

Algorithmen für Routenplanung

14. Vorlesung, Sommersemester 2017

Tobias Zündorf | 3. Juli 2017

INSTITUT FÜR THEORETISCHE INFORMATIK · ALGORITHMIK · PROF. DR. DOROTHEA WAGNER



Noch freie Klausurtermine:

- Mittwoch 09.08.2017 (nachmittags)
- Donnerstag 14.09.2017 (vormittags)

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

Besonderheiten:

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

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

⇒ Ladestand (state of charge, SoC) während Query berücksichtigen

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

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

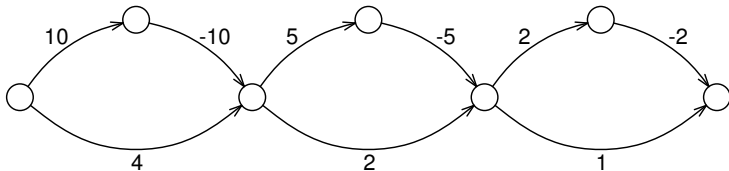
Dijkstra's Algorithmus:

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

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

Dijkstra's Algorithmus:

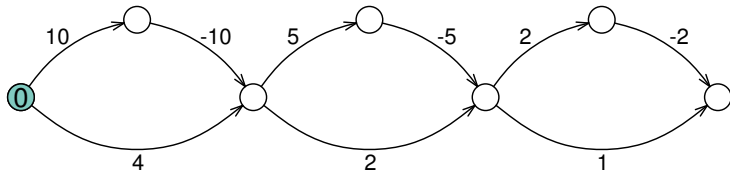
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

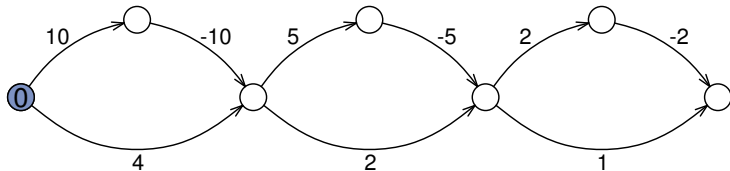
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

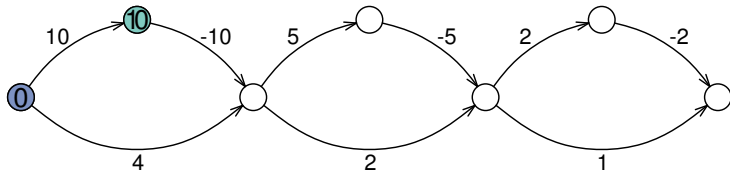
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

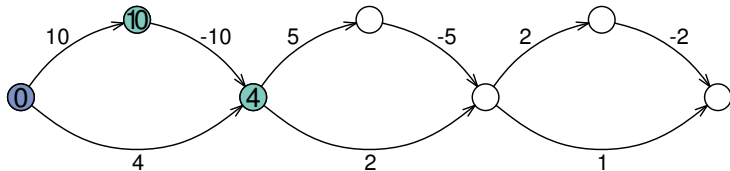
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

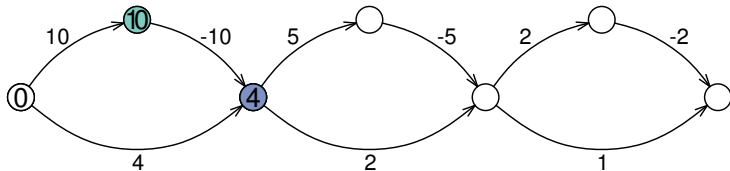
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

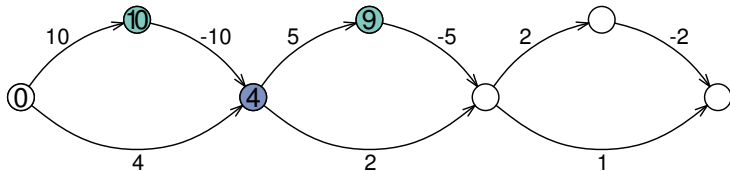
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

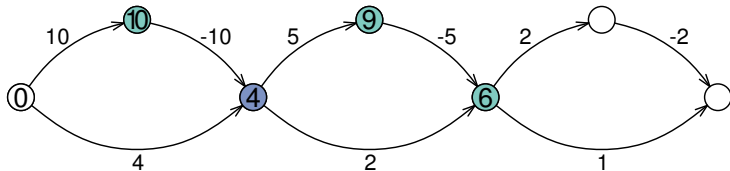
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

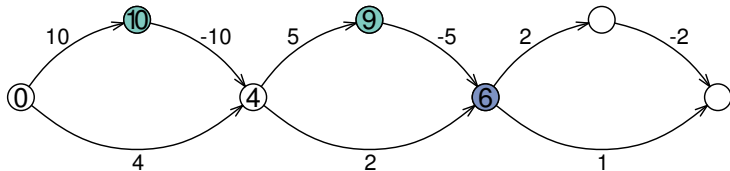
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

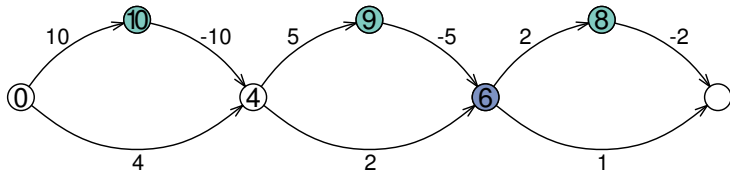
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

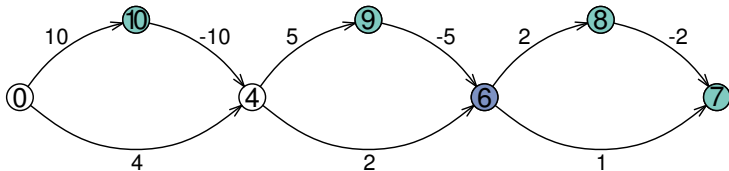
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

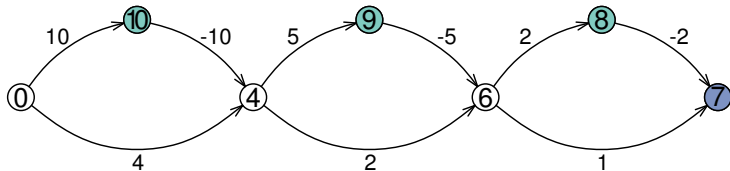
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

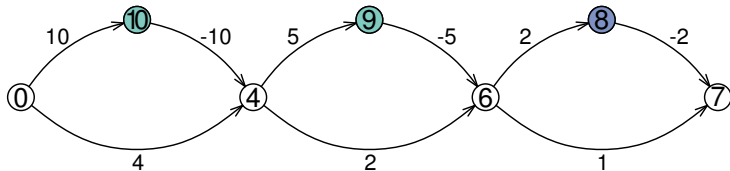
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

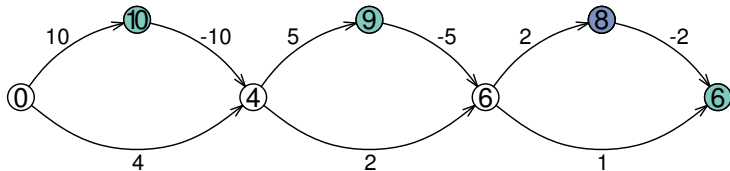
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

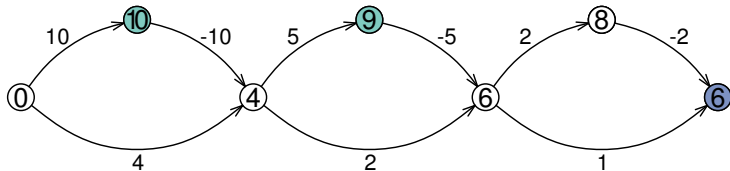
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

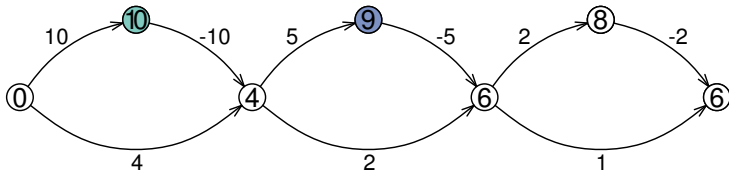
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

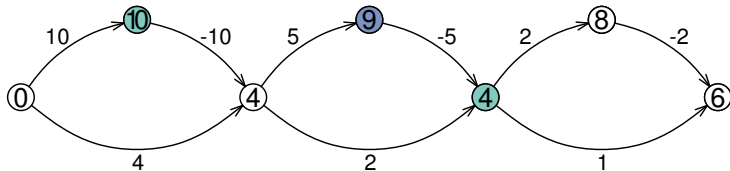
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

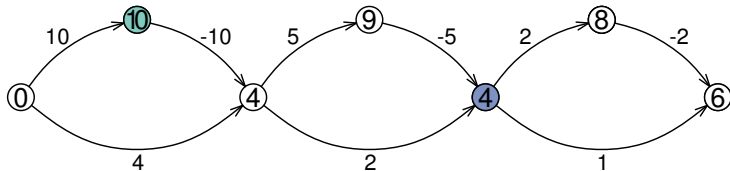
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

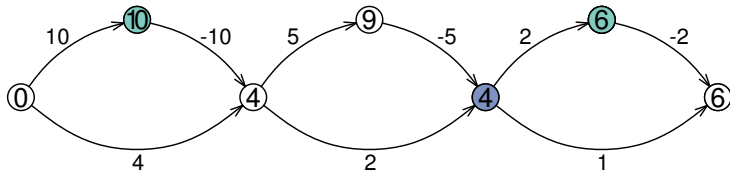
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

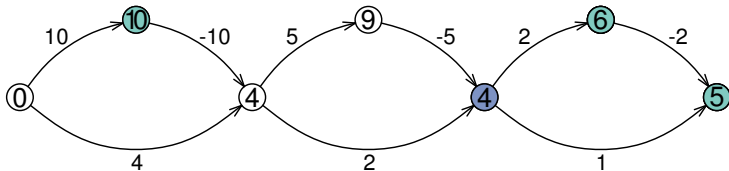
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

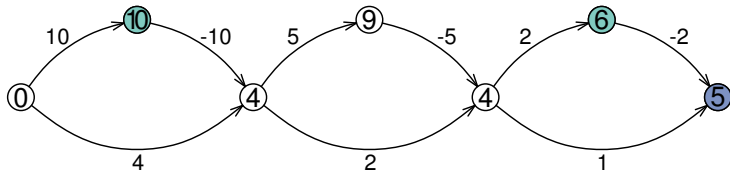
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

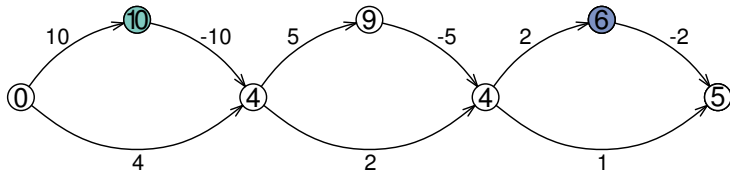
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

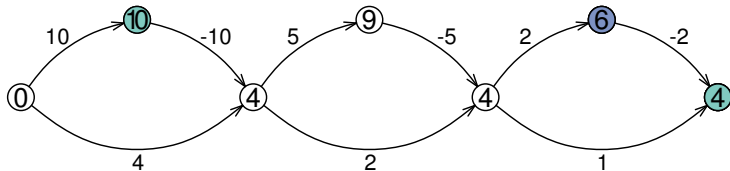
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

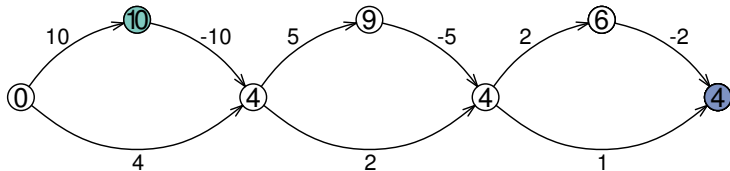
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

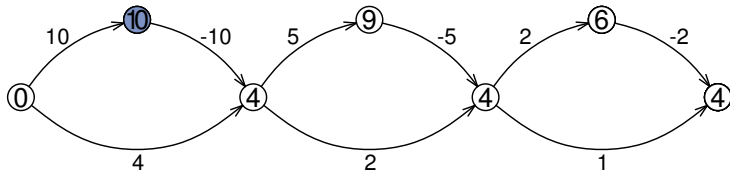
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

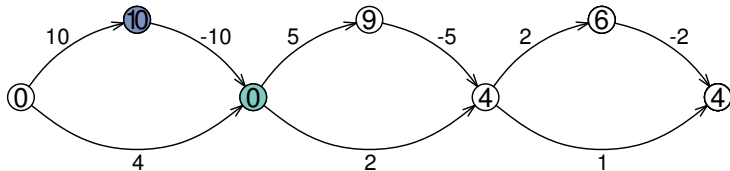
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

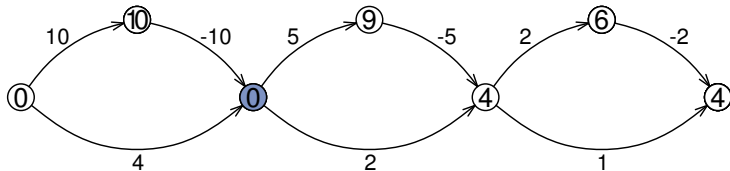
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

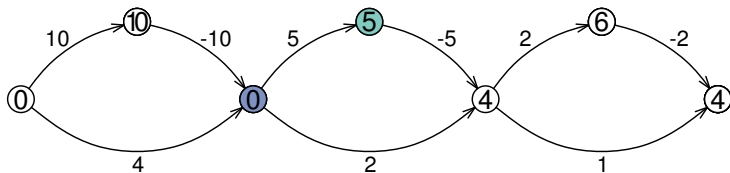
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

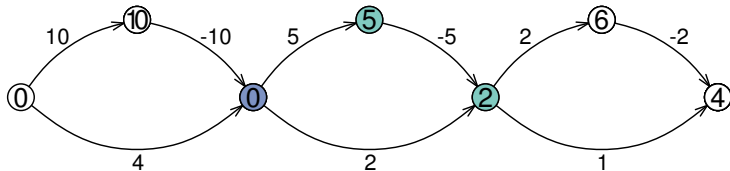
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

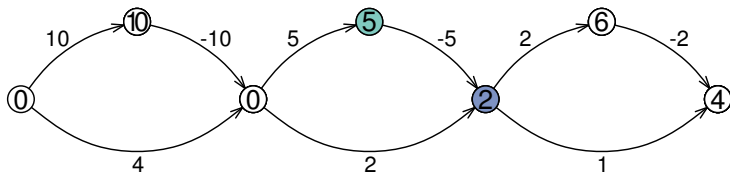
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

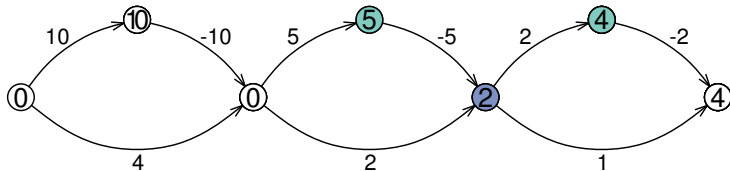
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

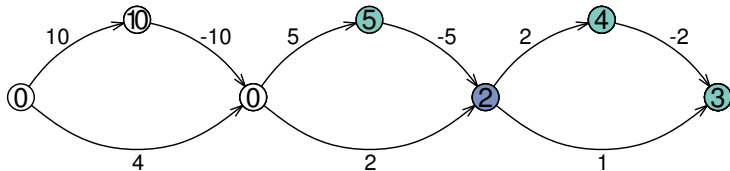
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

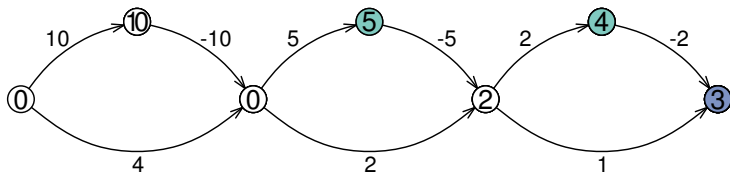
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

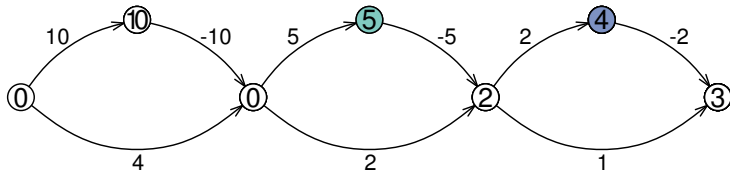
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

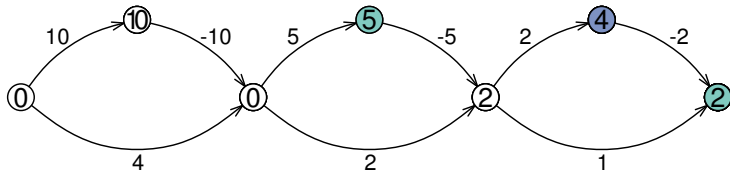
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

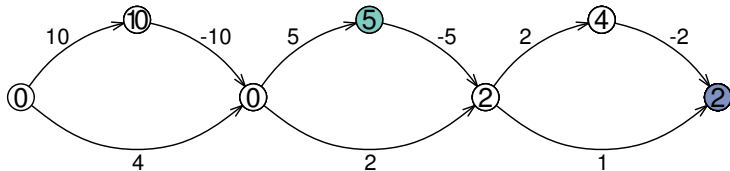
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

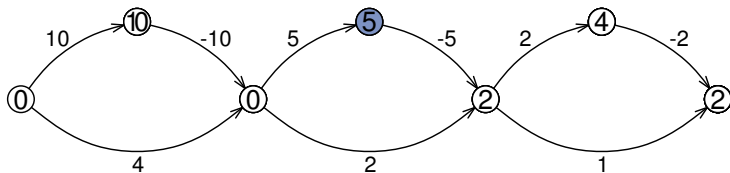
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

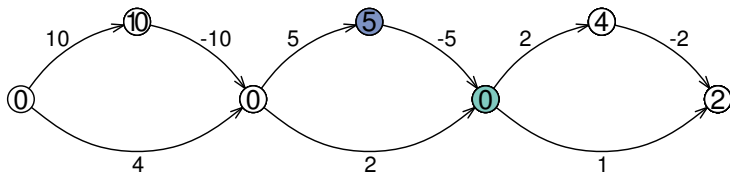
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

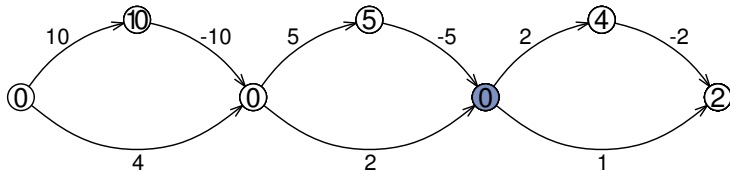
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

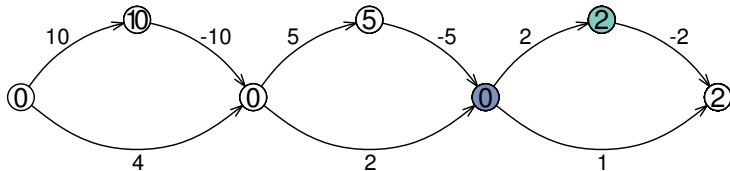
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

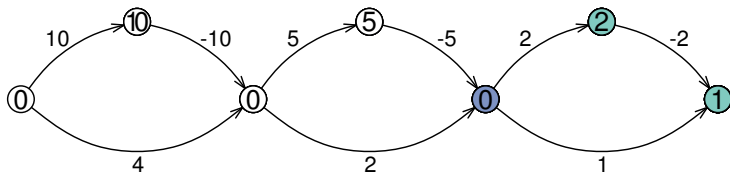
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

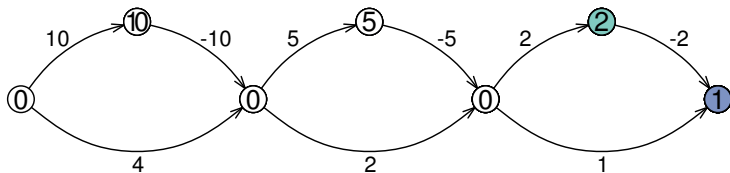
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

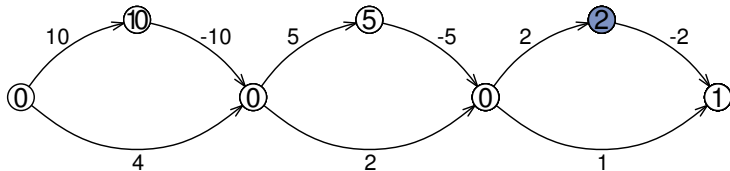
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

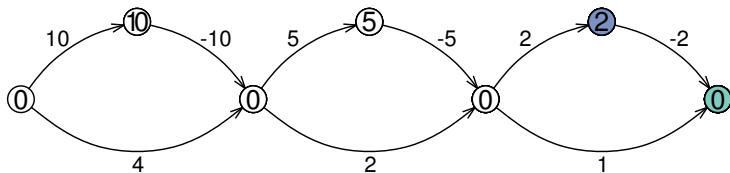
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

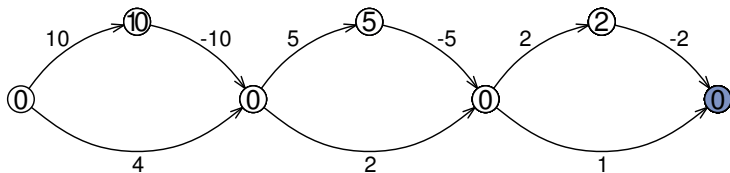
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

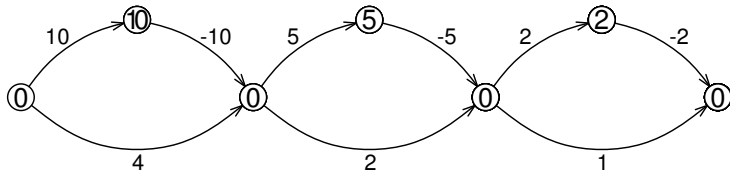
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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

Dijkstra's Algorithmus:

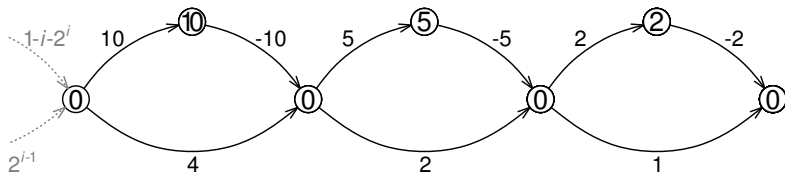
- Setzt nichtnegative Kantengewichte voraus
- Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr



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



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

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 ist Dijkstras Algorithmus schneller

Dijkstra's Algorithmus:

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

Frage: Stoppkriterium wiederherstellbar?

Dijkstra's Algorithmus:

- Setzt nichtnegative Kantengewichte voraus
 - Bleibt korrekt, aber keine polynomielle Laufzeitgarantie mehr
 - Nicht mehr Label-Setting
- ⇒ 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 $\text{minKey}(Q) \geq d[t] + P_{\min}$

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)

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: $P_{\min} := \min_{v \in V} \underline{d}[v]$

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

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: $\text{minKey}(Q) \leq \underline{d}[t] + \underline{d}[t]$

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?

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 $\pi: V \rightarrow \mathbb{R}$, sodass
 $len(u, v) - \pi(u) + \pi(v) \geq 0$ für jede Kante $(u, v) \in E$

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

Dann Dijkstras Algorithmus (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

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

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

2. 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)

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)

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

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:

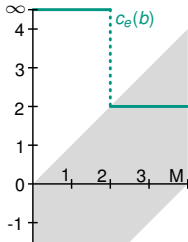
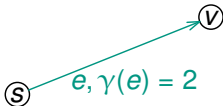
- Verbrauchsfunktionen
- Geeigneten Link/Merge-Operationen

Bisher:

- Konstanter Energieverbrauch pro Kante
- Explizites Überprüfen der Battery Constraints im Algorithmus

Jetzt:

- Integriere Battery Constraints in Verbrauchsfunktionen
- **Verbrauchsfunktion** bildet SoC auf tatsächlichen Verbrauch ab

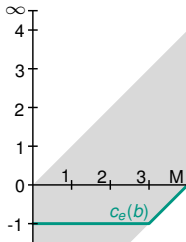
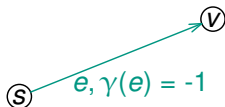


Bisher:

- Konstanter Energieverbrauch pro Kante
- Explizites Überprüfen der Battery Constraints im Algorithmus

Jetzt:

- Integriere Battery Constraints in Verbrauchsfunktionen
- **Verbrauchsfunktion** bildet SoC auf tatsächlichen Verbrauch ab

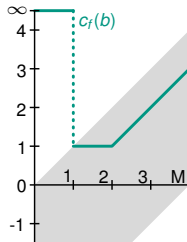
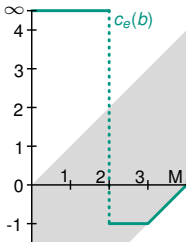


Bisher:

- Konstanter Energieverbrauch pro Kante
- Explizites Überprüfen der Battery Constraints im Algorithmus

Jetzt:

- Integriere Battery Constraints in Verbrauchsfunktionen
- Verbrauchsfunktion bildet SoC auf tatsächlichen Verbrauch ab

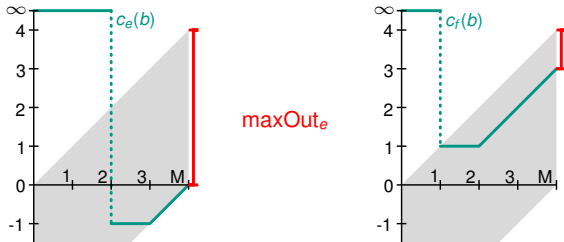


Bisher:

- Konstanter Energieverbrauch pro Kante
- Explizites Überprüfen der Battery Constraints im Algorithmus

Jetzt:

- Integriere Battery Constraints in Verbrauchsfunktionen
- Verbrauchsfunktion bildet SoC auf tatsächlichen Verbrauch ab



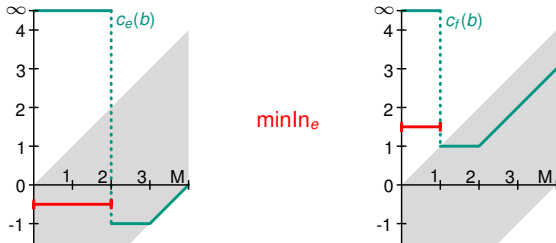
Anm: Potentiale weiterhin zulässig (cost_e ist untere Schranke auf Verbrauch)

Bisher:

- Konstanter Energieverbrauch pro Kante
- Explizites Überprüfen der Battery Constraints im Algorithmus

Jetzt:

- Integriere Battery Constraints in Verbrauchsfunktionen
- Verbrauchsfunktion bildet SoC auf tatsächlichen Verbrauch ab

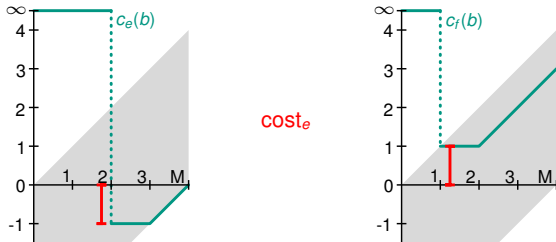


Bisher:

- Konstanter Energieverbrauch pro Kante
- Explizites Überprüfen der Battery Constraints im Algorithmus

Jetzt:

- Integriere Battery Constraints in Verbrauchsfunktionen
- Verbrauchsfunktion bildet SoC auf tatsächlichen Verbrauch ab

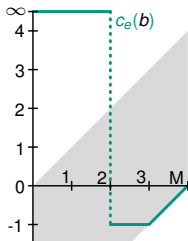
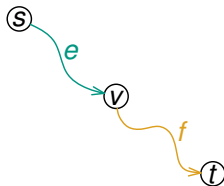


Gesucht: Verbrauch beim traversieren von e und f

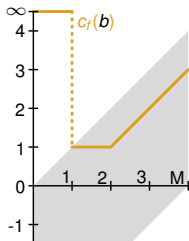
■ Verbrauch auf e gegeben durch $c_e(b)$

■ SoC nach $e = \text{SoC vor } f = b - c_e(b)$

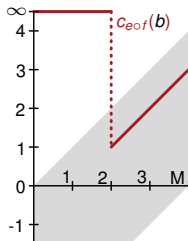
⇒ Verbrauch auf e UND f : $c_{e \circ f}(b) = c_f(b - c_e(b))$



\circ



$=$



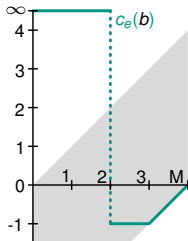
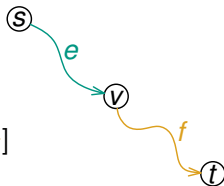
Formal: $c_{eof}(b)$ ist gegeben durch:

$$\min \text{In}_{eof} = \max[\min \text{In}_e, \min \text{In}_f + \text{cost}_e]$$

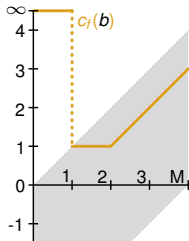
$$\max \text{Out}_{eof} = \min[\max \text{Out}_f, \max \text{Out}_e - \text{cost}_f]$$

$$\text{cost}_{eof} = \max[\text{cost}_e + \text{cost}_f, \min \text{In}_e - \max \text{Out}_f]$$

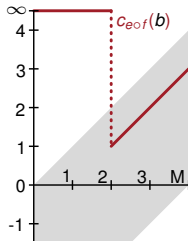
(Bedingung: $\max \text{Out}_e \geq \min \text{In}_f$)



\circ



$=$

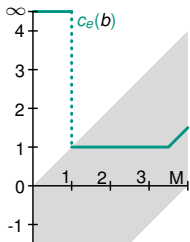
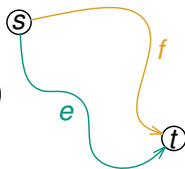


Verbrauchsfunktionen – Merging

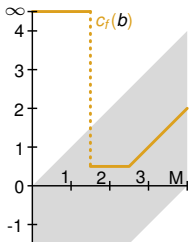
Gesucht: Verbrauch beim traversieren von e oder f

- Benutze Pfad mit geringerem Verbrauch

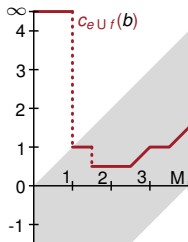
⇒ Verbrauch auf e ODER f : $c_{e \cup f}(b) = \min(c_f(b), c_e(b))$



U



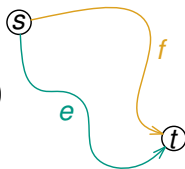
=



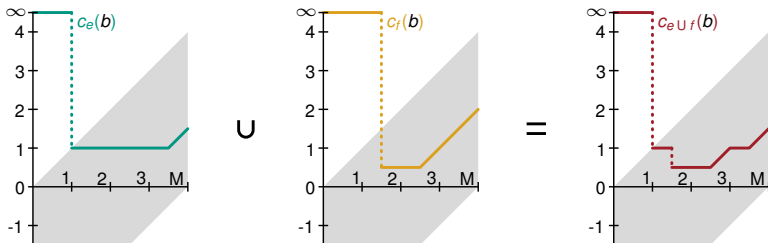
Gesucht: Verbrauch beim traversieren von e oder f

- Benutze Pfad mit geringerem Verbrauch

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



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



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, ...

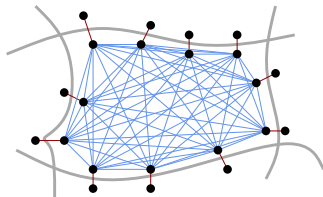
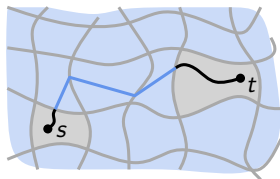
⇒ Schnelle Customization wichtig

Idee MLD:

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

Overlay Graph:

- Randknoten
- Cliques in jeder Zelle
- Schnittkanten



Suchgraph:

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

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)

Algorithm	CUSTOMIZING		QUERIES	
	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)

Bisher: Energieoptimale Routen

- Energiesparendes Fahren
- Wir finden einen zulässigen Pfad, falls dieser existiert

Problem: Wir versuchen Energie zu sparen selbst wenn:

- Die Strecke sehr kurz ist
- Der Akkustand mehr als ausreichend für die Strecke ist

Bisher: Energieoptimale Routen

- Energiesparendes Fahren
- Wir finden einen zulässigen Pfad, falls dieser existiert

Problem: Wir versuchen Energie zu sparen selbst wenn:

- Die Strecke sehr kurz ist
- Der Akkustand mehr als ausreichend für die Strecke ist

Alternativen?

Bisher: Energieoptimale Routen

- Energiesparendes Fahren
- Wir finden einen zulässigen Pfad, falls dieser existiert

Problem: Wir versuchen Energie zu sparen selbst wenn:

- Die Strecke sehr kurz ist
- Der Akkustand mehr als ausreichend für die Strecke ist

Alternativen?

- Berechne schnellste Route, überprüfe danach ob SoC ausreichend
- Schnellste Route mit Energieverbrauch als Nebenbedingung

Metric	Unreach.	Extra Energy	Extra T/D
Energy vs. Travel Time	60 %	62 %	63 %
Energy vs. Distance	25 %	15 %	4 %

Fazit:

- Energie explizit optimieren zahlt sich aus
- Kürzeste Wege energieeffizienter als schnellste

Aber:

- Fahrzeit viel höher auf energie-optimalen Wegen
- Nur eine Metrik optimieren ist nicht zufriedenstellend

Metric	Unreach.	Extra Energy	Extra T/D
Energy vs. Travel Time	60 %	62 %	63 %
Energy vs. Distance	25 %	15 %	4 %

Fazit:

- Energie explizit optimieren zahlt sich aus
- Kürzeste Wege energieeffizienter als schnellste

Aber:

- Fahrzeit viel höher auf energie-optimalen Wegen
- Nur eine Metrik optimieren ist nicht zufriedenstellend

⇒ Finde schnellste Route mit Energieverbrauch als Nebenbedingung

Ziel:

- Finde schnellste Route mit Energieverbrauch als Nebenbedingung
- Zwei Metriken auf den Kanten: *Fahrzeit* und *Energieverbrauch*
- Optimierte die Fahrzeit und beschränke den Energieverbrauch
- Erweiterung des *CSP Problems*

Definition: Constrained Shortest Path Problem

Gegeben: $G = (V, E)$, Länge $\ell: E \rightarrow \mathbb{N}_0$, Gewicht $\omega: E \rightarrow \mathbb{N}_0$,
Start und Ziel $s, t \in V$ sowie Schranken $L, W \in \mathbb{N}_0$

Problem: Existiert ein einfacher Pfad P von s nach t in G ,
für den $\ell(P) \leq L$ und $\omega(P) \leq W$ gelten?

Anmerkung: Das entsprechende Optimierungsproblem lautet:

- Finde einen s - t -Pfad P mit minimalem $\ell(P)$ und $\omega(P) \leq W$

Theorem

Constrained Shortest Path Problem ist (schwach) \mathcal{NP} -vollständig

CSP kann mit Multi-Criteria Dijkstra (MCD) gelöst werden

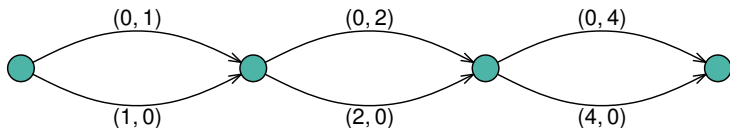
- Bikriterieller Ansatz mit Metriken: Länge & Gewicht
- Nutze Constraints zum prunen
 - Verwerfe Label mit Gewicht $> W$

CSP kann mit Multi-Criteria Dijkstra (MCD) gelöst werden

- Bikriterieller Ansatz mit Metriken: Länge & Gewicht
- Nutze Constraints zum prunen
 - Verwerfe Label mit Gewicht $> W$

Erinnerung:

- MCD hält pro Knoten eine Menge Pareto-Optimaler Pfade
- Pareto-Mengen können exponentiell groß werden

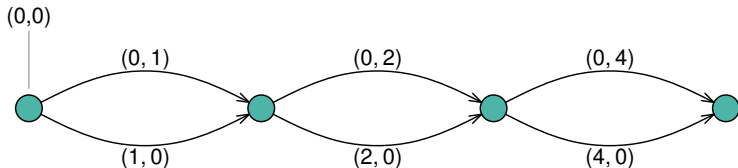


CSP kann mit Multi-Criteria Dijkstra (MCD) gelöst werden

- Bikriterieller Ansatz mit Metriken: Länge & Gewicht
- Nutze Constraints zum prunen
 - Verwerfe Label mit Gewicht $> W$

Erinnerung:

- MCD hält pro Knoten eine Menge Pareto-Optimaler Pfade
- Pareto-Mengen können exponentiell groß werden

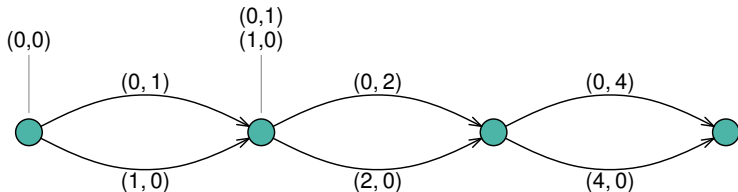


CSP kann mit Multi-Criteria Dijkstra (MCD) gelöst werden

- Bikriterieller Ansatz mit Metriken: Länge & Gewicht
- Nutze Constraints zum prunen
 - Verwerfe Label mit Gewicht $> W$

Erinnerung:

- MCD hält pro Knoten eine Menge Pareto-Optimaler Pfade
- Pareto-Mengen können exponentiell groß werden

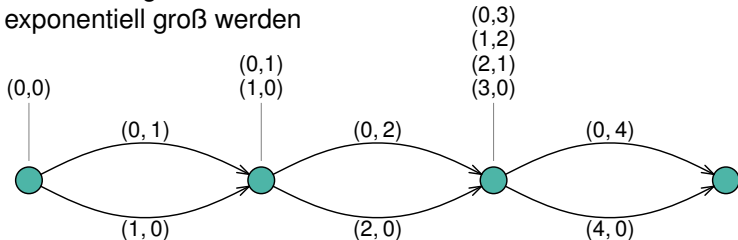


CSP kann mit Multi-Criteria Dijkstra (MCD) gelöst werden

- Bikriterieller Ansatz mit Metriken: Länge & Gewicht
- Nutze Constraints zum prunen
 - Verwerfe Label mit Gewicht $> W$

Erinnerung:

- MCD hält pro Knoten eine Menge Pareto-Optimaler Pfade
- Pareto-Mengen können exponentiell groß werden

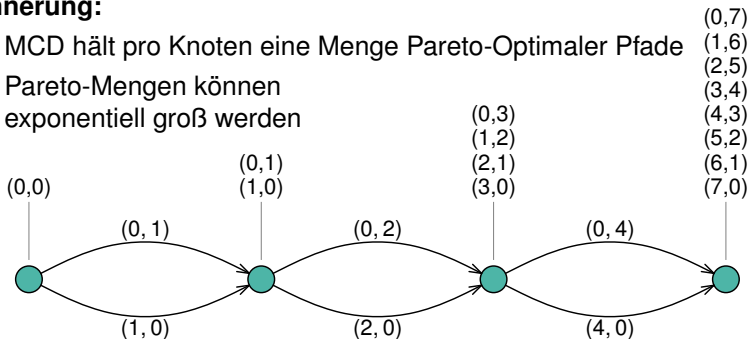


CSP kann mit Multi-Criteria Dijkstra (MCD) gelöst werden

- Bikriterieller Ansatz mit Metriken: Länge & Gewicht
- Nutze Constraints zum prunen
 - Verwerfe Label mit Gewicht $> W$

Erinnerung:

- MCD hält pro Knoten eine Menge Pareto-Optimaler Pfade
- Pareto-Mengen können exponentiell groß werden



Schnellste zulässige Route

Idee:

- Nutze gleichen Ansatz für EV routing
- Label sind Tupel (Fahrzeit, SoC)
- Falls $\text{SoC} < 0$, Pfad nicht weiter verfolgen

Idee:

- Nutze gleichen Ansatz für EV routing
- Label sind Tupel (Fahrzeit, SoC)
- Falls $\text{SoC} < 0$, Pfad nicht weiter verfolgen

Verbesserungen: Standard Beschleunigungen von MCD übertragbar:

- Hopping Reduction
- Nur ein Label pro Knoten in Queue
- Target Pruning (Nutze max. Rekuperation $\underline{d}[t]$)
- Knoten Kontraktionen (Nutze Verbrauchsfunktionen)

Idee:

- Nutze gleichen Ansatz für EV routing
- Label sind Tupel (Fahrzeit, SoC)
- Falls $\text{SoC} < 0$, Pfad nicht weiter verfolgen

Verbesserungen: Standard Beschleunigungen von MCD übertragbar:

- Hopping Reduction
- Nur ein Label pro Knoten in Queue
- Target Pruning (Nutze max. Rekuperation $\underline{d}[t]$)
- Knoten Kontraktionen (Nutze Verbrauchsfunktionen)

Beobachtung: Wir brauchen nicht alle Pareto-Optima an t :

- Sind nur an schnellster zulässiger Route interessiert
- Stoppe sobald erstes Label an t aus Queue genommen (Queue ist nach Fahrzeit sortiert)

Variable Geschwindigkeit:

- Bisher: Kanten mit fester Geschwindigkeit
- Idee: Erlaube langsamer zu fahren
- Ermöglicht Trade-off zwischen Fahrzeit und Energieverbrauch
- Mögliche Umsetzungen:
 - Multikanten mit verschiedenen Geschwindigkeits-/Verbrauchswerten
 - Funktionen an Kanten, die Fahrzeit auf Verbrauch abbilden

Ladestationen:

- Akku Kapazität ist stark begrenzt (~100 km, max. 400 km)
- Lange Strecken unmöglich, selbst mit verbrauchsoptimalen Routen
- Nutzung von Ladestationen nicht zu vermeiden
- Problem: Ladestationen sind langsam und wenig verbreitet

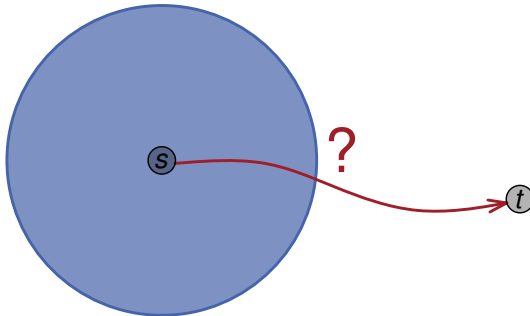
Finde schnellste Route von s nach t :



 Erreichbares Gebiet

 Ladestation

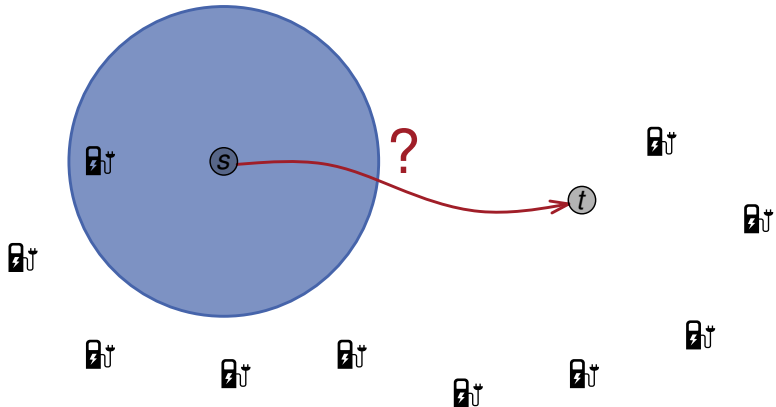
Finde schnellste Route von s nach t :



 Erreichbares Gebiet

 Ladestation

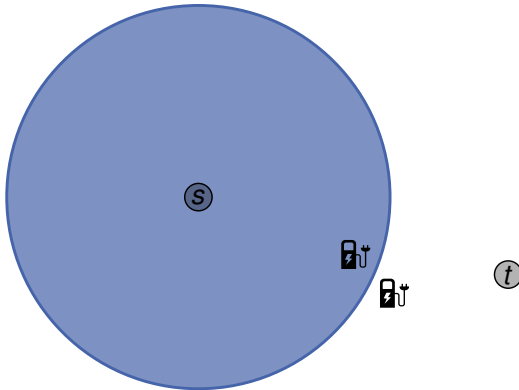
Finde schnellste Route von s nach t :



 Erreichbares Gebiet

 Ladestation

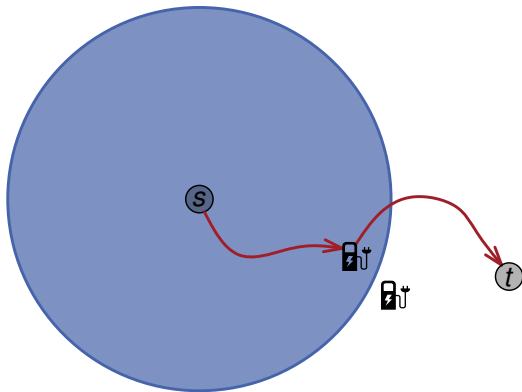
Finde schnellste Route von s nach t :



 Erreichbares Gebiet

 Ladestation

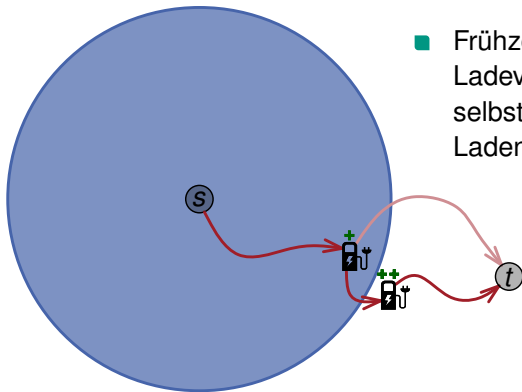
Finde schnellste Route von s nach t :



■ Erreichbares Gebiet


🔋 Ladestation


Finde schnellste Route von s nach t :



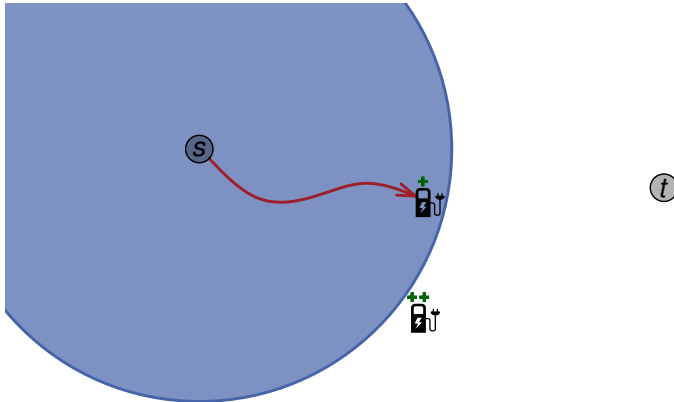
- Frühzeitiger Abbruch des Ladevorgangs kann lohnen, selbst wenn Ziel bei weiterem Laden direkt erreichbar wäre

 Erreichbares Gebiet


 Ladestation


 Super Charger / Swapping Station

Finde schnellste Route von s nach t :

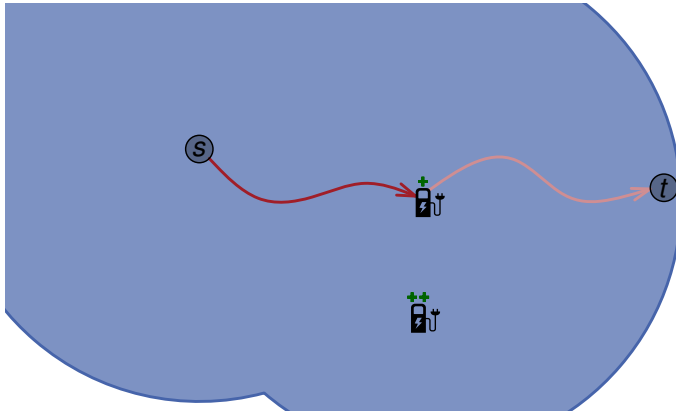


 Erreichbares Gebiet


 Ladestation


 Super Charger / Swapping Station

Finde schnellste Route von s nach t :

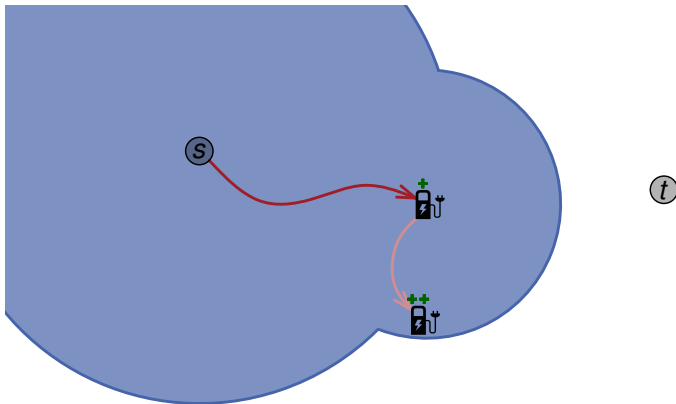


 Erreichbares Gebiet


 Ladestation


 Super Charger / Swapping Station

Finde schnellste Route von s nach t :

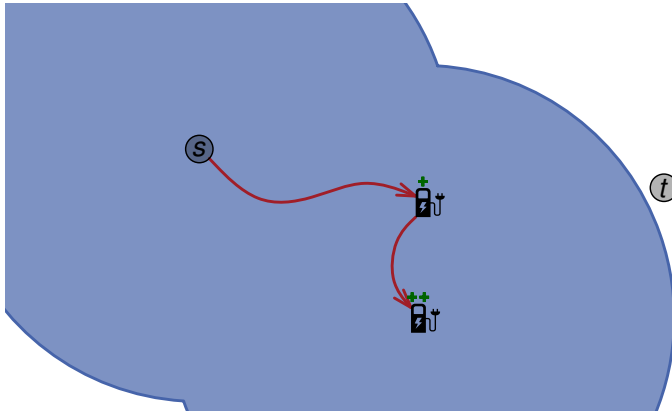


 Erreichbares Gebiet


 Ladestation


 Super Charger / Swapping Station

Finde schnellste Route von s nach t :

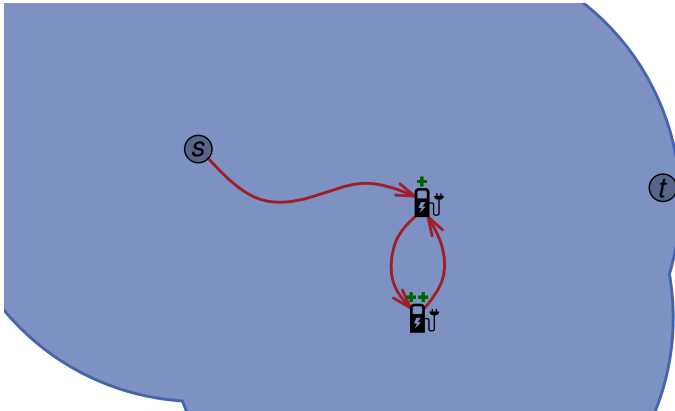


 Erreichbares Gebiet


 Ladestation


 Super Charger / Swapping Station

Finde schnellste Route von s nach t :



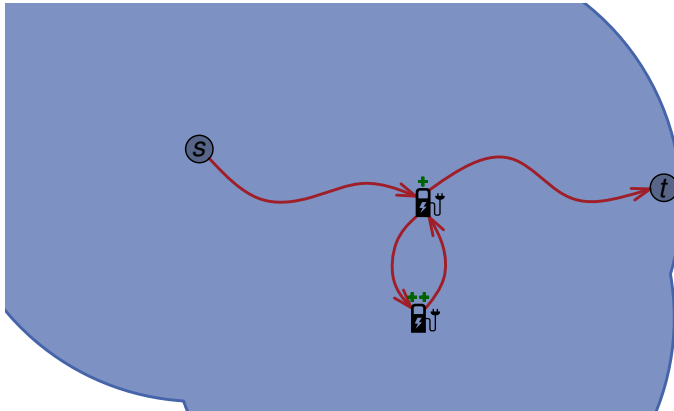
 Erreichbares Gebiet

 Ladestation


 Super Charger / Swapping Station


Finde schnellste Route von s nach t :

■ Zyklen möglich



 Erreichbares Gebiet

 Ladestation

 Super Charger / Swapping Station

Schwierigkeiten:

- Laden dauert lange (\Rightarrow lieber Energie sparen, laden vermeiden)
- Ladestationen sind selten (lohnt sich ein Umweg?)
- Laden jederzeit unterbrechbar

Ansatz:

- Ladezeiten müssen während Routenplanung berücksichtigt werden
- Optimierte Reisezeit = Fahrzeit + Ladezeit
- Teilmenge $S \subseteq V$ der Knoten sind Ladestationen
- Für jede Station eine Funktion, die das Ladeverhalten beschreibt

Formal:

- Eine Funktion $c_f: [0, M] \times \mathbb{R}_{\geq 0} \rightarrow [0, M]$, bildet
 - Initialen SoC b_s und
 - Gewünschte Ladezeit τ_c auf
 - Durch laden erreichten SoC ab
- Monoton steigend (Länger laden \Rightarrow mehr Energie)
- Konkav (Akku voller \Rightarrow langsames Laden)

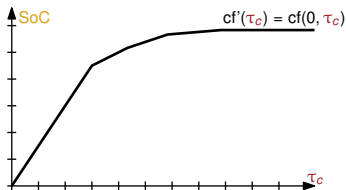
Formal:

- Eine Funktion $cf: [0, M] \times \mathbb{R}_{\geq 0} \rightarrow [0, M]$, bildet
 - Initialen SoC b_s und
 - Gewünschte Ladezeit τ_c auf
 - Durch laden erreichten SoC ab
- Monoton steigend (Länger laden \Rightarrow mehr Energie)
- Konkav (Akku voller \Rightarrow langsames Laden)

Anmerkung: Realistische Ladefunktionen darstellbar durch:

- Univariate Funktion $cf': \mathbb{R}_{\geq 0} \rightarrow [0, M]$

$$cf(b, \tau_c) := cf'(\tau_c + cf'^{-1}(b))$$



Formal:

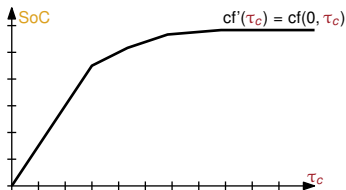
- Eine Funktion $cf: [0, M] \times \mathbb{R}_{\geq 0} \rightarrow [0, M]$, bildet
 - Initialen SoC b_s und
 - Gewünschte Ladezeit τ_c auf
 - Durch laden erreichten SoC ab
- Monoton steigend (Länger laden \Rightarrow mehr Energie)
- Konkav (Akku voller \Rightarrow langsames Laden)

Anmerkung: Realistische Ladefunktionen darstellbar durch:

- Univariate Funktion $cf': \mathbb{R}_{\geq 0} \rightarrow [0, M]$

$$cf(b, \tau_c) := cf'(\tau_c + cf'^{-1}(b))$$

$$cf(3, 2)$$



Formal:

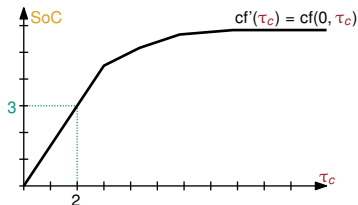
- Eine Funktion $cf: [0, M] \times \mathbb{R}_{\geq 0} \rightarrow [0, M]$, bildet
 - Initialen SoC b_s und
 - Gewünschte Ladezeit τ_c auf
 - Durch laden erreichten SoC ab
- Monoton steigend (Länger laden \Rightarrow mehr Energie)
- Konkav (Akku voller \Rightarrow langsames Laden)

Anmerkung: Realistische Ladefunktionen darstellbar durch:

- Univariate Funktion $cf': \mathbb{R}_{\geq 0} \rightarrow [0, M]$

$$cf(b, \tau_c) := cf'(\tau_c + cf'^{-1}(b))$$

$$cf(3, 2) = cf'(2 + cf'^{-1}(3))$$



Formal:

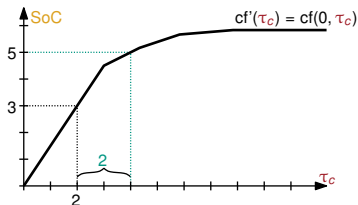
- Eine Funktion $cf: [0, M] \times \mathbb{R}_{\geq 0} \rightarrow [0, M]$, bildet
 - Initialen SoC b_s und
 - Gewünschte Ladezeit τ_c auf
 - Durch laden erreichten SoC ab
- Monoton steigend (Länger laden \Rightarrow mehr Energie)
- Konkav (Akku voller \Rightarrow langsames Laden)

Anmerkung: Realistische Ladefunktionen darstellbar durch:

- Univariate Funktion $cf': \mathbb{R}_{\geq 0} \rightarrow [0, M]$

$$cf(b, \tau_c) := cf'(\tau_c + cf'^{-1}(b))$$

$$\begin{aligned} cf(3, 2) &= cf'(2 + cf'^{-1}(3)) \\ &= cf'(2 + 2) \end{aligned}$$



Formal:

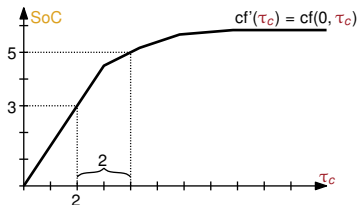
- Eine Funktion $cf: [0, M] \times \mathbb{R}_{\geq 0} \rightarrow [0, M]$, bildet
 - Initialen SoC b_s und
 - Gewünschte Ladezeit τ_c auf
 - Durch laden erreichten SoC ab
- Monoton steigend (Länger laden \Rightarrow mehr Energie)
- Konkav (Akku voller \Rightarrow langsames Laden)

Anmerkung: Realistische Ladefunktionen darstellbar durch:

- Univariate Funktion $cf': \mathbb{R}_{\geq 0} \rightarrow [0, M]$

$$cf(b, \tau_c) := cf'(\tau_c + cf'^{-1}(b))$$

$$\begin{aligned} cf(3, 2) &= cf'(2 + cf'^{-1}(3)) \\ &= cf'(2 + 2) \\ &= 5 \end{aligned}$$



Algorithmus:

- Basiert auf Dijkstra's Algorithmus bzw. MCD
- Solange keine Ladestation Besucht: Label = Tupel (Reisezeit, SoC)
- Battery Constraints, Pareto-Optimierung wie bisher

Algorithmus:

- Basiert auf Dijkstra's Algorithmus bzw. MCD
- Solange keine Ladestation Besucht: Label = Tupel (Reisezeit, SoC)
- Battery Constraints, Pareto-Optimierung wie bisher

Problem: Wenn Ladestation erreicht: Wie lange laden?

- Hängt vom gewähltem Pfad zu t ab
- Optimaler SoC zum Weiterfahren unbekannt

Algorithmus:

- Basiert auf Dijkstra's Algorithmus bzw. MCD
- Solange keine Ladestation Besucht: Label = Tupel (Reisezeit, SoC)
- Battery Constraints, Pareto-Optimierung wie bisher

Problem: Wenn Ladestation erreicht: Wie lange laden?

- Hängt vom gewähltem Pfad zu t ab
- Optimaler SoC zum Weiterfahren unbekannt

Lösung:

- Verschiebe die Entscheidung auf später!
- Merke zuletzt gesehene Ladestation

Label: Ein Label ℓ am Knoten v ist ein Tupel $(\tau_t, b_u, u, c_{(u, \dots, v)})$ mit:

- Reisezeit τ_t von s nach v (Inklusive Ladezeiten außer an u)
- SoC b_u mit dem die letzte Ladestation (u) erreicht wurde
- Zu letzt passierte Ladestation u (Initial \perp)
- Verbrauchsfunktion $c_{(u, \dots, v)}$ für den Pfad von u nach v (Initial \perp)

Label: Ein Label ℓ am Knoten v ist ein Tupel $(\tau_t, b_u, u, c_{(u, \dots, v)})$ mit:

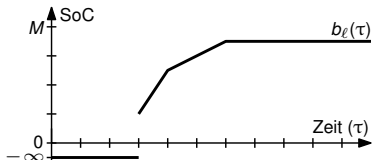
- Reisezeit τ_t von s nach v (Inklusive Ladezeiten außer an u)
- SoC b_u mit dem die letzte Ladestation (u) erreicht wurde
- Zu letzt passierte Ladestation u (Initial \perp)
- Verbrauchsfunktion $c_{(u, \dots, v)}$ für den Pfad von u nach v (Initial \perp)

Interpretation:

- Label beschreibt eine *verschobene* Ladefunktion
- Bildet Reisezeit auf SoC ab (Daher auch SoC-Funktion genannt)
- Funktion repräsentiert Menge von Pareto-Optimalen Punkten
- Definition der SoC-Funktion $b_\ell(\tau)$:

$$b_\ell(\tau) := b' - c_{(u, \dots, v)}(b')$$

$$b' := \text{cf}_u(b_u, \tau - \tau_t)$$



Label: Ein Label ℓ am Knoten v ist ein Tupel $(\tau_t, b_u, u, c_{(u, \dots, v)})$ mit:

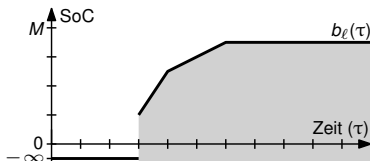
- **Reisezeit** τ_t von s nach v (Inklusive Ladezeiten außer an u)
- **SoC** b_u mit dem die letzte Ladestation (u) erreicht wurde
- Zu letzt passierte **Ladestation** u (Initial \perp)
- **Verbrauchsfunktion** $c_{(u, \dots, v)}$ für den Pfad von u nach v (Initial \perp)

Interpretation:

- Label beschreibt eine *verschobene* Ladefunktion
- Bildet Reisezeit auf SoC ab (Daher auch SoC-Funktion genannt)
- Funktion repräsentiert Menge von Pareto-Optimalen Punkten
- Definition der SoC-Funktion $b_\ell(\tau)$:

$$b_\ell(\tau) := b' - c_{(u, \dots, v)}(b')$$

$$b' := \text{cf}_u(b_u, \tau - \tau_t)$$

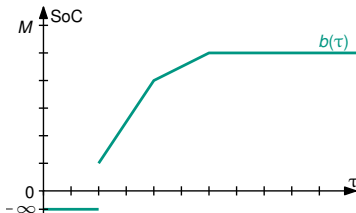


Kanten Relaxierung: (Label $\ell = (\tau_t, b_U, u, c_{(u, \dots, v)})$)

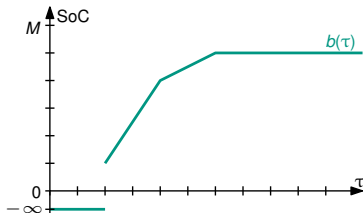
- Relaxieren der Kante $e = (v, w)$ verschiebt die SoC-Funktion $b_\ell(\tau)$
 - $b_\ell(\tau)$ wird um Fahrzeit $\tau_d(e)$ nach rechts verschoben
 - $b_\ell(\tau)$ wird um Verbrauch $\gamma(e)$ nach unten verschoben
- Anschließend werden Battery Constraints überprüft

Formal: $\tau_t \leftarrow \tau_t + \tau_d(e)$ und $c_{(u, \dots, w)} \leftarrow c_{(u, \dots, v)} \circ c_e$

$$\tau_d(e) = 3, \gamma(e) = 2$$



$$\tau_d(e) = 3, \gamma(e) = -2$$

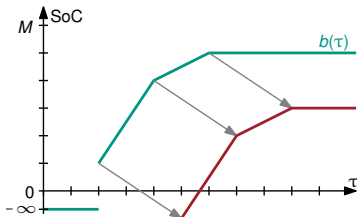


Kanten Relaxierung: (Label $\ell = (\tau_t, b_U, u, c_{(u, \dots, v)})$)

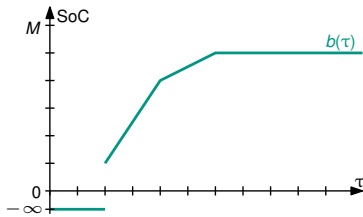
- Relaxieren der Kante $e = (v, w)$ verschiebt die SoC-Funktion $b_\ell(\tau)$
 - $b_\ell(\tau)$ wird um Fahrzeit $\tau_d(e)$ nach rechts verschoben
 - $b_\ell(\tau)$ wird um Verbrauch $\gamma(e)$ nach unten verschoben
- Anschließend werden Battery Constraints überprüft

Formal: $\tau_t \leftarrow \tau_t + \tau_d(e)$ und $c_{(u, \dots, w)} \leftarrow c_{(u, \dots, v)} \circ c_e$

$$\tau_d(e) = 3, \gamma(e) = 2$$



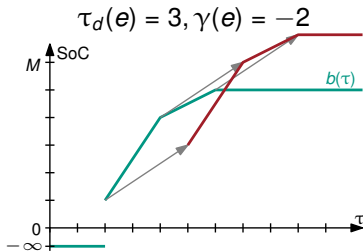
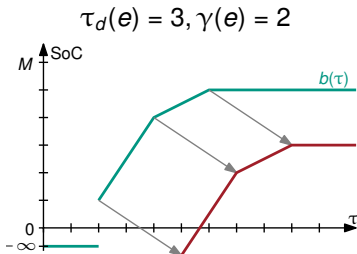
$$\tau_d(e) = 3, \gamma(e) = -2$$



Kanten Relaxierung: (Label $\ell = (\tau_t, b_U, u, c_{(u, \dots, v)})$)

- Relaxieren der Kante $e = (v, w)$ verschiebt die SoC-Funktion $b_\ell(\tau)$
 - $b_\ell(\tau)$ wird um Fahrzeit $\tau_d(e)$ nach rechts verschoben
 - $b_\ell(\tau)$ wird um Verbrauch $\gamma(e)$ nach unten verschoben
- Anschließend werden Battery Constraints überprüft

Formal: $\tau_t \leftarrow \tau_t + \tau_d(e)$ und $c_{(u, \dots, w)} \leftarrow c_{(u, \dots, v)} \circ c_e$

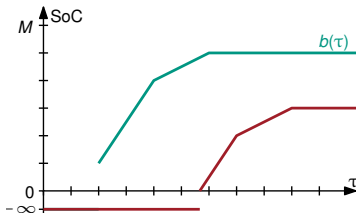


Kanten Relaxierung: (Label $\ell = (\tau_t, b_U, u, c_{(u, \dots, v)})$)

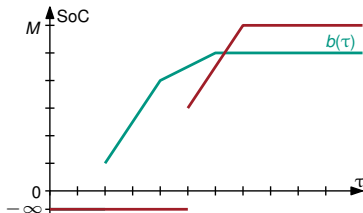
- Relaxieren der Kante $e = (v, w)$ verschiebt die SoC-Funktion $b_\ell(\tau)$
 - $b_\ell(\tau)$ wird um Fahrzeit $\tau_d(e)$ nach rechts verschoben
 - $b_\ell(\tau)$ wird um Verbrauch $\gamma(e)$ nach unten verschoben
- Anschließend werden Battery Constraints überprüft

Formal: $\tau_t \leftarrow \tau_t + \tau_d(e)$ und $c_{(u, \dots, w)} \leftarrow c_{(u, \dots, v)} \circ c_e$

$$\tau_d(e) = 3, \gamma(e) = 2$$

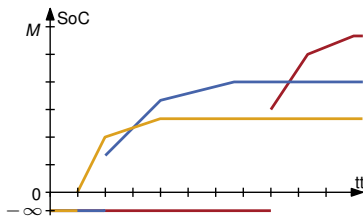


$$\tau_d(e) = 3, \gamma(e) = -2$$



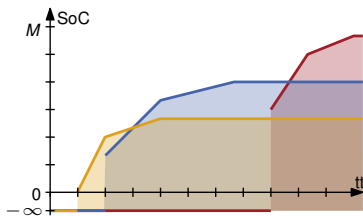
Dominanz von SoC-Funktionen:

- Eine SoC-Funktion $b_e(\tau)$ dominiert $b_e(\tau)$ ($b_e(\tau) \propto b_e(\tau)$) gdw.:
$$\forall \tau \geq 0: b_e(\tau) \geq b_e(\tau)$$
- Pro Knoten eine Pareto-Menge von SoC-Funktionen
- Ein neues Label wird erzeugt (Durch eine Kanten Relaxierung)
⇒ Überprüfe Dominanz,
Lösche dominierte Label



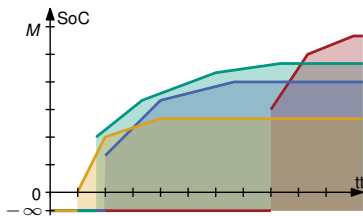
Dominanz von SoC-Funktionen:

- Eine SoC-Funktion $b_e(\tau)$ dominiert $b_e(\tau)$ ($b_e(\tau) \propto b_e(\tau)$) gdw.:
$$\forall \tau \geq 0: b_e(\tau) \geq b_e(\tau)$$
- Pro Knoten eine Pareto-Menge von SoC-Funktionen
- Ein neues Label wird erzeugt (Durch eine Kanten Relaxierung)
⇒ Überprüfe Dominanz,
Lösche dominierte Label



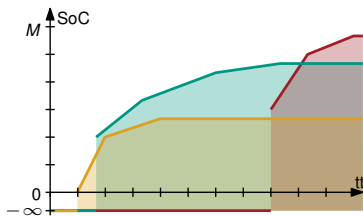
Dominanz von SoC-Funktionen:

- Eine SoC-Funktion $b_e(\tau)$ dominiert $b_e(\tau)$ ($b_e(\tau) \propto b_e(\tau)$) gdw.:
$$\forall \tau \geq 0: b_e(\tau) \geq b_e(\tau)$$
- Pro Knoten eine Pareto-Menge von SoC-Funktionen
- Ein neues Label wird erzeugt (Durch eine Kanten Relaxierung)
⇒ Überprüfe Dominanz,
Lösche dominierte Label



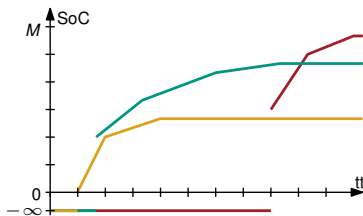
Dominanz von SoC-Funktionen:

- Eine SoC-Funktion $b_e(\tau)$ dominiert $b_e(\tau)$ ($b_e(\tau) \propto b_e(\tau)$) gdw.:
$$\forall \tau \geq 0: b_e(\tau) \geq b_e(\tau)$$
- Pro Knoten eine Pareto-Menge von SoC-Funktionen
- Ein neues Label wird erzeugt (Durch eine Kanten Relaxierung)
⇒ Überprüfe Dominanz,
Lösche dominierte Label



Dominanz von SoC-Funktionen:

- Eine SoC-Funktion $b_e(\tau)$ dominiert $b_e(\tau)$ ($b_e(\tau) \propto b_e(\tau)$) gdw.:
$$\forall \tau \geq 0: b_e(\tau) \geq b_e(\tau)$$
- Pro Knoten eine Pareto-Menge von SoC-Funktionen
- Ein neues Label wird erzeugt (Durch eine Kanten Relaxierung)
⇒ Überprüfe Dominanz,
Lösche dominierte Label



Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation \Rightarrow Bestimme τ_C für die letzte Station

Problem:

- Am ursprünglichem Problem hat sich nichts geändert
- Auch für vorletzte Ladestation ist die Ladezeit unklar

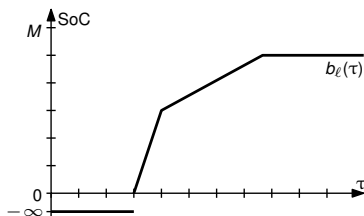
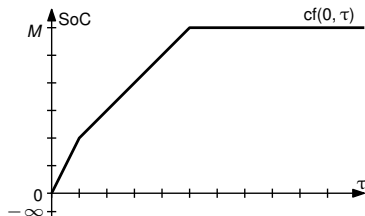
Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation \Rightarrow Bestimme τ_C für die letzte Station

Problem:

- Am ursprünglichem Problem hat sich nichts geändert
- Auch für vorletzte Ladestation ist die Ladezeit unklar

Aber: Wechsel der Station nur sinnvoll, wenn neue Station besser



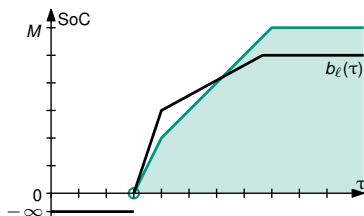
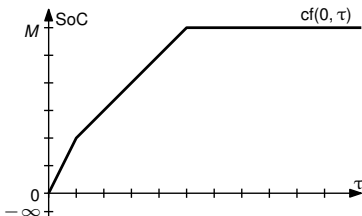
Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation \Rightarrow Bestimme τ_C für die letzte Station

Problem:

- Am ursprünglichem Problem hat sich nichts geändert
- Auch für vorletzte Ladestation ist die Ladezeit unklar

Aber: Wechsel der Station nur sinnvoll, wenn neue Station besser



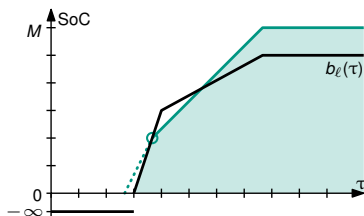
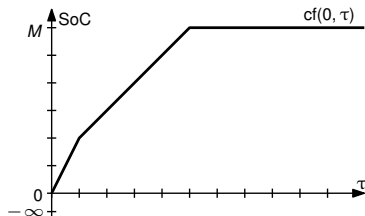
Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation \Rightarrow Bestimme τ_C für die letzte Station

Problem:

- Am ursprünglichem Problem hat sich nichts geändert
- Auch für vorletzte Ladestation ist die Ladezeit unklar

Aber: Wechsel der Station nur sinnvoll, wenn neue Station besser



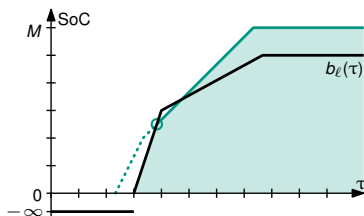
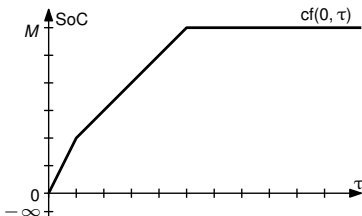
Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation \Rightarrow Bestimme τ_C für die letzte Station

Problem:

- Am ursprünglichem Problem hat sich nichts geändert
- Auch für vorletzte Ladestation ist die Ladezeit unklar

Aber: Wechsel der Station nur sinnvoll, wenn neue Station besser



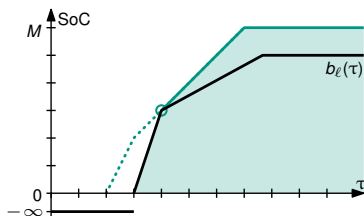
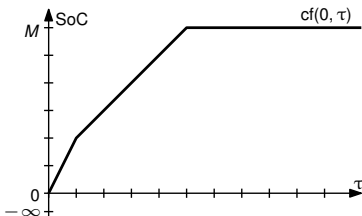
Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation \Rightarrow Bestimme τ_C für die letzte Station

Problem:

- Am ursprünglichem Problem hat sich nichts geändert
- Auch für vorletzte Ladestation ist die Ladezeit unklar

Aber: Wechsel der Station nur sinnvoll, wenn neue Station besser



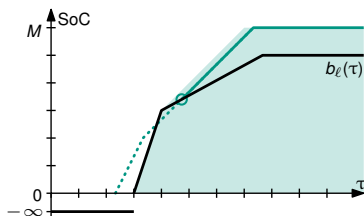
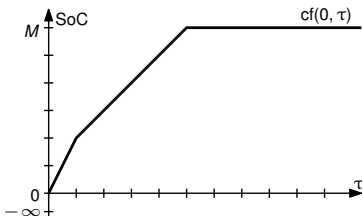
Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation \Rightarrow Bestimme τ_C für die letzte Station

Problem:

- Am ursprünglichem Problem hat sich nichts geändert
- Auch für vorletzte Ladestation ist die Ladezeit unklar

Aber: Wechsel der Station nur sinnvoll, wenn neue Station besser



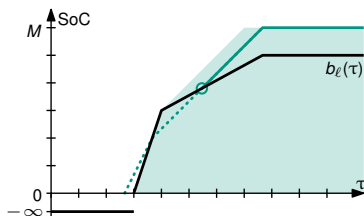
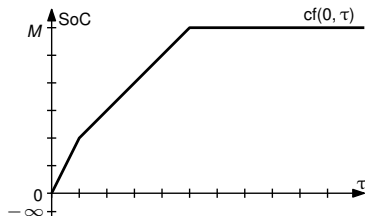
Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation \Rightarrow Bestimme τ_C für die letzte Station

Problem:

- Am ursprünglichem Problem hat sich nichts geändert
- Auch für vorletzte Ladestation ist die Ladezeit unklar

Aber: Wechsel der Station nur sinnvoll, wenn neue Station besser



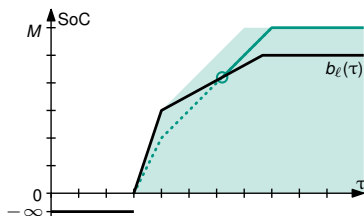
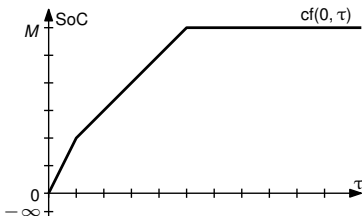
Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation \Rightarrow Bestimme τ_C für die letzte Station

Problem:

- Am ursprünglichem Problem hat sich nichts geändert
- Auch für vorletzte Ladestation ist die Ladezeit unklar

Aber: Wechsel der Station nur sinnvoll, wenn neue Station besser



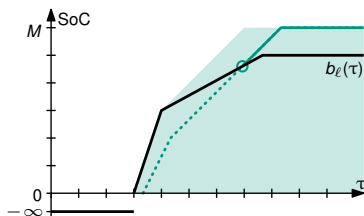
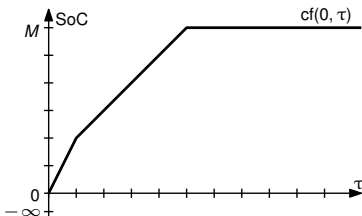
Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation \Rightarrow Bestimme τ_C für die letzte Station

Problem:

- Am ursprünglichem Problem hat sich nichts geändert
- Auch für vorletzte Ladestation ist die Ladezeit unklar

Aber: Wechsel der Station nur sinnvoll, wenn neue Station besser



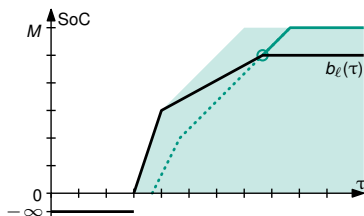
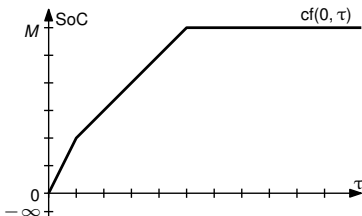
Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation \Rightarrow Bestimme τ_C für die letzte Station

Problem:

- Am ursprünglichem Problem hat sich nichts geändert
- Auch für vorletzte Ladestation ist die Ladezeit unklar

Aber: Wechsel der Station nur sinnvoll, wenn neue Station besser



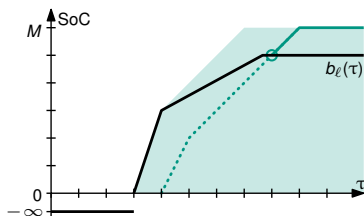
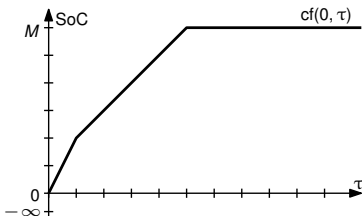
Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation \Rightarrow Bestimme τ_C für die letzte Station

Problem:

- Am ursprünglichem Problem hat sich nichts geändert
- Auch für vorletzte Ladestation ist die Ladezeit unklar

Aber: Wechsel der Station nur sinnvoll, wenn neue Station besser



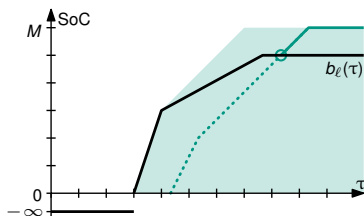
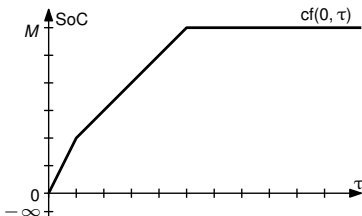
Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation \Rightarrow Bestimme τ_C für die letzte Station

Problem:

- Am ursprünglichem Problem hat sich nichts geändert
- Auch für vorletzte Ladestation ist die Ladezeit unklar

Aber: Wechsel der Station nur sinnvoll, wenn neue Station besser



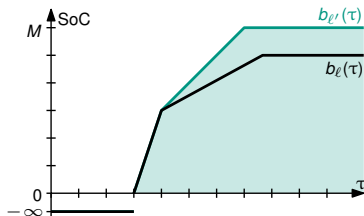
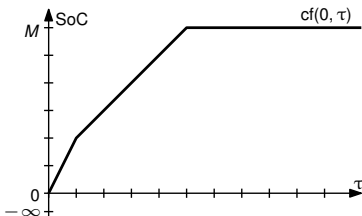
Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation \Rightarrow Bestimme τ_C für die letzte Station

Problem:

- Am ursprünglichem Problem hat sich nichts geändert
- Auch für vorletzte Ladestation ist die Ladezeit unklar

Aber: Wechsel der Station nur sinnvoll, wenn neue Station besser



Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation \Rightarrow Bestimme τ_C für die letzte Station

Problem:

- Am ursprünglichem Problem hat sich nichts geändert
- Auch für vorletzte Ladestation ist die Ladezeit unklar

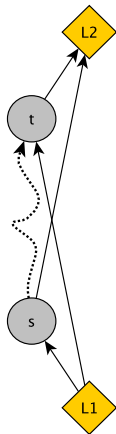
Wechsel der Ladestation lohnt sich nur an Stützpunkten von $b_\ell(\tau)$

Gegeben:

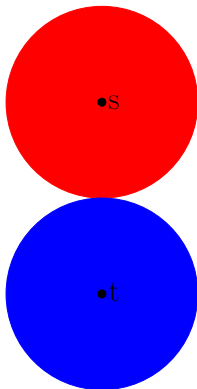
- Label $\ell = (\tau_t, b_u, u, c_{(u, \dots, v)})$ an Knoten v
- v ist Ladestation
- τ ist Stützstelle von b_ℓ

\Rightarrow Erzeuge neues Label $\ell' = (\tau, b_\ell(\tau), v, 0)$

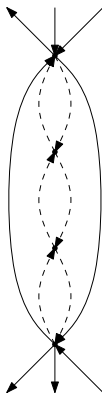
Landmarken



Bidirektionale Suche



Kontraktion



Arc-Flags

