Karlsruher Institut für Technologie

Algorithmen für Routenplanung

13. Vorlesung, Sommersemester 2023

Jonas Sauer | 12. Juni 2023

Elektromobilität

Elektrofahrzeuge (EVs):

- Transportmittel der Zukunft
- Emissionsfreie Mobilität

Aber:

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

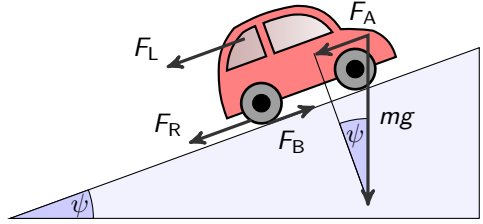
⇒ Berücksichtigung von Energieverbrauch bei der Routenplanung

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

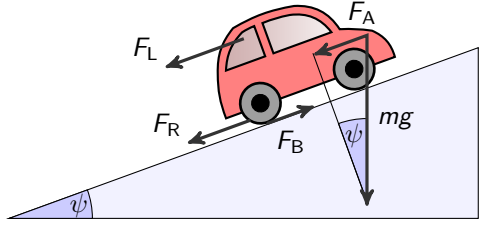
Nötige Leistung: $P = \frac{F \cdot v}{\eta} + P_0$

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

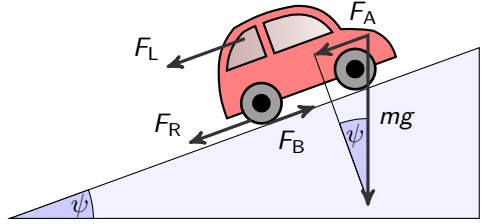
Alle anderen Parameter sind konstant oder vom Fahrzeugzustand abhängig (Masse, Nebenverbraucher)

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

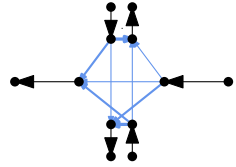


Insgesamt benötigte Kraft: $F = F_R + F_L + F_A + F_B$

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
- Zusätzliche Kanten für Brems-/Beschleunigungskosten

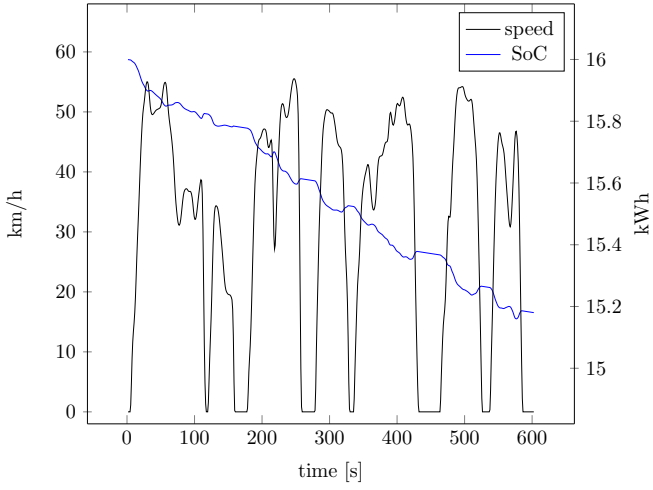


$\Rightarrow$ Verbrauch auf Segment vereinfacht (für realistische Steigung) als

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

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

Alternative Datenquellen



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

Besonderheiten von Elektrofahrzeugen

Rekuperation:

- Verbrauch kann auch negativ werden
  - F_A ist negativ beim Bergabfahren
  - F_B ist negativ beim Bremsen
- Elektromotor fungiert als Generator
- Bei negativem Verbrauch wird Akku aufgeladen (Rekuperation)

Aber: keine negativen Zyklen (physikalische Gesetze)

Akkukapazität (Battery Constraints):

- Akku darf nicht leer laufen
  (Andernfalls Kante nicht benutzbar)
- Kapazität kann nicht überschritten werden
  (Kanten können genutzt werden, aber überschüssige Energie verfällt)

$\Rightarrow$ 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

Dijkstras Algorithmus:

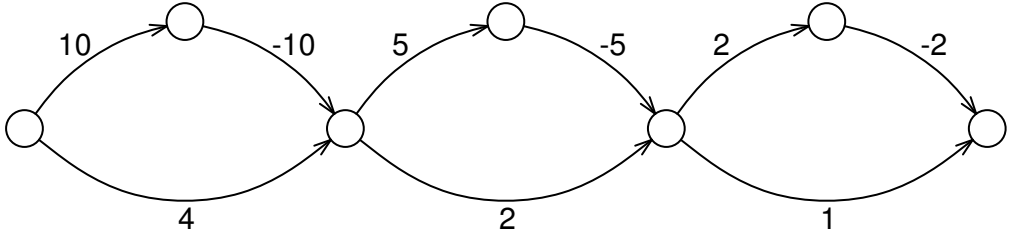
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr

Energieoptimale Routen

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

Dijkstras Algorithmus:

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

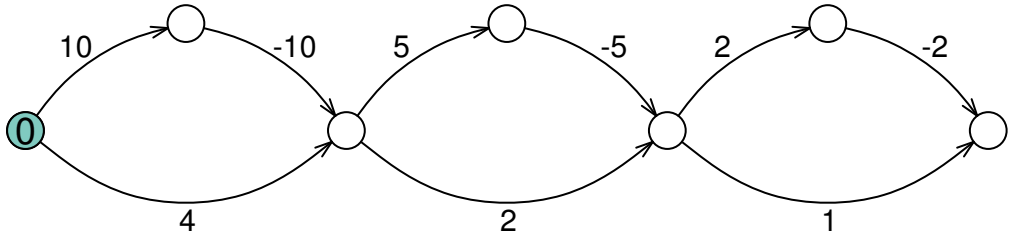


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

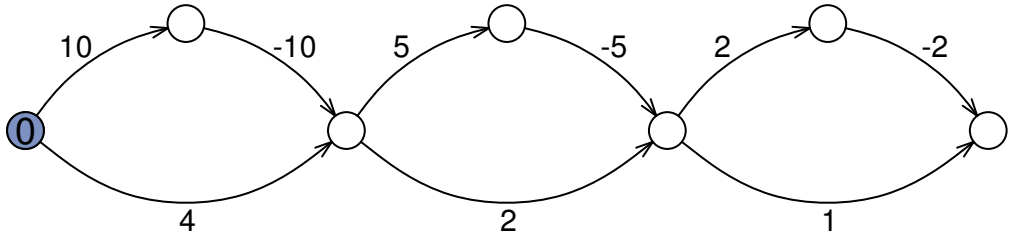


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

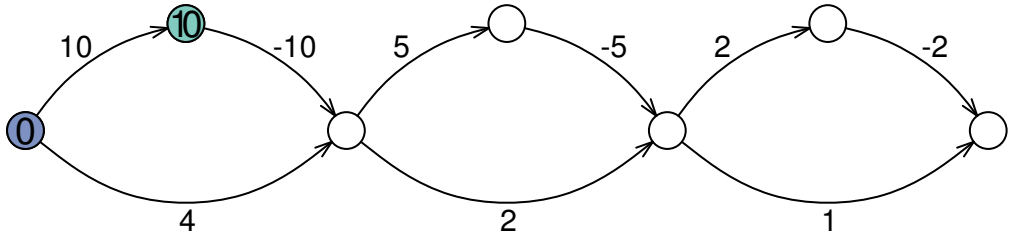


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

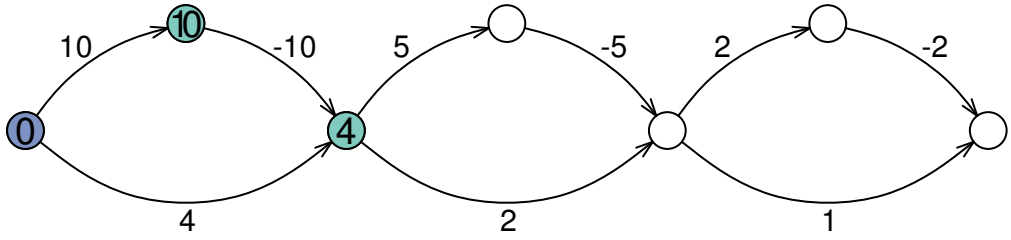


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

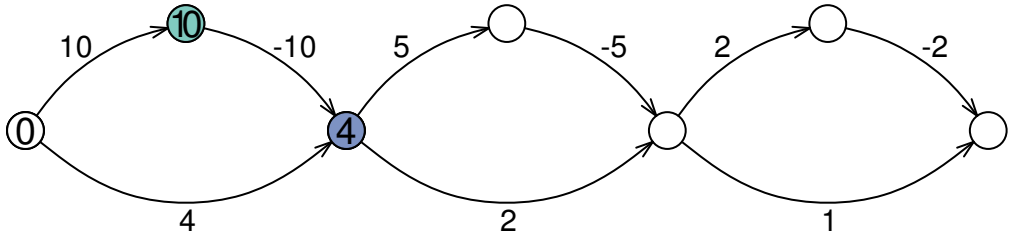


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

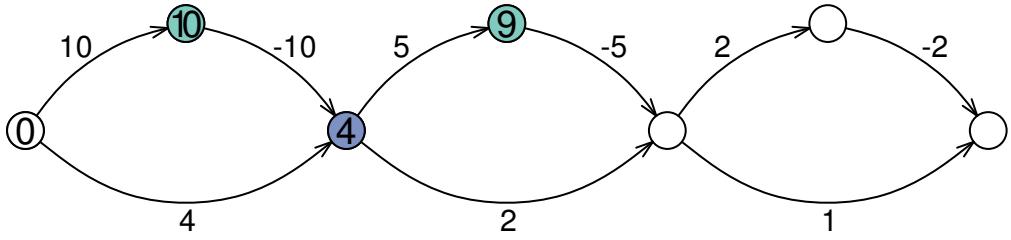


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

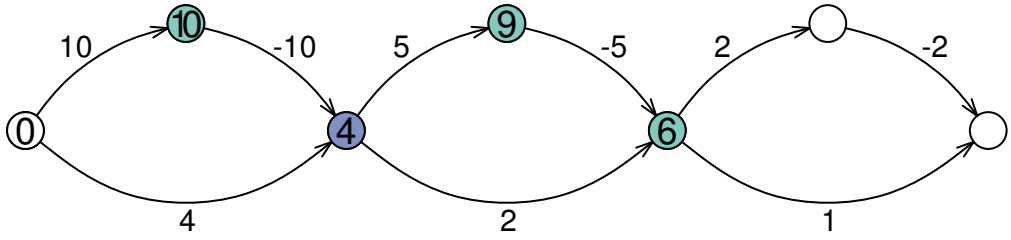


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

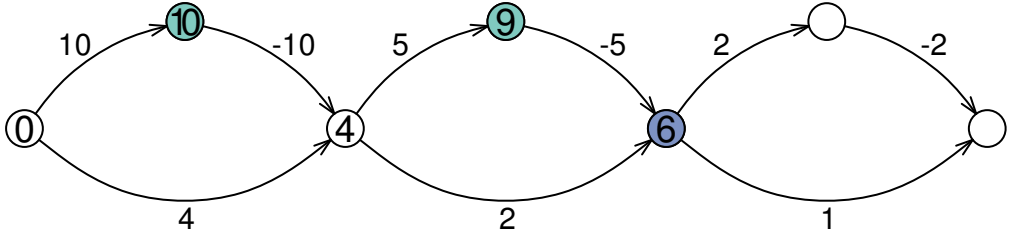


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

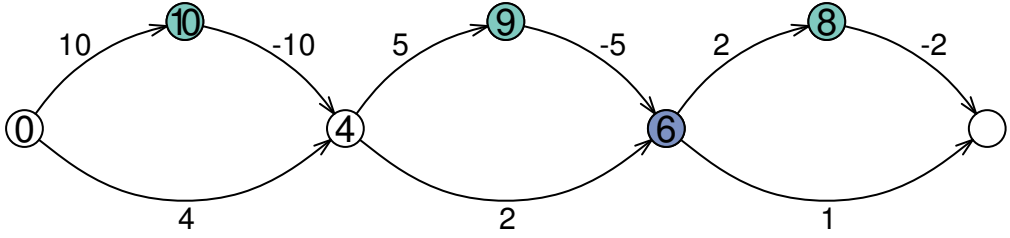


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

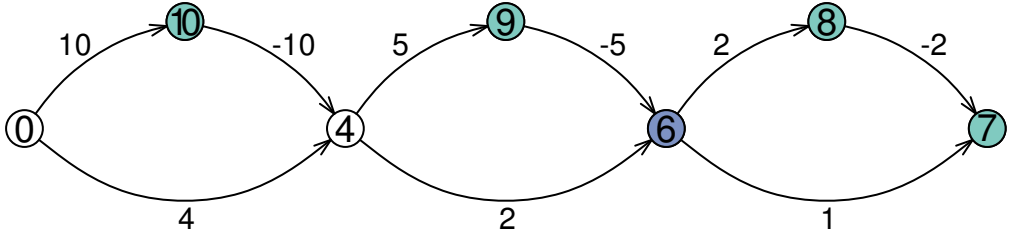


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

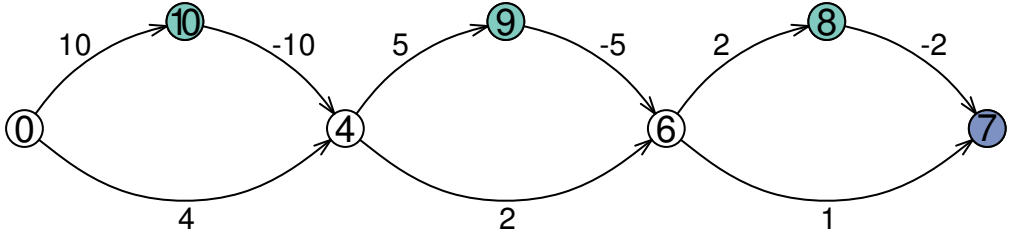


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

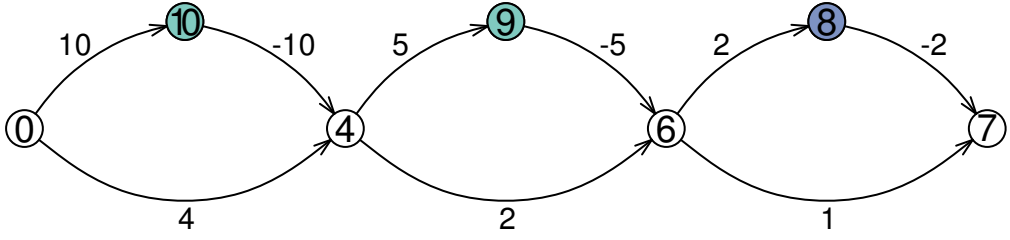


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

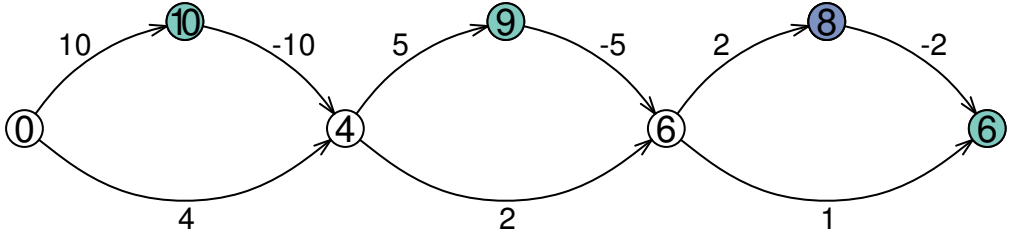


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

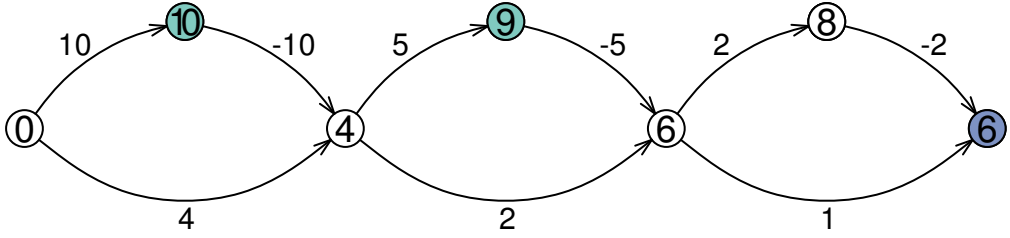


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

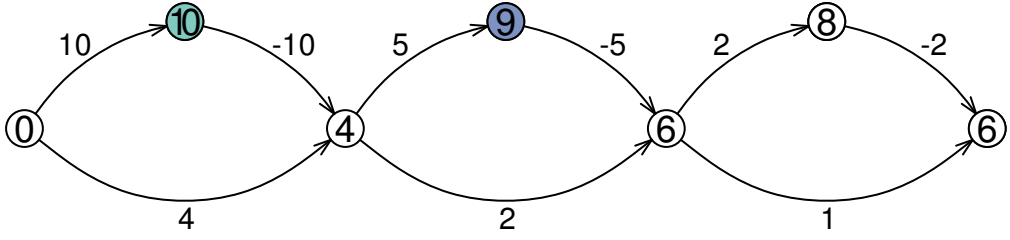


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

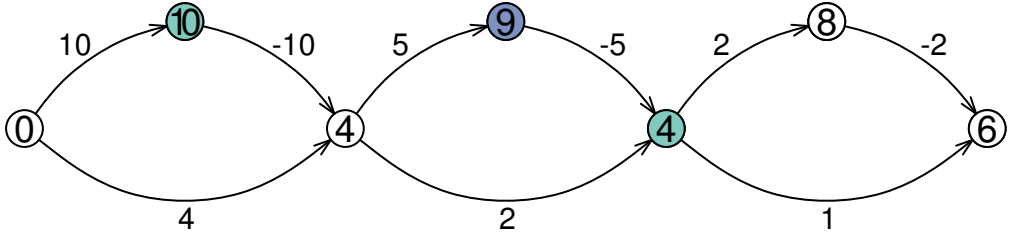


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

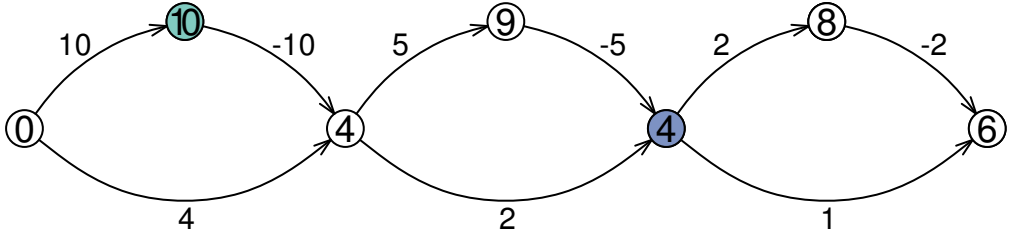


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

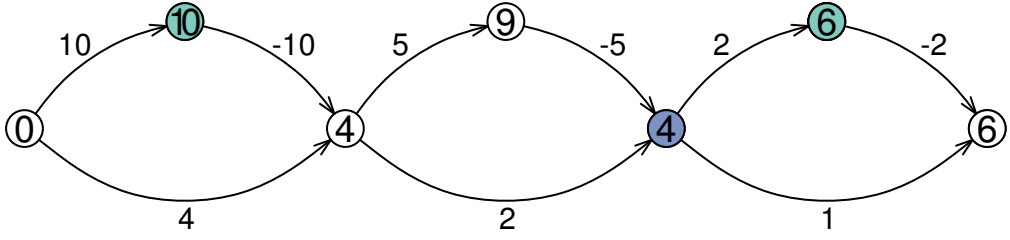


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

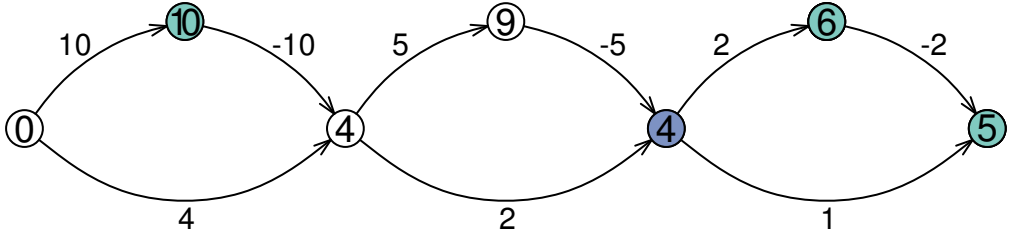


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

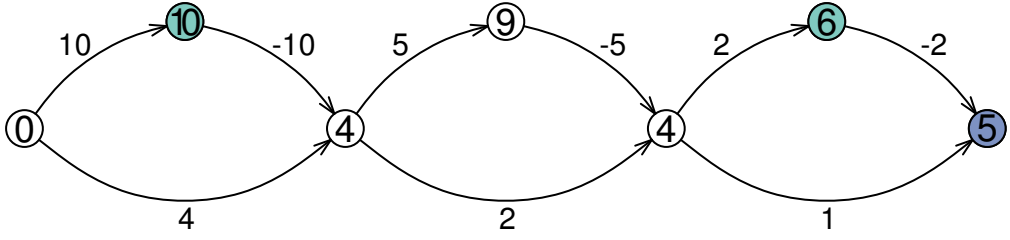


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

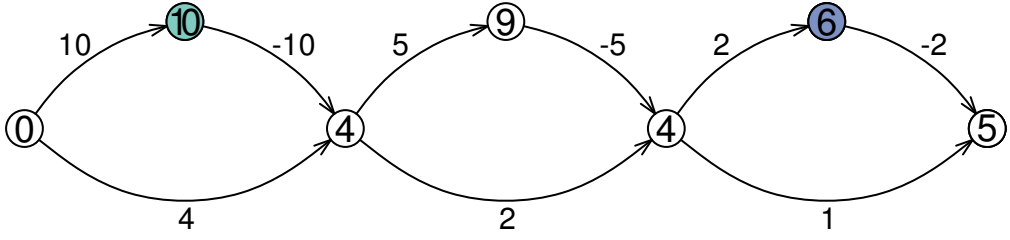


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

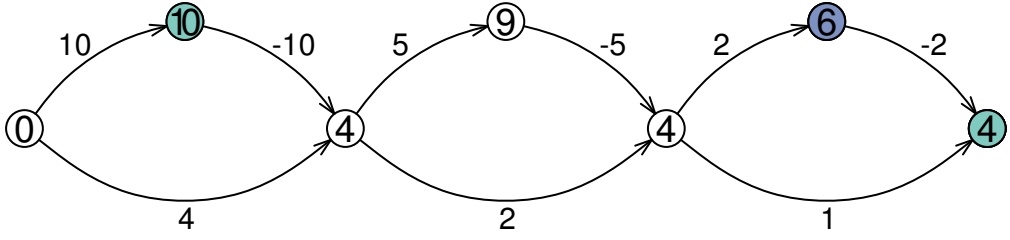


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

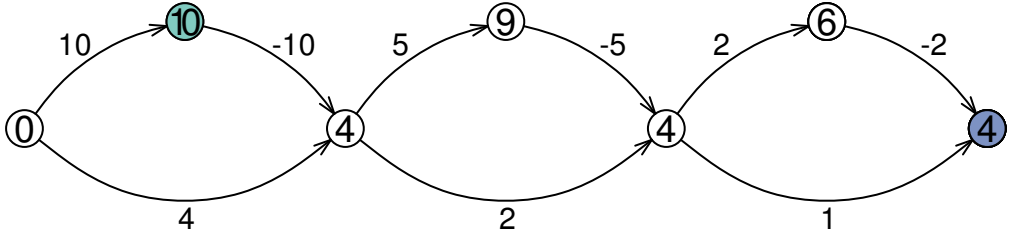


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

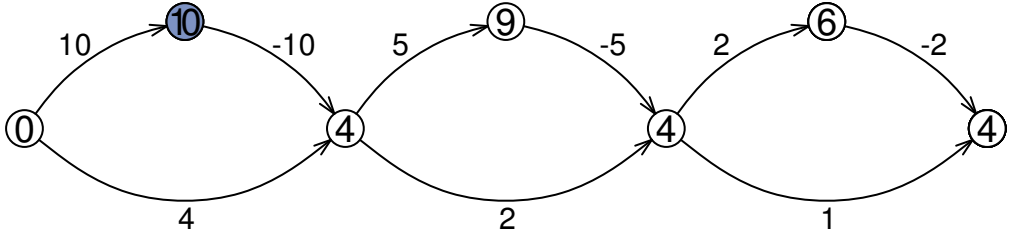


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

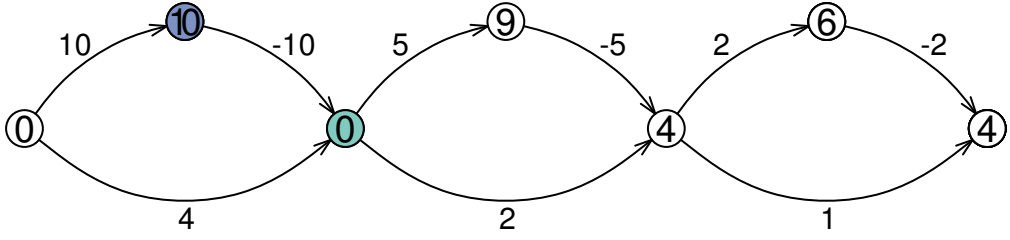


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

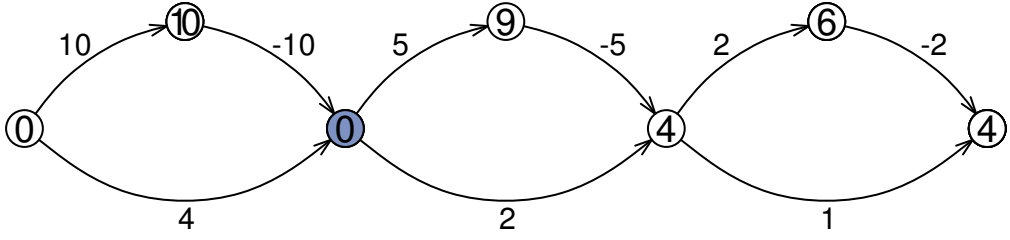


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

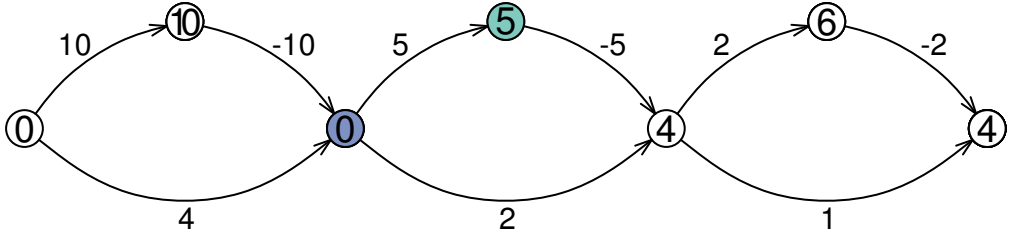


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

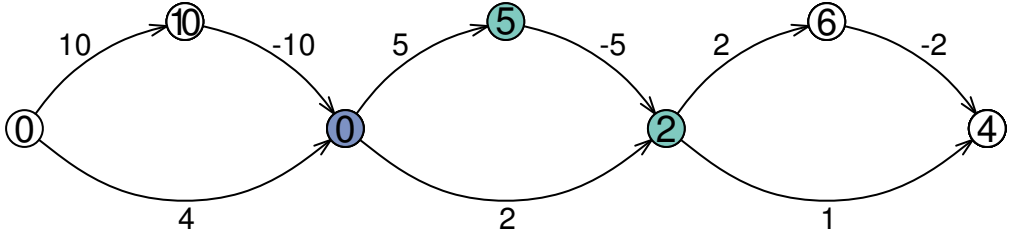


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

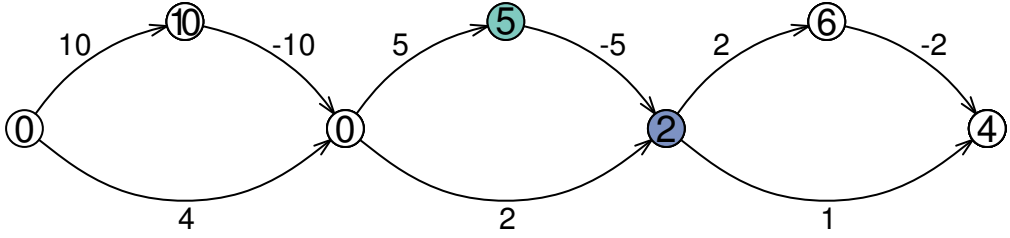


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

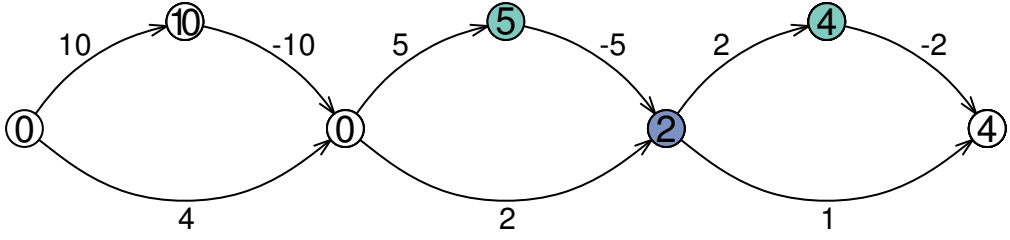


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

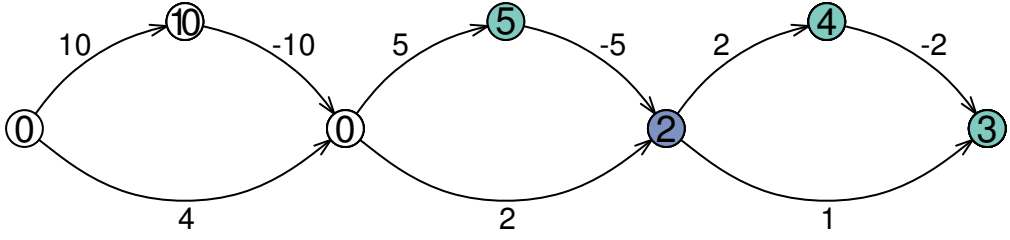


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

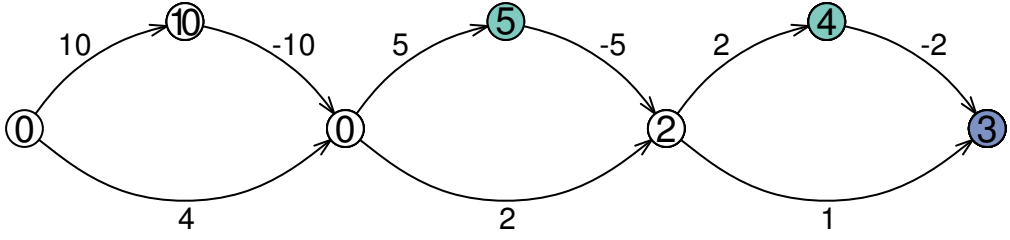


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

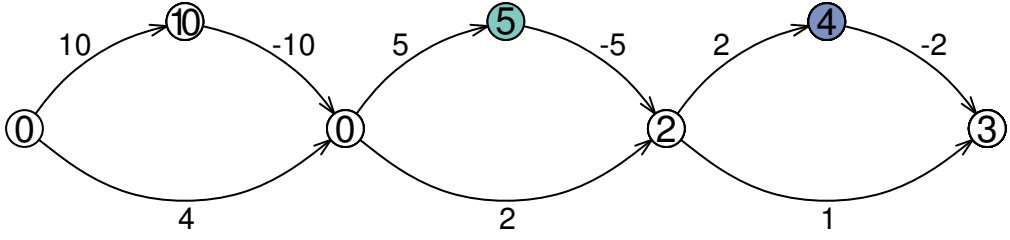


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

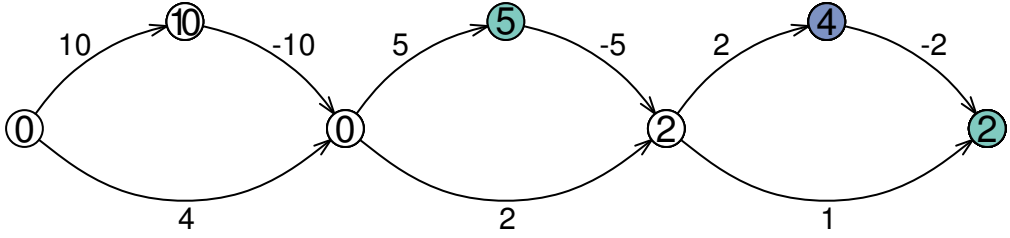


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

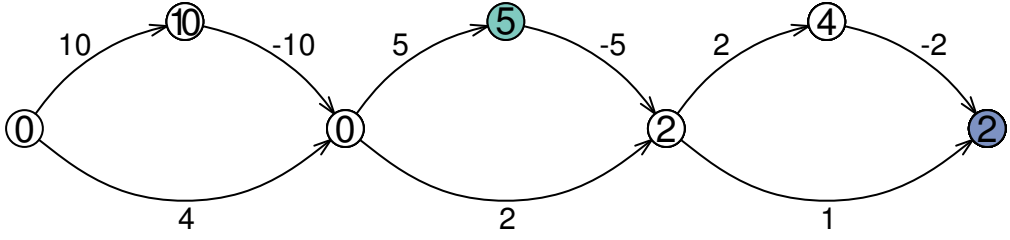


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

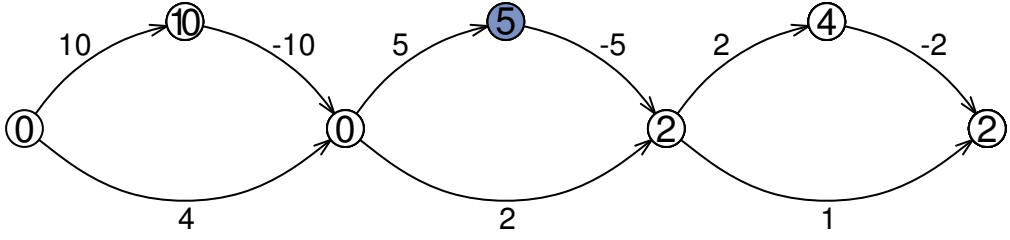


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

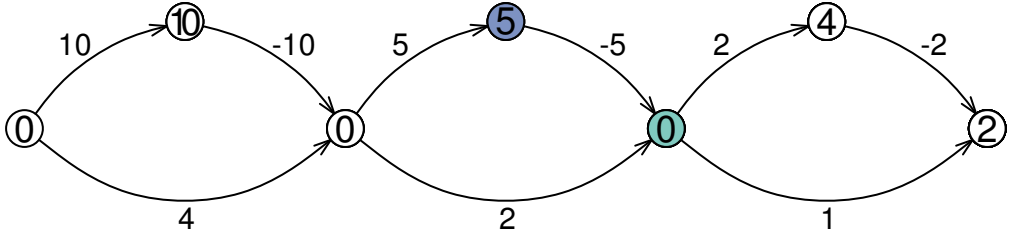


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

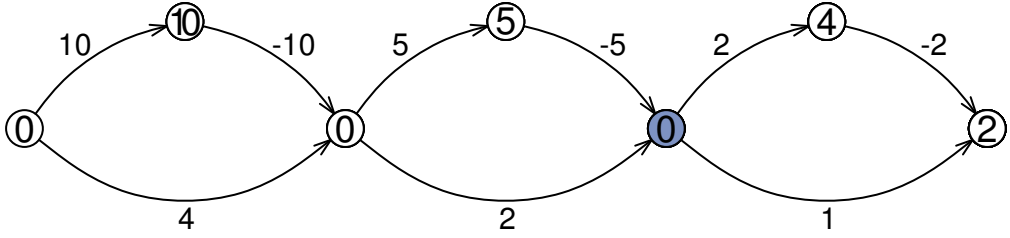


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

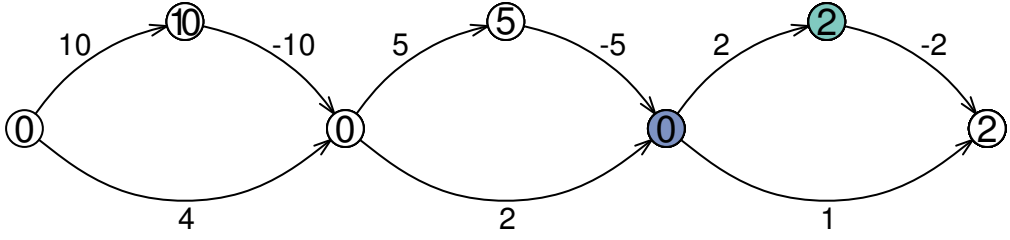


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

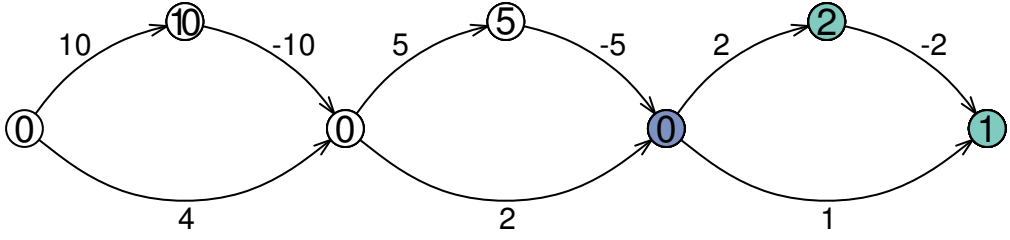


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

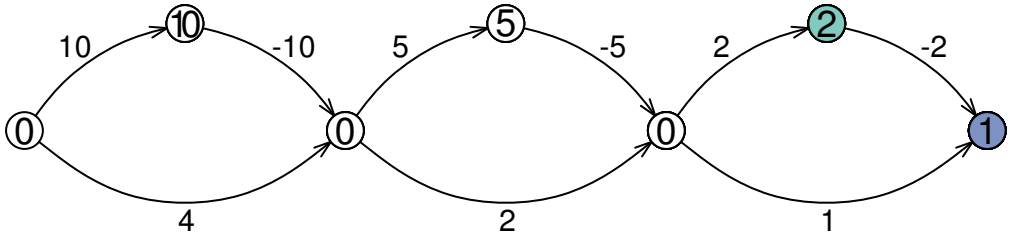


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

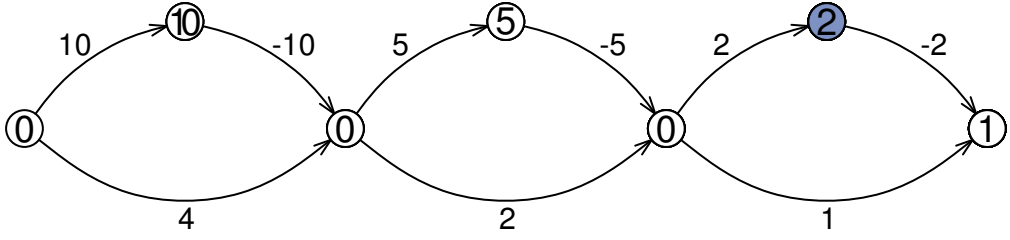


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

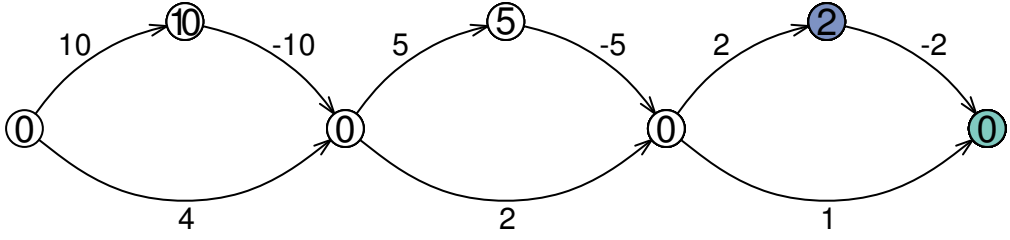


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

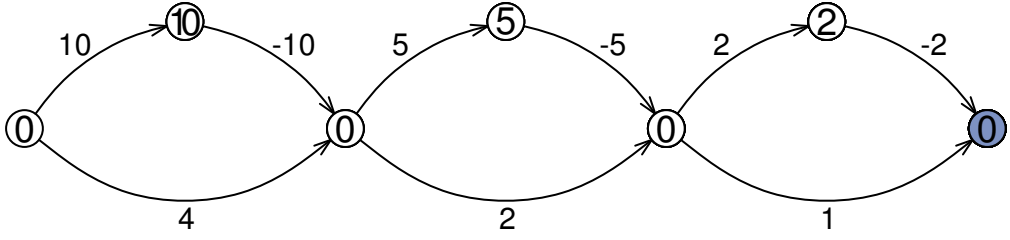


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

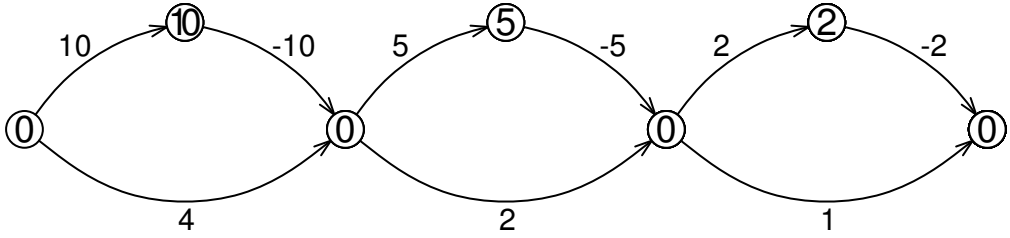


Energieoptimale Routen

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

Dijkstras Algorithmus:

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

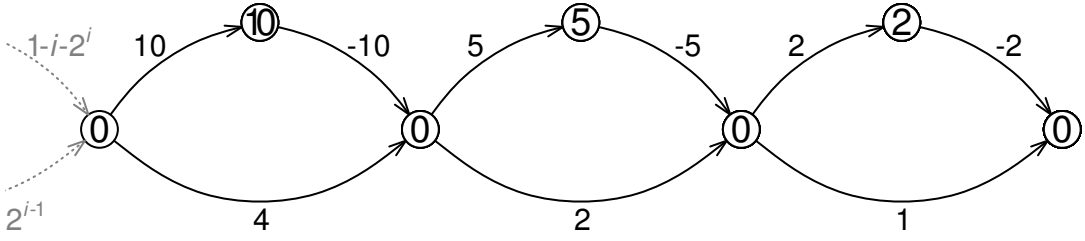


Energieoptimale Routen

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

Dijkstras 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

Dijkstras 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

Dijkstras 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 negativem Gewicht) ist Dijkstras Algorithmus schneller

Energieoptimale Routen

Dijkstras 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

Dijkstras 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 Knoten in der Queue befinden, die Label $d[t]$ durch Anhängen eines negativen Subpfades verbessern
- Ziel: Berechne Schranke $P_{\text{min}} \leq 0$ für Länge dieses Subpfades
- Dann Abbruch, sobald $\text{minKey}(Q) + P_{\text{min}} \geq d[t]$

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: $\underline{d}[t] \leq \text{dist}(v, t), \ \forall v \in V$

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

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

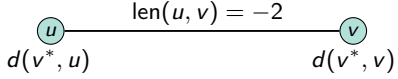
Knotenpotentiale

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

1. Distanzbasiertes Potential:

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



$$\text{len}'(u,v) := -2 + d(v^*, u) - d(v^*, v)$$

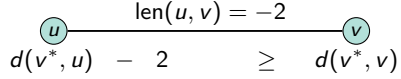
Knotenpotentiale

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

1. Distanzbasiertes Potential:

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



$$\text{len}'(u,v) := -2 + d(v^*, u) - d(v^*, v)$$

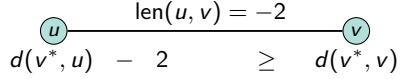
Knotenpotentiale

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

1. Distanzbasiertes Potential:

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



$$\begin{aligned}\text{len}'(u,v) &:= -2 + d(v^*,u) - d(v^*,v) \\ &\geq -2 + d(v^*,v) + 2 - d(v^*,v)\end{aligned}$$

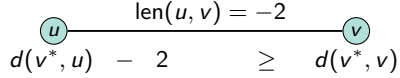
Knotenpotentiale

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



$$\begin{aligned}\text{len}'(u,v) &:= -2 + d(v^*,u) - d(v^*,v) \\ &\geq \cancel{-2} + \cancel{d(v^*,v)} + \cancel{2} - \cancel{d(v^*,v)} = 0\end{aligned}$$

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:

- Annahme: Höhenkoordinate $h(v)$ für jeden Knoten gegeben
- $\pi(v) := \alpha \cdot h(v)$ für alle v
- Wähle α so, dass $\alpha(h(u) - h(v)) \leq \text{len}(u, v)$ für alle $(u, v) \in E$
- Keine Garantie, dass das erfüllbar ist
  (klappt für realistische Kantengewichte)
- α kann mit Sweep über alle Kanten berechnet werden

Energieoptimale Routen

Bisher:

- Route minimiert Energieverbrauch für unbegrenzt großen Akku
- Tatsächlicher Verbrauch hängt von Kapazität ab
- Route ist für konkreten Ladestand an s ggf. nicht realisierbar

Jetzt:

- Berechne Route abhängig vom initialen Ladestand (SoC)
- Maximiere resultierenden SoC an t
- Berücksichtige Battery Constraints

Battery Constraints

Fahrzeug hat aktuellen SoC b und Akku-Kapazität M

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

Ausnahmen:

- Falls $b - \text{len}(e) < 0 \Rightarrow$ Kante nicht befahrbar
- Falls $b - \text{len}(e) > M \Rightarrow$ SoC ist M

$\Rightarrow$ Explizites Überprüfen in Dijkstras Algorithmus

Energieoptimale Routen

Anfragetypen:

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

Berechnung mit angepasstem Dijkstra

Energieoptimale Routen

Anfragetypen:

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

Berechnung mit angepasstem Dijkstra

Profilsuche: Gegeben 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

Battery Constraints

Konstanter Energieverbrauch pro Kante, aber:

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

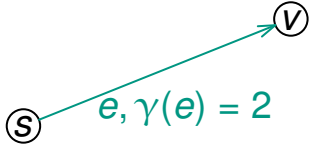
Battery Constraints

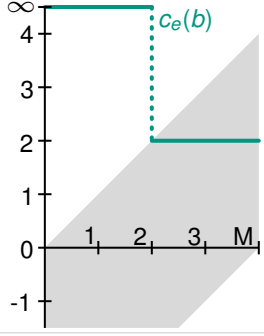
Konstanter Energieverbrauch pro Kante, aber:

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

Alternative 1:

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





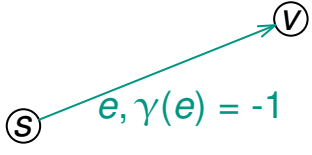
Battery Constraints

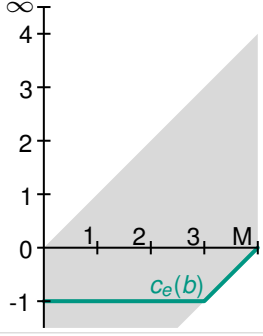
Konstanter Energieverbrauch pro Kante, aber:

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

Alternative 1:

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





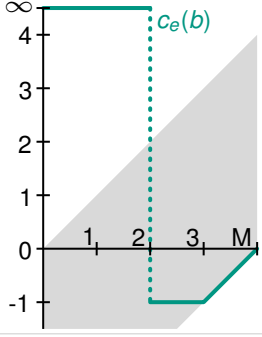
Battery Constraints

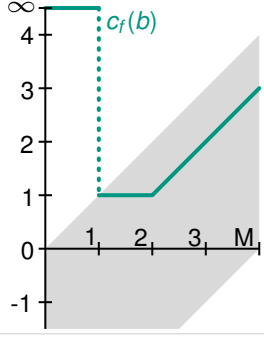
Konstanter Energieverbrauch pro Kante, aber:

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

Alternative 1:

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





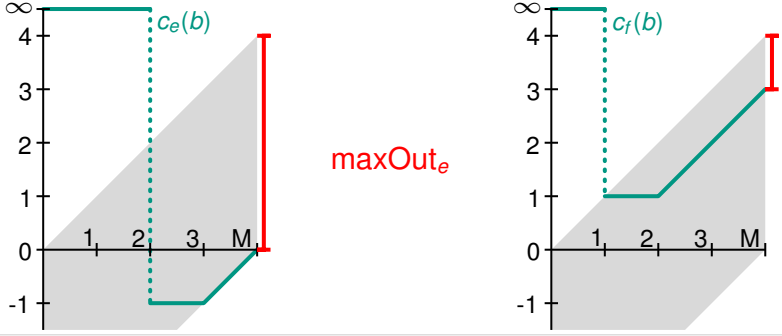
Battery Constraints

Konstanter Energieverbrauch pro Kante, aber:

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

Alternative 1:

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



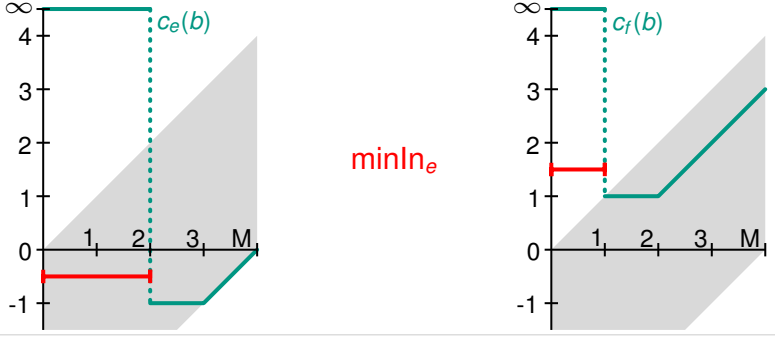
Battery Constraints

Konstanter Energieverbrauch pro Kante, aber:

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

Alternative 1:

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



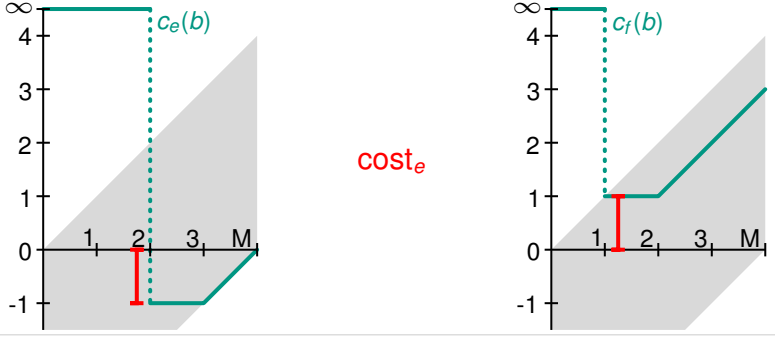
Battery Constraints

Konstanter Energieverbrauch pro Kante, aber:

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

Alternative 1:

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



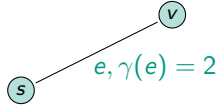
Battery Constraints

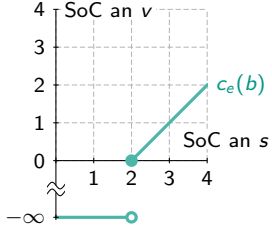
Konstanter Energieverbrauch pro Kante, aber:

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

Alternative 2:

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





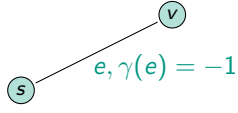
Battery Constraints

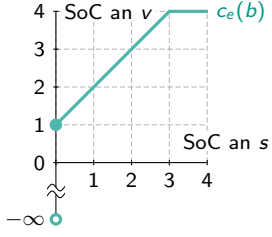
Konstanter Energieverbrauch pro Kante, aber:

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

Alternative 2:

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



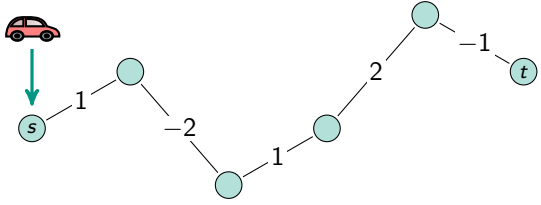


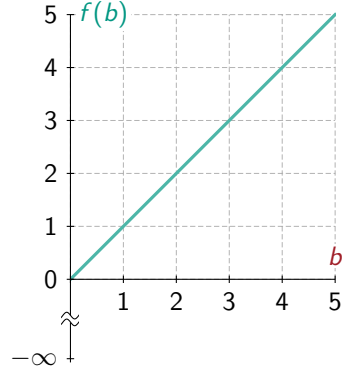
SoC-Profil

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: $M = 5$



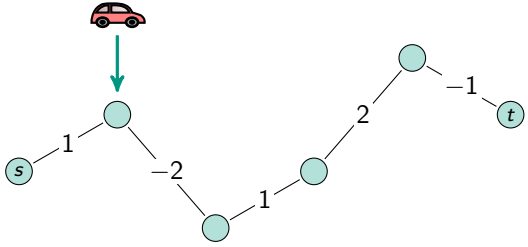


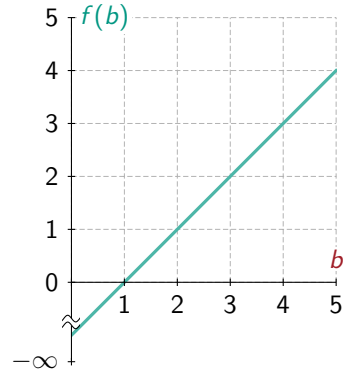
SoC-Profil

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: $M = 5$



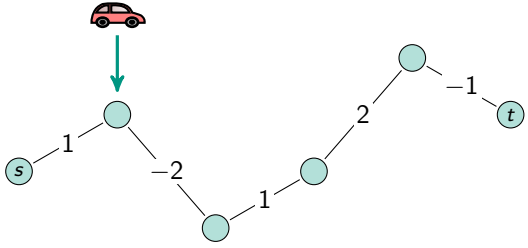


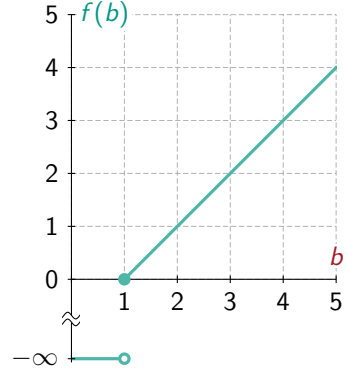
SoC-Profil

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: $M = 5$



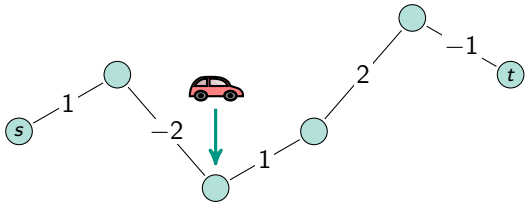


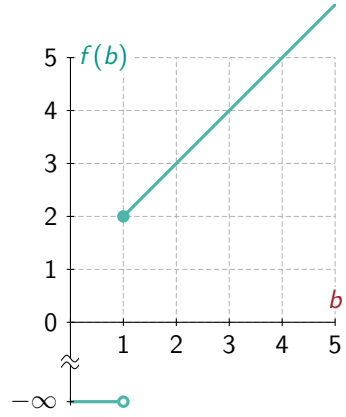
SoC-Profil

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: $M = 5$



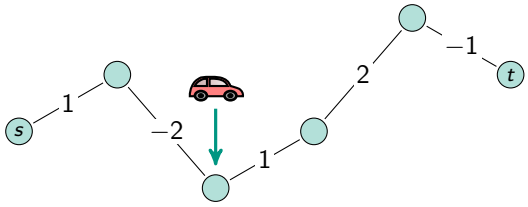


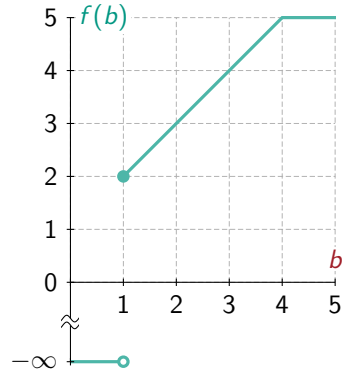
SoC-Profil

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: $M = 5$



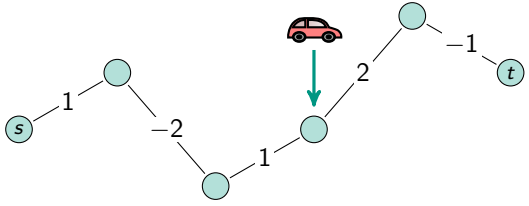


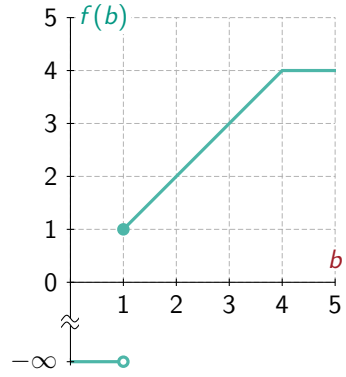
SoC-Profil

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: $M = 5$



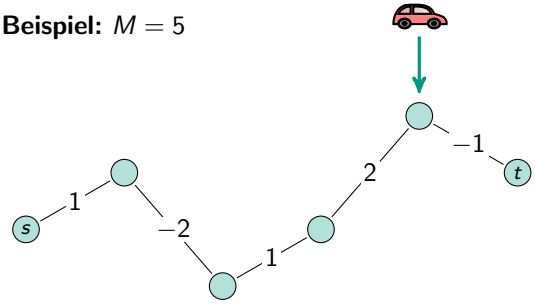


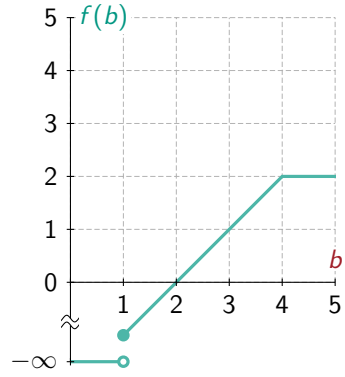
SoC-Profil

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: $M = 5$



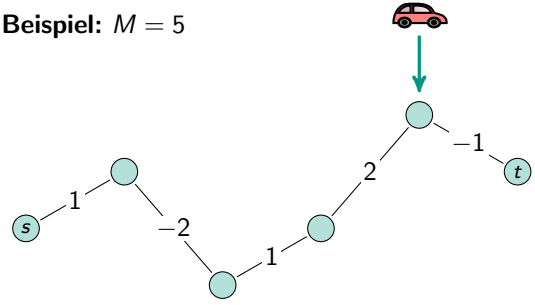


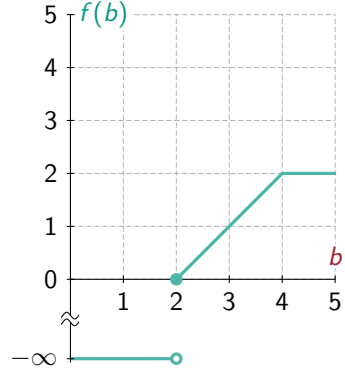
SoC-Profil

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: $M = 5$



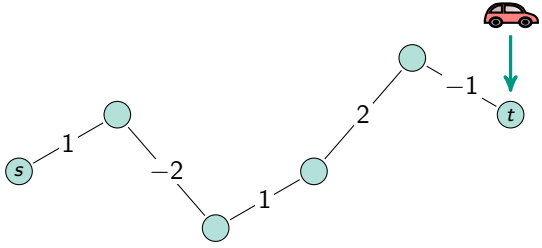


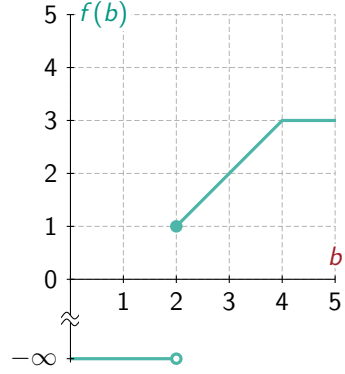
SoC-Profil

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: $M = 5$

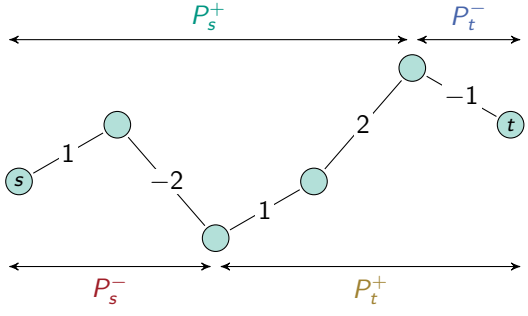


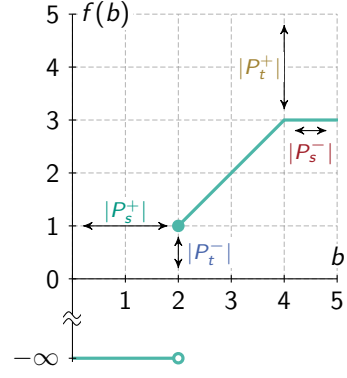


SoC-Profil eines Pfads

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

Beispiel:





$\Rightarrow$ 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:

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

SoC-Profil $\leftrightarrow$ Verbrauchsfunktion

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

SoC-Profil $\rightarrow$ Verbrauchsfunktion:

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

Verbrauchsfunktion $\rightarrow$ SoC-Profil:

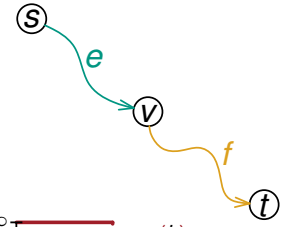
- $|P_s^+| = \text{minIn}_P$
- $|P_s^-| = M - \text{maxOut}_P - \text{cost}_P$
- $|P_t^+| = M - \text{maxOut}_P$
- $|P_t^-| = \text{minIn}_P - \text{cost}_P$

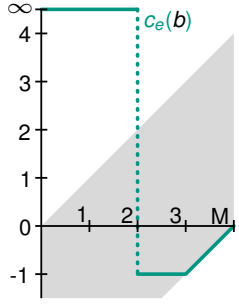
Verbrauchsfunktionen – Linking

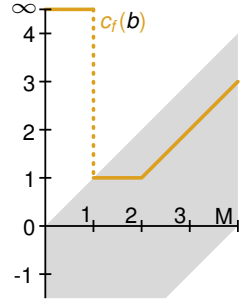
Gesucht: Verbrauch beim Traversieren von e *und* f

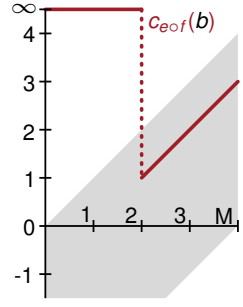
- Verbrauch auf e gegeben durch $c_e(b)$
- SoC nach e = SoC vor f = $b - c_e(b)$

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





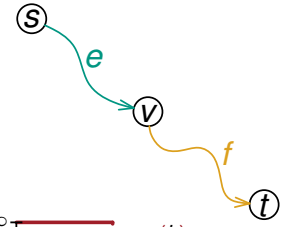


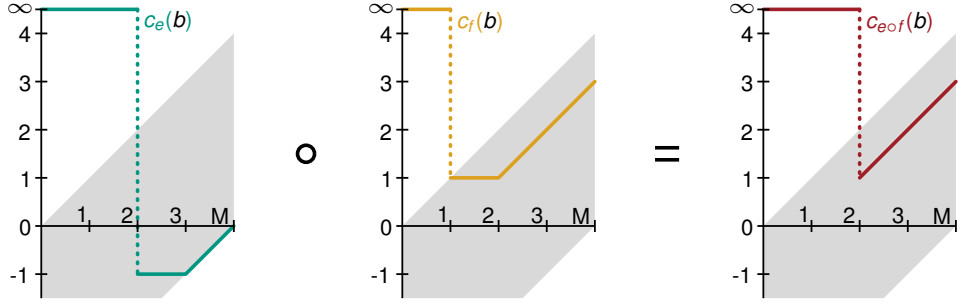


Verbrauchsfunktionen – Linking

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

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



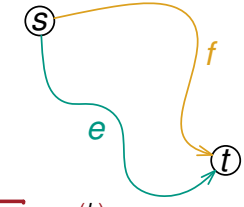


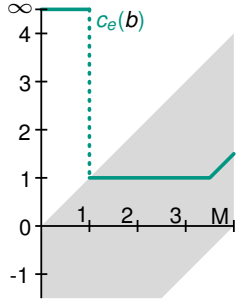
Verbrauchsfunktionen – Merging

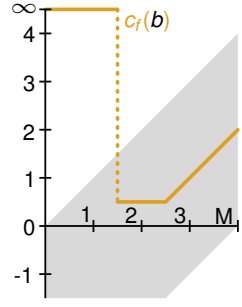
Gesucht: Verbrauch beim Traversieren von e *oder* f

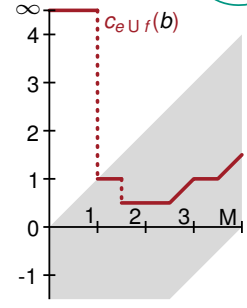
- Benutze Pfad mit geringerem Verbrauch

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







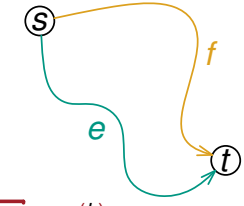


Verbrauchsfunktionen – Merging

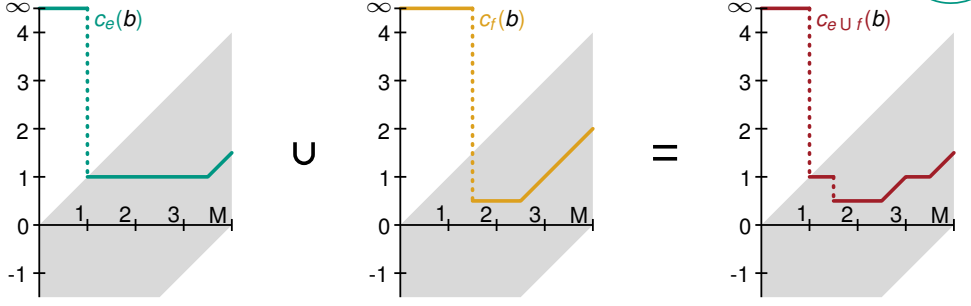
Gesucht: Verbrauch beim Traversieren von e *oder* f

- Benutze Pfad mit geringerem Verbrauch

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



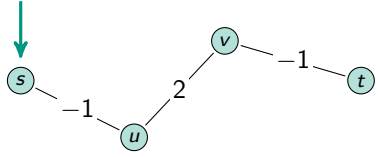
Im Allgemeinen $\mathcal{O}(m)$ Stützstellen

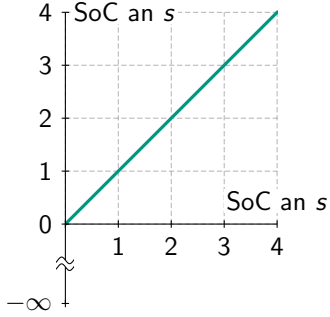


Allgemeine Profile

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

Beispiel: $M = 4$

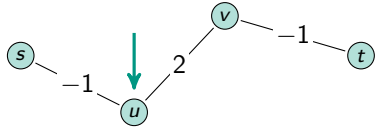


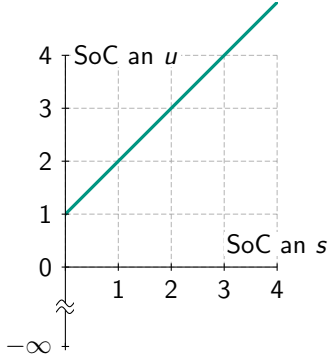


Allgemeine Profile

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

Beispiel: $M = 4$

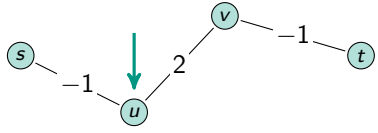


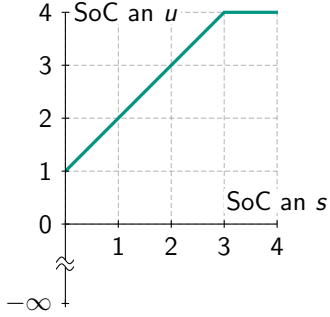


Allgemeine Profile

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

Beispiel: $M = 4$

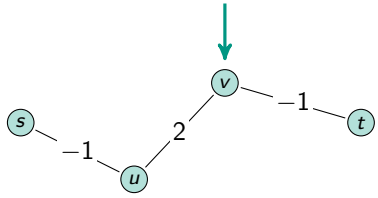


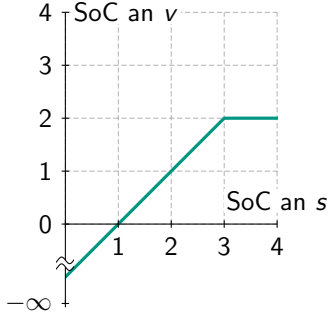


Allgemeine Profile

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

Beispiel: $M = 4$

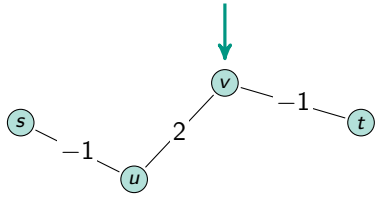


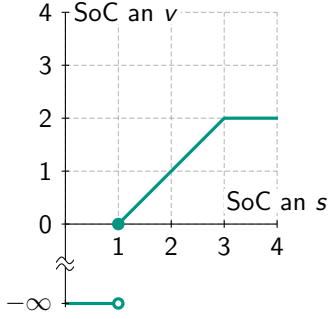


Allgemeine Profile

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

Beispiel: $M = 4$

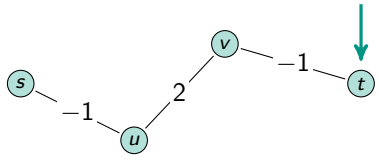


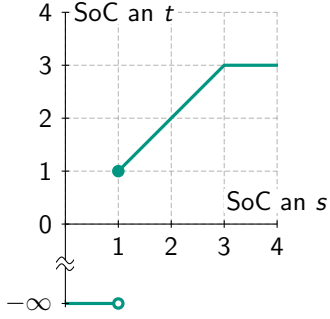


Allgemeine Profile

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

Beispiel: $M = 4$

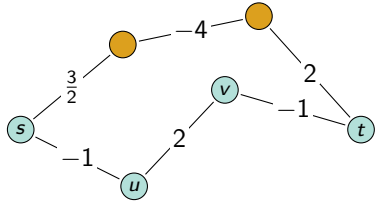


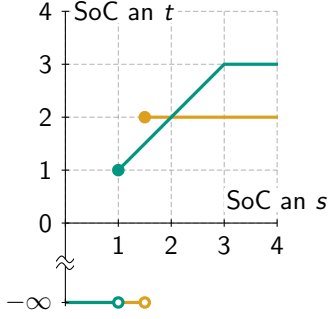


Allgemeine Profile

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

Beispiel: $M = 4$

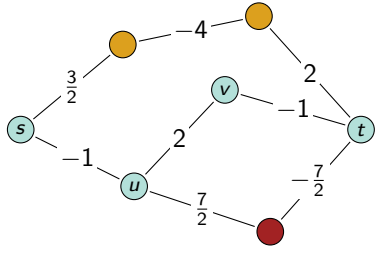


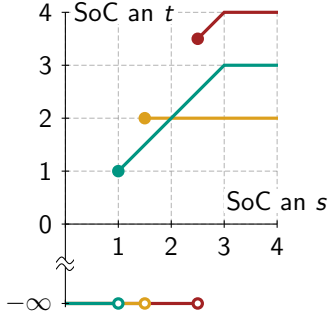


Allgemeine Profile

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

Beispiel: $M = 4$

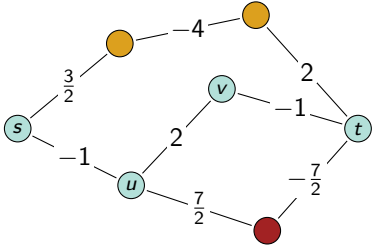


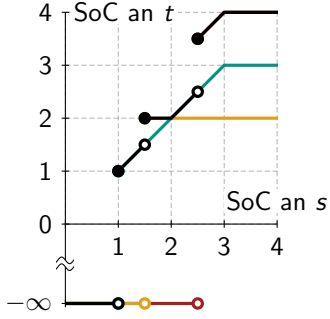


Allgemeine Profile

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

Beispiel: $M = 4$

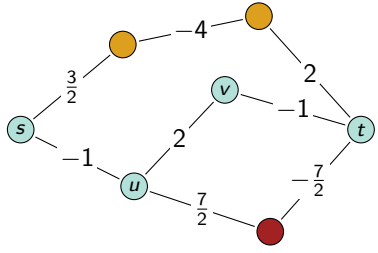


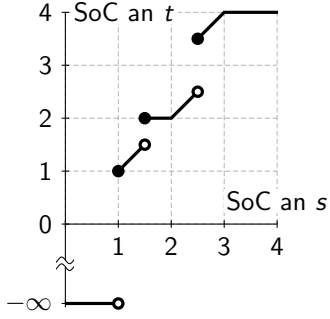


Allgemeine Profile

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

Beispiel: $M = 4$

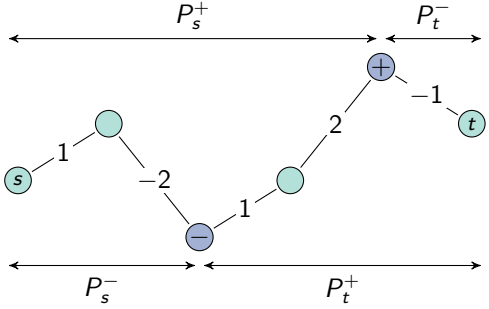




Anzahl Stützpunkte ist linear in Anzahl der Pfade
⇒ Wie viele verschiedene Pfade tragen zum Profil bei?

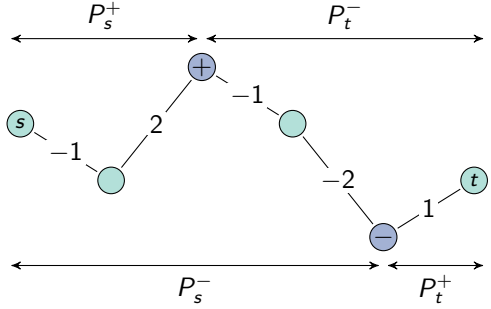
Typen von Pfaden

Tal-Berg-Pfad:



Min. Präfix endet vor max. Präfix

Berg-Tal-Pfad:



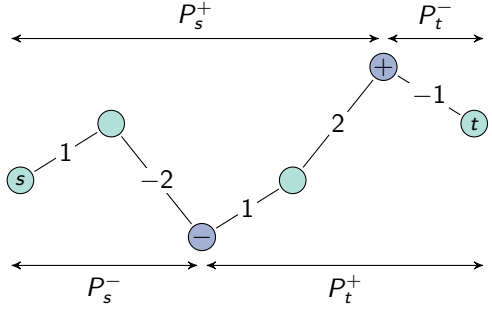
Max. Präfix endet vor min. Präfix

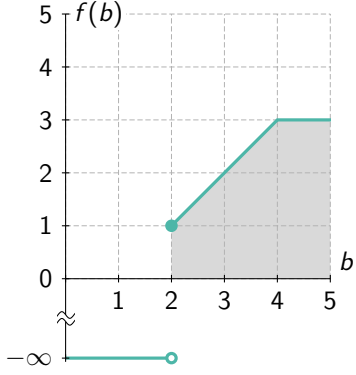
- Letzter Knoten des max. Präfix heißt Bergknoten
- Letzter Knoten des min. Präfix heißt Talknoten

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

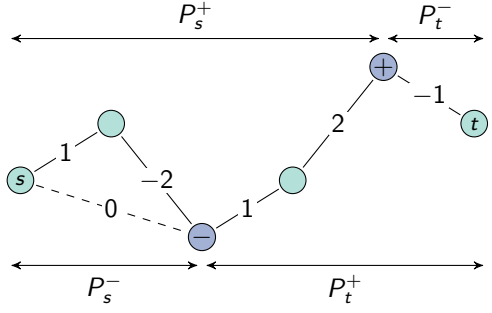


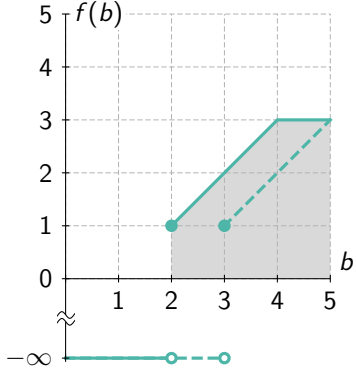


Komplexität von SoC-Profilen

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

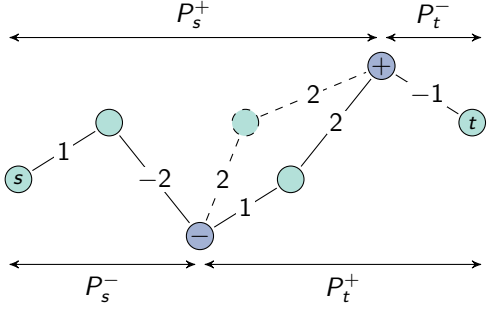


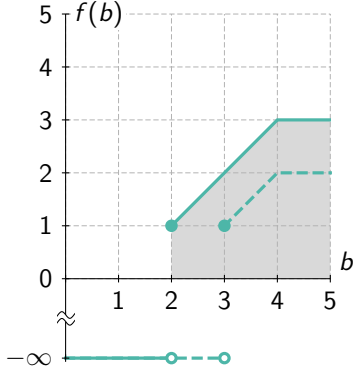


Komplexität von SoC-Profilen

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

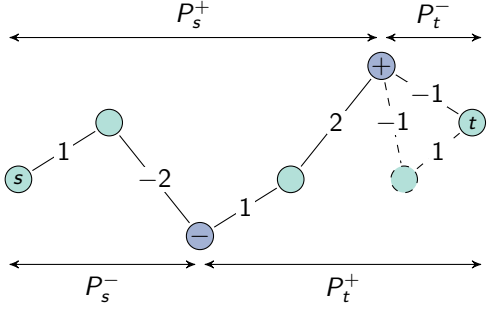


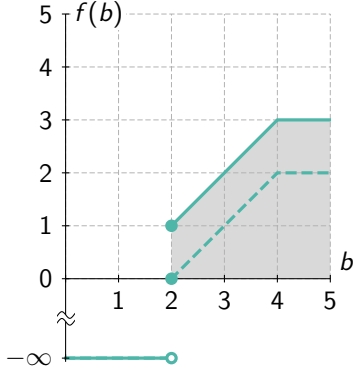


Komplexität von SoC-Profilen

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)

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.