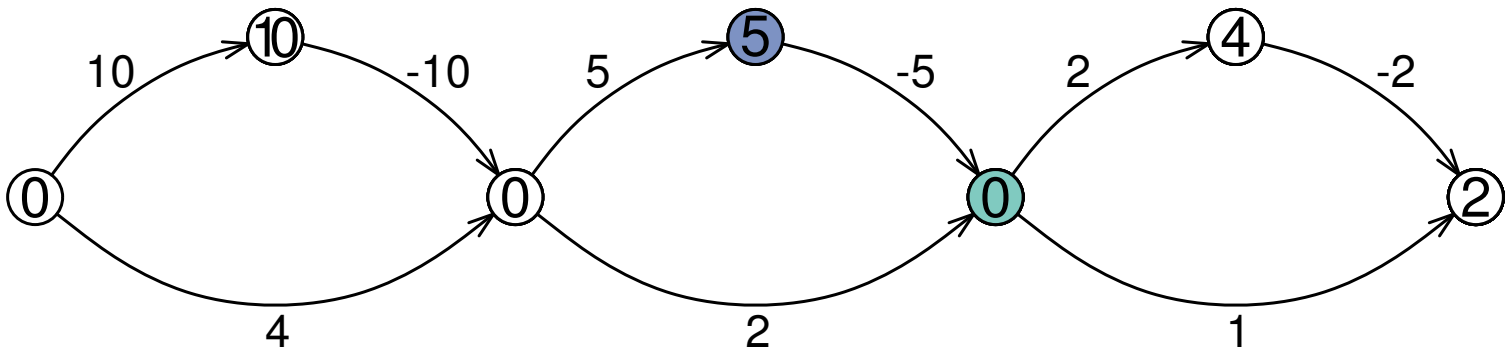


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

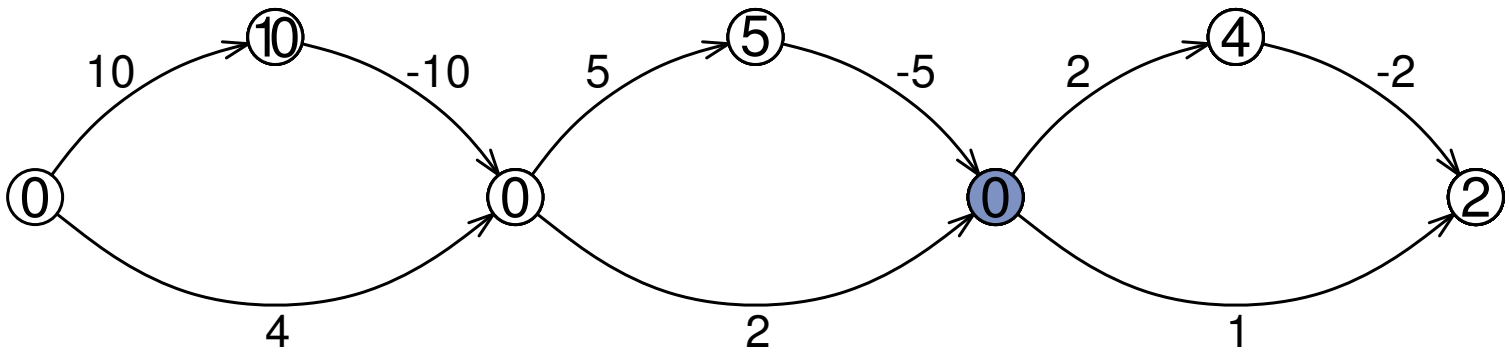


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

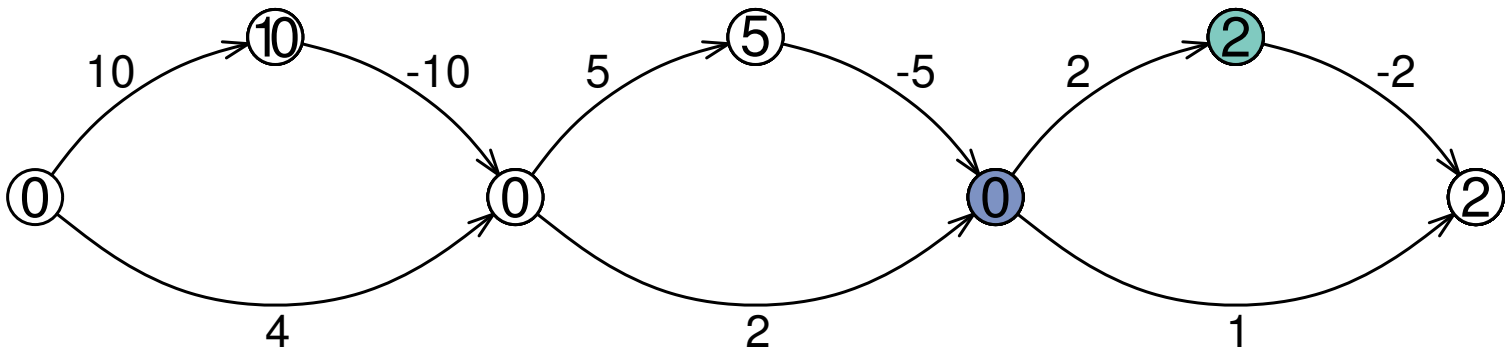


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

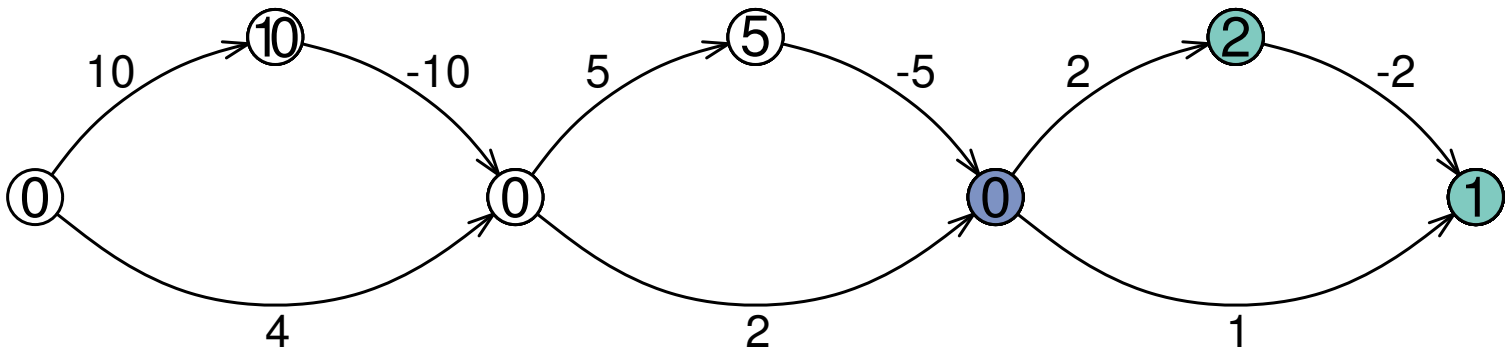


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

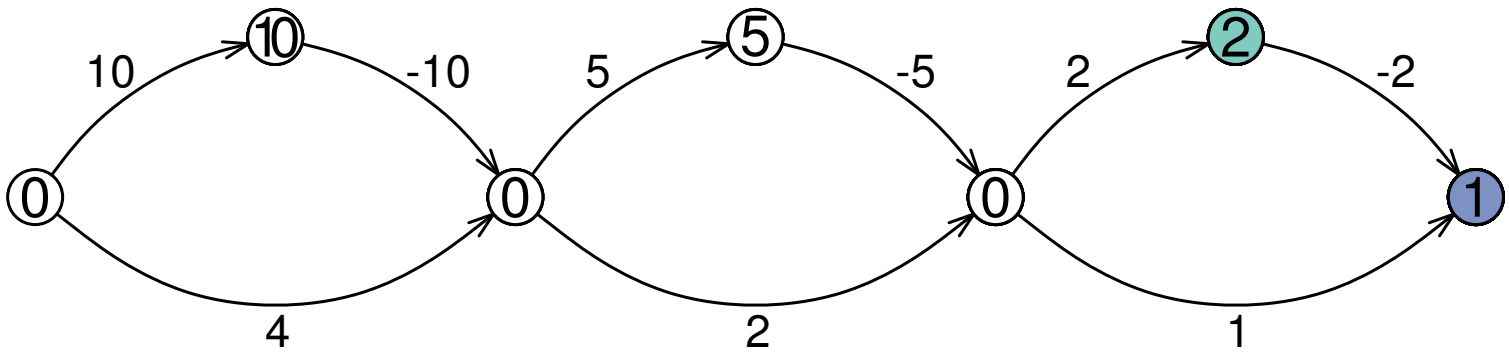


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

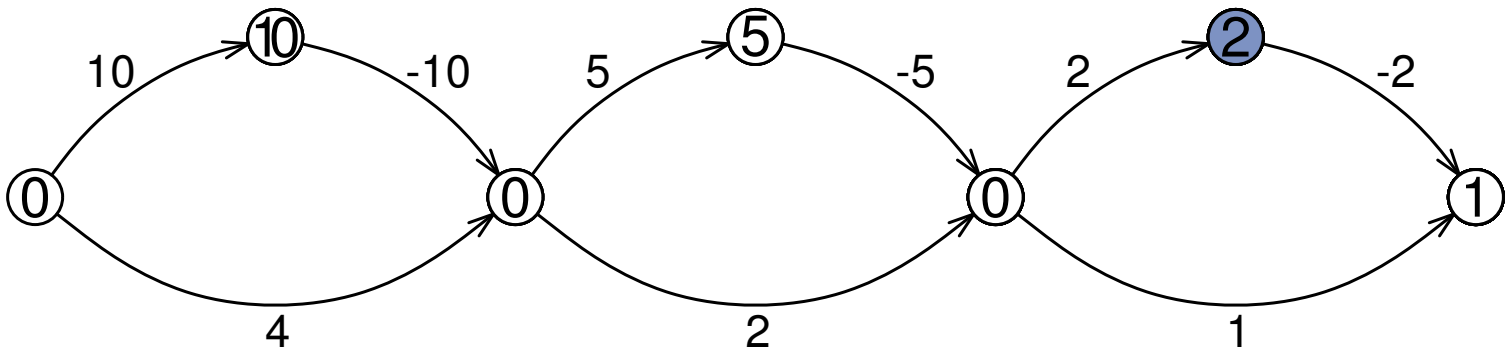


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

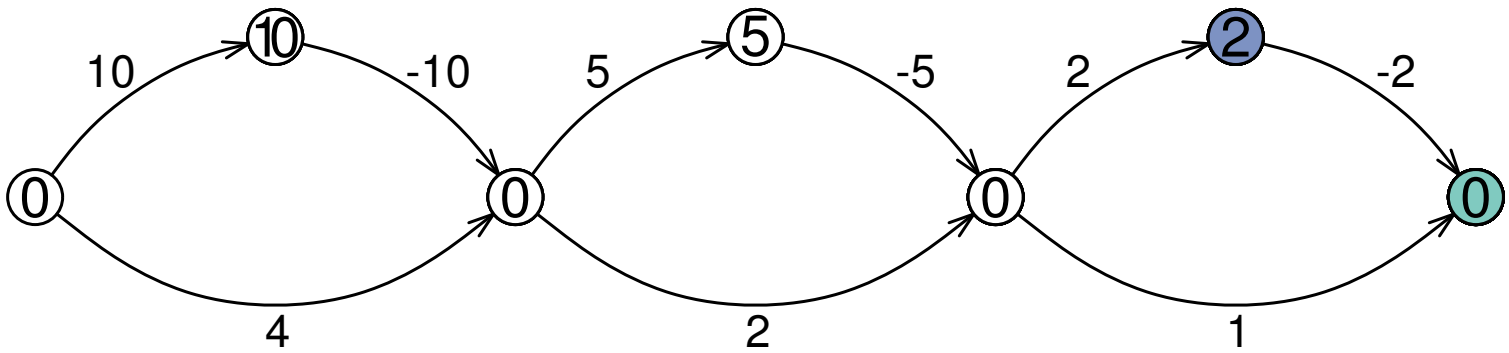


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

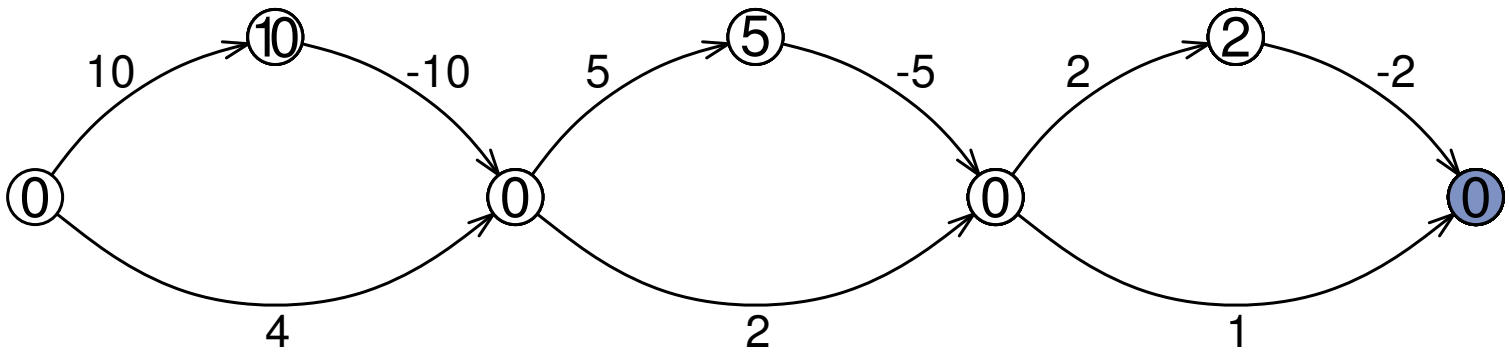


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

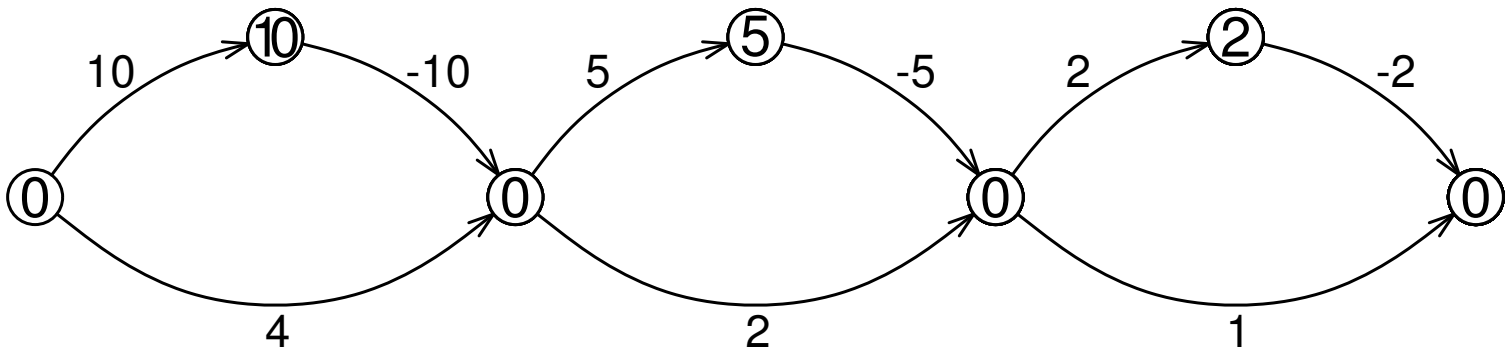


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

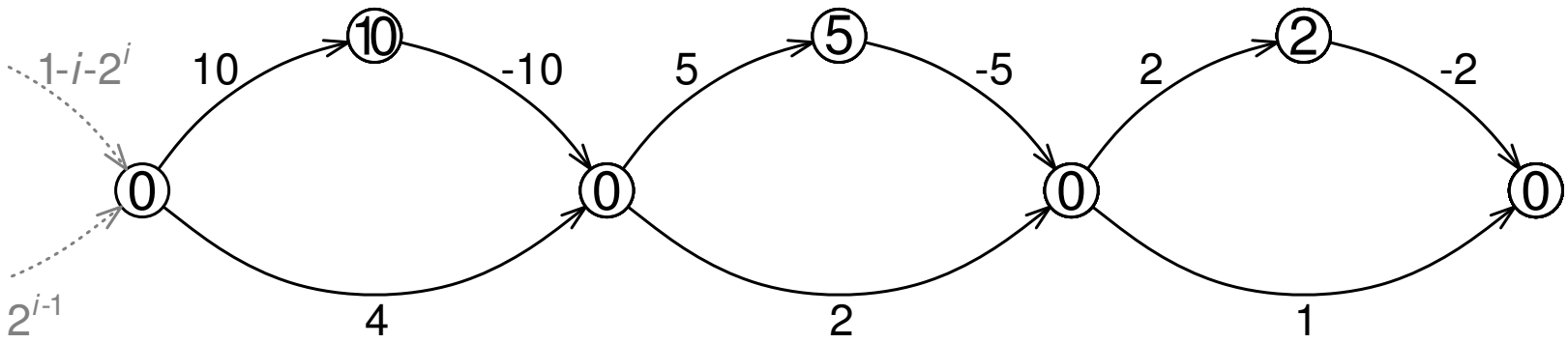


Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

Bellman Ford:

- Funktioniert auch mit negativen Kantengewichten

Energieoptimale Routen

Ziel: Gegeben Startknoten s und Zielknoten t,
berechne eine Route die den Energieverbrauch minimiert

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

Bellman Ford:

- Funktioniert auch mit negativen Kantengewichten

Auf Realwelt-Instanzen (wenige Kanten mit neg. Gewicht) ist Dijkstras Algorithmus schneller

Energieoptimale Routen

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr
- Nicht mehr Label-Setting

$\Rightarrow$ Stoppkriterium nicht mehr korrekt

Frage: Stoppkriterium wiederherstellbar?

Energieoptimale Routen

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr
- Nicht mehr Label-Setting

$\Rightarrow$ Stoppkriterium nicht mehr korrekt

Frage: Stoppkriterium wiederherstellbar?

- Wenn t abgearbeitet wird, könnten sich noch Pfade in der Queue befinden, die Label $d[t]$ durch Anhängen eines negativen Subpfades verbessern
- Ziel: berechne Schranke $P_{\min} \leq 0$ für Länge dieses Subpfades
- Dann Abbruch, sobald $\text{minKey}(Q) \geq d[t] + P_{\min}$

Berechnung von $P_{\min}$

Gegeben: Graph $G = (V, E)$ (ohne negative Zyklen)
Gesucht: (Global) kürzester Pfad in G

1. Ansatz:

- Füge (virtuelle) Knoten s', t' zu G hinzu
- Mit Kanten (s', u), (u, t') für alle $u \in V$ (mit Energieverbrauch 0)
- Berechne kürzesten s'-t'-Weg (mit Label-Correcting Dijkstra)

Berechnung von $P_{\min}$

Gegeben: Graph $G = (V, E)$ (ohne negative Zyklen)
Gesucht: (Global) kürzester Pfad in G

2. Ansatz (simuliert 1. Ansatz, ist in der Praxis schneller):

- Initialisiere Distanzlabel $\underline{d}[v] = 0$ für alle $v \in V$
- Für jeden Knoten $v \in V$:
  - Starte (Label-Correcting) Suche von v
  - Distanzlabel $\underline{d}[\cdot]$ wird zwischen den Suchen *nicht* reinitialisiert
- Berechne währenddessen: $P_{min} := min_{v \in V}\ \underline{d}[v]$

Berechnung von $P_{\min}$

Gegeben: Graph $G = (V, E)$ (ohne negative Zyklen)
Gesucht: (Global) kürzester Pfad in G

2. Ansatz (simuliert 1. Ansatz, ist in der Praxis schneller):

- Initialisiere Distanzlabel $\underline{d}[v] = 0$ für alle $v \in V$
- Für jeden Knoten $v \in V$:
  - Starte (Label-Correcting) Suche von v
  - Distanzlabel $\underline{d}[\cdot]$ wird zwischen den Suchen *nicht* reinitialisiert
- Berechne währenddessen: $P_{min} := min_{v \in V}\ \underline{d}[v]$

Danach Stoppkriterium wieder anwendbar

Berechnung von $P_{\min}$

Gegeben: Graph $G = (V, E)$ (ohne negative Zyklen)
Gesucht: (Global) kürzester Pfad in G

2. Ansatz (simuliert 1. Ansatz, ist in der Praxis schneller):

- Initialisiere Distanzlabel $\underline{d}[v] = 0$ für alle $v \in V$
- Für jeden Knoten $v \in V$:
  - Starte (Label-Correcting) Suche von v
  - Distanzlabel $\underline{d}[\cdot]$ wird zwischen den Suchen *nicht* reinitialisiert
- Berechne währenddessen: $P_{min} := min_{v \in V}\ \underline{d}[v]$

Danach Stoppkriterium wieder anwendbar

Beobachtung: Nach Berechnung von P_{min} gilt: $\underline{d}[t] \leq dist(v, t),\ \forall v \in V$

$\Rightarrow$ Stoppkriterium lässt sich verbessern zu: $\text{minKey}(Q) \leq d[t] + \underline{d}[t]$

Energieoptimale Routen

Dijkstras Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Stoppkriterium lässt sich wiederherstellen
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr
- Nicht mehr Label-Setting

Frage: Label-Setting Eigenschaft wiederherstellbar?

Energieoptimale Routen

Dijkstras Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
- Stoppkriterium lässt sich wiederherstellen
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr
- Nicht mehr Label-Setting

Frage: Label-Setting Eigenschaft wiederherstellbar?

Idee: Finde zulässiges Potential $\pi\colon V \to \mathbb{R}$, sodass $\text{len}(u,v) - \pi(u) + \pi(v) \geq 0$ für jede Kante $(u,v) \in E$

Knotenpotentiale

Idee: Finde zulässiges Potential $\pi\colon V \to \mathbb{R}$, sodass
$\text{len}(u, v) - \pi(u) + \pi(v) \geq 0$ für jede Kante $(u, v) \in E$

Dann Dijkstras Algorihmus (mit Stoppkriterium) auf Graph mit reduzierten Gewichten anwendbar

1. Distanzbasiertes Potential:

- Setze $\pi(v) = d(v, v^*)$, für beliebigen Knoten v^*
- Berechnung mittels (Label-Correcting) Query von v^*
- Zulässigkeit folgt aus Dreiecksungleichung

Knotenpotentiale

Idee: Finde zulässiges Potential $\pi\colon V \to \mathbb{R}$, sodass
$\text{len}(u,v) - \pi(u) + \pi(v) \geq 0$ für jede Kante $(u,v) \in E$

Dann Dijkstras Algorihmus (mit Stoppkriterium) auf Graph mit reduzierten Gewichten anwendbar

2. P_{min}-basiertes Potential:

- Setze $\pi(v) = -\underline{d}[v]$ für alle v
- Berechnung wie vorher beschrieben
- Zulässigkeit folgt wieder aus Dreiecksungleichung:
  $\underline{d}[u] + \text{len}(u,v) \geq \underline{d}[v]$

Knotenpotentiale

Idee: Finde zulässiges Potential $\pi\colon V \to \mathbb{R}$, sodass
$\text{len}(u, v) - \pi(u) + \pi(v) \geq 0$ für jede Kante $(u, v) \in E$

Dann Dijkstras Algorihmus (mit Stoppkriterium) auf Graph mit reduzierten Gewichten anwendbar

3. Höheninduziertes Potential:

- Ann.: Höhenkoordinate $h(v)$ eines jeden Knoten gegeben
- $\pi(v) := \alpha \cdot h(v)$ für alle v, so dass $c(u, v) + \alpha(h(v) - h(u)) \geq 0$
- Kann mit Sweep über alle Kanten berechnet werden
- Keine Garantie für Zulässigkeit des Potentials
  (klappt für realistische Kantengewichte)

Energieoptimale Routen

Bisher:

- Route minimiert den absoluten Energieverbrauch
- Route ist für konkreten Ladezustand an s ggf. nicht realisierbar

Jetzt:

- Berechne Route abhängig vom initialen Ladezustand (SoC)
- Maximiere resultierenden SoC an t
- Betrachte nur Pfade die *zulässig* sind:
  - Energie ist ausreichend: SoC wird nie negativ
  - Akku wird nie überschritten: $\text{SoC} > M \rightarrow \text{SoC} = M$
    ($M :=$ Akku-Kapazität)

Battery Constraints

Akku hat begrenzte Kapazität

- Akku darf nicht leer laufen (andernfalls Kante nicht benutzbar)
- Kapazität kann nicht überschritten werden (Energie verfällt ggf.)
- Muss für jeden Knoten eines Pfades gelten

$\Rightarrow$ Ladestand (state of charge, SoC) in der Suche berücksichtigen

Fahrzeug hat aktuellen SoC b

$\Rightarrow$ Befahren von Kante e mit Gewicht $\text{len}(e)$ ergibt SoC $b - \text{len}(e)$

Ausnahmen:

- Falls $b - \text{len}(e)$ unterhalb Grenzwert $\Rightarrow$ Kante nicht befahrbar
- Falls $b - \text{len}(e)$ oberhalb Kapazität $\Rightarrow$ SoC ist 100%

Battery Constraints

Konstanter Energieverbrauch pro Kante, aber:

- Kante kann nicht benutzt werden falls Akkustand $<$ Kantengewicht
- Akkustand kann das Maximum M nicht überschreiten

$\Rightarrow$ Explizites überprüfen der Battery Constraints im Algorithmus

Battery Constraints

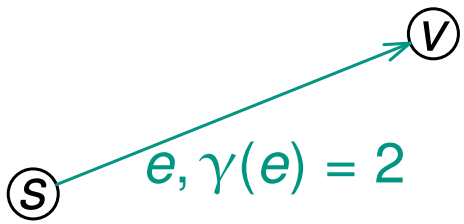
Konstanter Energieverbrauch pro Kante, aber:

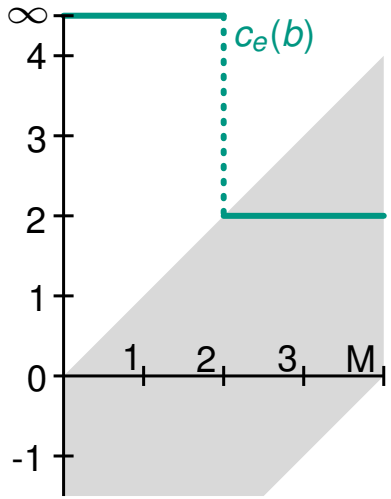
- Kante kann nicht benutzt werden falls Akkustand $<$ Kantengewicht
- Akkustand kann das Maximum M nicht überschreiten

$\Rightarrow$ Explizites überprüfen der Battery Constraints im Algorithmus

Alternative 1:

- Modelliere Gewicht einer Kante als Verbrauchsfunktion
- Verbrauchsfunktion bildet SoC auf tatsächlichen Verbrauch ab





Battery Constraints

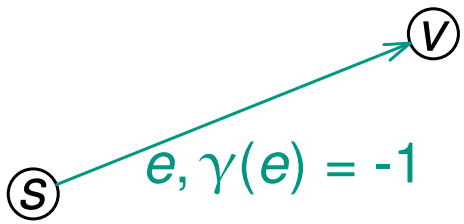
Konstanter Energieverbrauch pro Kante, aber:

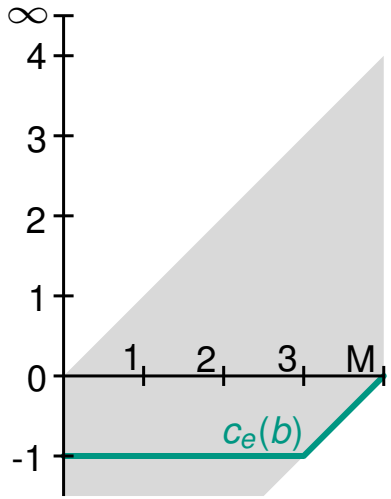
- Kante kann nicht benutzt werden falls Akkustand $<$ Kantengewicht
- Akkustand kann das Maximum M nicht überschreiten

$\Rightarrow$ Explizites überprüfen der Battery Constraints im Algorithmus

Alternative 1:

- Modelliere Gewicht einer Kante als Verbrauchsfunktion
- Verbrauchsfunktion bildet SoC auf tatsächlichen Verbrauch ab





Battery Constraints

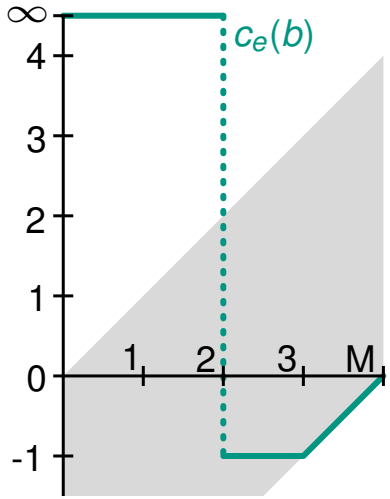
Konstanter Energieverbrauch pro Kante, aber:

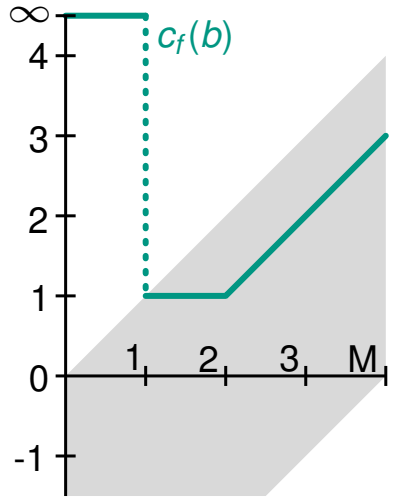
- Kante kann nicht benutzt werden falls Akkustand $<$ Kantengewicht
- Akkustand kann das Maximum M nicht überschreiten

$\Rightarrow$ Explizites überprüfen der Battery Constraints im Algorithmus

Alternative 1:

- Modelliere Gewicht einer Kante als Verbrauchsfunktion
- Verbrauchsfunktion bildet SoC auf tatsächlichen Verbrauch ab





Battery Constraints

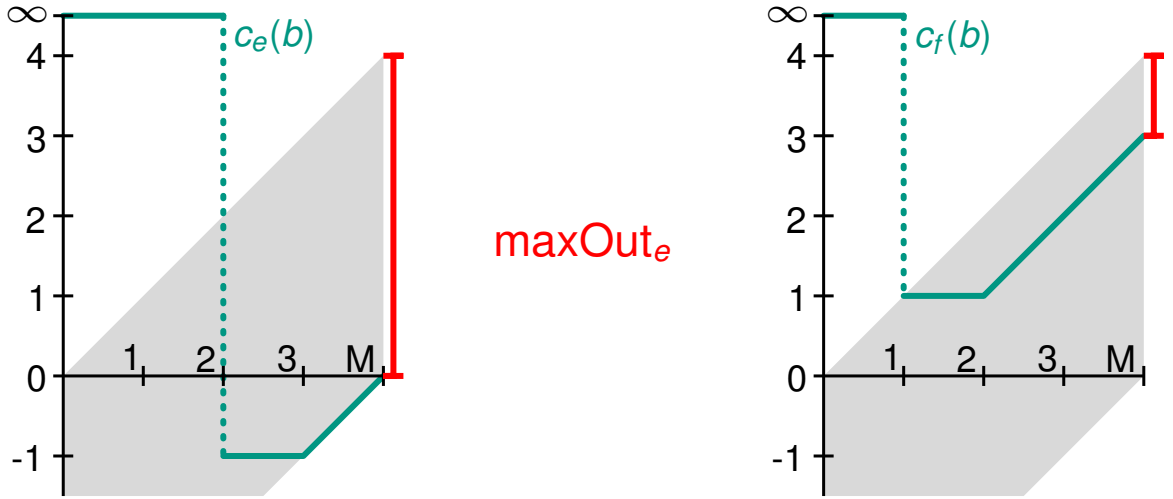
Konstanter Energieverbrauch pro Kante, aber:

- Kante kann nicht benutzt werden falls Akkustand $<$ Kantengewicht
- Akkustand kann das Maximum M nicht überschreiten

$\Rightarrow$ Explizites überprüfen der Battery Constraints im Algorithmus

Alternative 1:

- Modelliere Gewicht einer Kante als Verbrauchsfunktion
- Verbrauchsfunktion bildet SoC auf tatsächlichen Verbrauch ab



Battery Constraints

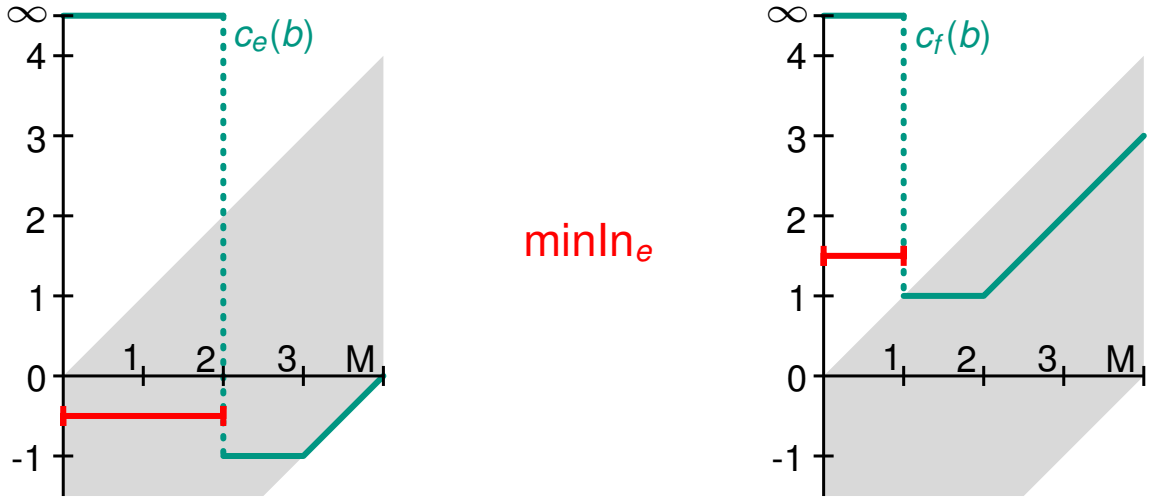
Konstanter Energieverbrauch pro Kante, aber:

- Kante kann nicht benutzt werden falls Akkustand $<$ Kantengewicht
- Akkustand kann das Maximum M nicht überschreiten

$\Rightarrow$ Explizites überprüfen der Battery Constraints im Algorithmus

Alternative 1:

- Modelliere Gewicht einer Kante als Verbrauchsfunktion
- Verbrauchsfunktion bildet SoC auf tatsächlichen Verbrauch ab



Battery Constraints

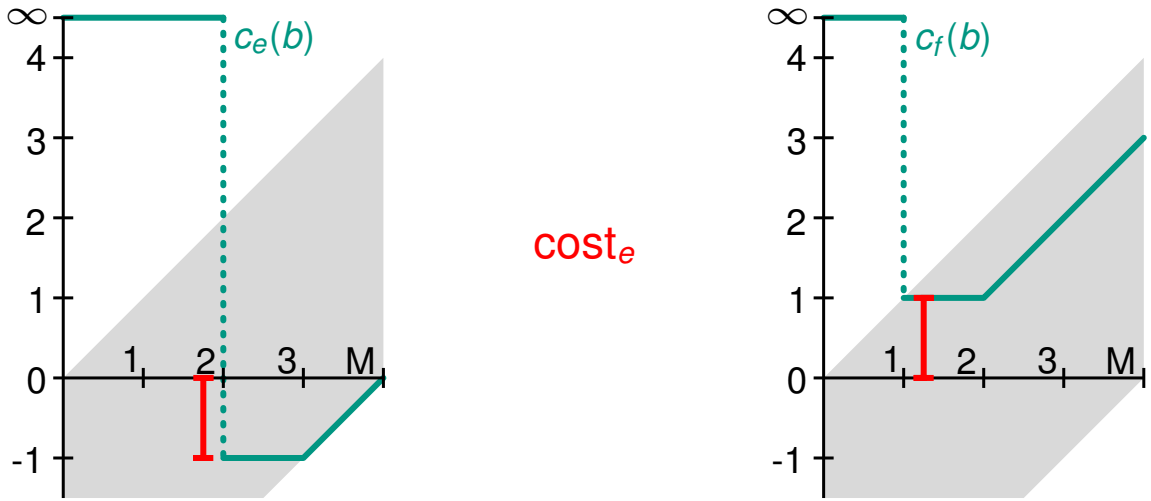
Konstanter Energieverbrauch pro Kante, aber:

- Kante kann nicht benutzt werden falls Akkustand $<$ Kantengewicht
- Akkustand kann das Maximum M nicht überschreiten

$\Rightarrow$ Explizites überprüfen der Battery Constraints im Algorithmus

Alternative 1:

- Modelliere Gewicht einer Kante als Verbrauchsfunktion
- Verbrauchsfunktion bildet SoC auf tatsächlichen Verbrauch ab



Battery Constraints

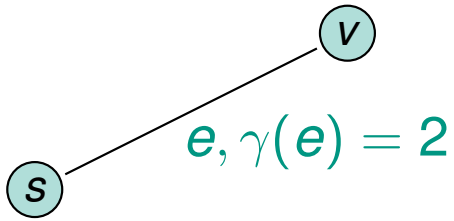
Konstanter Energieverbrauch pro Kante, aber:

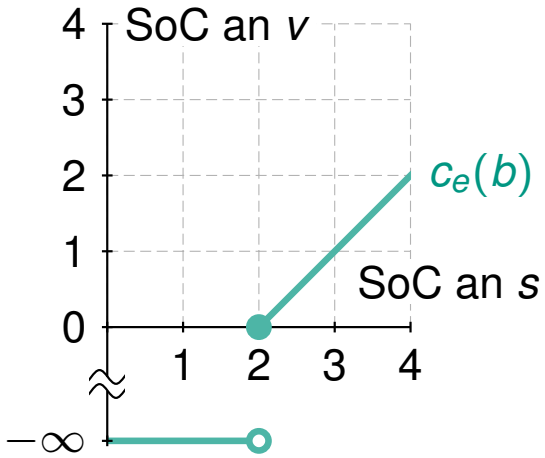
- Kante kann nicht benutzt werden falls Akkustand $<$ Kantengewicht
- Akkustand kann das Maximum M nicht überschreiten

$\Rightarrow$ Explizites überprüfen der Battery Constraints im Algorithmus

Alternative 2:

- Modelliere Gewicht einer Kante als SoC-Funktion
- SoC-Funktion bildet SoC vor der Kante auf SoC nach der Kante ab





Battery Constraints

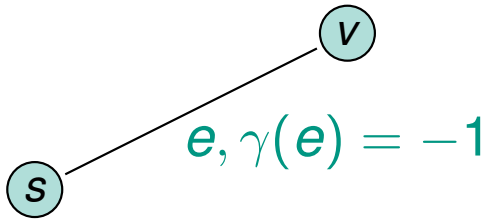
Konstanter Energieverbrauch pro Kante, aber:

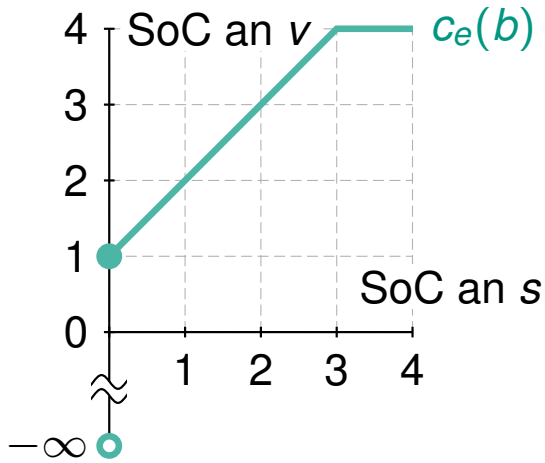
- Kante kann nicht benutzt werden falls Akkustand $<$ Kantengewicht
- Akkustand kann das Maximum M nicht überschreiten

$\Rightarrow$ Explizites überprüfen der Battery Constraints im Algorithmus

Alternative 2:

- Modelliere Gewicht einer Kante als SoC-Funktion
- SoC-Funktion bildet SoC vor der Kante auf SoC nach der Kante ab





Energieoptimale Routen

Anfragetypen (analog zu Zeitabhängigkeit):

SoC Query: Gegeben Start s, Ziel t und initialen Ladezustand b_s finde zulässige s-t-Pfad mit maximalem SoC an t

Berechnung mit angepasstem Dijkstra

Energieoptimale Routen

Anfragetypen (analog zu Zeitabhängigkeit):

SoC Query: Gegeben Start s, Ziel t und initialen Ladezustand b_s finde zulässige s-t-Pfad mit maximalem SoC an t

Berechnung mit angepasstem Dijkstra

Profilsuche: Für Start s und Ziel t, finde zulässige s-t-Pfade mit maximalem SoC an t für *alle* $b_s \in [0, M]$

Berechnung mit Hilfe von:

- SoC-Funktionen
- Geeigneten Link/Merge-Operationen

SoC-Profile

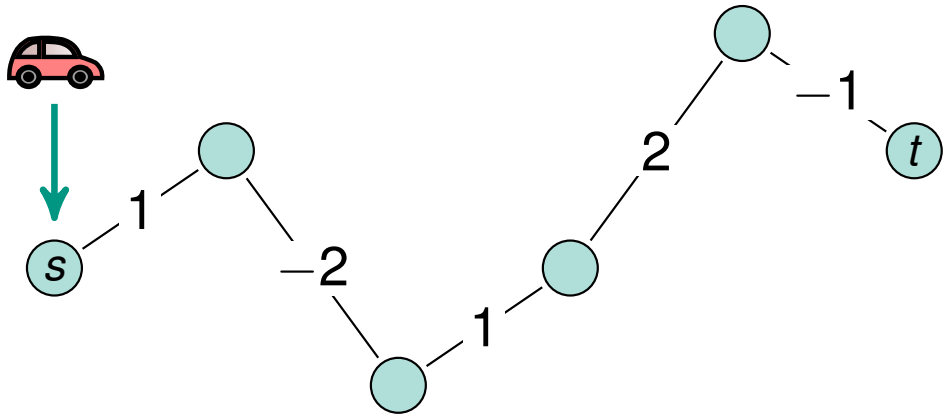
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

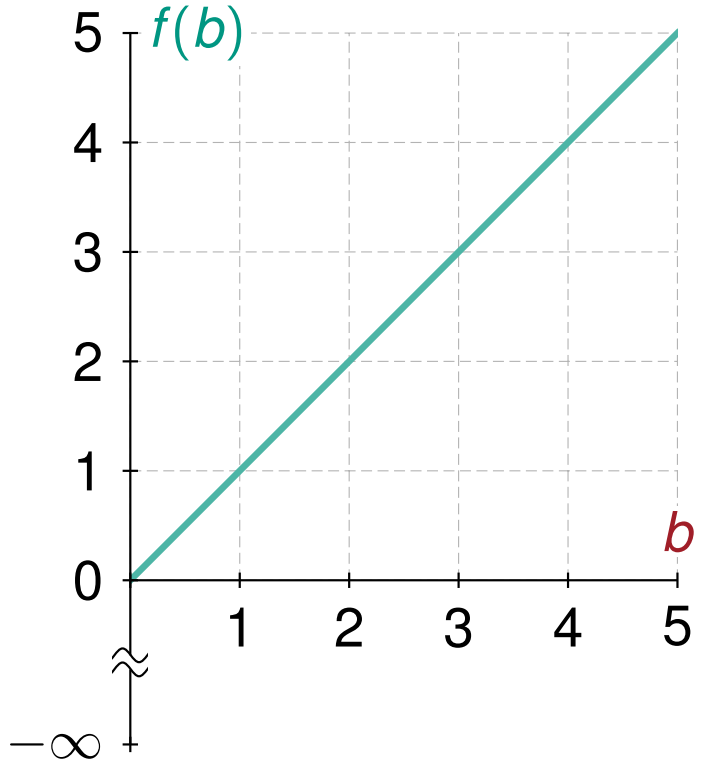
SoC-Profil

Optimaler SoC am Ziel hängt vom SoC am Startknoten ab

SoC-Profil f bildet SoC b am Start auf SoC $f(b)$ am Ziel ab

Beispiel: $M = 5$



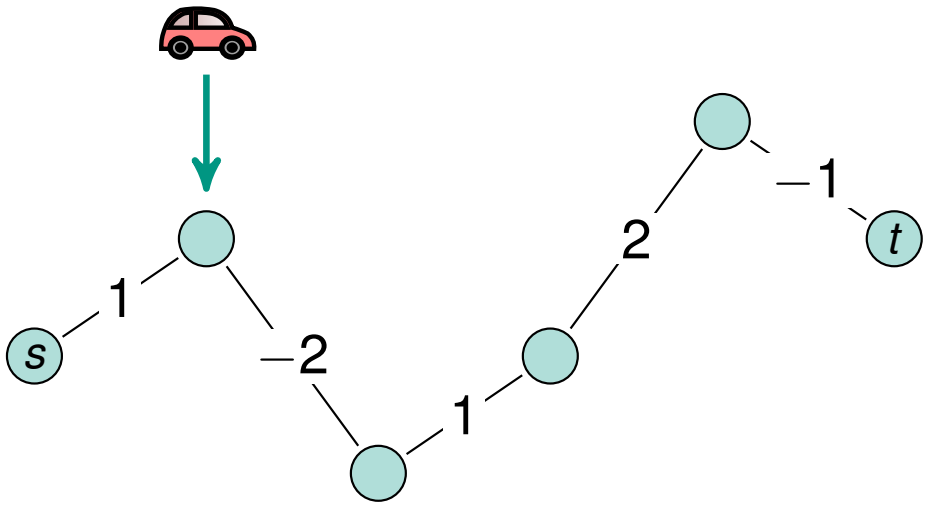


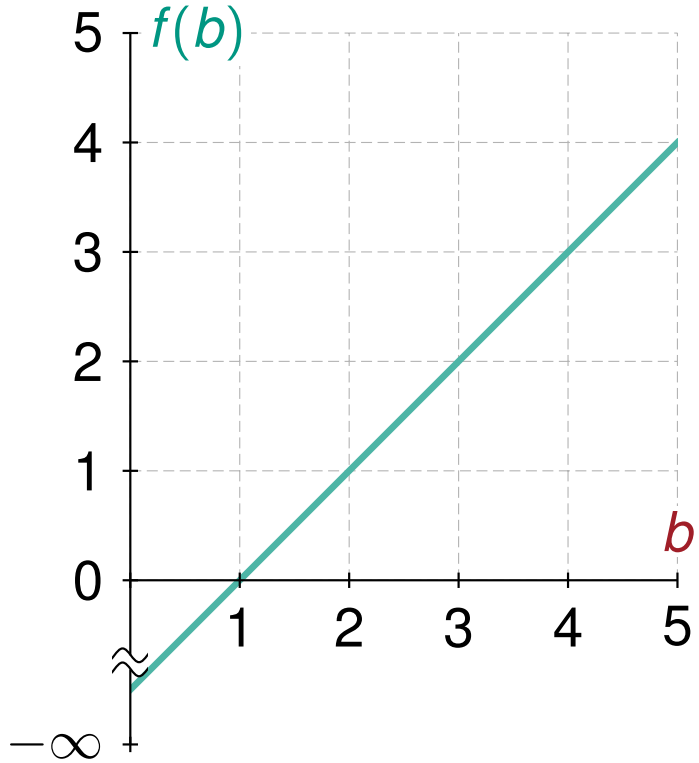
SoC-Profil

Optimaler SoC am Ziel hängt vom SoC am Startknoten ab

SoC-Profil f bildet SoC b am Start auf SoC $f(b)$ am Ziel ab

Beispiel: $M = 5$



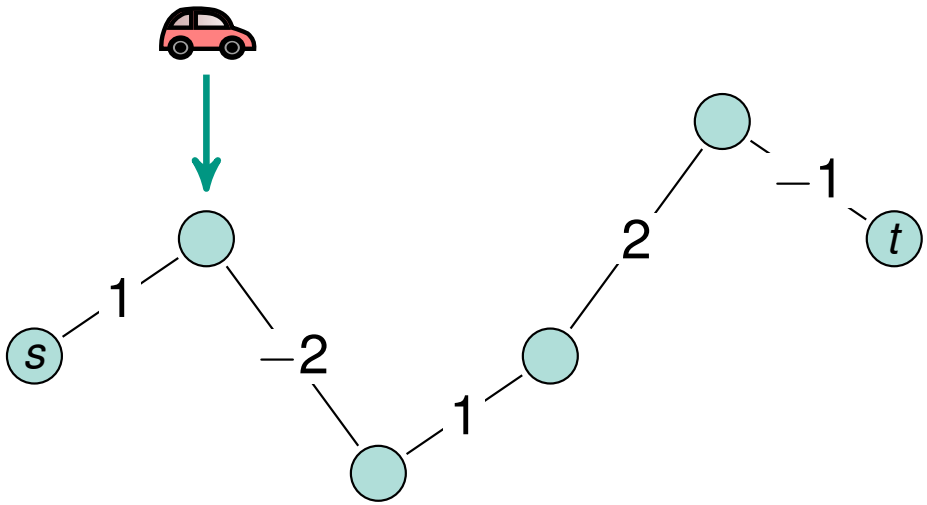


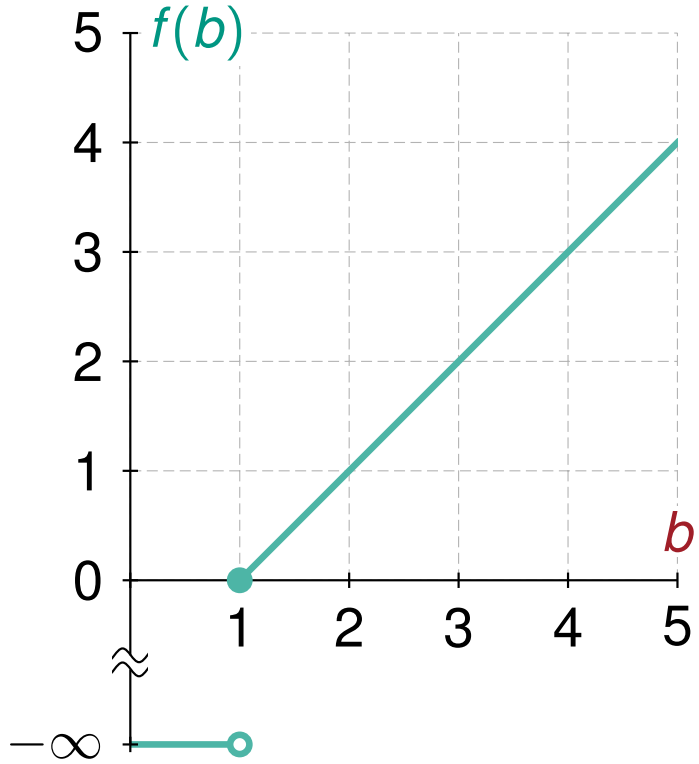
SoC-Profil

Optimaler SoC am Ziel hängt vom SoC am Startknoten ab

SoC-Profil f bildet SoC b am Start auf SoC $f(b)$ am Ziel ab

Beispiel: $M = 5$



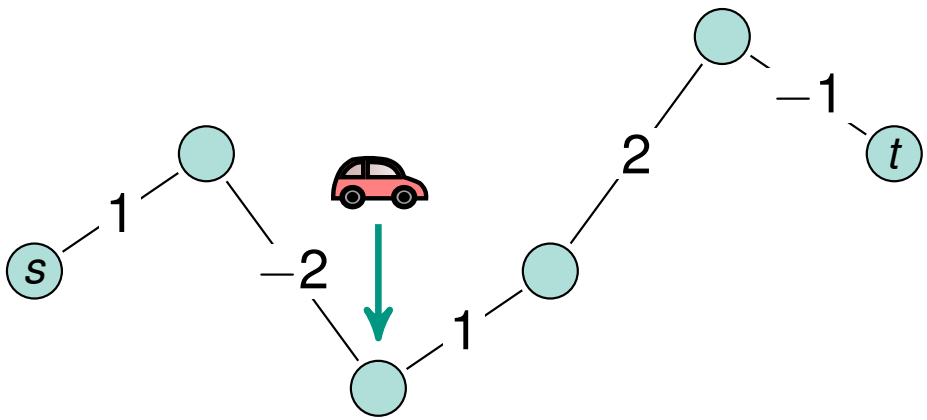


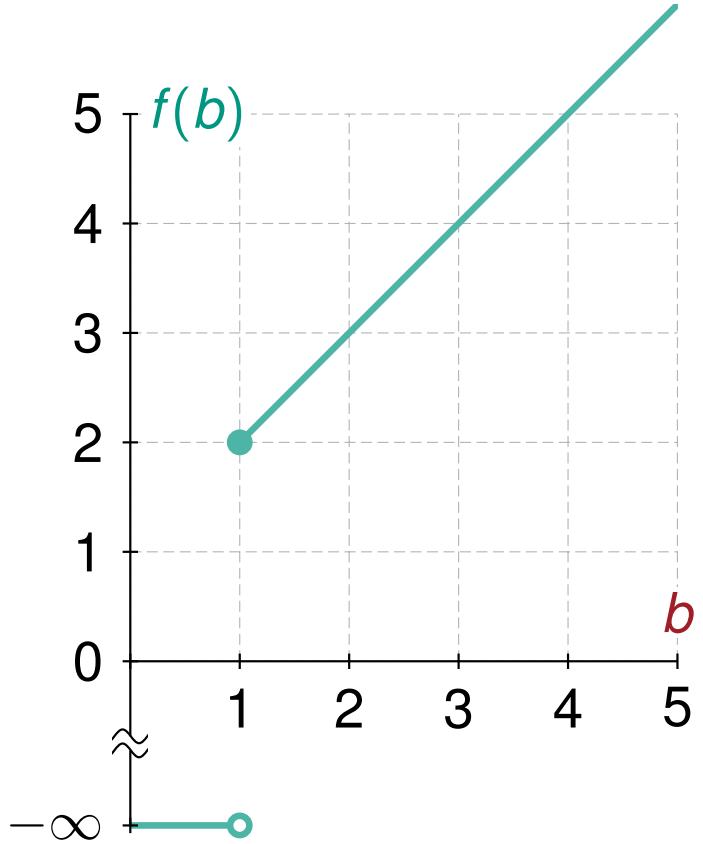
SoC-Profil

Optimaler SoC am Ziel hängt vom SoC am Startknoten ab

SoC-Profil f bildet SoC b am Start auf SoC $f(b)$ am Ziel ab

Beispiel: $M = 5$



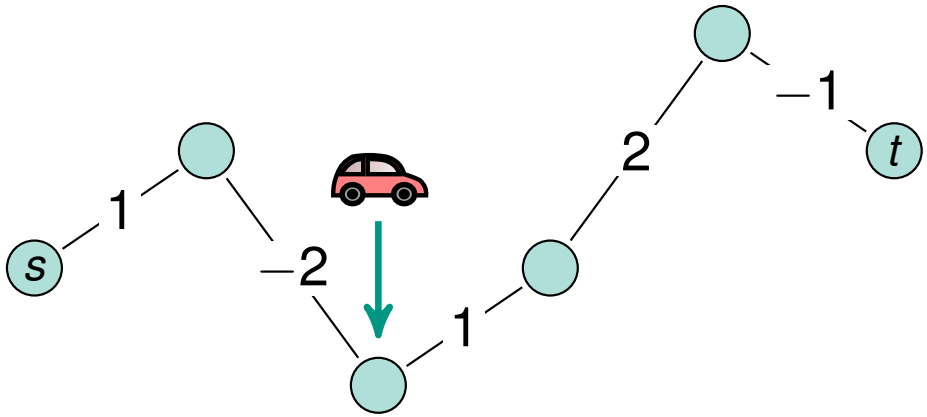


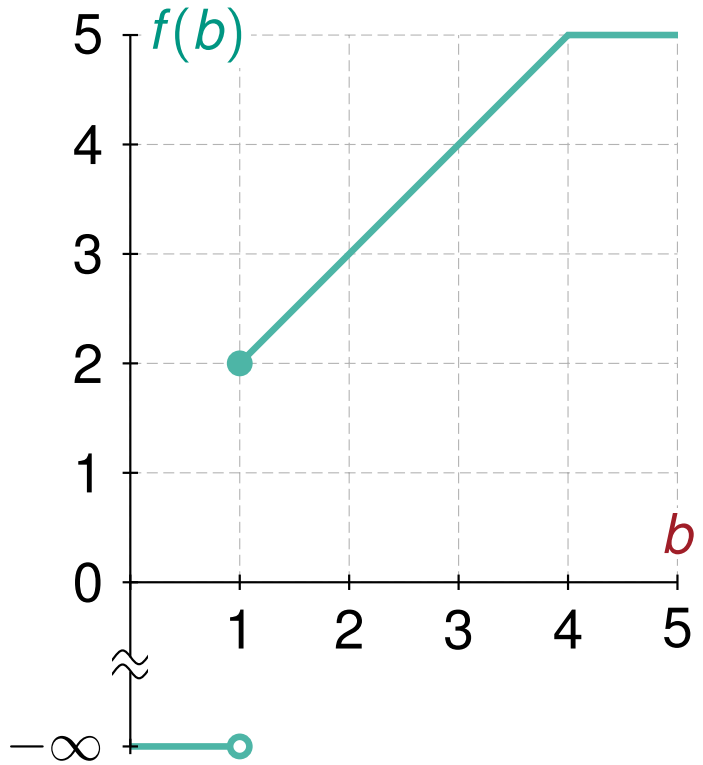
SoC-Profil

Optimaler SoC am Ziel hängt vom SoC am Startknoten ab

SoC-Profil f bildet SoC b am Start auf SoC $f(b)$ am Ziel ab

Beispiel: $M = 5$



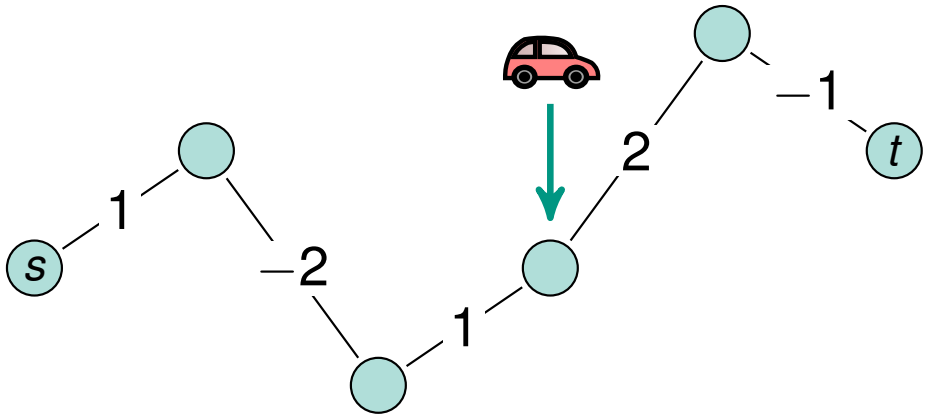


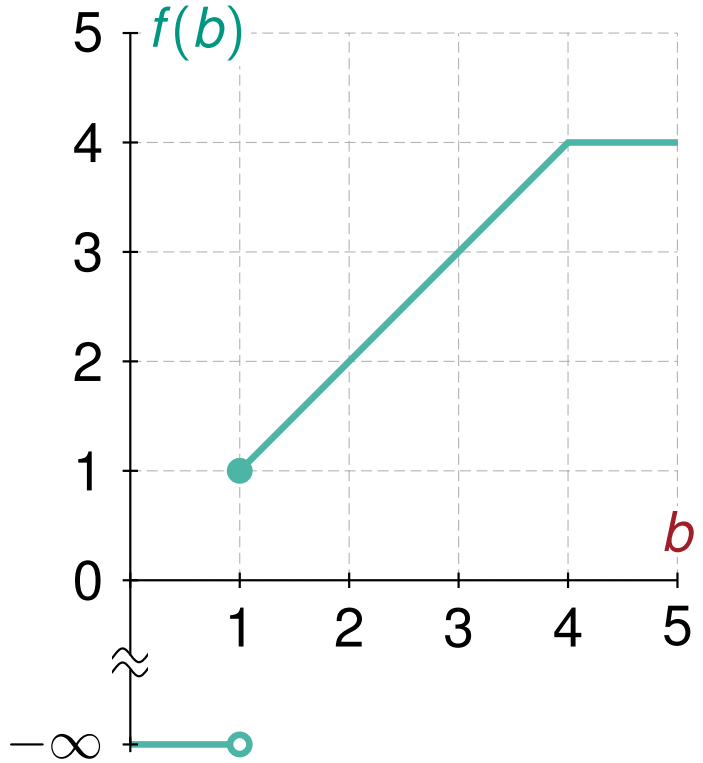
SoC-Profil

Optimaler SoC am Ziel hängt vom SoC am Startknoten ab

SoC-Profil f bildet SoC b am Start auf SoC $f(b)$ am Ziel ab

Beispiel: $M = 5$



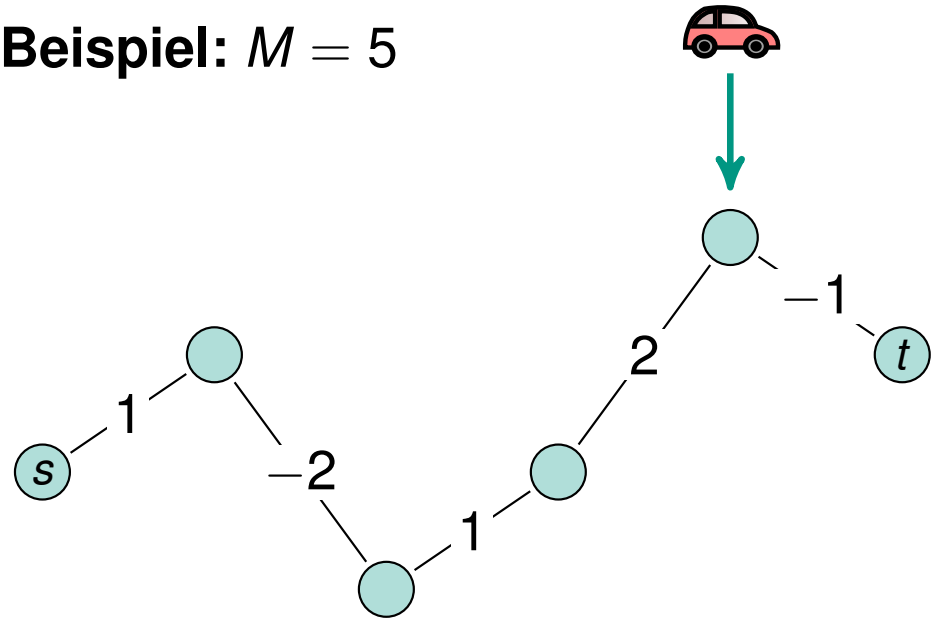


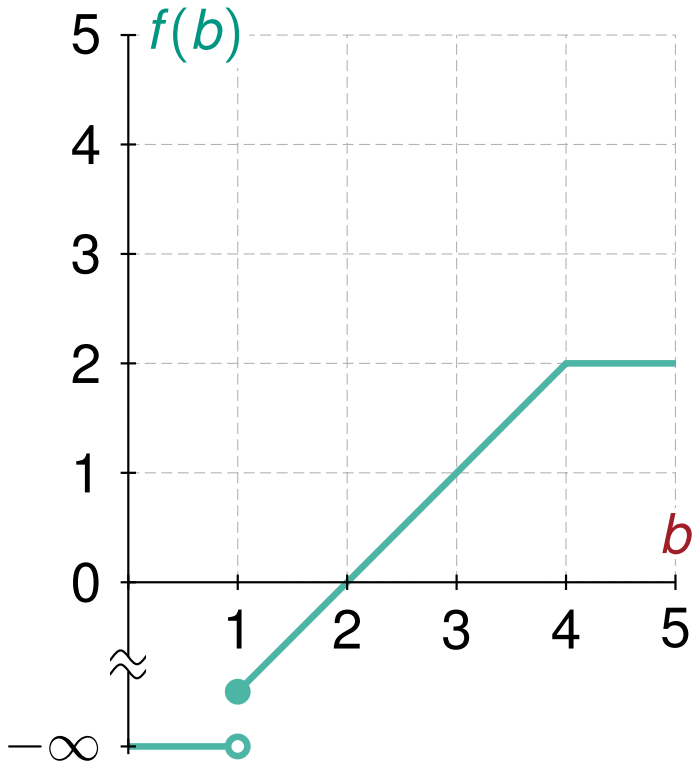
SoC-Profil

Optimaler SoC am Ziel hängt vom SoC am Startknoten ab

SoC-Profil f bildet SoC b am Start auf SoC $f(b)$ am Ziel ab

Beispiel: $M = 5$



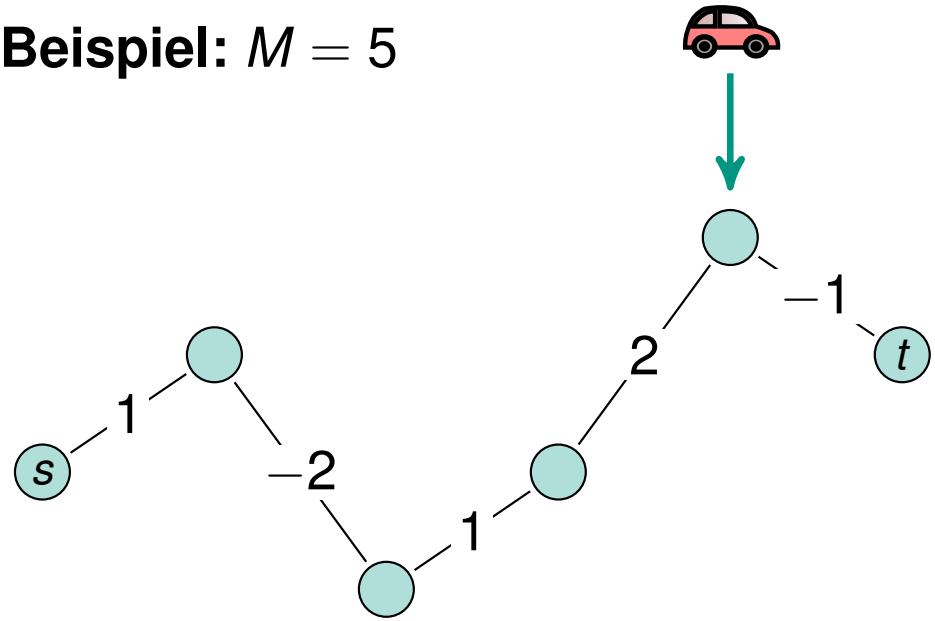


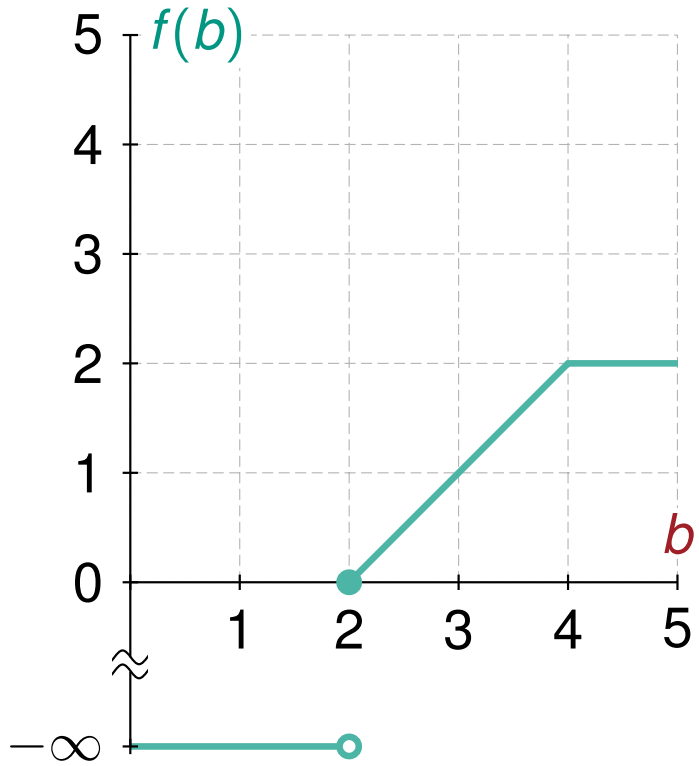
SoC-Profil

Optimaler SoC am Ziel hängt vom SoC am Startknoten ab

SoC-Profil f bildet SoC b am Start auf SoC $f(b)$ am Ziel ab

Beispiel: $M = 5$



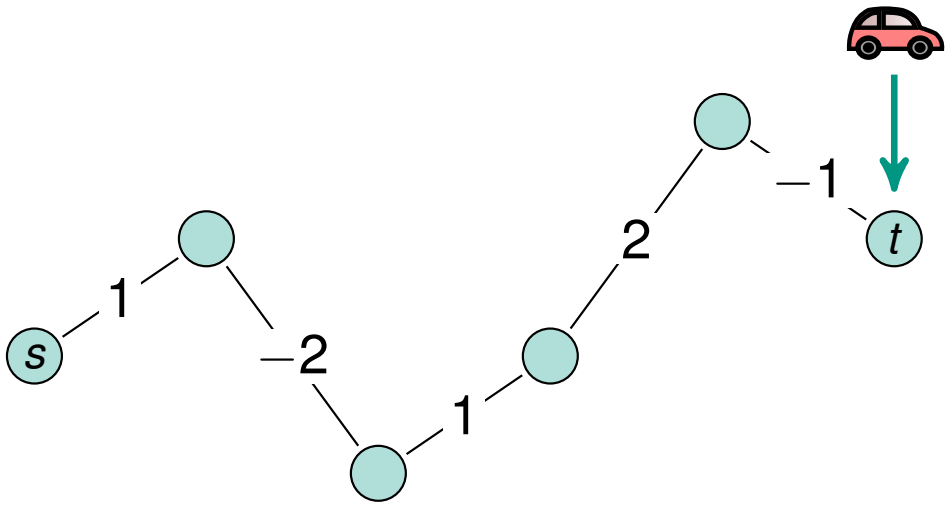


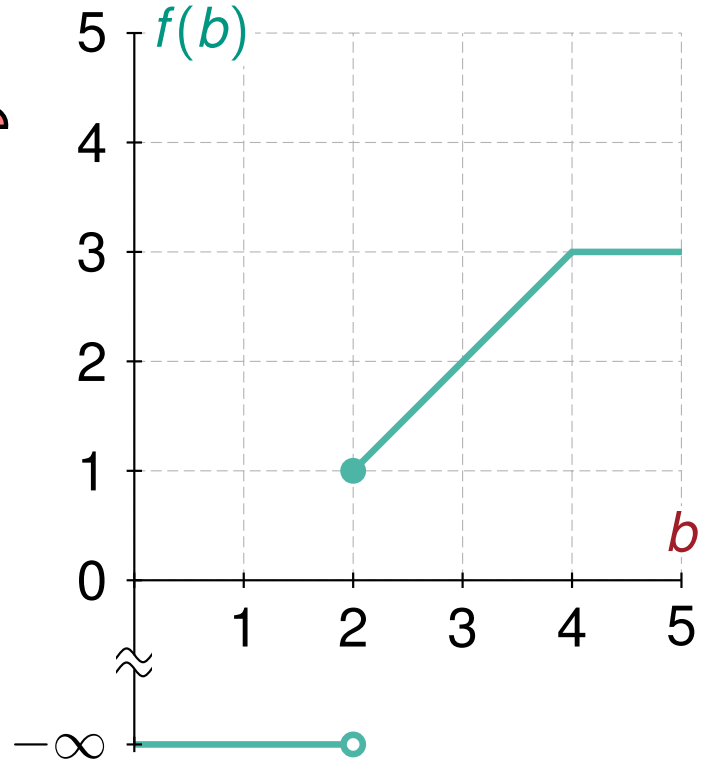
SoC-Profil

Optimaler SoC am Ziel hängt vom SoC am Startknoten ab

SoC-Profil f bildet SoC b am Start auf SoC $f(b)$ am Ziel ab

Beispiel: $M = 5$

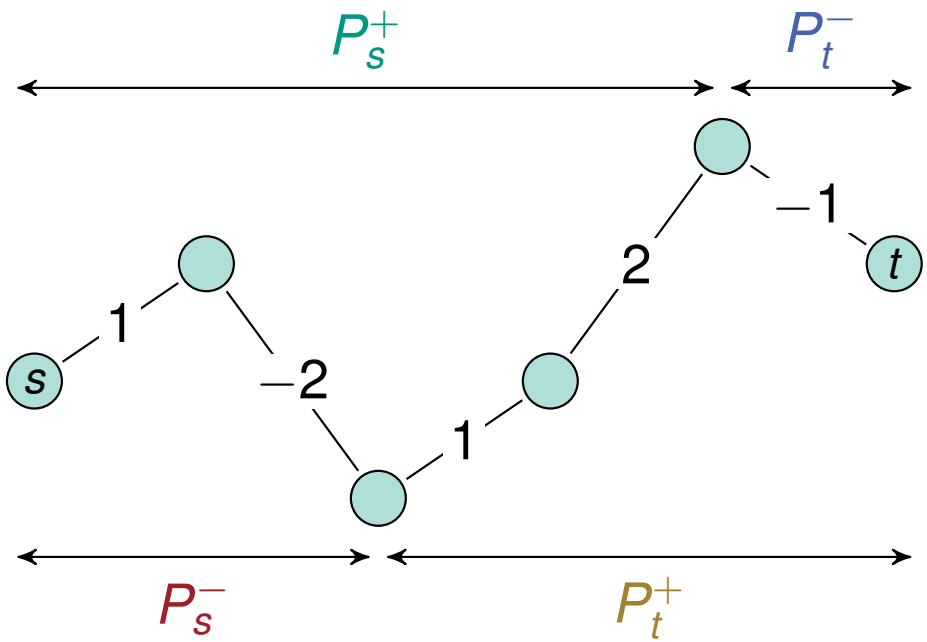


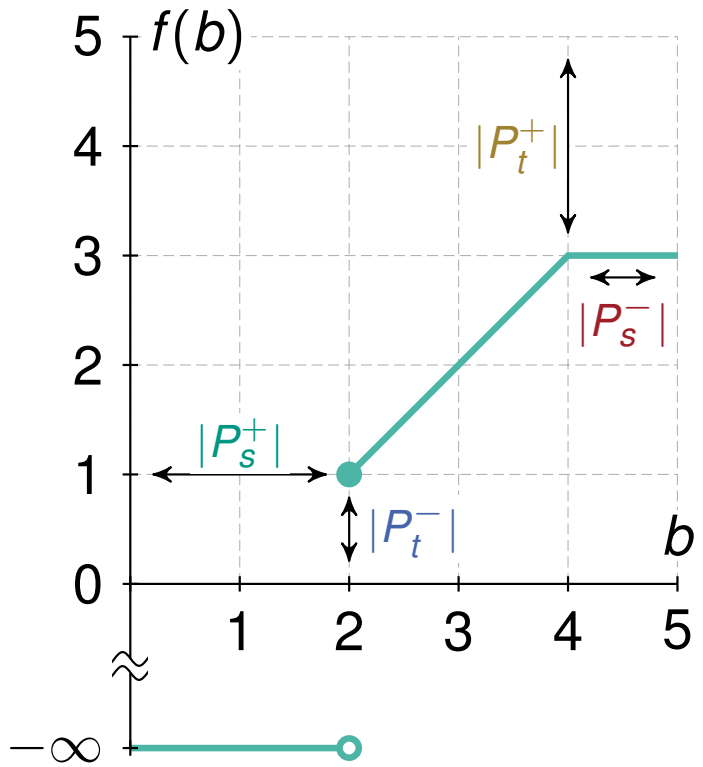


SoC-Profil eines Pfads

SoC-Profil von P bestimmt durch min./max. Präfix/Suffix

Beispiel:





⇒ Komplexität des SoC-Profils ist konstant

SoC-Profil eines Pfads

SoC-Profil entlang eines Pfads P hängt nur von 4 Subpfaden ab:

- max. Präfix P_s^+ von P:
  P befahrbar gdw. $b \geq |P_s^+|$ (Anm.: es gilt $|P_s^+| \geq 0$)
- min. Präfix P_s^- von P:
  Akku mindestens an einem Knoten auf P voll geladen gdw.
  $b - |P_s^-| \geq M$ (Anm.: es gilt $|P_s^-| \leq 0$)
- max. Suffix P_t^+ von P:
  Bestimmt, wie viel SoC nach Befahren höchstens übrig bleibt
  (Anm.: es gilt $|P_t^+| \geq 0$)
- min. Suffix P_t^- von P:
  Bestimmt, wie viel SoC nach Befahren mindestens übrig bleibt
  (Anm.: es gilt $|P_t^-| \leq 0$)

SoC-Profil $\leftrightarrow$ Verbrauchsfunktion

SoC-Profil entlang eines Pfads P kann mit 3 Werten beschrieben werden:

SoC-Profil $\rightarrow$ Verbrauchsfunktion:

- $\text{minIn}_P = |P_s^+|$
- $\text{maxOut}_P = M - |P_t^+|$
- $\text{cost}_P = |P_s^+| - |P_t^-|$

SoC-Profil $\leftrightarrow$ Verbrauchsfunktion

SoC-Profil entlang eines Pfads P kann mit 3 Werten beschrieben werden:

SoC-Profil $\rightarrow$ Verbrauchsfunktion:

- $\text{minIn}_P = |P_s^+|$
- $\text{maxOut}_P = M - |P_t^+|$
- $\text{cost}_P = |P_s^+| - |P_t^-|$

Verbrauchsfunktion $\rightarrow$ SoC-Profil:

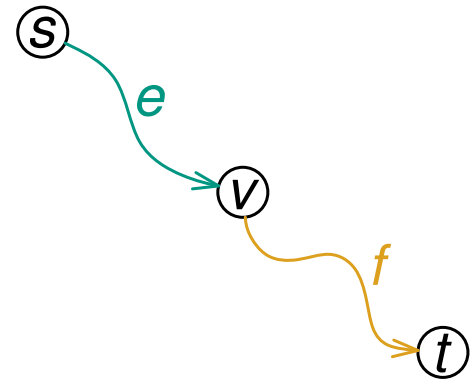
- $|P_s^+| = \text{minIn}_P$
- $|P_s^-| = M - \text{maxOut}_P - \text{cost}_P$
- $|P_t^+| = M - \text{maxOut}_P$
- $|P_t^-| = \text{minIn}_P - \text{cost}_P$

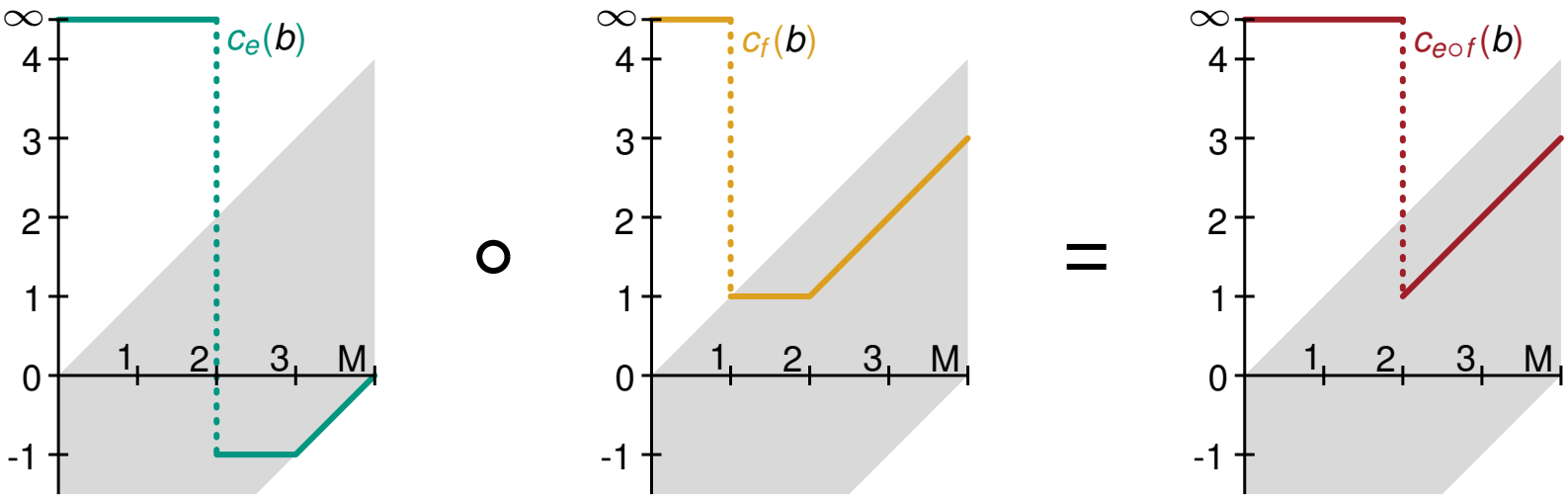
Verbrauchsfunktionen – Linking

Gesucht: Verbrauch beim traversieren von e *und* f

- Verbrauch auf e gegeben durch $c_e(b)$
- SoC nach e $=$ SoC vor f $=$ $b - c_e(b)$

$\Rightarrow$ Verbrauch auf e UND f: $c_{e \circ f}(b) = c_f(b - c_e(b))$

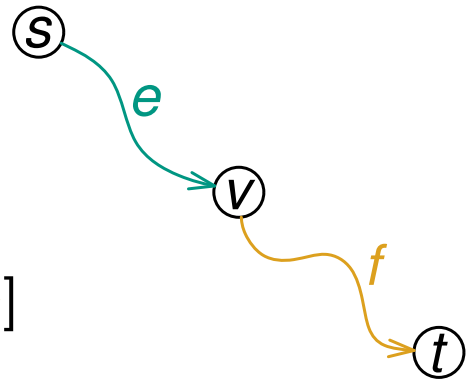


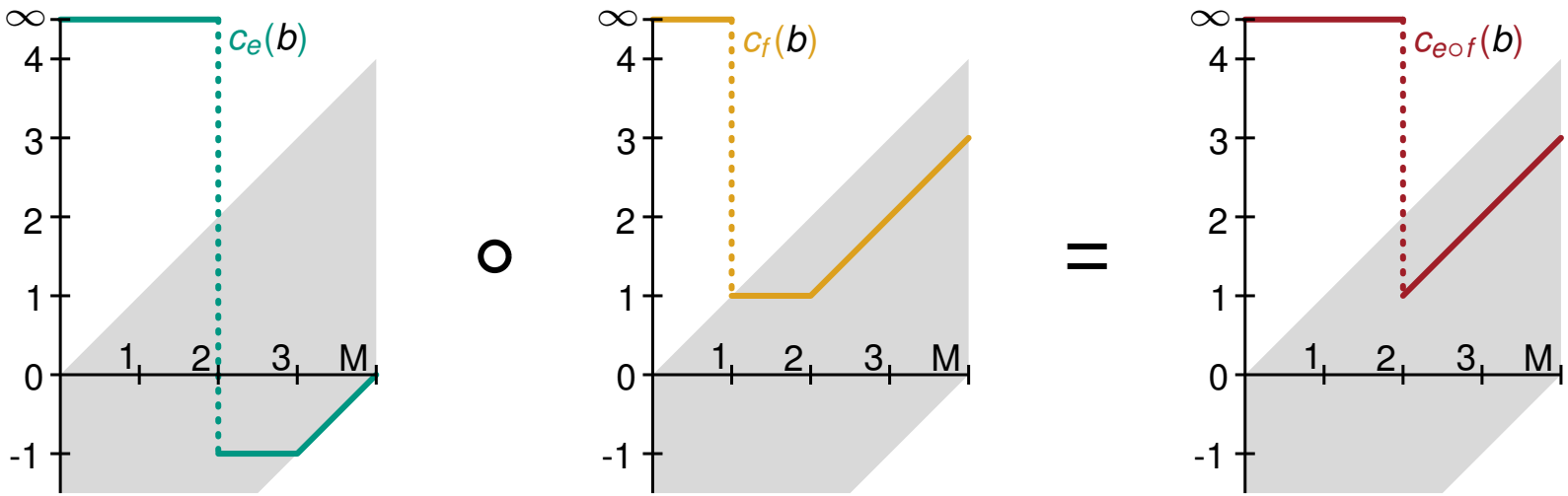


Verbrauchsfunktionen – Linking

Formal: $c_{e \circ f}(b)$ ist gegeben durch:

$$\begin{aligned} \text{minIn}_{e \circ f} &= \max[\text{minIn}_e,\ \text{minIn}_f + \text{cost}_e] \\ \text{maxOut}_{e \circ f} &= \min[\text{maxOut}_f,\ \text{maxOut}_e - \text{cost}_f] \\ \text{cost}_{e \circ f} &= \max[\text{cost}_e + \text{cost}_f,\ \text{minIn}_e - \text{maxOut}_f] \end{aligned}$$



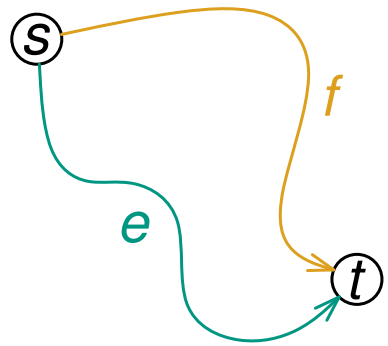


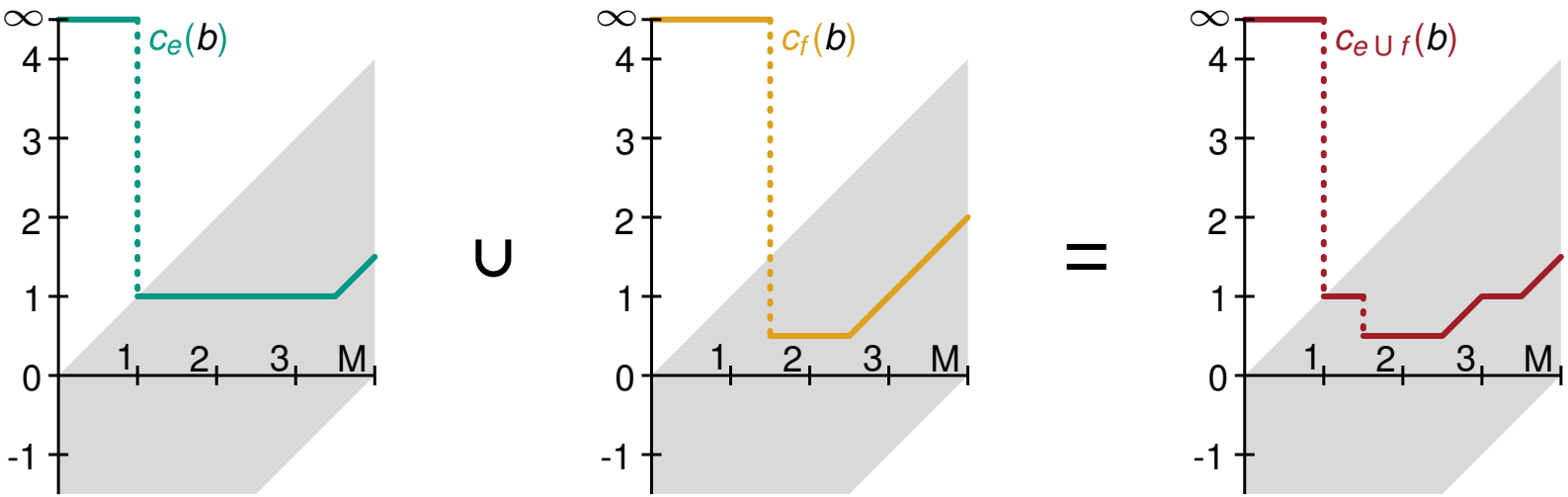
Verbrauchsfunktionen – Merging

Gesucht: Verbrauch beim traversieren von e *oder* f

- Benutze Pfad mit geringerem Verbrauch

$\Rightarrow$ Verbrauch auf e ODER f: $c_{e \cup f}(b) = \min(c_f(b), c_e(b))$



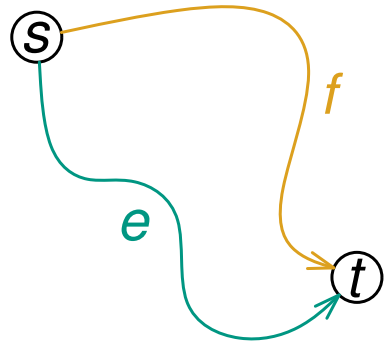


Verbrauchsfunktionen – Merging

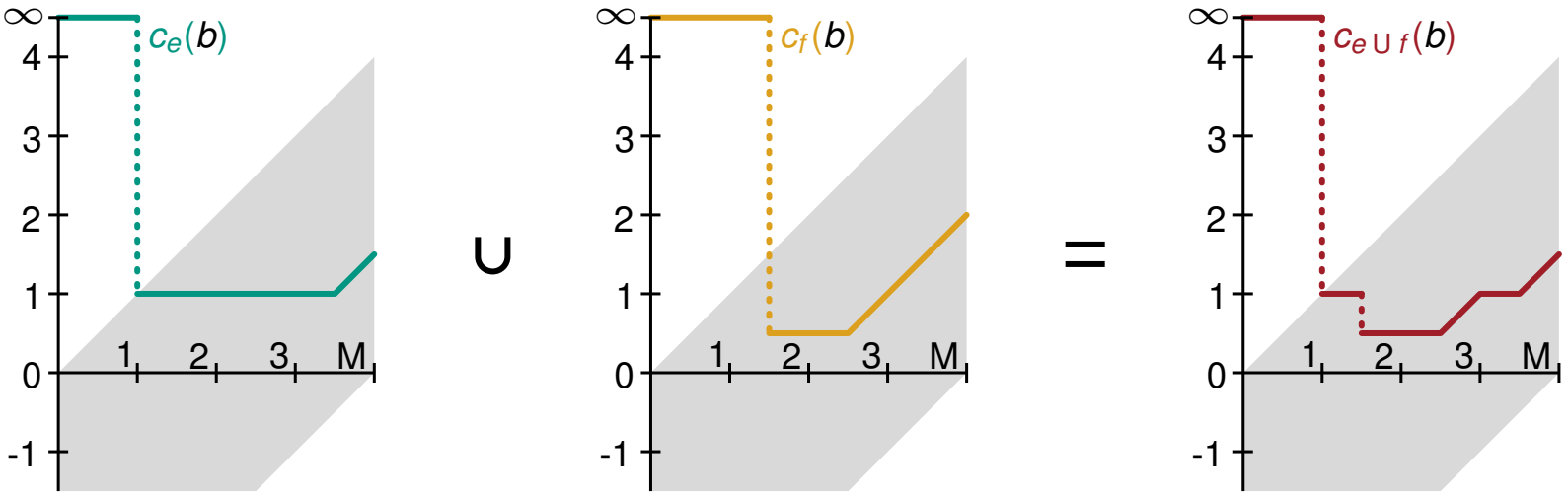
Gesucht: Verbrauch beim traversieren von e *oder* f

- Benutze Pfad mit geringerem Verbrauch

$\Rightarrow$ Verbrauch auf e ODER f: $c_{e \cup f}(b) = \min(c_f(b), c_e(b))$



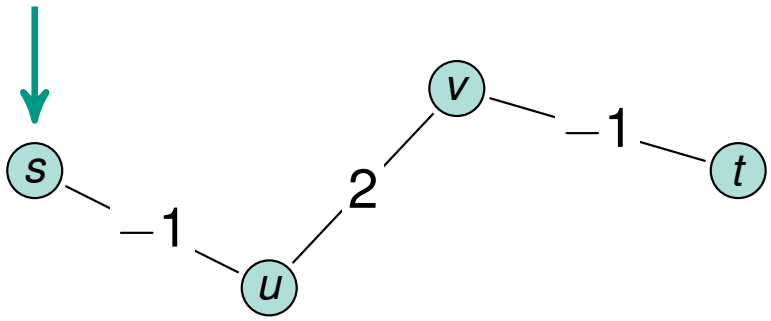
Aber: Im Allgemeinem $\mathcal{O}(m)$ Stützstellen

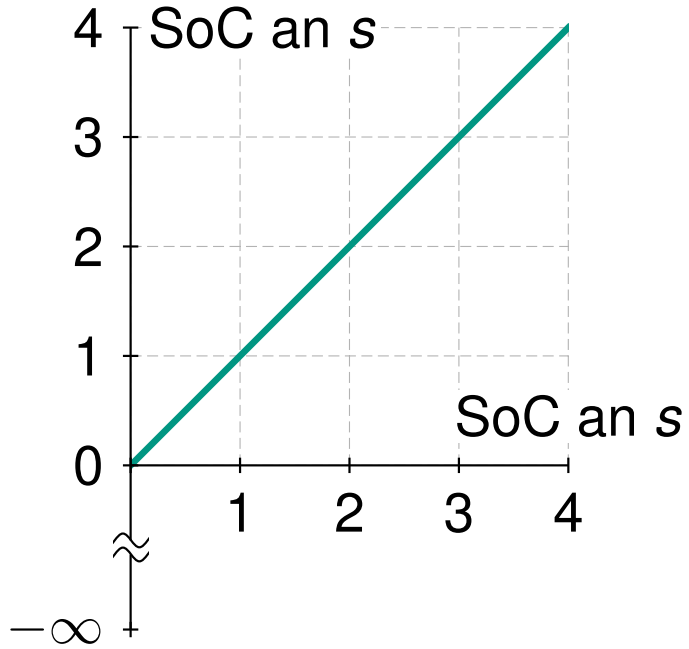


Allgemeine Profile

Profilsuche: Für Start s und Ziel t, finde zulässige s-t-Pfade mit maximalem SoC an t für *alle* $b_s \in [0, M]$

Beispiel: $M = 4$



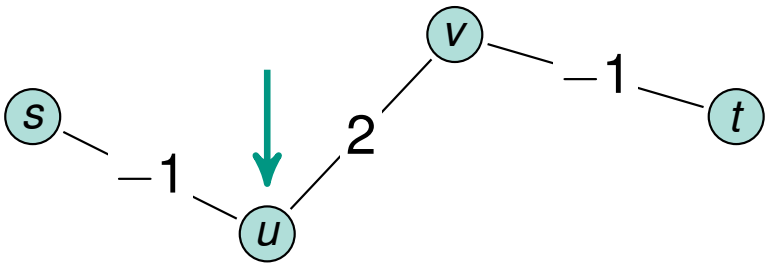


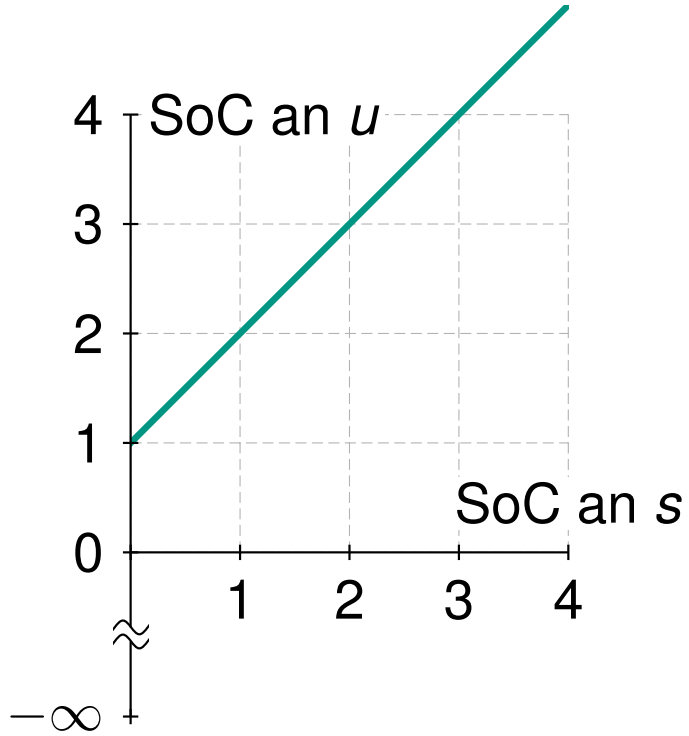
Anzahl Stützpunkte ist linear in der Anzahl Pfade
$\Rightarrow$ Wie viele verschiedene Pfade tragen zum Profil bei?

Allgemeine Profile

Profilsuche: Für Start s und Ziel t, finde zulässige s-t-Pfade mit maximalem SoC an t für *alle* $b_s \in [0, M]$

Beispiel: $M = 4$



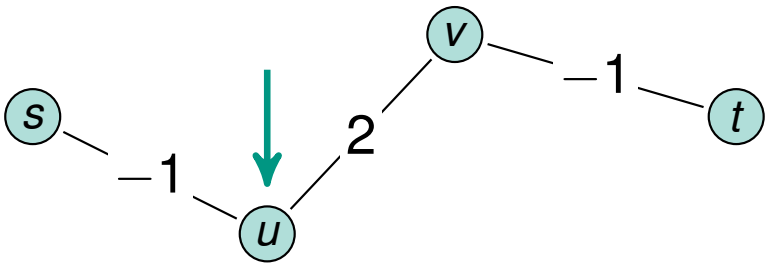


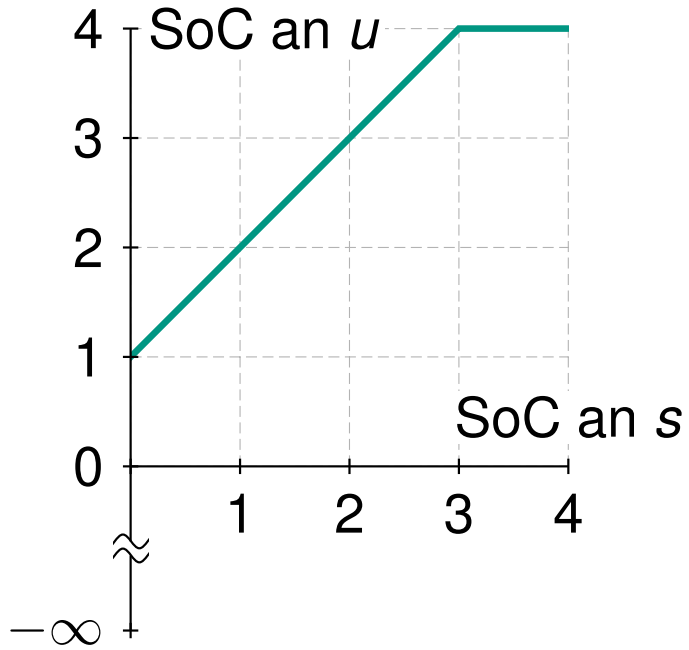
Anzahl Stützpunkte ist linear in der Anzahl Pfade
$\Rightarrow$ Wie viele verschiedene Pfade tragen zum Profil bei?

Allgemeine Profile

Profilsuche: Für Start s und Ziel t, finde zulässige s-t-Pfade mit maximalem SoC an t für *alle* $b_s \in [0, M]$

Beispiel: $M = 4$



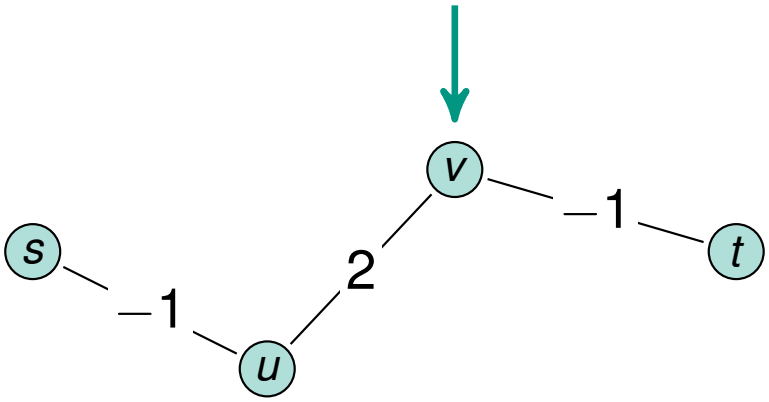


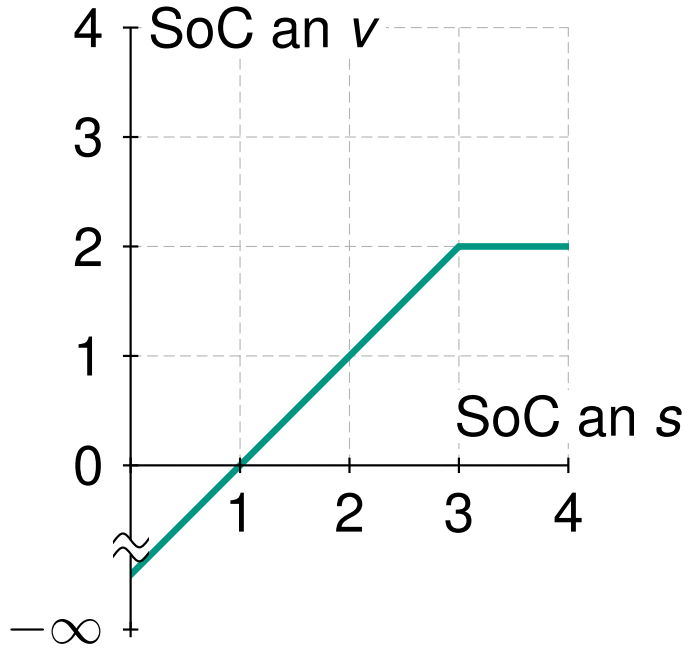
Anzahl Stützpunkte ist linear in der Anzahl Pfade
$\Rightarrow$ Wie viele verschiedene Pfade tragen zum Profil bei?

Allgemeine Profile

Profilsuche: Für Start s und Ziel t, finde zulässige s-t-Pfade mit maximalem SoC an t für *alle* $b_s \in [0, M]$

Beispiel: $M = 4$



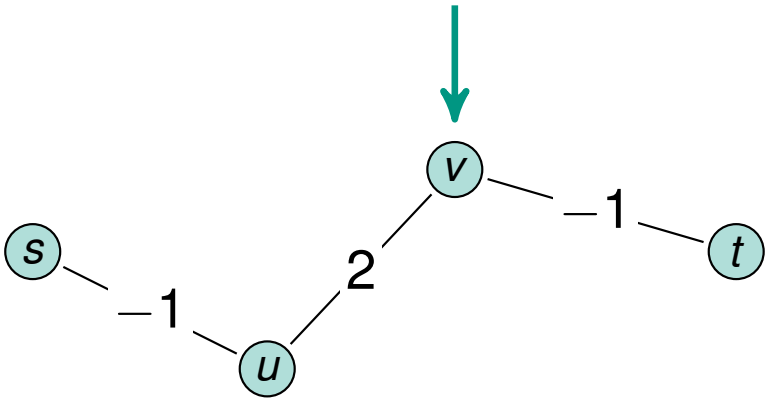


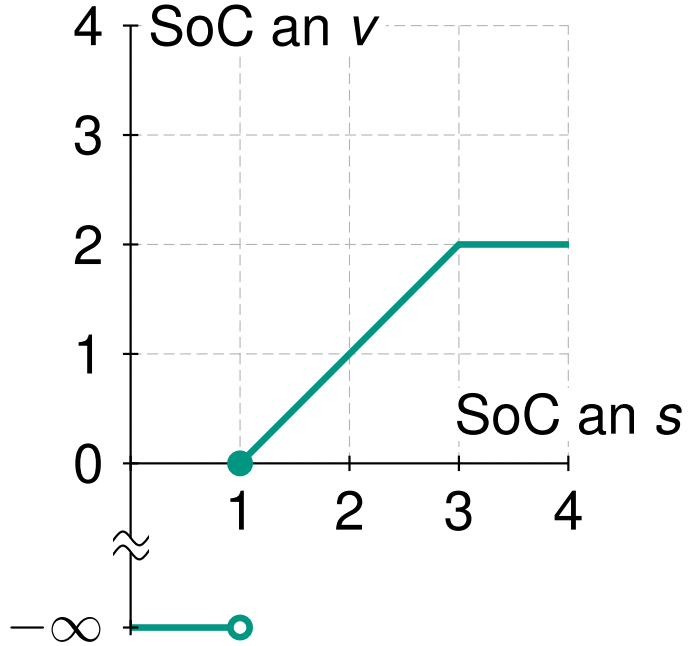
Anzahl Stützpunkte ist linear in der Anzahl Pfade
$\Rightarrow$ Wie viele verschiedene Pfade tragen zum Profil bei?

Allgemeine Profile

Profilsuche: Für Start s und Ziel t, finde zulässige s-t-Pfade mit maximalem SoC an t für *alle* $b_s \in [0, M]$

Beispiel: $M = 4$



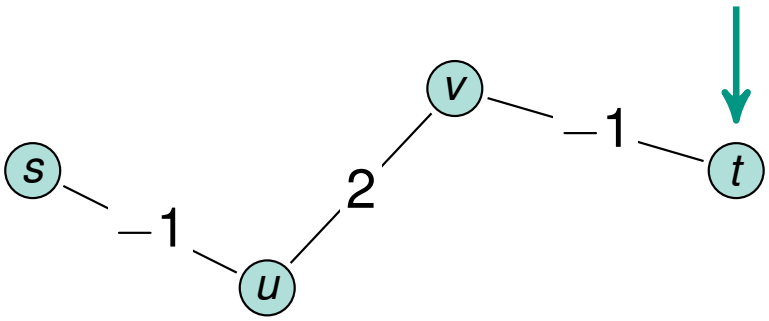


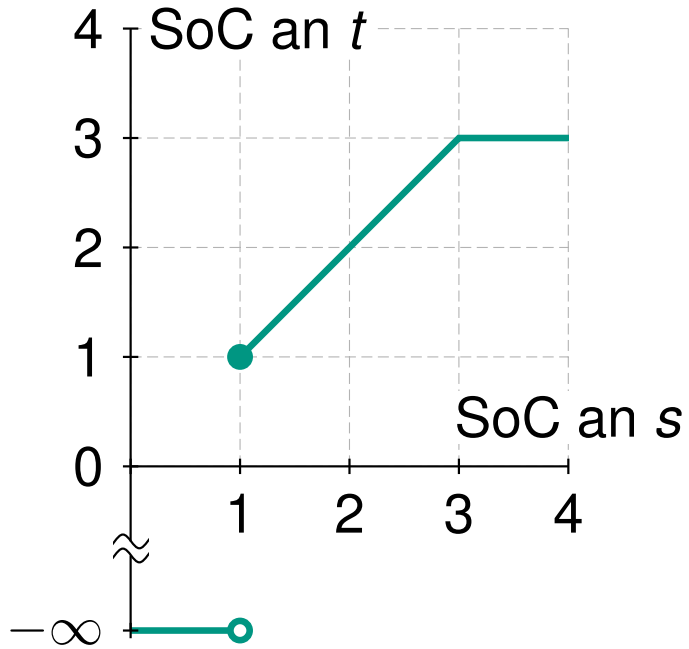
Anzahl Stützpunkte ist linear in der Anzahl Pfade
$\Rightarrow$ Wie viele verschiedene Pfade tragen zum Profil bei?

Allgemeine Profile

Profilsuche: Für Start s und Ziel t, finde zulässige s-t-Pfade mit maximalem SoC an t für *alle* $b_s \in [0, M]$

Beispiel: $M = 4$



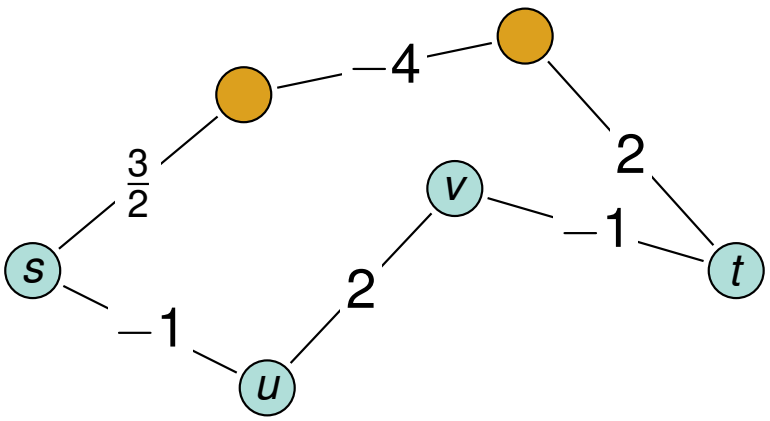


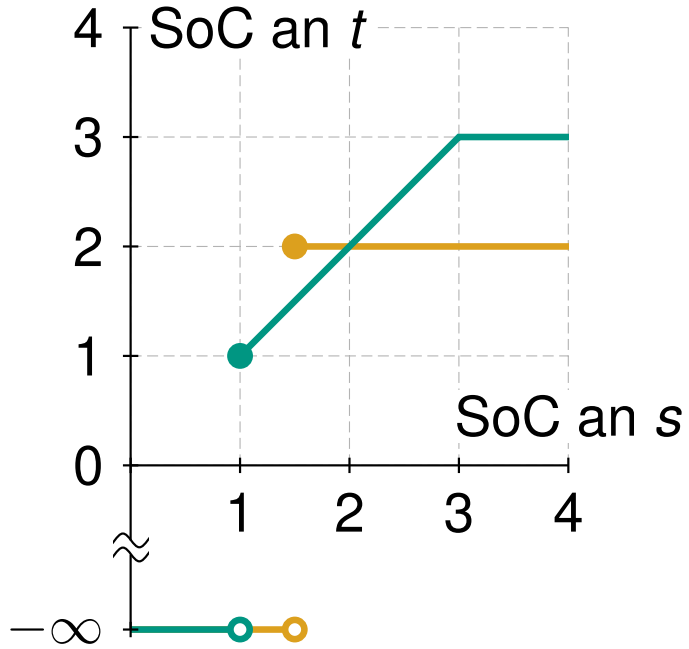
Anzahl Stützpunkte ist linear in der Anzahl Pfade
$\Rightarrow$ Wie viele verschiedene Pfade tragen zum Profil bei?

Allgemeine Profile

Profilsuche: Für Start s und Ziel t, finde zulässige s-t-Pfade mit maximalem SoC an t für *alle* $b_s \in [0, M]$

Beispiel: $M = 4$



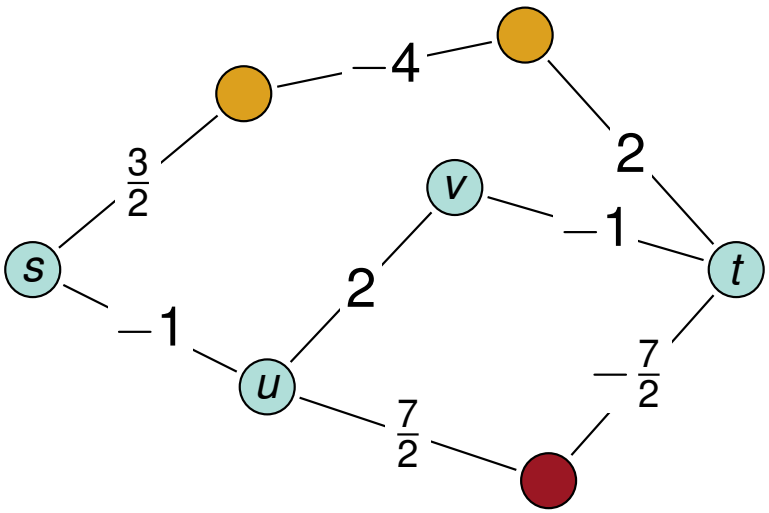


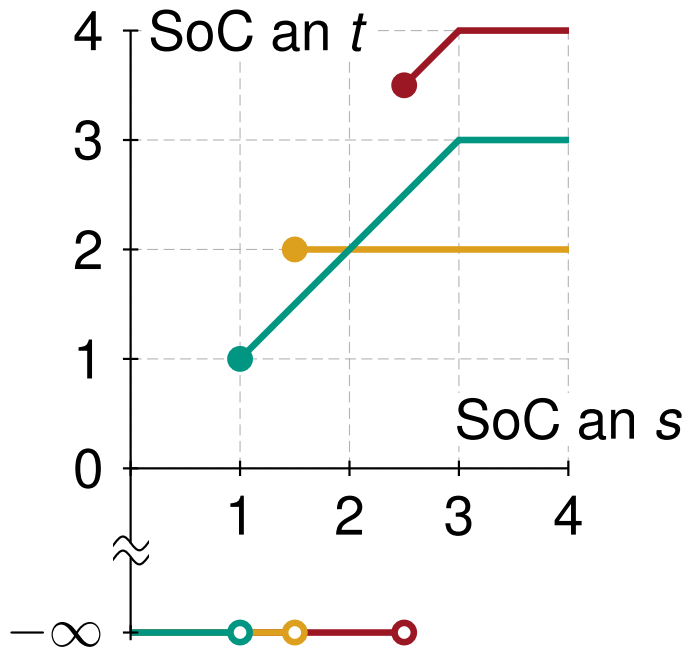
Anzahl Stützpunkte ist linear in der Anzahl Pfade
$\Rightarrow$ Wie viele verschiedene Pfade tragen zum Profil bei?

Allgemeine Profile

Profilsuche: Für Start s und Ziel t, finde zulässige s-t-Pfade mit maximalem SoC an t für *alle* $b_s \in [0, M]$

Beispiel: $M = 4$



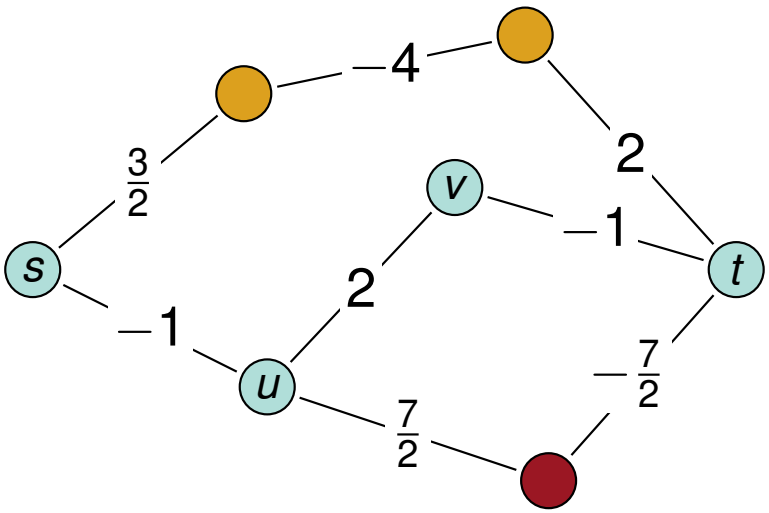


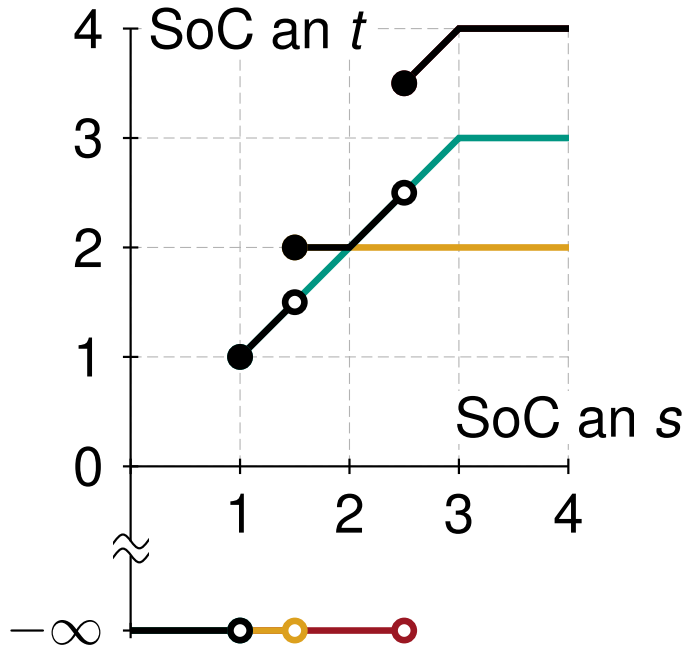
Anzahl Stützpunkte ist linear in der Anzahl Pfade
$\Rightarrow$ Wie viele verschiedene Pfade tragen zum Profil bei?

Allgemeine Profile

Profilsuche: Für Start s und Ziel t, finde zulässige s-t-Pfade mit maximalem SoC an t für *alle* $b_s \in [0, M]$

Beispiel: $M = 4$



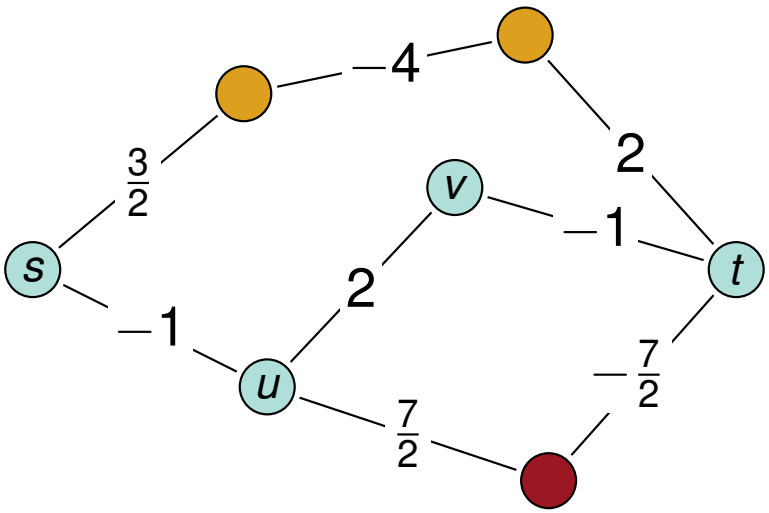


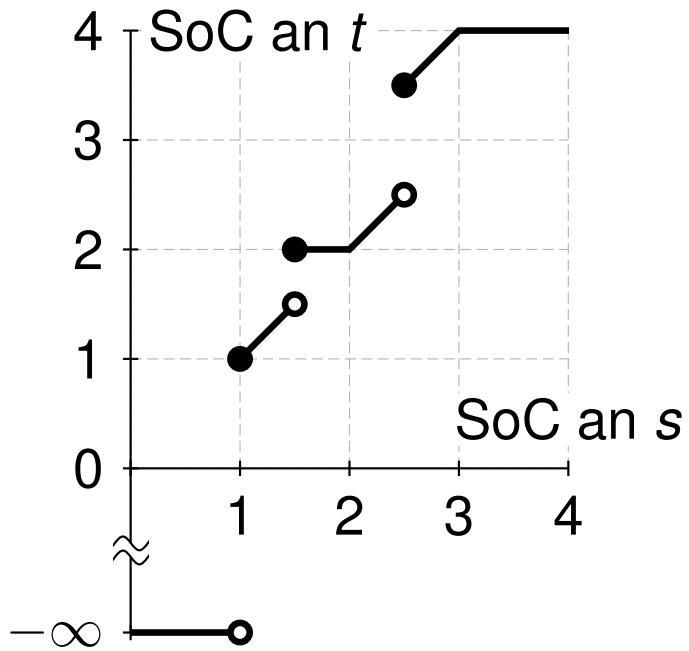
Anzahl Stützpunkte ist linear in der Anzahl Pfade
$\Rightarrow$ Wie viele verschiedene Pfade tragen zum Profil bei?

Allgemeine Profile

Profilsuche: Für Start s und Ziel t, finde zulässige s-t-Pfade mit maximalem SoC an t für *alle* $b_s \in [0, M]$

Beispiel: $M = 4$

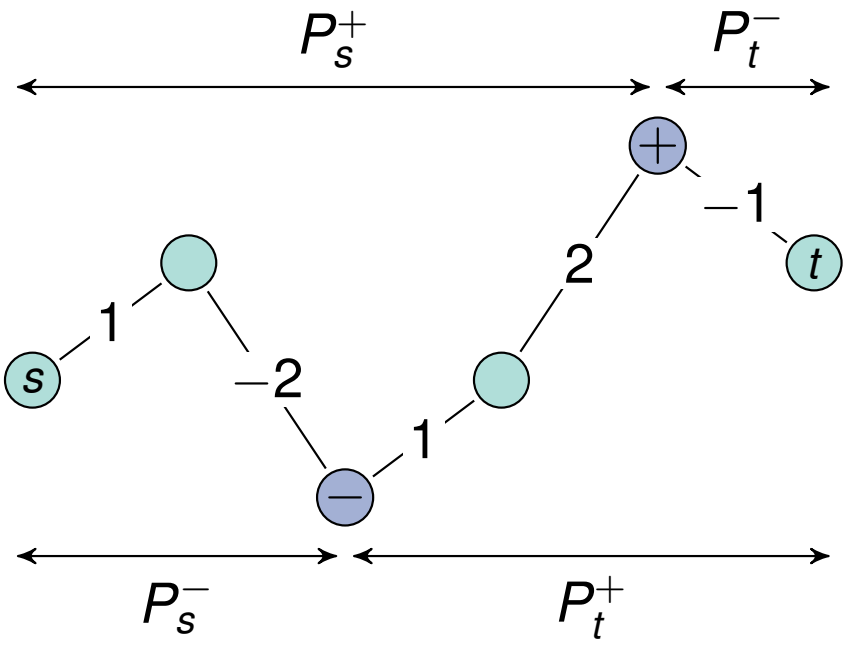




Anzahl Stützpunkte ist linear in der Anzahl Pfade
$\Rightarrow$ Wie viele verschiedene Pfade tragen zum Profil bei?

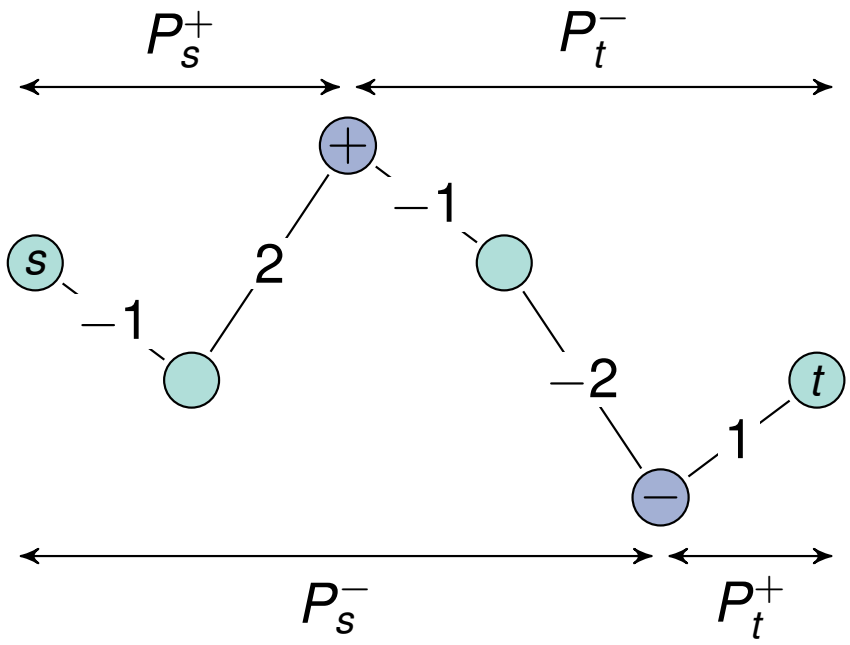
Typen von Pfaden

Tal-Berg-Pfad:



Min. Präfix endet vor max. Präfix

Berg-Tal-Pfad:



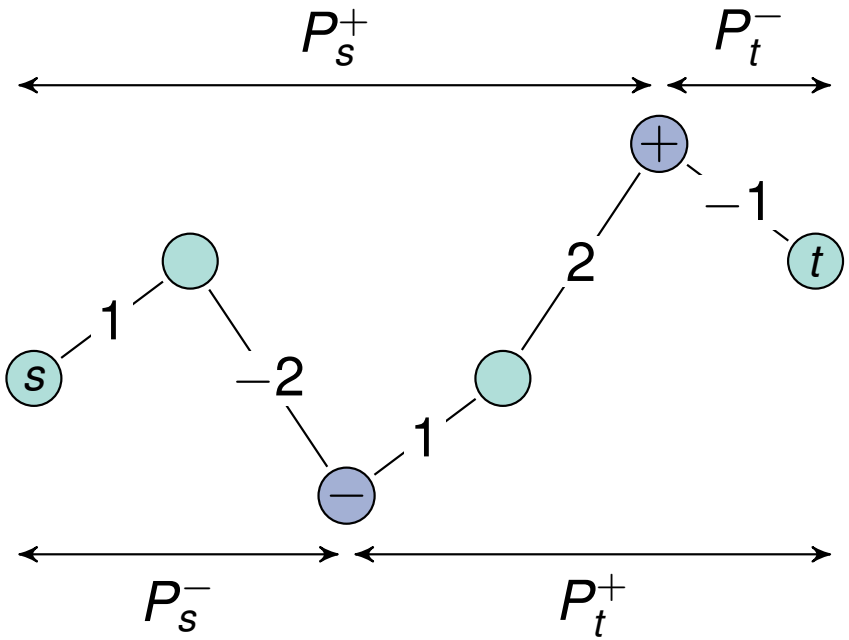
Max. Präfix endet vor min. Präfix

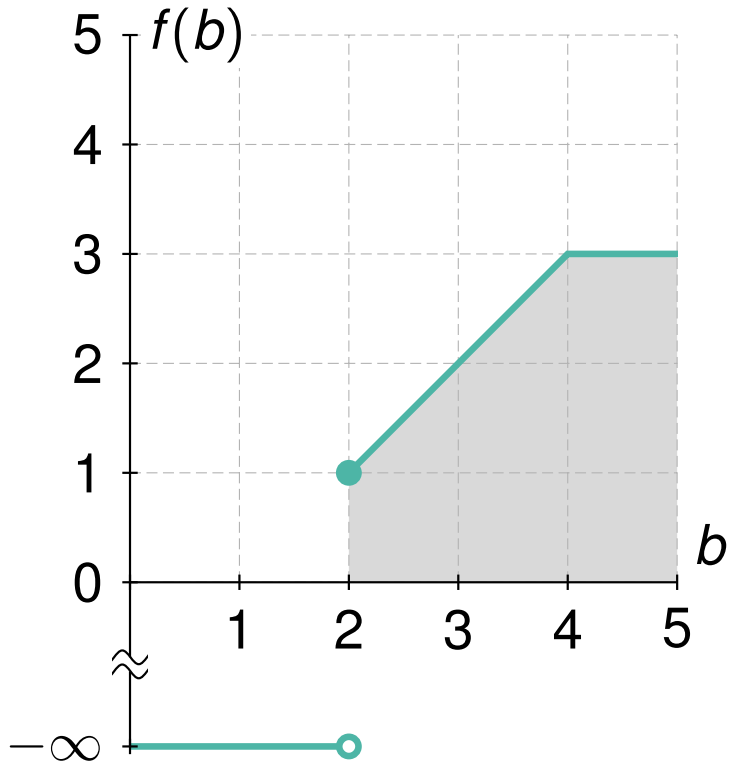
- Letzter Knoten des max. Präfix heißt Bergknoten
- Letzter Knoten des min. Präfix heißt Talknoten

Komplexität von SoC-Profilen

Betrachte beliebiges paar von Bergknoten $\overline{v}$ und Talknoten $\underline{v}$:

Höchstens ein Tal-Berg-Pfad und ein Berg-Tal-Pfad benutzen $\overline{v}$ und $\underline{v}$ als Berg- bzw. Talknoten

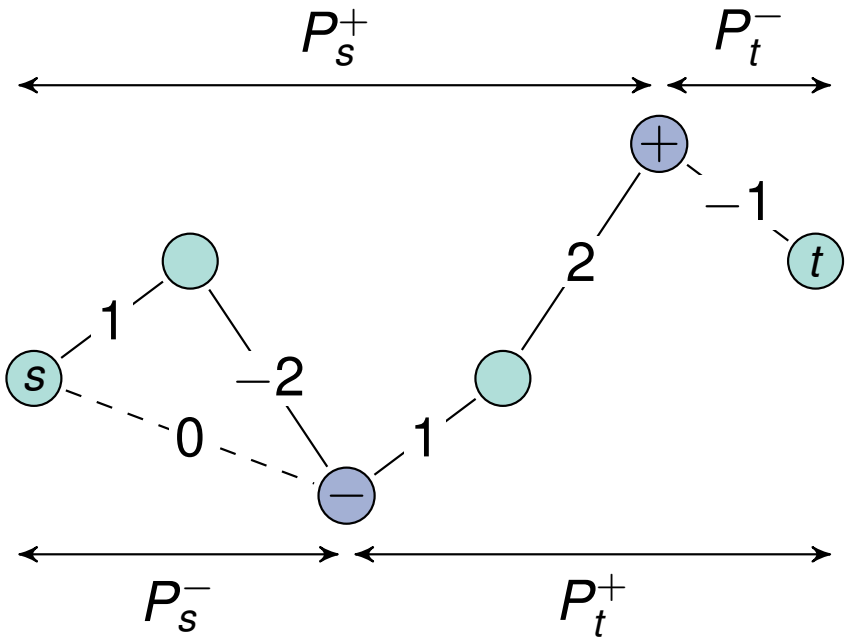


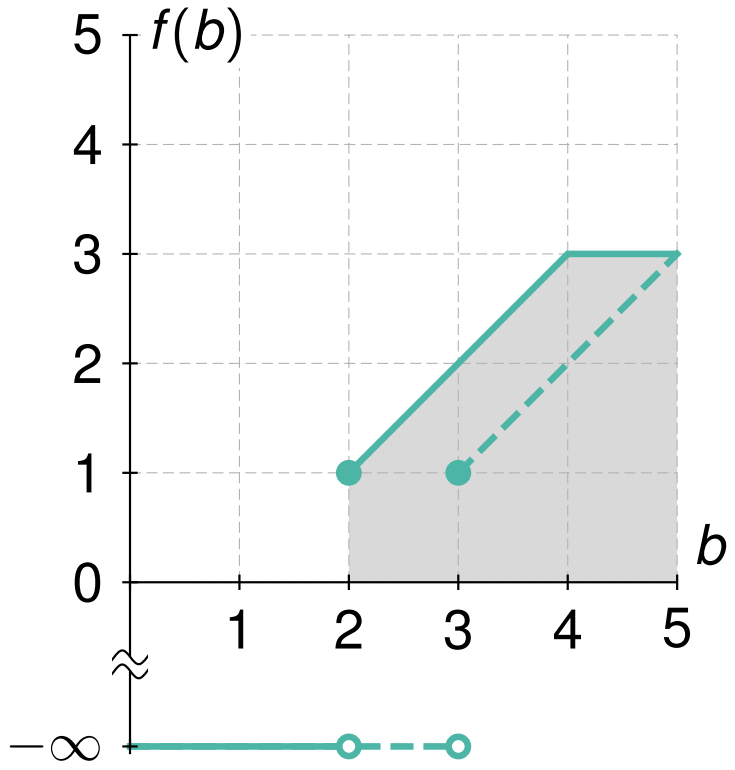


Komplexität von SoC-Profilen

Betrachte beliebiges paar von Bergknoten $\overline{v}$ und Talknoten $\underline{v}$:

Höchstens ein Tal-Berg-Pfad und ein Berg-Tal-Pfad benutzen $\overline{v}$ und $\underline{v}$ als Berg- bzw. Talknoten

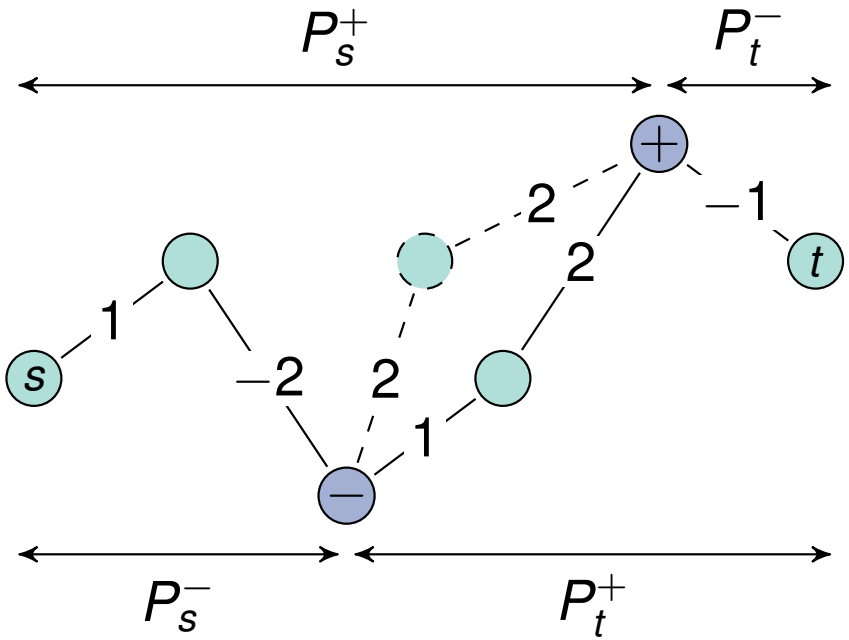


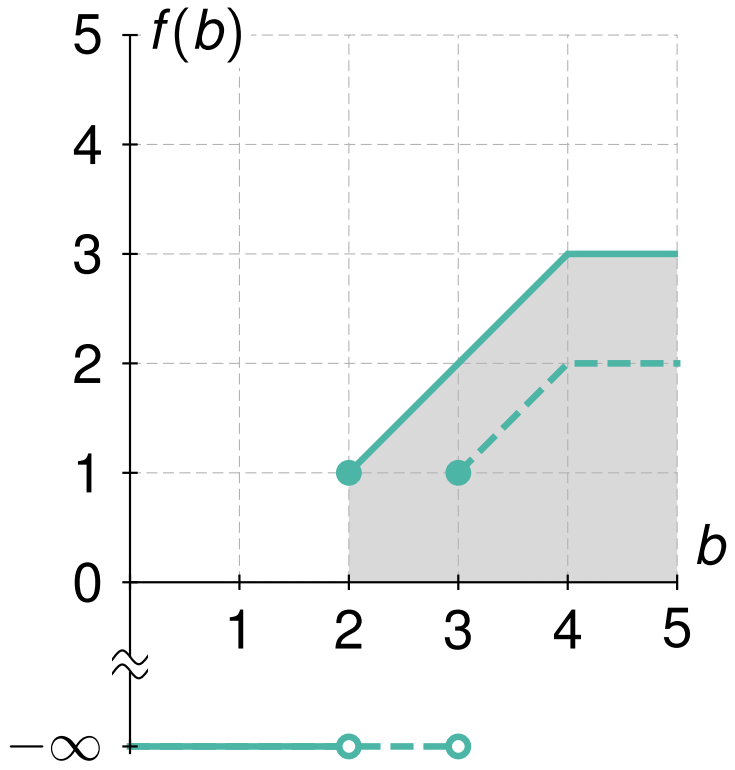


Komplexität von SoC-Profilen

Betrachte beliebiges paar von Bergknoten $\overline{v}$ und Talknoten $\underline{v}$:

Höchstens ein Tal-Berg-Pfad und ein Berg-Tal-Pfad benutzen $\overline{v}$ und $\underline{v}$ als Berg- bzw. Talknoten

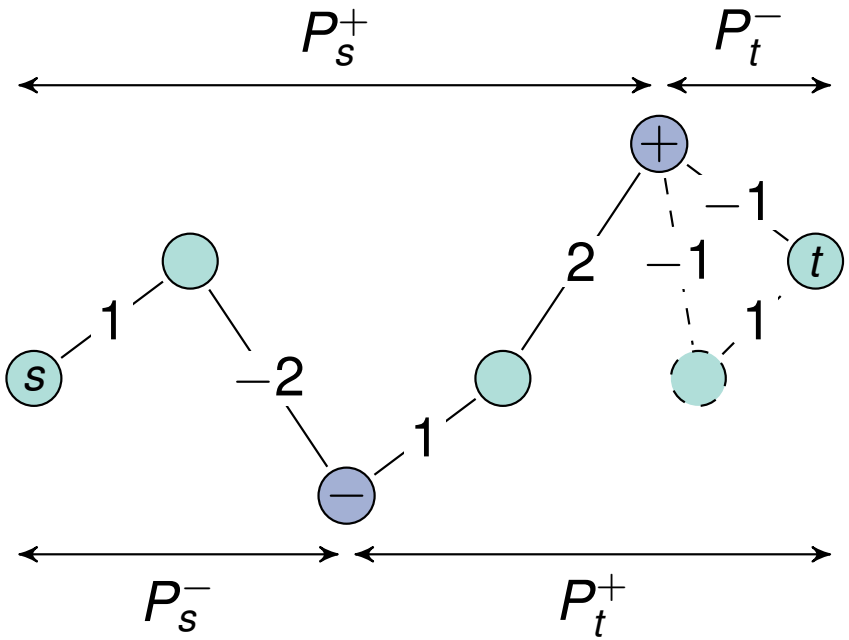


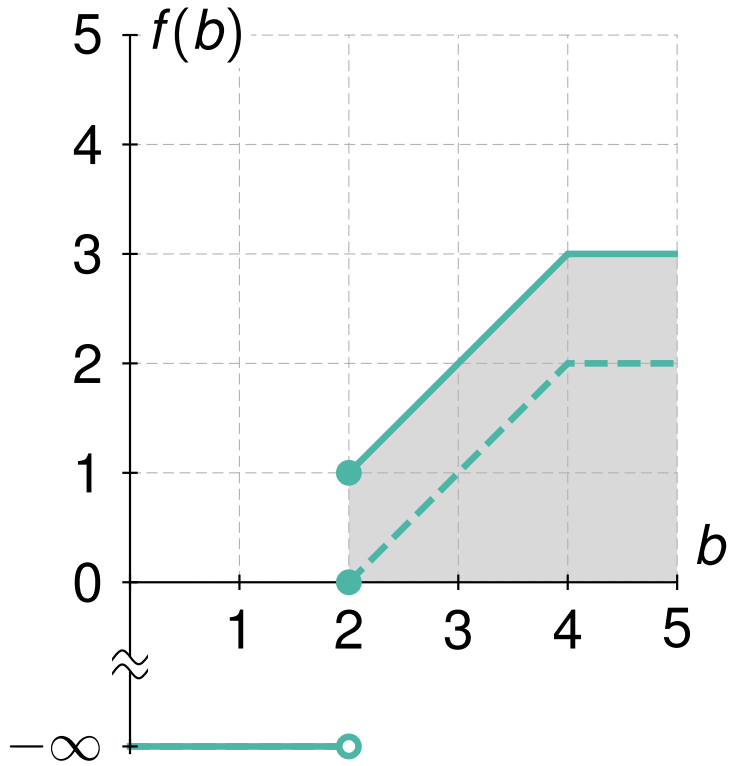


Komplexität von SoC-Profilen

Betrachte beliebiges paar von Bergknoten $\overline{v}$ und Talknoten $\underline{v}$:

Höchstens ein Tal-Berg-Pfad und ein Berg-Tal-Pfad benutzen $\overline{v}$ und $\underline{v}$ als Berg- bzw. Talknoten





Komplexität von SoC-Profilen

Lemma

Für bel. Knoten $s, t, \underline{v}, \overline{v}$ gilt:
Das SoC-Profil für Start s und Ziel t enthält

- *höchstens einen Tal-Berg-Pfad mit Talknoten $\underline{v}$ und Bergknoten $\overline{v}$*
- *höchstens einen Berg-Tal-Pfad mit Bergknoten $\overline{v}$ und Talknoten $\underline{v}$*

$\Rightarrow$ höchstens $\mathcal{O}(|V|^2)$ Pfade (und somit Stützpunkte) im Profil

Komplexität von SoC-Profilen

Lemma

Für bel. Knoten $s, t, \underline{v}, \overline{v}$ gilt:
Das SoC-Profil für Start s und Ziel t enthält

- *höchstens einen Tal-Berg-Pfad mit Talknoten $\underline{v}$ und Bergknoten $\overline{v}$*
- *höchstens einen Berg-Tal-Pfad mit Bergknoten $\overline{v}$ und Talknoten $\underline{v}$*

$\Rightarrow$ höchstens $\mathcal{O}(|V|^2)$ Pfade (und somit Stützpunkte) im Profil

Man kann sogar zeigen:

Theorem

Die Anzahl der Pfade (und Stützpunkte) eines SoC-Profils liegt in $\mathcal{O}(|V|)$.

$\Rightarrow$ SoC-Profile haben lineare Komplexität (und sind in der Praxis klein)

MLD für energieoptimale Routen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Energieoptimale Routen

Anfragetypen (analog zu Zeitabhängigkeit):

SoC Query: Gegeben Start s, Ziel t und initialen Ladezustand b_s finde zulässige s-t-Pfad mit maximalem SoC an t

Berechnung mit angepasstem Dijkstra

Energieoptimale Routen

Anfragetypen (analog zu Zeitabhängigkeit):

SoC Query: Gegeben Start s, Ziel t und initialen Ladezustand b_s finde zulässige s-t-Pfad mit maximalem SoC an t

Berechnung mit angepasstem Dijkstra

Profilsuche: Für Start s und Ziel t, finde zulässige s-t-Pfade mit maximalem SoC an t für *alle* $b_s \in [0, M]$

Berechnung mit Hilfe von:

- SoC-Funktionen
- Geeigneten Link/Merge-Operationen

Energieoptimale Routen

Ziel: Gegeben Start s, Ziel t und initialen Ladezustand b_s
finde zulässigen s-t-Pfad mit maximalem SoC an t

Besonderheiten: Energieverbrauch hängt von vielen Parametern ab

- Temperatur, Wetterbedingungen, Beladung, ...
- Abhängig von Fahrzeugtyp, Fahrstil, ...
- Verkehrslage, ...

$\Rightarrow$ Schnelle Customization wichtig

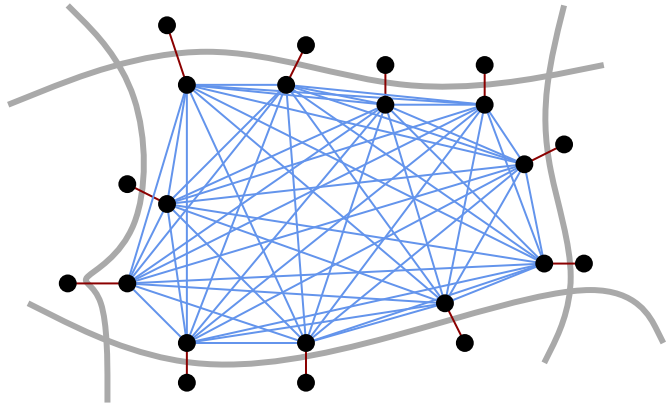
MLD für energieoptimale Routen

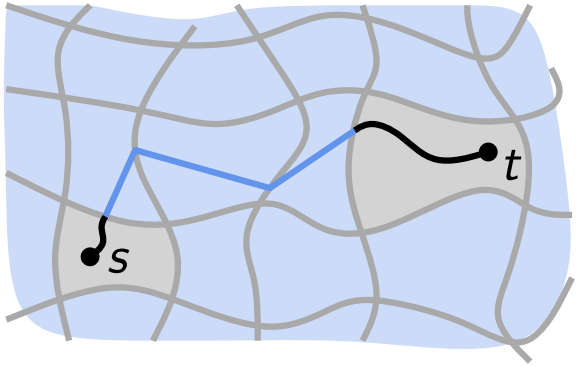
Idee MLD:

- Zerteile den Graphen in mehrere Zellen
- Berechne Distanzen zwischen Randknoten *in jeder Zelle*

Overlay Graph:

- Randknoten
- Cliquen in jeder Zelle
- Schnittkanten





Suchgraph:

- Start- und Zielzelle...
- ...plus Overlaygraph.
- (bidirektionaler) Dijkstra

MLD für energieoptimale Routen

1: Metrik-unabhängige Vorberechnung

- Metrikunabhängig, keine Anpassung nötig

2: Metrik-abhängige Vorberechnung (Customization)

- Cliquenkanten sind Profile
- Lokale Profilsuchen
- Anpassung der Datenstrukturen

3: Queries

- Knotenpotentiale für Stoppkriterium
- Varianten:
  - Unidirektionale Query
  - Bidirektionale Query mit Rückwärts-Profilsuche
  - Bidirektionale Query mit Rückwärtssuche auf Schranken
    (ähnlich wie bei zeitabhängigen Techniken)

Experimente

	CUSTOMIZING		QUERIES	
Algorithm	space [B/n]	time [s]	vertex scans	time [ms]
LC	—	—	4 471 230	709.6
LC (stop. cr.)	0.0	5.20	3 001 014	486.6
Dijkstra π_{v*}	4.0	3.91	2 361 997	288.0
Dijkstra π_h	4.0	0.69	2 359 140	380.6
Prof. π_h	4.0	0.70	2 904 764	741.2
MLD (LC)	10.5	4.42	3 676	2.7
MLD Uni π_h	14.5	5.12	2 410	1.9
MLD BPE π_h	14.5	5.12	2 266	1.4
MLD BDB π_h	14.5	5.12	2 917	1.1

für EV mit 85kWh Akku (ca. 400 km Reichweite)

Ladestopps

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Problemstellung

Bisher:

SoC Query: Gegeben Start s, Ziel t und initialen Ladezustand b_s finde zulässige s-t-Pfad mit maximalem SoC an t

Immer noch sinnvoll wenn wir Ladestopps erlauben?

Problemstellung

Bisher:

SoC Query: Gegeben Start s, Ziel t und initialen Ladezustand b_s finde zulässige s-t-Pfad mit maximalem SoC an t

Immer noch sinnvoll wenn wir Ladestopps erlauben?

- Finde Ladestation in der Nähe vom Ziel
- Route (beliebig?) zu dieser Station, lade voll auf, fahre zum Ziel

Problemstellung

Bisher:

SoC Query: Gegeben Start s, Ziel t und initialen Ladezustand b_s
finde zulässige s-t-Pfad mit maximalem SoC an t

Immer noch sinnvoll wenn wir Ladestopps erlauben?

- Finde Ladestation in der Nähe vom Ziel
- Route (beliebig?) zu dieser Station, lade voll auf, fahre zum Ziel

Stattdessen:

SoC Query: Gegeben Start s, Ziel t und initialen Ladezustand b_s
finde zulässigen s-t-Pfad der Energieverbrauch
+ gesamte Menge geladener Energie minimiert

Analog: Maximiere $b_t - r_t$, wobei

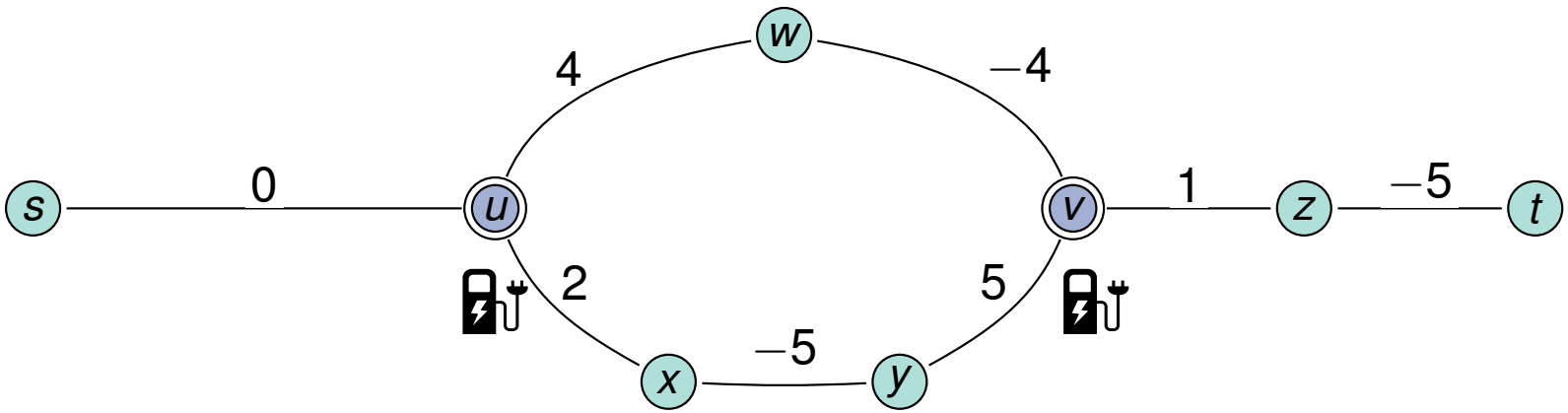
- b_t: SoC an t
- r_t: Geladene Energie um t zu erreichen

Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$



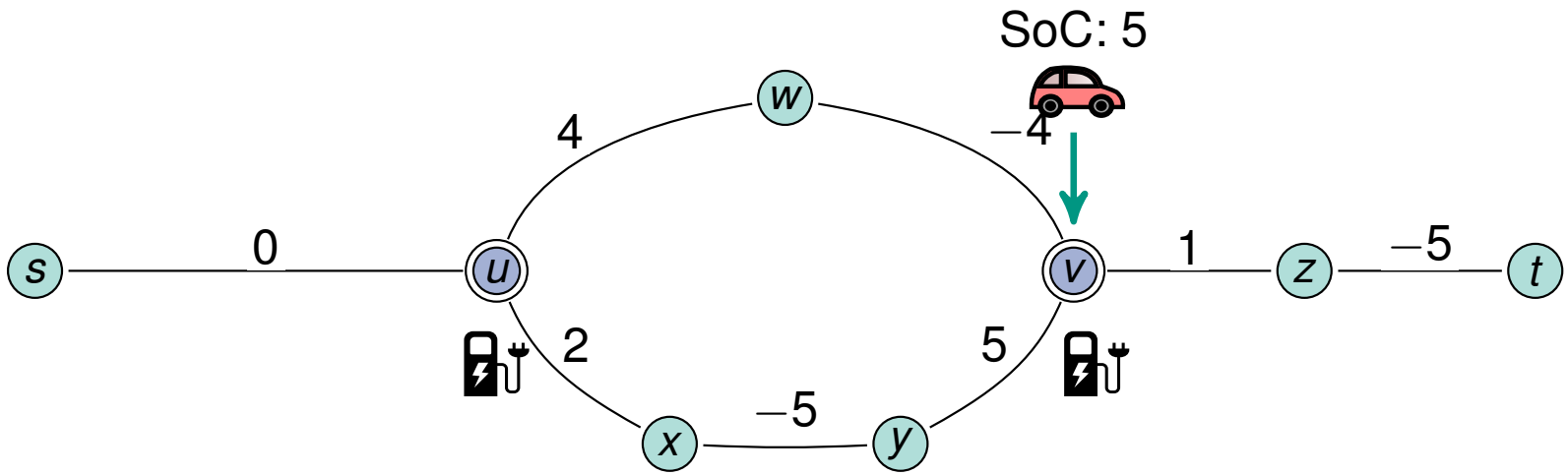
Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$

$b_v - r_v = 5 - 0 = 5$



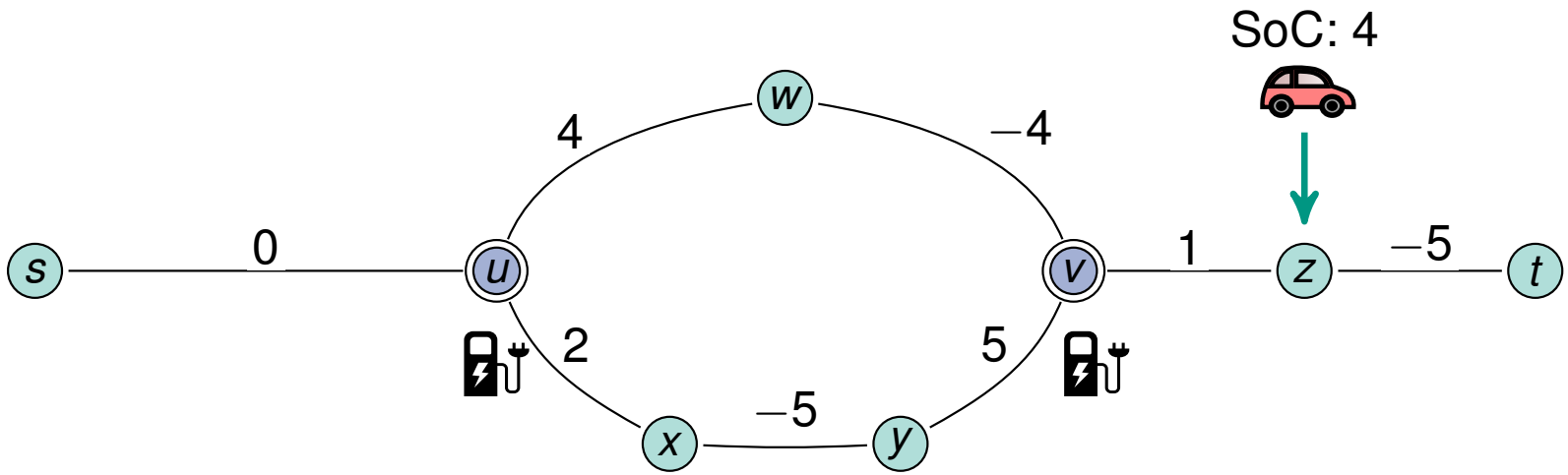
Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$

$b_z - r_z = 4 - 0 = 4$



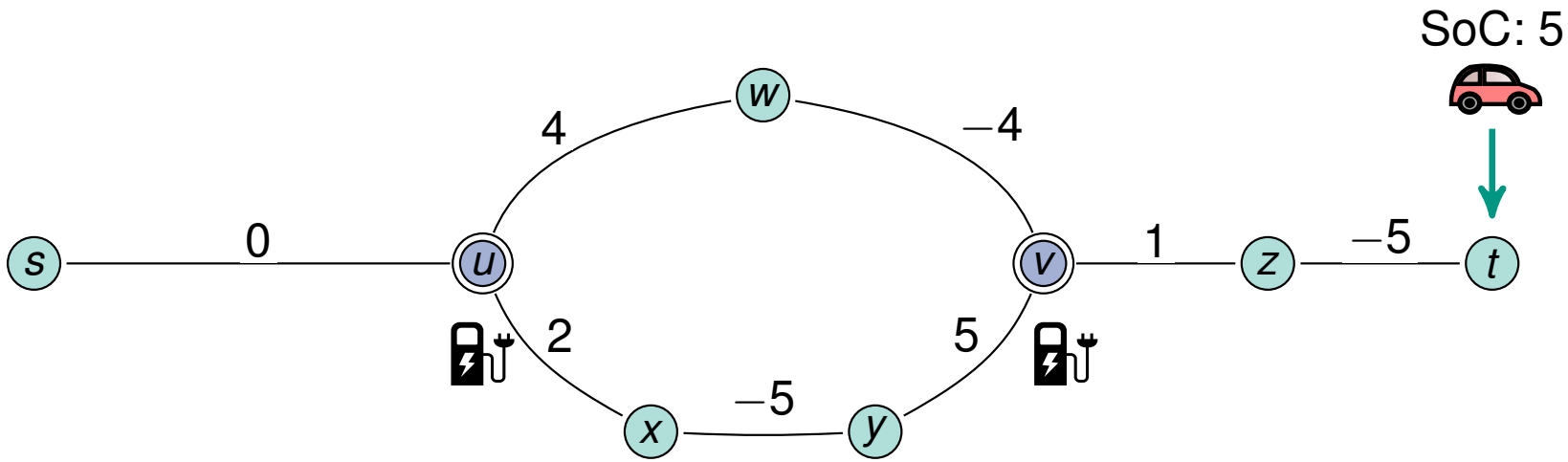
Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$

$b_t - r_t = 5 - 0 = 5$



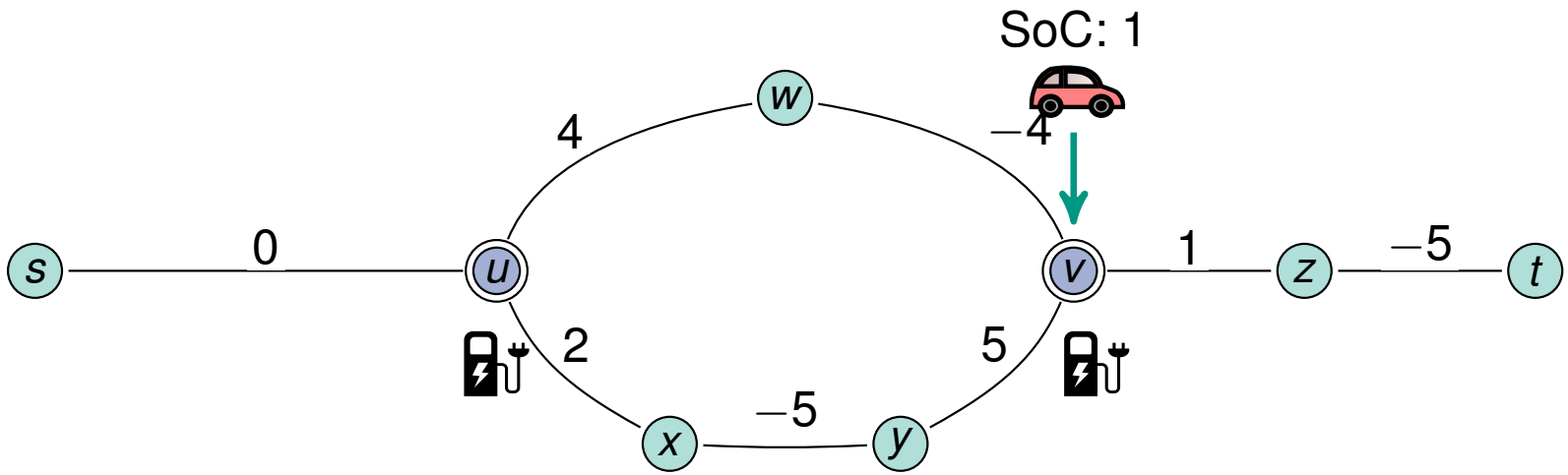
Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$

$b_v - r_v = 1 - 0 = 1$



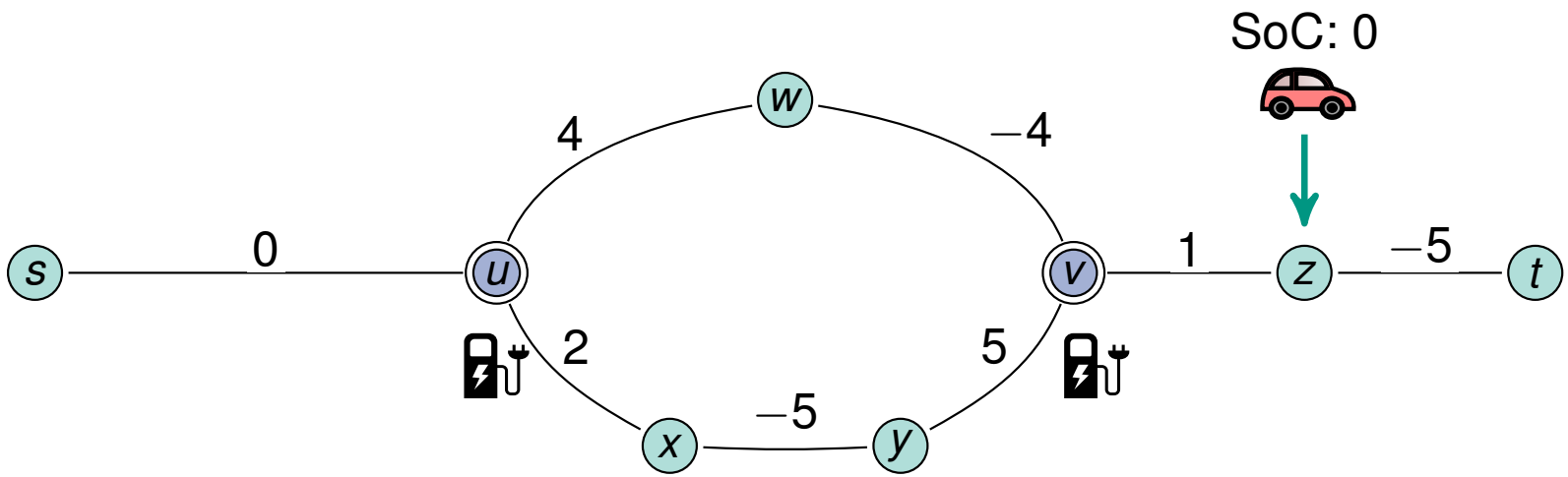
Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$

$$b_z - r_z = 0 - 0 = 0$$



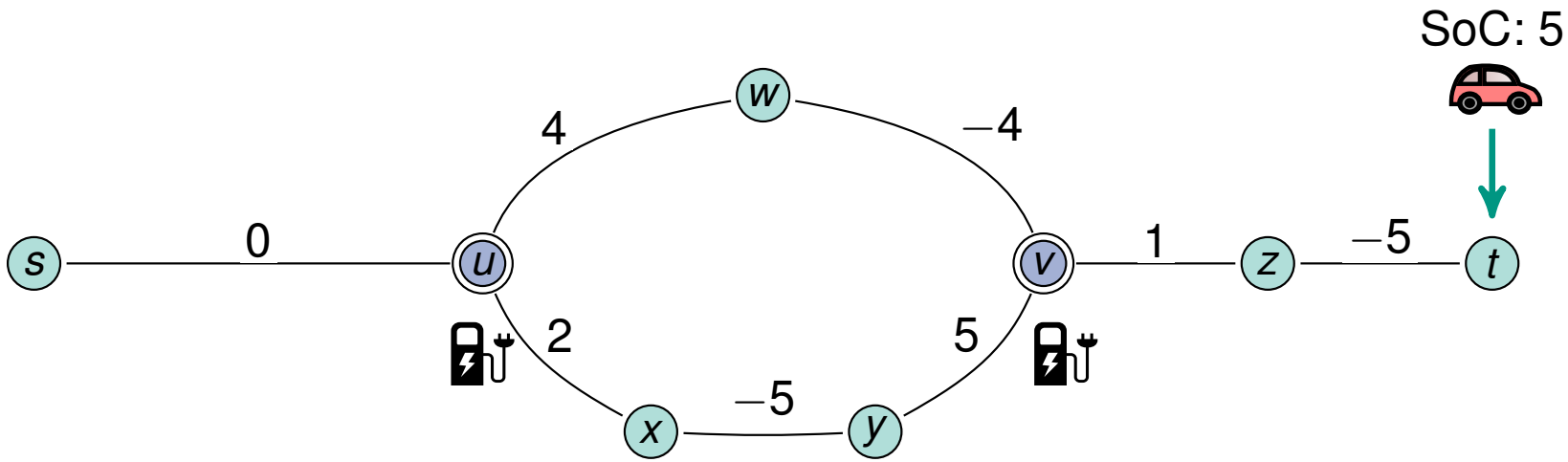
Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$

$b_t - r_t = 5 - 0 = 5$



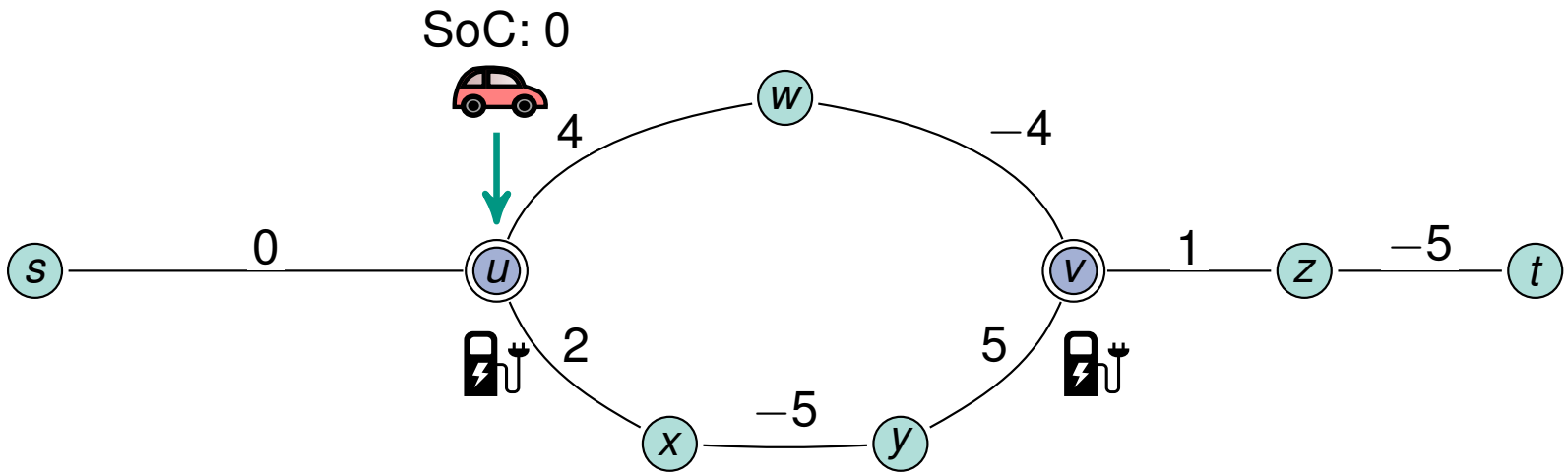
Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$

$b_u - r_u = 0 - 0 = 0$



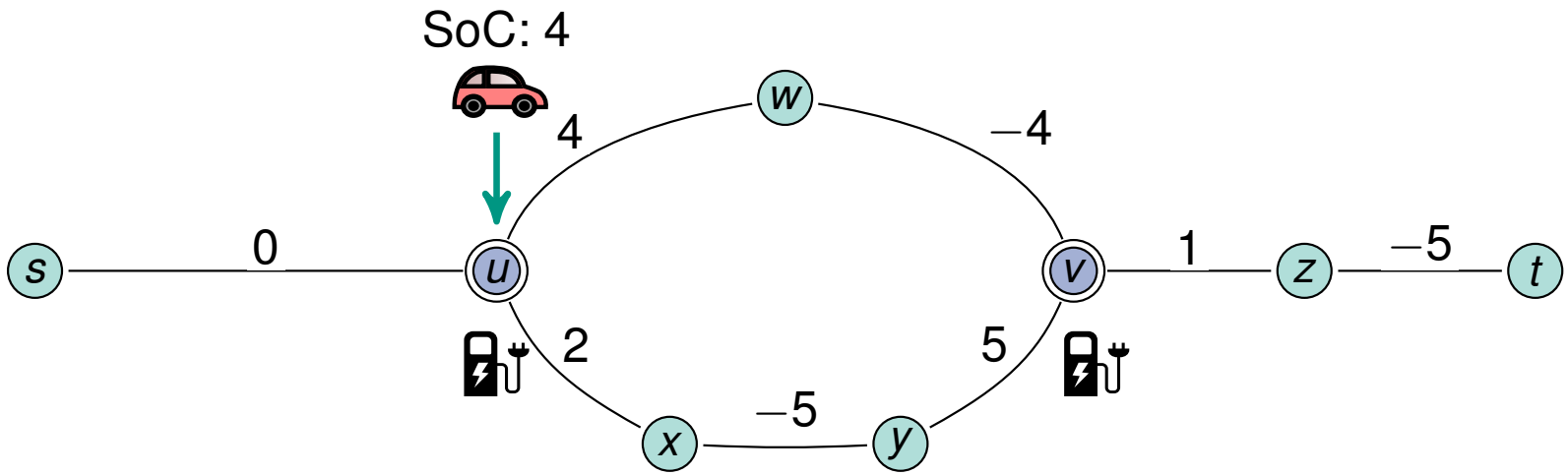
Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$

$b_u - r_u = 4 - 4 = 0$



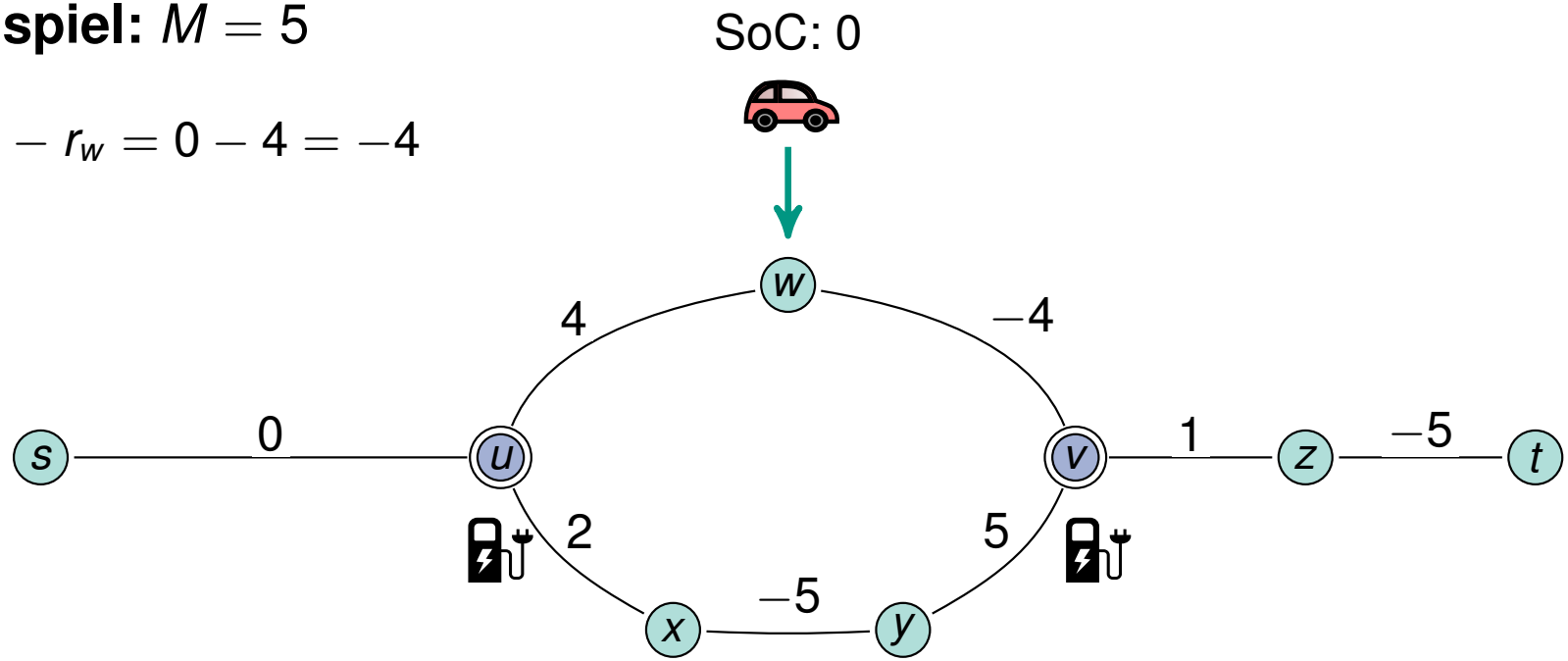
Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$

$b_w - r_w = 0 - 4 = -4$



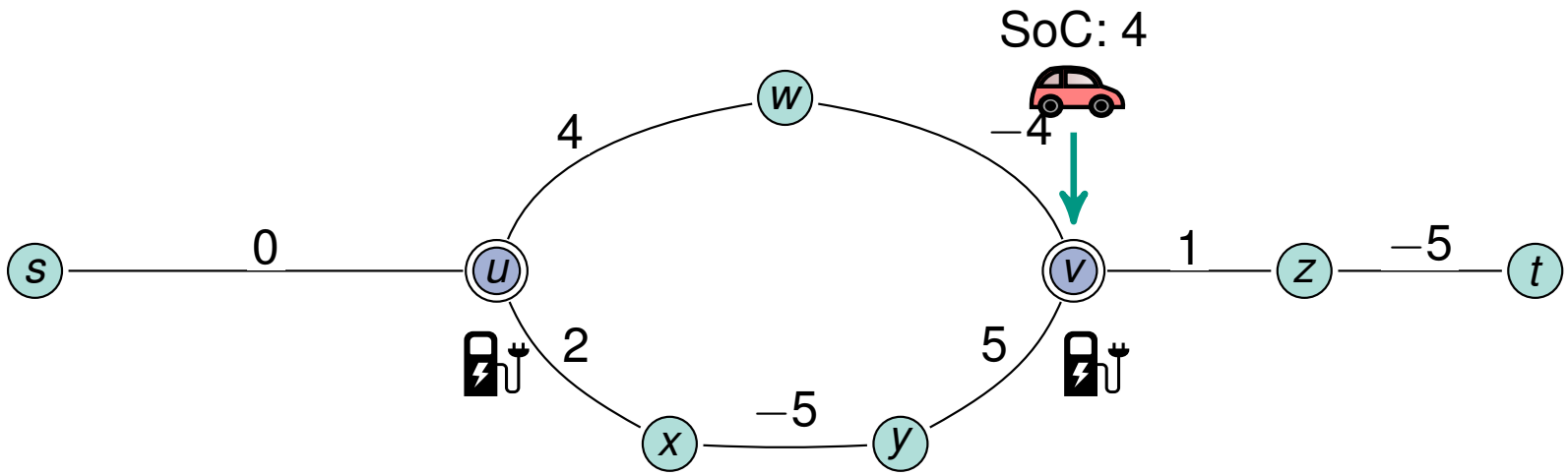
Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$

$b_v - r_v = 4 - 4 = 0$



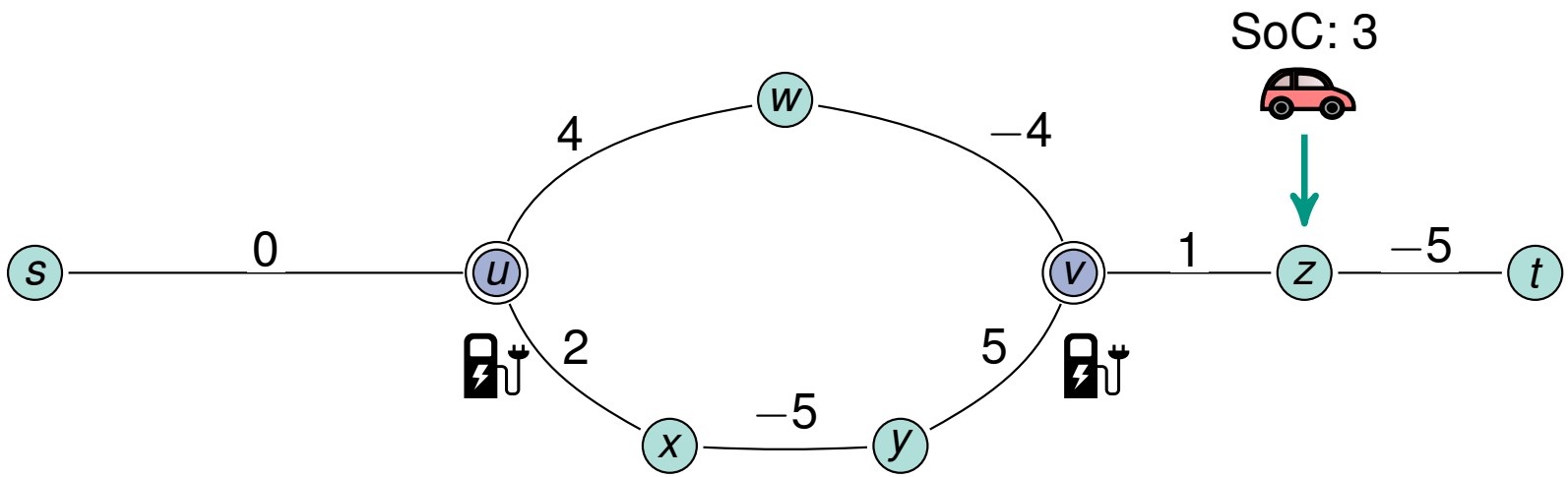
Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$

$$b_z - r_z = 3 - 4 = -1$$



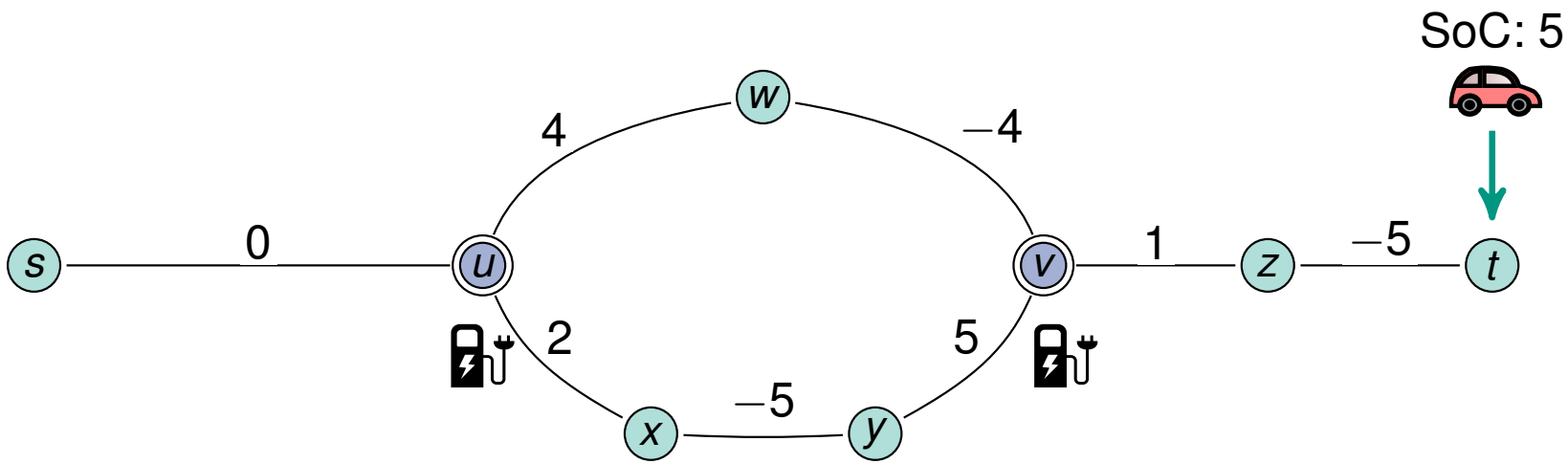
Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$

$b_t - r_t = 5 - 4 = 1$



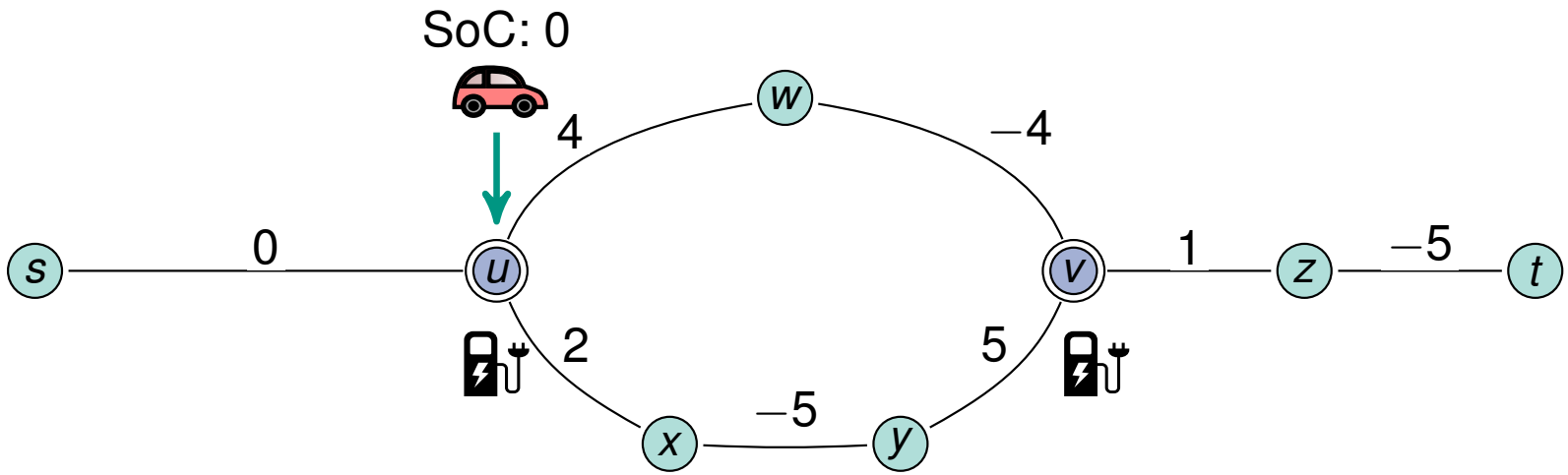
Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$

$b_u - r_u = 0 - 0 = 0$



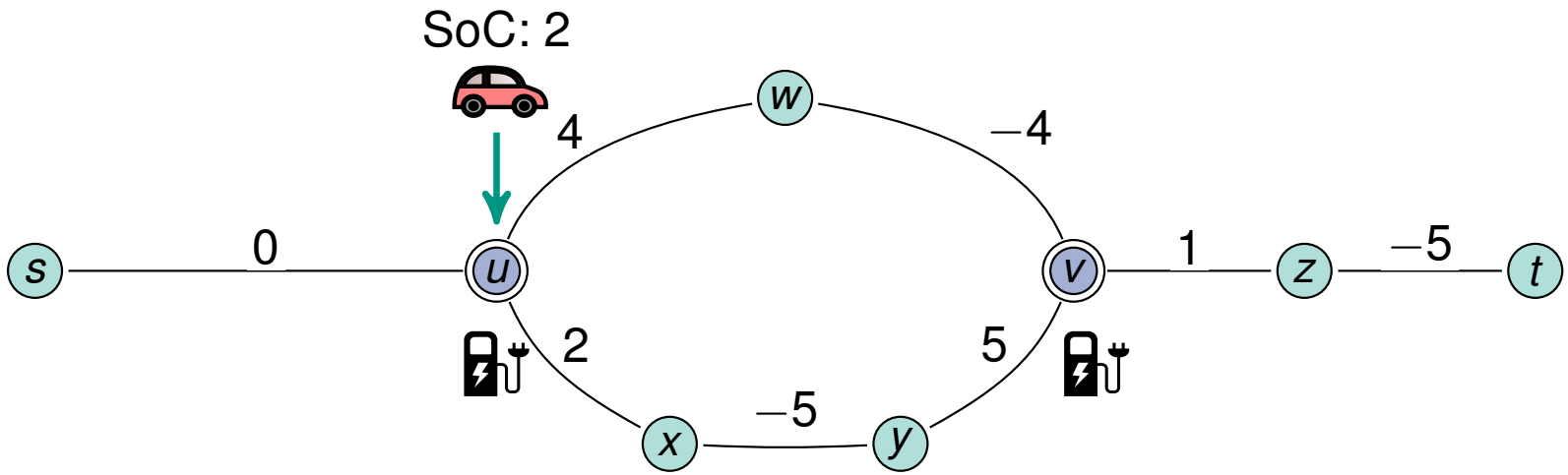
Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$

$b_u - r_u = 2 - 2 = 0$



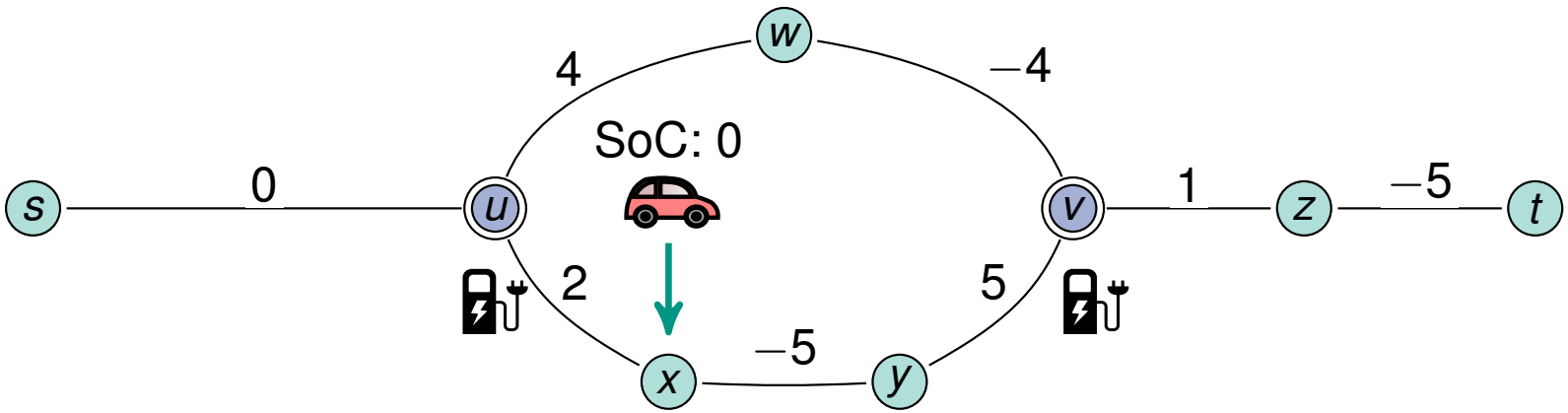
Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$

$$b_x - r_x = 0 - 2 = -2$$



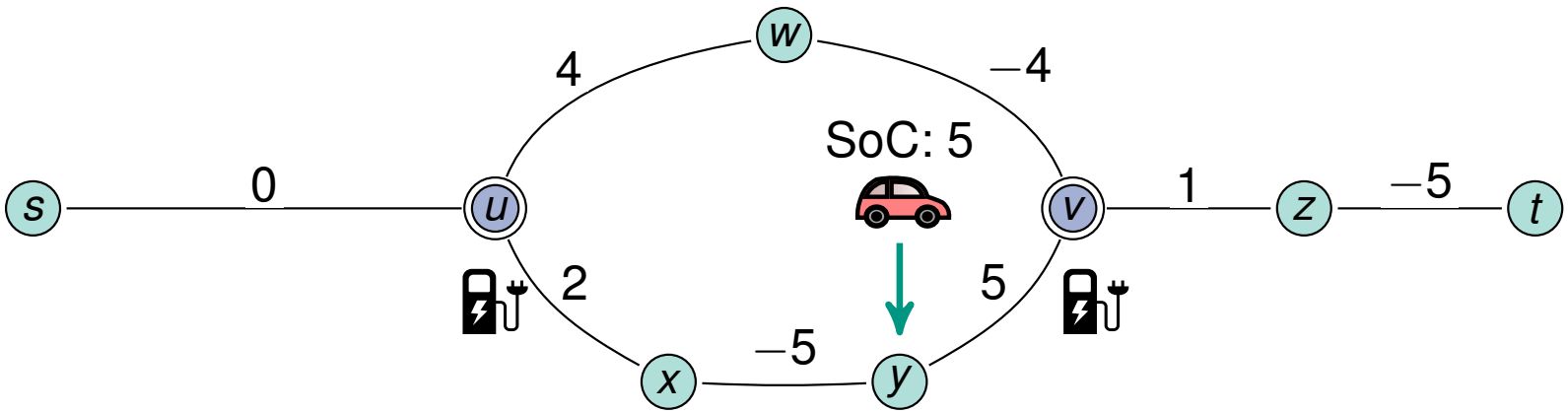
Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$

$$b_y - r_y = 5 - 2 = 3$$



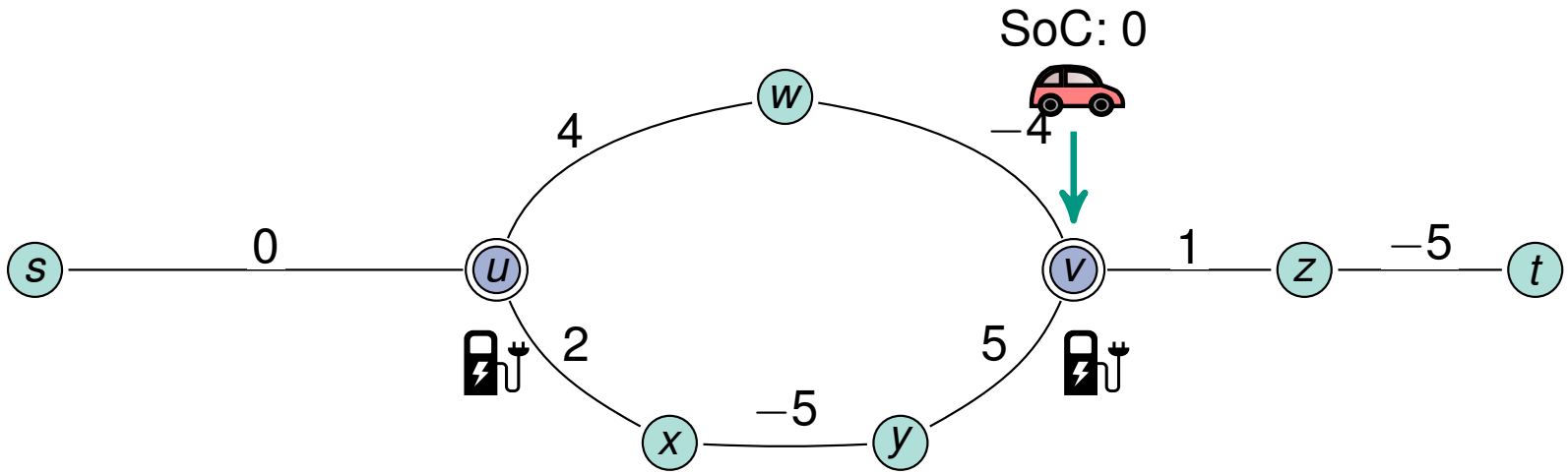
Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$

$b_v - r_v = 0 - 2 = -2$



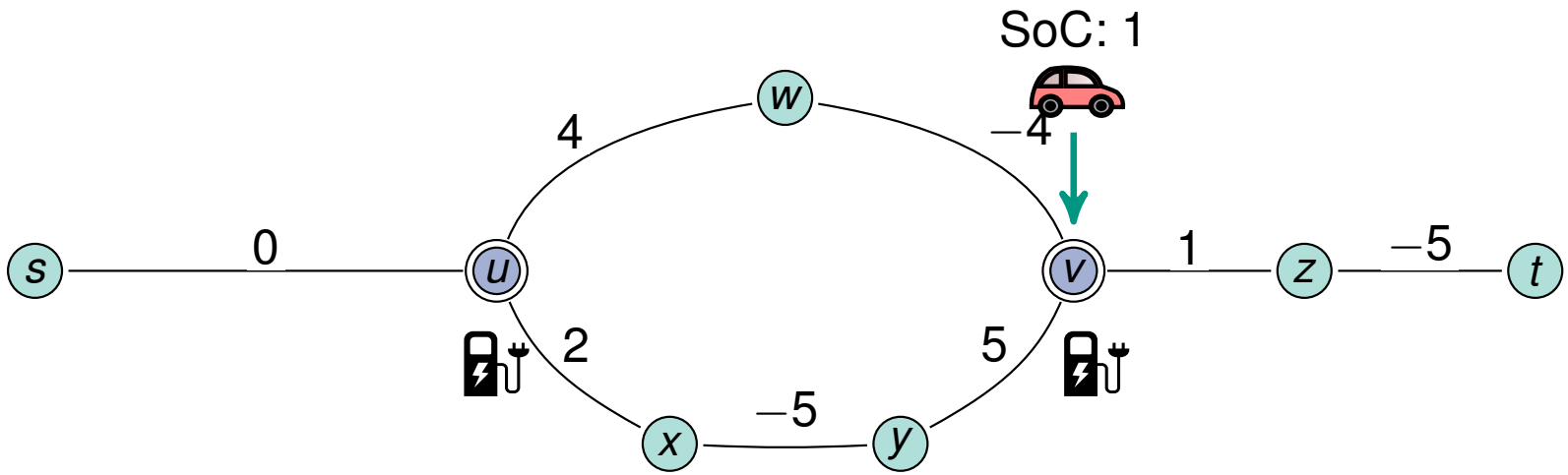
Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$

$b_v - r_v = 1 - 3 = -2$



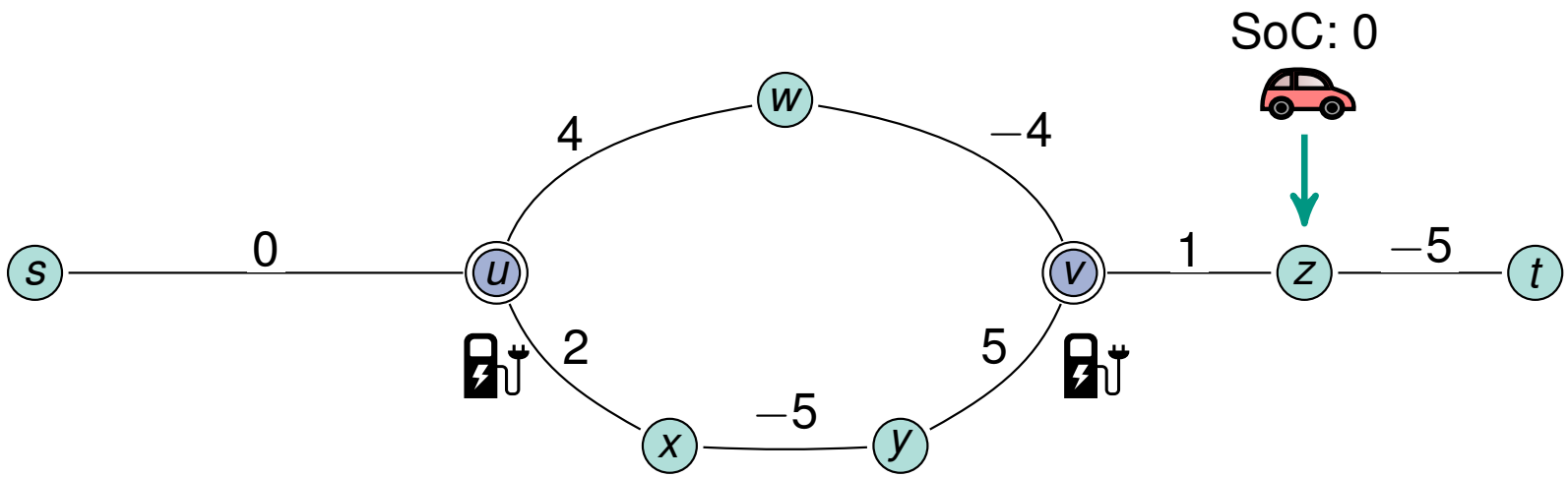
Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$

$$b_z - r_z = 0 - 3 = -3$$



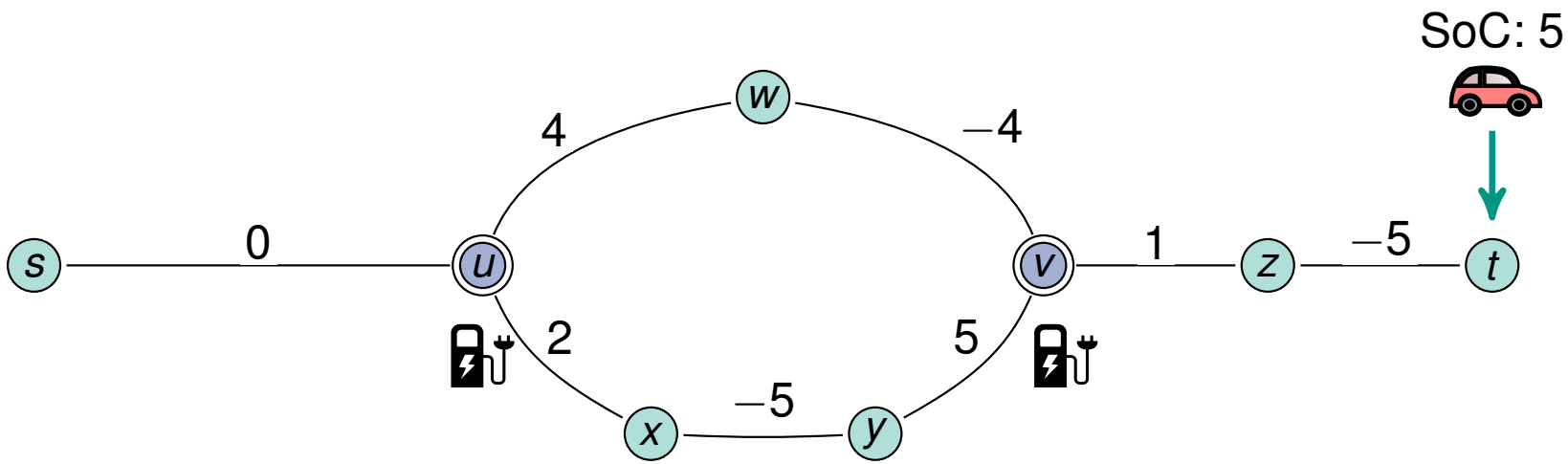
Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$

$b_t - r_t = 5 - 3 = 2$

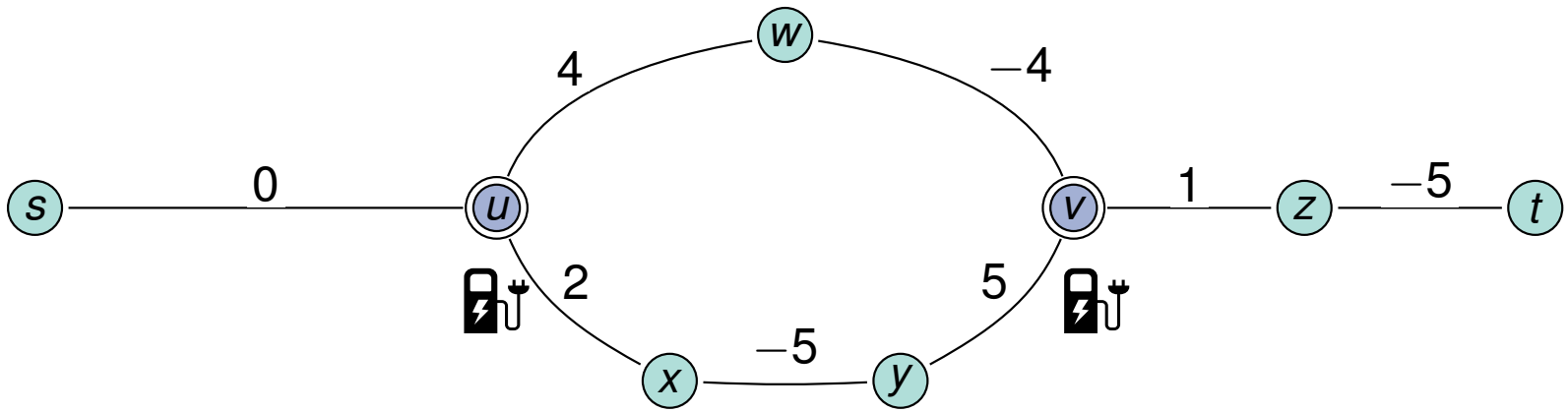


Planung von Ladestopps

Zwei Probleme:

- Volles Aufladen ist ggf. nicht optimal
- Subpfade von optimalen Pfaden ggf. selbst nicht optimal

Beispiel: $M = 5$



Für Zielknoten v**:** u–v-Pfad über w immer die optimal Wahl
Für Zielknoten t**:** u–v-Pfad über x und y ggf. besser

Polynomialzeitalgorithmus

Beobachtung: Teilpfade zwischen Ladestationen (oder von Station zum Ziel) müssen energieoptimal sein

- Berechne alle energieoptimalen Pfade zwischen Ladestationen (+Ziel)
  (linear pro Paar von Stationen)
- Berechne für jeden dieser Pfade den minimal nötigen SoC zum Befahren
- Mehr als das muss nicht geladen werden
  (man kann an der nächsten Station weiter laden bzw. ist am Ziel)

$\Rightarrow$ pro Ladestation polynomiell viele “Abfahrt-SoCs”

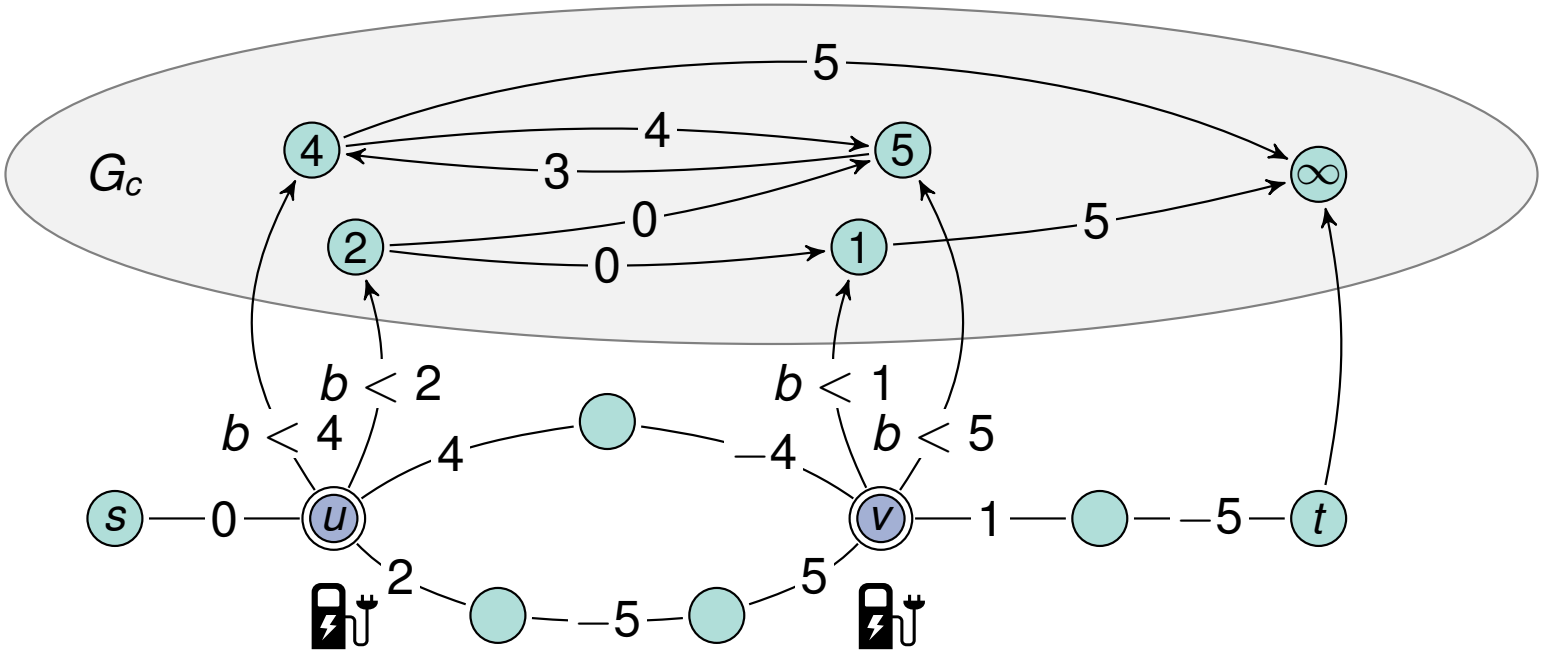
$\Rightarrow$ konstruiere polynomiellen Suchgraph:

- Ein Knoten pro Station und Abfahrt-SoC
- Speichere Ankunfts-SoC an der Kante

Suchgraph

Für jeden Pfad im Profil zwischen Ladestationen (+Ziel):

- Berechne Ankunfts-SoC (Wert an der Kante)
- Füge Ladeknoten hinzu (Mindest-Ladewert im Knoten)

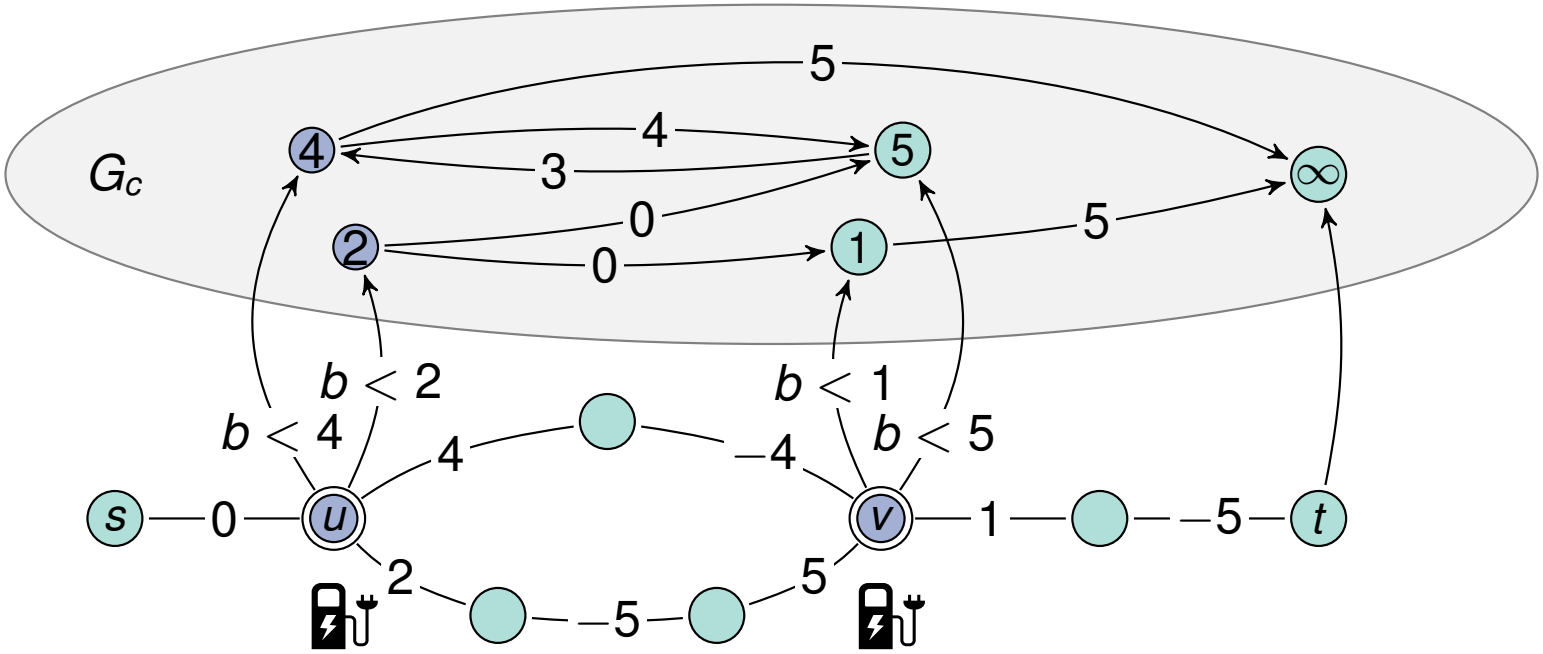


$\mathcal{O}(|V|)$ Pfade pro Ladestationspaar $\Rightarrow$ Suchgraph hat poly. Größe

Suchgraph

Für jeden Pfad im Profil zwischen Ladestationen (+Ziel):

- Berechne Ankunfts-SoC (Wert an der Kante)
- Füge Ladeknoten hinzu (Mindest-Ladewert im Knoten)

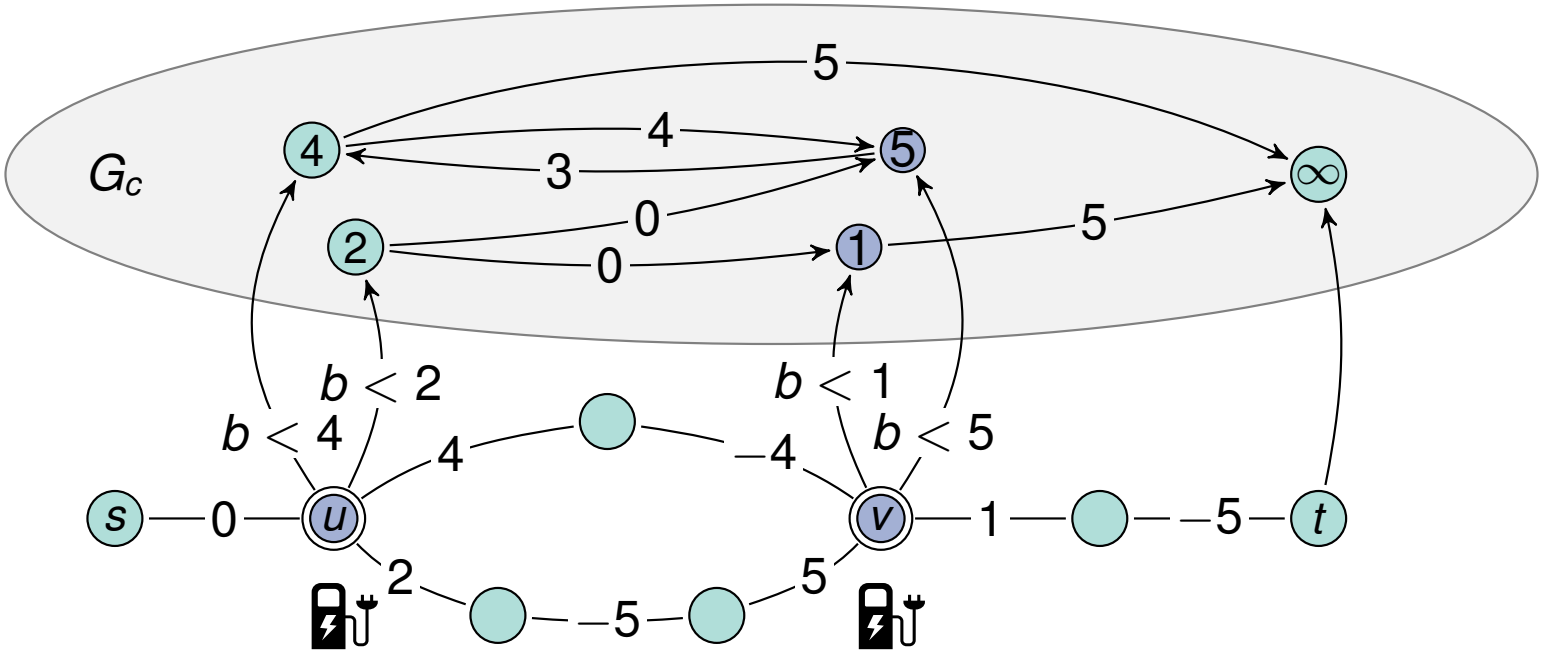


$\mathcal{O}(|V|)$ Pfade pro Ladestationspaar $\Rightarrow$ Suchgraph hat poly. Größe

Suchgraph

Für jeden Pfad im Profil zwischen Ladestationen (+Ziel):

- Berechne Ankunfts-SoC (Wert an der Kante)
- Füge Ladeknoten hinzu (Mindest-Ladewert im Knoten)

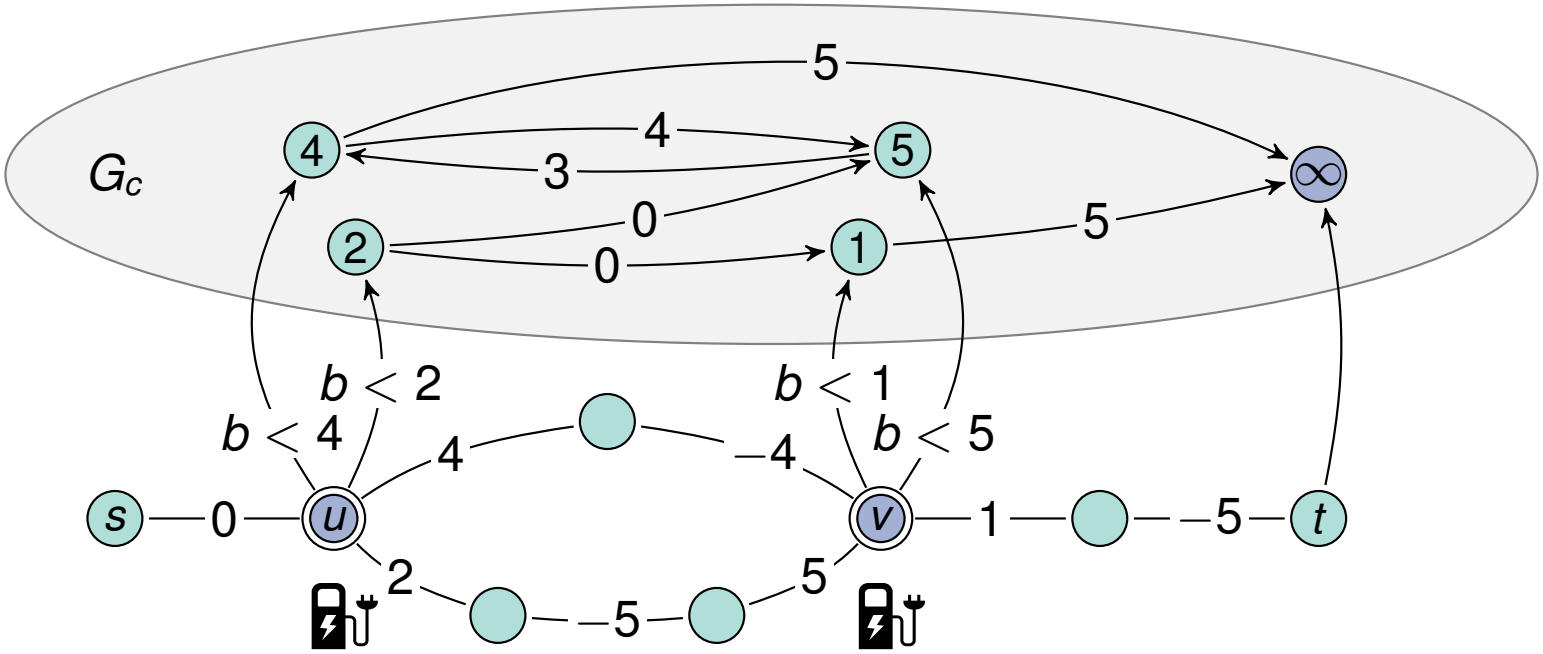


$\mathcal{O}(|V|)$ Pfade pro Ladestationspaar $\Rightarrow$ Suchgraph hat poly. Größe

Suchgraph

Für jeden Pfad im Profil zwischen Ladestationen (+Ziel):

- Berechne Ankunfts-SoC (Wert an der Kante)
- Füge Ladeknoten hinzu (Mindest-Ladewert im Knoten)

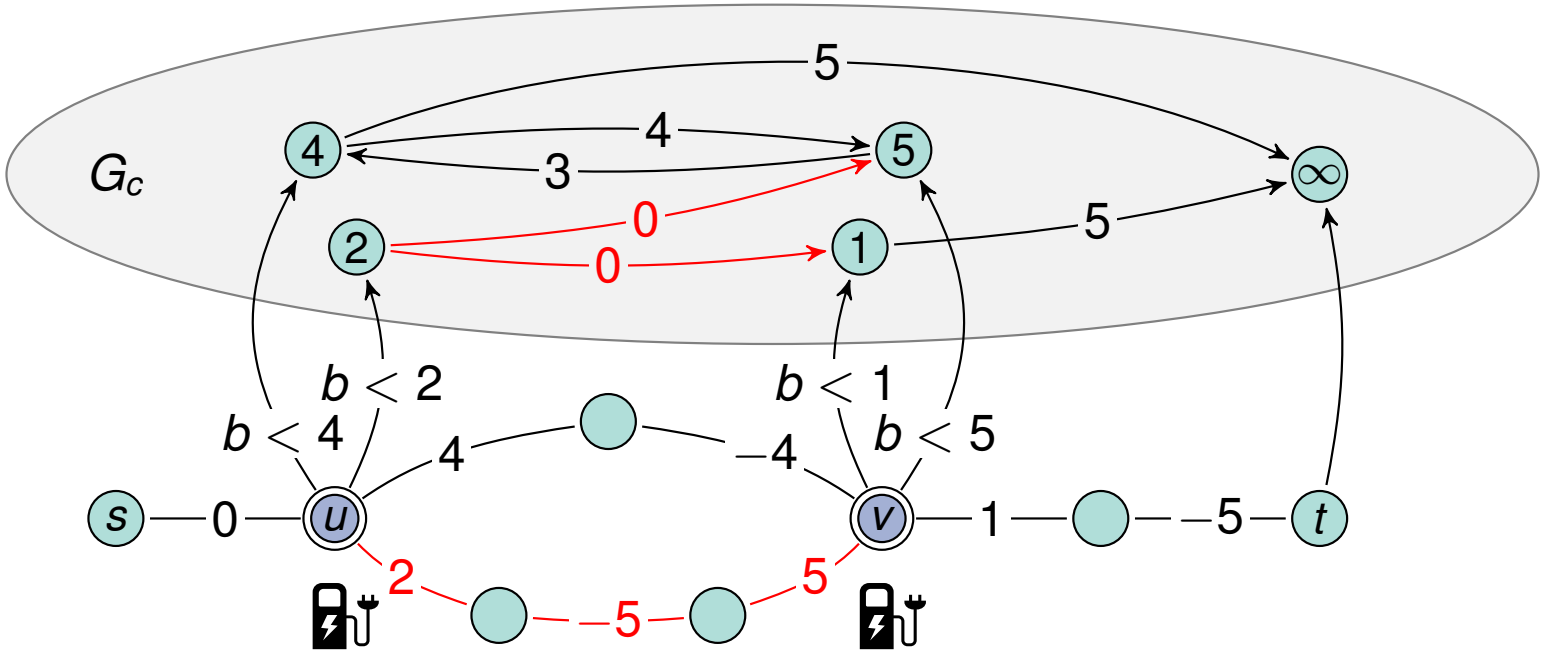


$\mathcal{O}(|V|)$ Pfade pro Ladestationspaar $\Rightarrow$ Suchgraph hat poly. Größe

Suchgraph

Für jeden Pfad im Profil zwischen Ladestationen (+Ziel):

- Berechne Ankunfts-SoC (Wert an der Kante)
- Füge Ladeknoten hinzu (Mindest-Ladewert im Knoten)

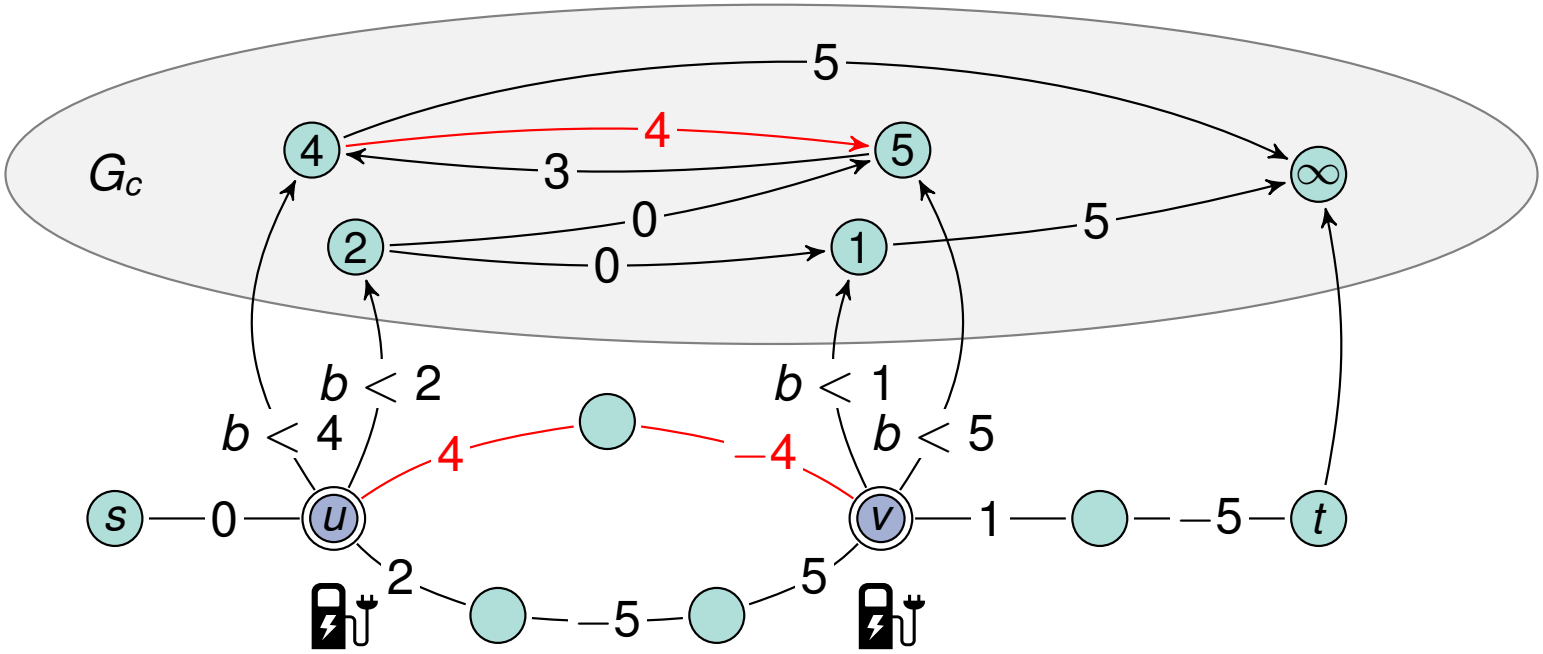


$\mathcal{O}(|V|)$ Pfade pro Ladestationspaar $\Rightarrow$ Suchgraph hat poly. Größe

Suchgraph

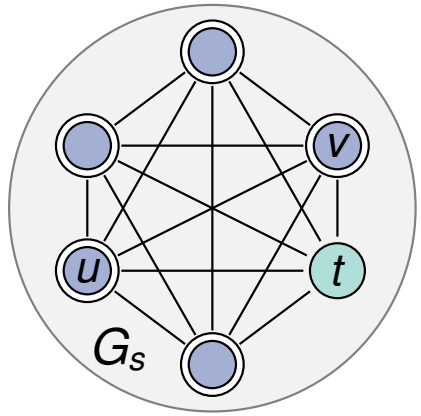
Für jeden Pfad im Profil zwischen Ladestationen (+Ziel):

- Berechne Ankunfts-SoC (Wert an der Kante)
- Füge Ladeknoten hinzu (Mindest-Ladewert im Knoten)



$\mathcal{O}(|V|)$ Pfade pro Ladestationspaar $\Rightarrow$ Suchgraph hat poly. Größe

Heuristische Implementierung

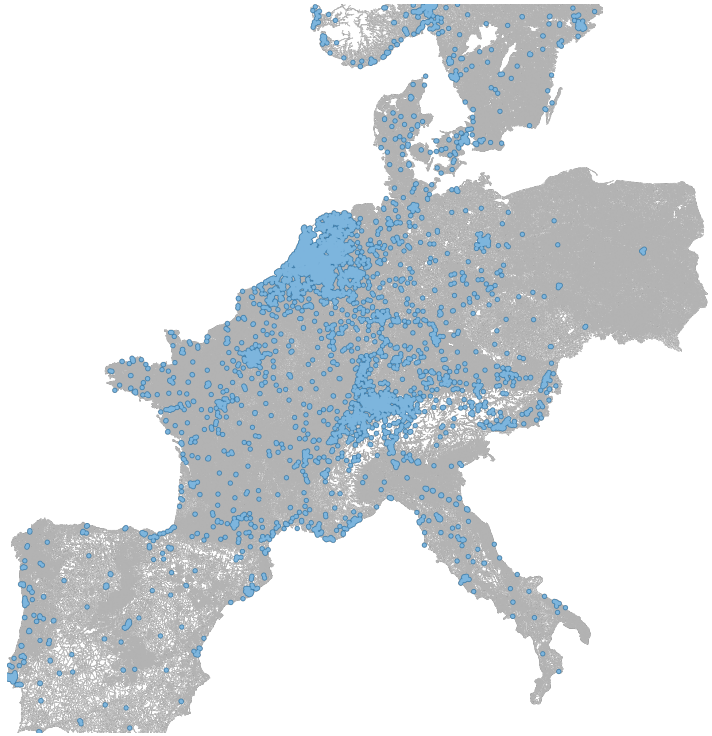


- Suchgraph G_s:
  - Knoten: Alle Ladestationen und Zielknoten t
  - Clique
  - An den Kanten: SoC-Profile
- Traversieren von Kanten zwischen Stationen u und v:
  - Lade genug Energie, um Verbrauch auf s–v Pfad zu minimieren
  - Nicht beweisbar optimal, aber (fast) immer optimal in der Praxis
- Speedup-Techniken:
  - Contraction Hierarchies (CH): Partielle CH; kontrahiere keine Ladestationen
  - A* Suche: Potentiale basierend Rückwärtssuche (ähnlich wie bei multikriterieller Suche)

Experimente

Input:

- Straßennetzwerk Europa,
  (ähnlich wie DIMACS)
- 22.2M Knoten, 51.1M Kanten
- Verbrauchsdaten
  aus PHEM Emissionsmodel
- Kantensteigung aus SRTM
  (Höhendaten)
- Ladestationen von ChargeMap:
  13 810 in Europa



Performance

Fahrzeugtypen:

- Peugeot iOn, 16 kWh (100-150 km)
- Künstliches Modell, 85 kWh (400-500 km, ähnl. wie Tesla)

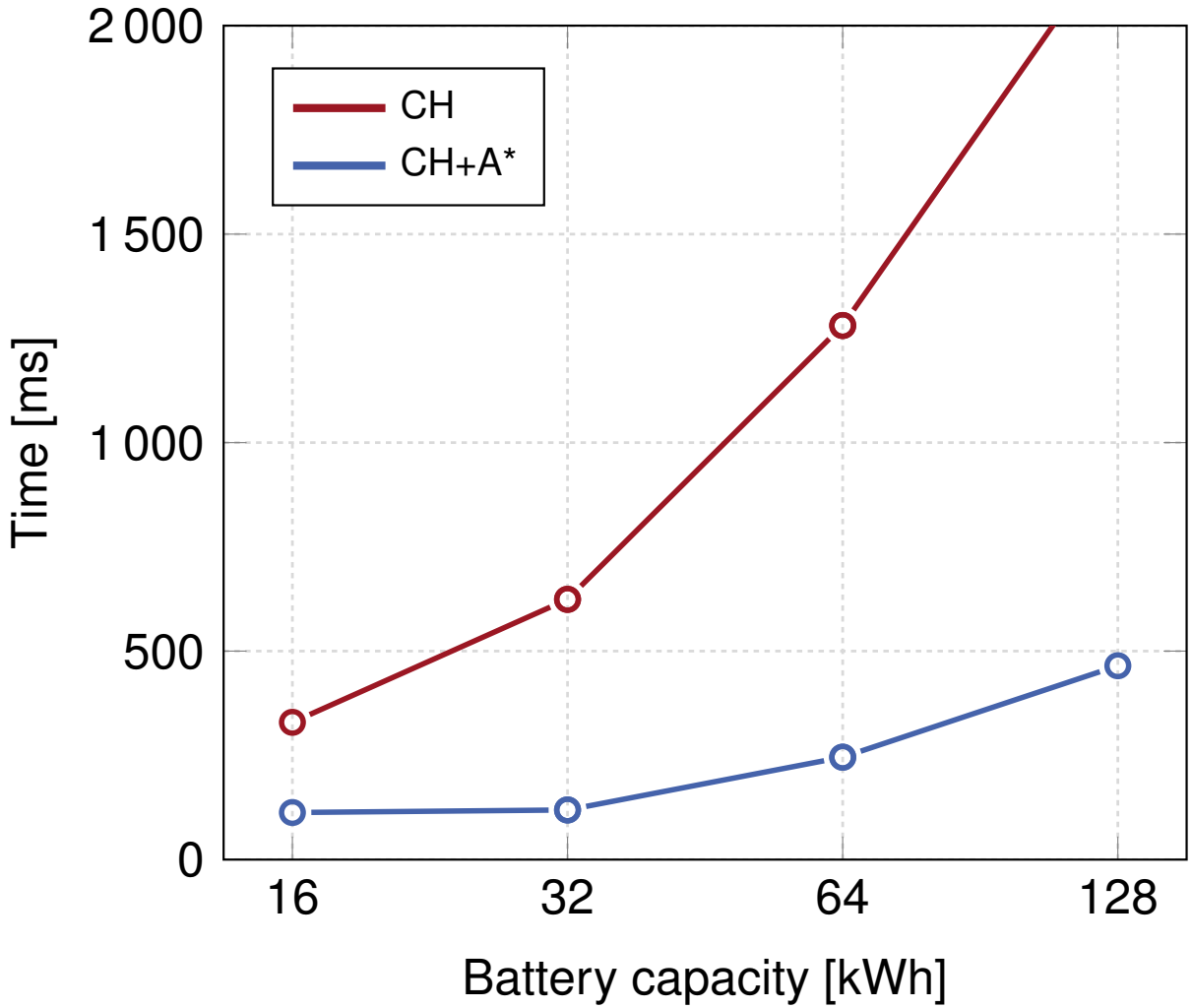
Techniques			Peugeot iOn			Artificial		
G_S	CH	A*	Prepr. [s]	# V. Sc.	Q. [ms]	Prepr. [s]	# V. Sc.	Q. [ms]
○	○	○	—	8 895k	20 161	—	11 034k	32 929
●	○	○	1 487	760k	710	15 062	7 754k	6 286
●	●	○	2 860	8k	310	3 246	20k	1 282
●	●	●	2 860	4k	128	3 246	10k	298

Ladestationsverteilungen

Subgraph Deutschland (4.7M vertices, 10.8M edges)

		Prepr.		Queries		
Scenario	$\|S\|$	T. [s]	$\|E_S\|$	# V. Sc.	# E. Sc.	T. [ms]
ChargeMap	1 966	549	539k	5k	126k	4.22
random-0.01	469	487	22k	2k	50k	1.30
random-0.1	4 692	583	2 263k	9k	224k	7.97
random-1.0	46 920	965	227 514k	61k	1 829k	73.46

Batteriegröße



Zusammenfassung

Ergebnisse:

- Energieverbrauch modellieren ist nicht trivial:
  - Negative Kantengewichte
  - Battery Constraints
- SoC Profile haben linear Komplexität in $|V|$
- Energieoptimale Pfade mit Ladestopps in Polynomialzeital
- Praktische Implementierung: < 300 ms auf Europa
- Anpassung von MLD möglich: < 1 ms auf Europa
  (ohne Ladestopps)

Offene Fragen:

- Suchgraph ausdünnen $\rightarrow$ bessere Performance/Skalierbarkeit?
- Integration CCH $\rightarrow$ schnellere Vorberechnung?
- Profilsuche mit Ladestopps?

Literatur I

Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner.
Energy-optimal routes for electric vehicles.
Technical Report 2013-06, Faculty of Informatics, Karlsruhe Institute of Technology, 2013.

Moritz Baum, Jonas Sauer, Dorothea Wagner, and Tobias Zündorf.
Consumption profiles in route planning for electric vehicles: Theory and applications.
In *Proceedings of the 16th International Symposium on Experimental Algorithms (SEA'17)*, volume 75 of *Leibniz International Proceedings in Informatics*, pages 19:1–19:18, 2017.

Jochen Eisner, Stefan Funke, and Sabine Storandt.
Optimal route planning for electric vehicles in large networks.
In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 1108–1113. AAAI Press, August 2011.