

# Algorithmen für Routenplanung

16. Vorlesung, Sommersemester 2019

Tobias Zündorf | 26. Juni 2019

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



## 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

## **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

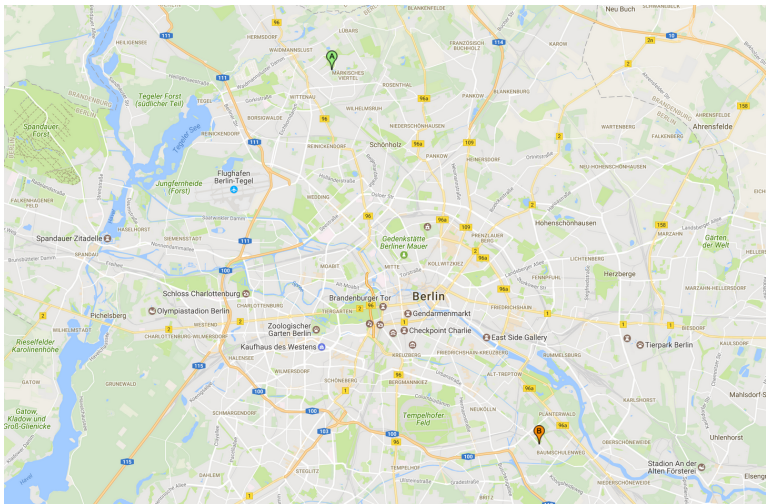
## **Problem:** Wir versuchen Energie zu sparen selbst wenn:

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

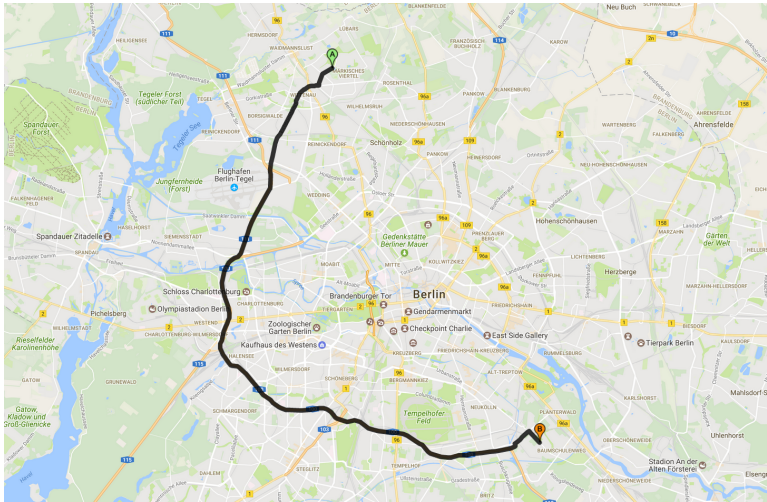
## **Alternativen?**

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

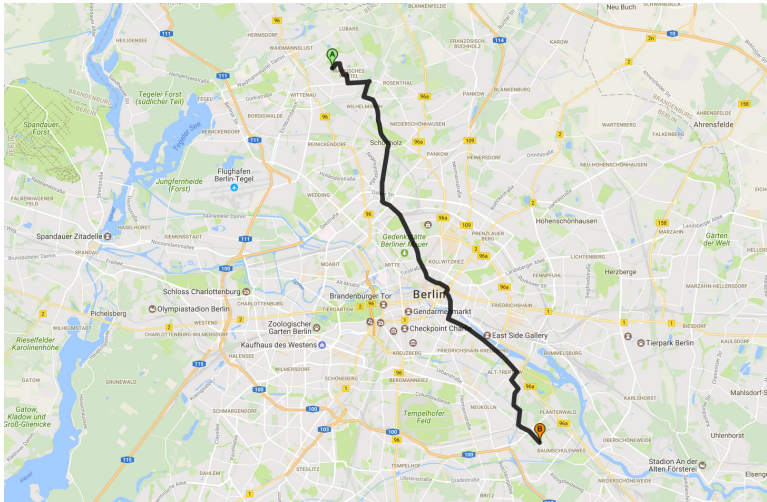
# Beispielrouten



# Beispielrouten



# Beispielrouten





Optimierte Metrik	Unerreichbar	Extra Energie	Extra Zeit/Dist.
Fahrzeit	60 %	62 %	63 %
Distanz	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

Optimierte Metrik	Unerreichbar	Extra Energie	Extra Zeit/Dist.
Fahrzeit	60 %	62 %	63 %
Distanz	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

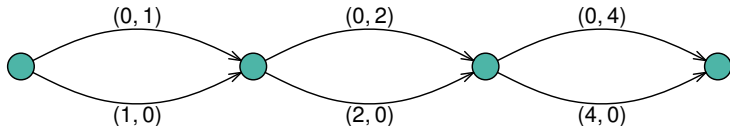
- Bikriterieller Ansatz mit Metriken: Länge & Gewicht
- Nutze Constraints zum prunen
  - Verwerfe Label mit Gewicht  $> W$

## 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

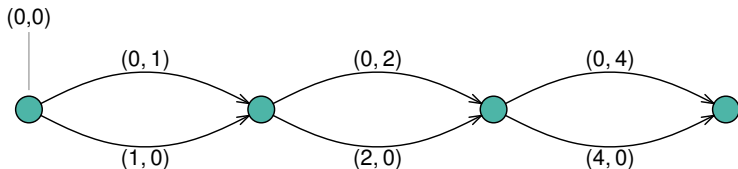


## CSP kann mit Multi-Criteria Dijkstra (MCD) gelöst werden

- 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

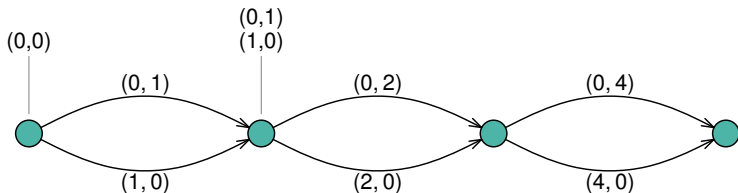


## CSP kann mit Multi-Criteria Dijkstra (MCD) gelöst werden

- 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



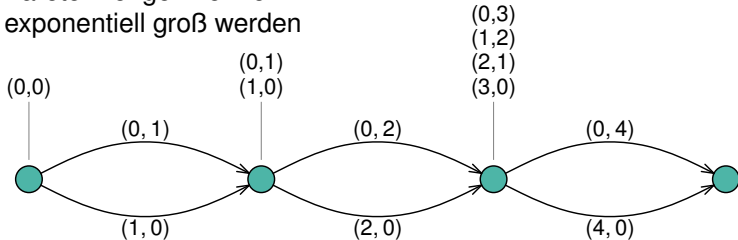


## CSP kann mit Multi-Criteria Dijkstra (MCD) gelöst werden

- 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

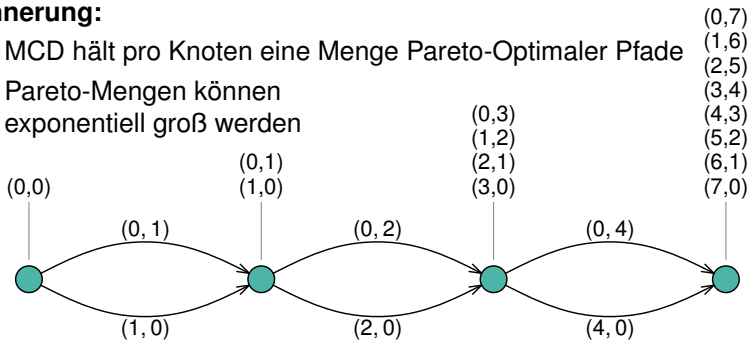


## CSP kann mit Multi-Criteria Dijkstra (MCD) gelöst werden

- 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)

Mögliche (alternative) Problemstellungen:

- Finde die schnellste Route, so dass das Ziel erreichbar ist
- Finde die schnellstmögliche Route, so dass der SoC am Ziel mindestens  $x\%$  ist
- Finde die schnellstmögliche Route, so dass der SoC einen bestimmten SoC niemals unterschreitet
- Finde eine Route mit minimalem Verbrauch, wobei eine vorgegebene Zeit nicht überschritten wird
- Berechne alle **nichtdominierten** Lösungen

Alle Probleme sind  $\mathcal{NP}$ -schwer

⇒ Auch mit Speedup Techniken ggf nur Laufzeiten im Sekundenbereich

⇒ Heuristiken

## 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

# Ladestopps





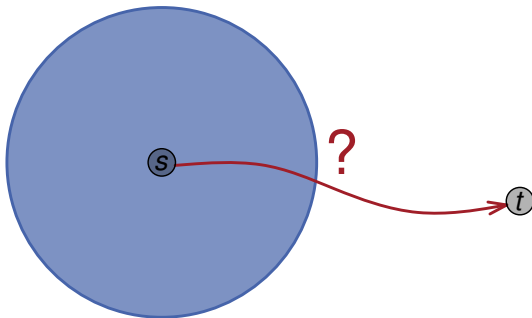
Finde schnellste Route von  $s$  nach  $t$ :



 Erreichbares Gebiet

 Ladestation

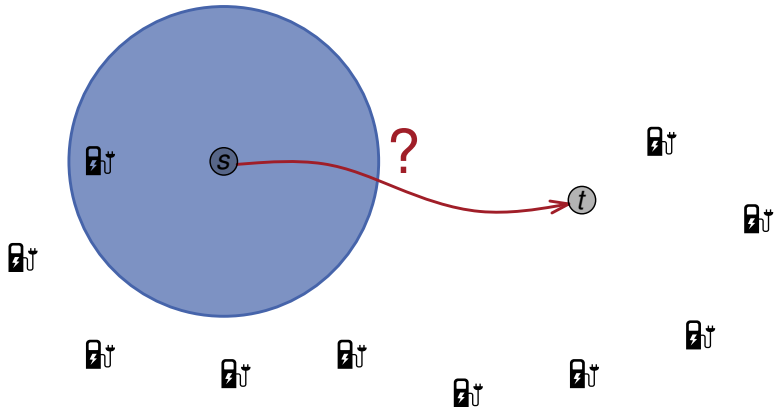
Finde schnellste Route von  $s$  nach  $t$ :



 Erreichbares Gebiet

 Ladestation

Finde schnellste Route von  $s$  nach  $t$ :

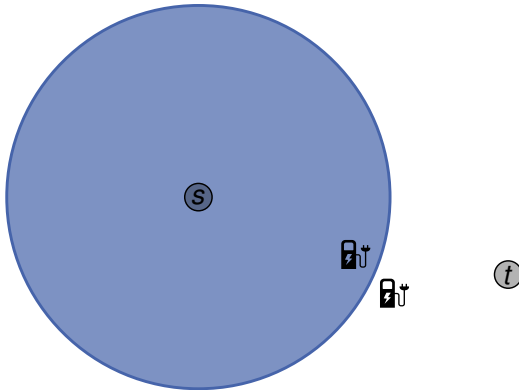


 Erreichbares Gebiet

 Ladestation



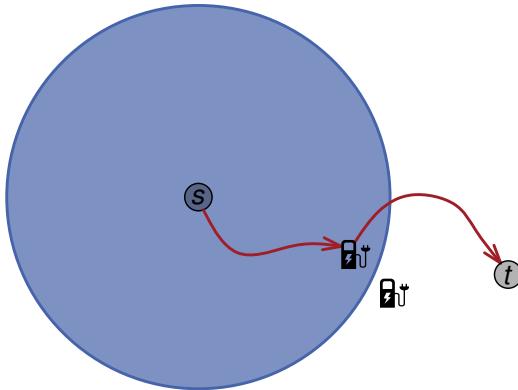
Finde schnellste Route von  $s$  nach  $t$ :



 Erreichbares Gebiet

 Ladestation

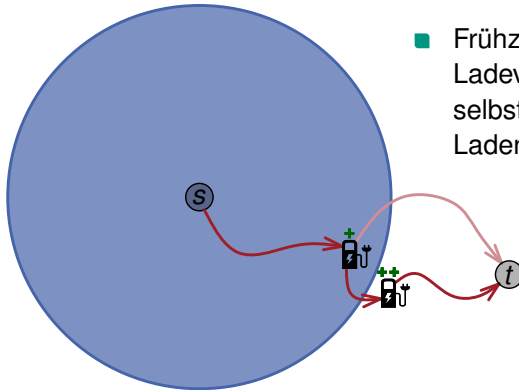
Finde schnellste Route von  $s$  nach  $t$ :



 Erreichbares Gebiet


 Ladestation


Finde schnellste Route von  $s$  nach  $t$ :



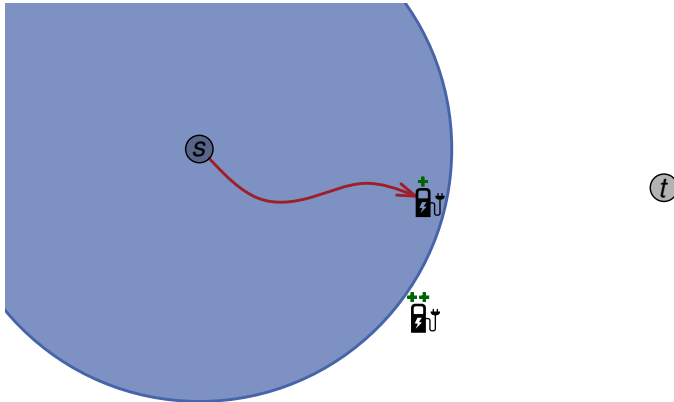
- Frühzeitiger Abbruch des Ladevorgangs kann lohnen, selbst wenn Ziel bei weiterem Laden direkt erreichbar wäre

 Erreichbares Gebiet


 Ladestation


 Super Charger / Swapping Station

Finde schnellste Route von  $s$  nach  $t$ :



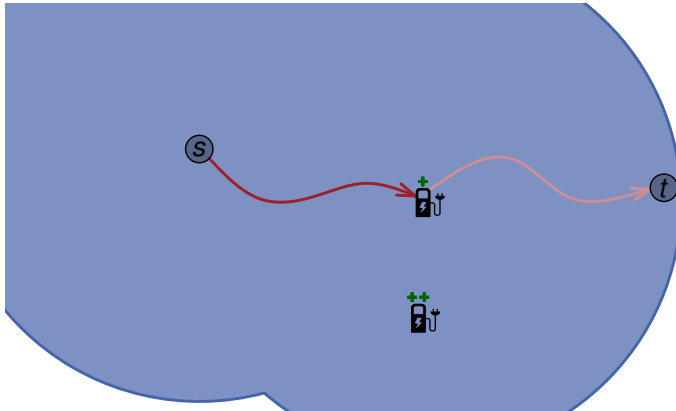
 Erreichbares Gebiet

 Ladestation


 Super Charger / Swapping Station




Finde schnellste Route von  $s$  nach  $t$ :

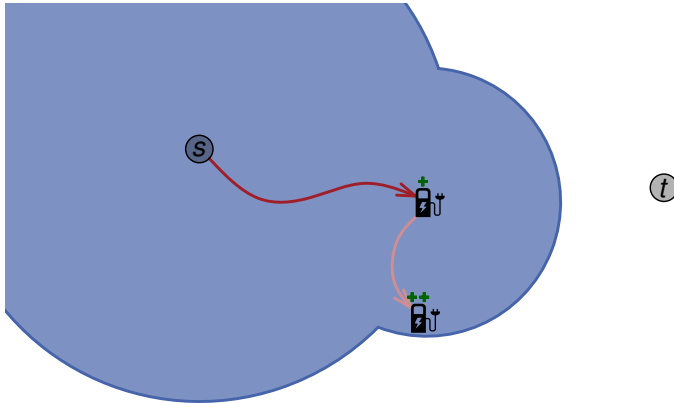


 Erreichbares Gebiet


 Ladestation


 Super Charger / Swapping Station

Finde schnellste Route von  $s$  nach  $t$ :

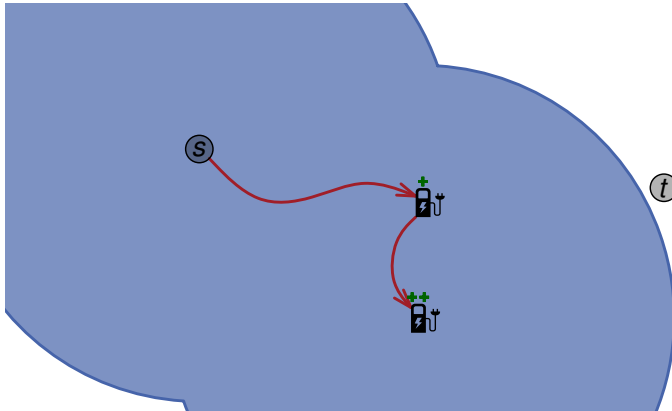


 Erreichbares Gebiet


 Ladestation


 Super Charger / Swapping Station

Finde schnellste Route von  $s$  nach  $t$ :

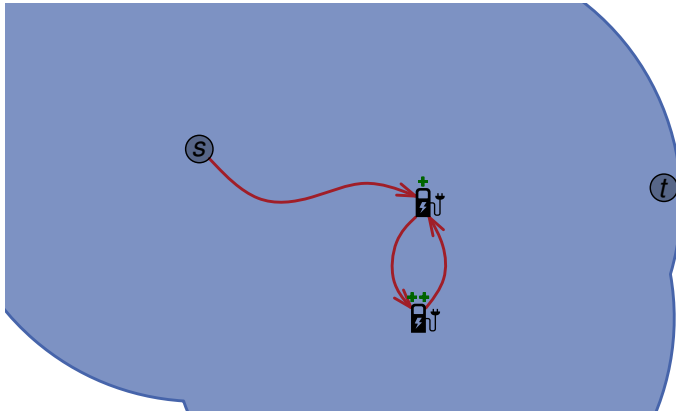


 Erreichbares Gebiet


 Ladestation


 Super Charger / Swapping Station

Finde schnellste Route von  $s$  nach  $t$ :



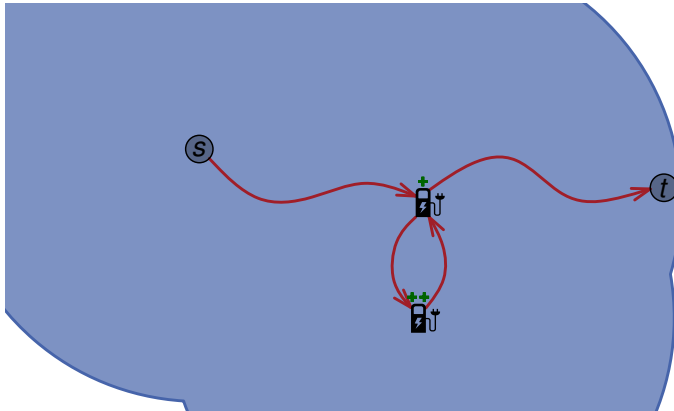
 Erreichbares Gebiet

 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:

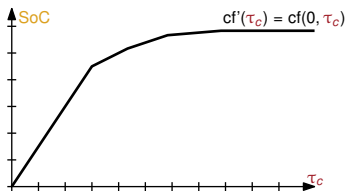
- Eine Funktion  $cf: [0, M] \times \mathbb{R}_{\geq 0} \rightarrow [0, M]$ , bildet
  - Initialen SoC  $b_s$  und
  - Gewünschte Ladezeit  $\tau_c$  auf
  - Durch laden erreichten SoC ab
- Monoton steigend (Länger laden  $\Rightarrow$  mehr Energie)
- Konkav (Akku voller  $\Rightarrow$  langsames Laden)

## Formal:

- Eine Funktion  $cf: [0, M] \times \mathbb{R}_{\geq 0} \rightarrow [0, M]$ , bildet
  - Initialen SoC  $b_s$  und
  - Gewünschte Ladezeit  $\tau_c$  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  $cf': \mathbb{R}_{\geq 0} \rightarrow [0, M]$   
 $cf(b, \tau_c) := cf'(\tau_c + cf'^{-1}(b))$



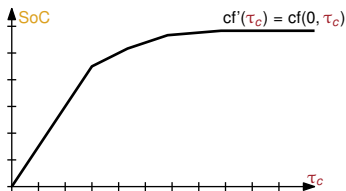


## Formal:

- Eine Funktion  $cf: [0, M] \times \mathbb{R}_{\geq 0} \rightarrow [0, M]$ , bildet
  - Initialen SoC  $b_s$  und
  - Gewünschte Ladezeit  $\tau_c$  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:

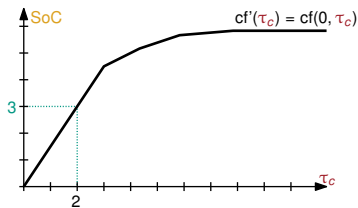
- Eine Funktion  $cf: [0, M] \times \mathbb{R}_{\geq 0} \rightarrow [0, M]$ , bildet
  - Initialen SoC  $b_s$  und
  - Gewünschte Ladezeit  $\tau_c$  auf
  - Durch laden erreichten SoC ab
- Monoton steigend (Länger laden  $\Rightarrow$  mehr Energie)
- Konkav (Akku voller  $\Rightarrow$  langsames Laden)

## 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:

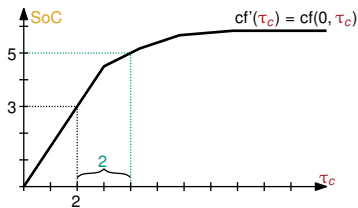
- Eine Funktion  $cf: [0, M] \times \mathbb{R}_{\geq 0} \rightarrow [0, M]$ , bildet
  - Initialen SoC  $b_s$  und
  - Gewünschte Ladezeit  $\tau_c$  auf
  - Durch laden erreichten SoC ab
- Monoton steigend (Länger laden  $\Rightarrow$  mehr Energie)
- Konkav (Akku voller  $\Rightarrow$  langsames Laden)

## 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:

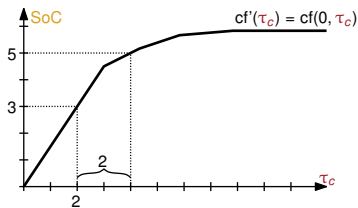
- Eine Funktion  $cf: [0, M] \times \mathbb{R}_{\geq 0} \rightarrow [0, M]$ , bildet
  - Initialen SoC  $b_s$  und
  - Gewünschte Ladezeit  $\tau_c$  auf
  - Durch laden erreichten SoC ab
- Monoton steigend (Länger laden  $\Rightarrow$  mehr Energie)
- Konkav (Akku voller  $\Rightarrow$  langsames Laden)

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

- Reisezeit  $\tau_t$  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  $\ell$  am Knoten  $v$  ist ein Tupel  $(\tau_t, b_u, u, c_{(u,\dots,v)})$  mit:

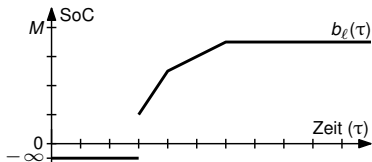
- Reisezeit  $\tau_t$  von  $s$  nach  $v$  (Inklusive Ladezeiten außer an  $u$ )
- SoC  $b_u$  mit dem die letzte Ladestation ( $u$ ) erreicht wurde
- Zu letzt passierte Ladestation  $u$  (Initial  $\perp$ )
- Verbrauchsfunktion  $c_{(u,\dots,v)}$  für den Pfad von  $u$  nach  $v$  (Initial  $\perp$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

## 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  $b_\ell(\tau)$ :

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

$$b' := cf_u(b_u, \tau - \tau_t)$$



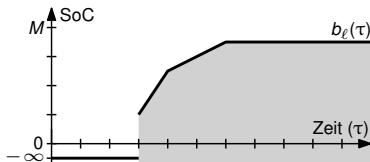
**Label:** Ein Label  $\ell$  am Knoten  $v$  ist ein Tupel  $(\tau_t, b_u, u, c_{(u,\dots,v)})$  mit:

- Reisezeit  $\tau_t$  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  $b_\ell(\tau)$ :

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

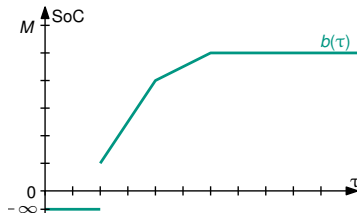


**Kanten Relaxierung:** (Label  $\ell = (\tau_t, b_u, u, c_{(u,\dots,v)})$ )

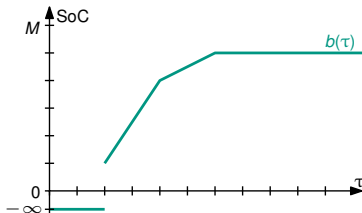
- Relaxieren der Kante  $e = (v, w)$  verschiebt die SoC-Funktion  $b_\ell(\tau)$ 
  - $b_\ell(\tau)$  wird um Fahrzeit  $\tau_d(e)$  nach rechts verschoben
  - $b_\ell(\tau)$  wird um Verbrauch  $\gamma(e)$  nach unten verschoben
- Anschließend werden Battery Constraints überprüft

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

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



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

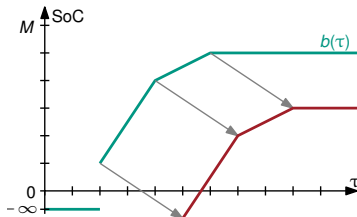


**Kanten Relaxierung:** (Label  $\ell = (\tau_t, b_U, u, c_{(u, \dots, v)})$ )

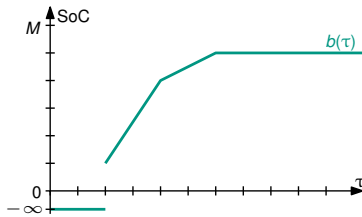
- Relaxieren der Kante  $e = (v, w)$  verschiebt die SoC-Funktion  $b_\ell(\tau)$ 
  - $b_\ell(\tau)$  wird um Fahrzeit  $\tau_d(e)$  nach rechts verschoben
  - $b_\ell(\tau)$  wird um Verbrauch  $\gamma(e)$  nach unten verschoben
- Anschließend werden Battery Constraints überprüft

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

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



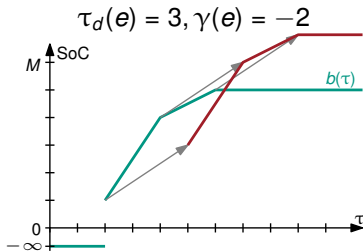
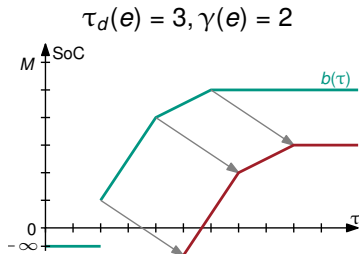
$$\tau_d(e) = 3, \gamma(e) = -2$$



**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  $c_{(u, \dots, w)} \leftarrow c_{(u, \dots, v)} \circ c_e$

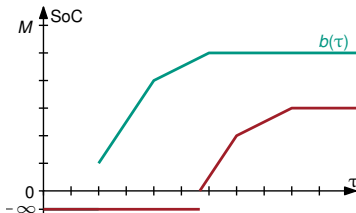


**Kanten Relaxierung:** (Label  $\ell = (\tau_t, b_U, u, c_{(u, \dots, v)})$ )

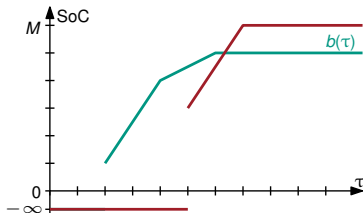
- Relaxieren der Kante  $e = (v, w)$  verschiebt die SoC-Funktion  $b_\ell(\tau)$ 
  - $b_\ell(\tau)$  wird um Fahrzeit  $\tau_d(e)$  nach rechts verschoben
  - $b_\ell(\tau)$  wird um Verbrauch  $\gamma(e)$  nach unten verschoben
- Anschließend werden Battery Constraints überprüft

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

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



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

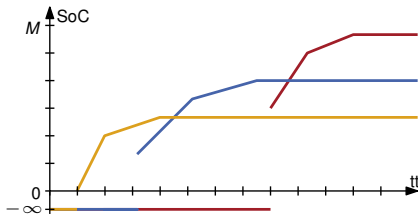


## Dominanz von SoC-Funktionen:

- Für SoC-Funktion  $b_e(\tau)$  und  $b_e(\tau)$  definieren wir Dominanz ( $\propto$ ) als:

$$b_e(\tau) \propto b_e(\tau) \Leftrightarrow \forall \tau \geq 0: b_e(\tau) \geq b_e(\tau)$$

- Pro Knoten eine Menge von SoC-Funktionen
- Ein neues Label wird erzeugt (Durch eine Kanten Relaxierung)  
 $\Rightarrow$  Überprüfe Dominanz (Nur paarweise)  
Lösche dominierte Label

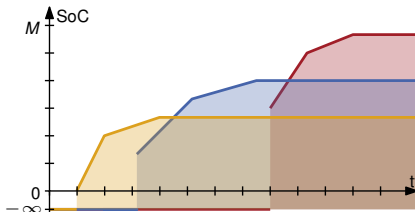


## Dominanz von SoC-Funktionen:

- Für SoC-Funktion  $b_e(\tau)$  und  $b_e(\tau)$  definieren wir Dominanz ( $\propto$ ) als:

$$b_e(\tau) \propto b_e(\tau) \Leftrightarrow \forall \tau \geq 0: b_e(\tau) \geq b_e(\tau)$$

- Pro Knoten eine Menge von SoC-Funktionen
- Ein neues Label wird erzeugt (Durch eine Kanten Relaxierung)  
 $\Rightarrow$  Überprüfe Dominanz (Nur paarweise)  
Lösche dominierte Label



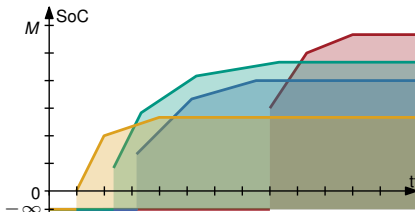


## Dominanz von SoC-Funktionen:

- Für SoC-Funktion  $b_e(\tau)$  und  $b_e(\tau)$  definieren wir Dominanz ( $\propto$ ) als:

$$b_e(\tau) \propto b_e(\tau) \Leftrightarrow \forall \tau \geq 0: b_e(\tau) \geq b_e(\tau)$$

- Pro Knoten eine Menge von SoC-Funktionen
- Ein neues Label wird erzeugt (Durch eine Kanten Relaxierung)  
 $\Rightarrow$  Überprüfe Dominanz (Nur paarweise)  
Lösche dominierte Label

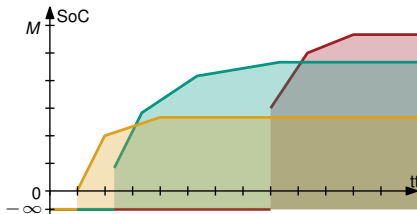


## Dominanz von SoC-Funktionen:

- Für SoC-Funktion  $b_e(\tau)$  und  $b_{\ell}(\tau)$  definieren wir Dominanz ( $\propto$ ) als:

$$b_e(\tau) \propto b_{\ell}(\tau) \Leftrightarrow \forall \tau \geq 0: b_e(\tau) \geq b_{\ell}(\tau)$$

- Pro Knoten eine Menge von SoC-Funktionen
- Ein neues Label wird erzeugt (Durch eine Kanten Relaxierung)  
 $\Rightarrow$  Überprüfe Dominanz (Nur paarweise)  
Lösche dominierte Label

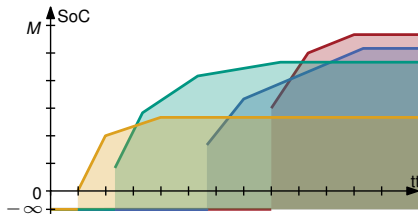


## Dominanz von SoC-Funktionen:

- Für SoC-Funktion  $b_e(\tau)$  und  $b_e(\tau)$  definieren wir Dominanz ( $\propto$ ) als:

$$b_e(\tau) \propto b_e(\tau) \Leftrightarrow \forall \tau \geq 0: b_e(\tau) \geq b_e(\tau)$$

- Pro Knoten eine Menge von SoC-Funktionen
- Ein neues Label wird erzeugt (Durch eine Kanten Relaxierung)  
 $\Rightarrow$  Überprüfe Dominanz (Nur paarweise)  
Lösche dominierte Label

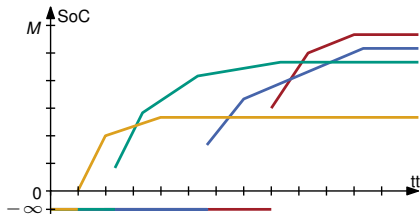


## Dominanz von SoC-Funktionen:

- Für SoC-Funktion  $b_e(\tau)$  und  $b_e(\tau)$  definieren wir Dominanz ( $\propto$ ) als:

$$b_e(\tau) \propto b_e(\tau) \Leftrightarrow \forall \tau \geq 0: b_e(\tau) \geq b_e(\tau)$$

- Pro Knoten eine Menge von SoC-Funktionen
- Ein neues Label wird erzeugt (Durch eine Kanten Relaxierung)  
 $\Rightarrow$  Überprüfe Dominanz (Nur paarweise)  
Lösche dominierte Label



## Settling von Ladestationen

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

## Problem:

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

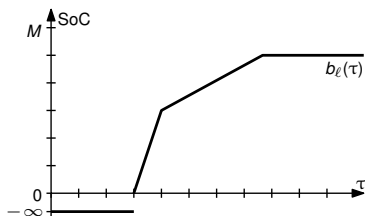
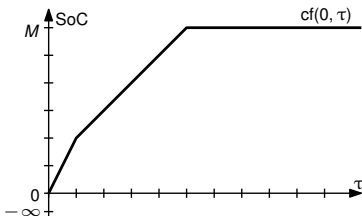
## Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation  $\Rightarrow$  Bestimme  $\tau_C$  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



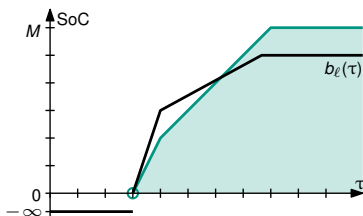
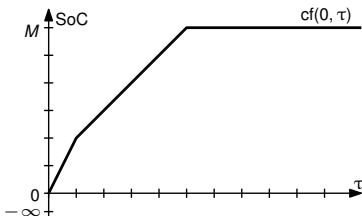
## Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation  $\Rightarrow$  Bestimme  $\tau_C$  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



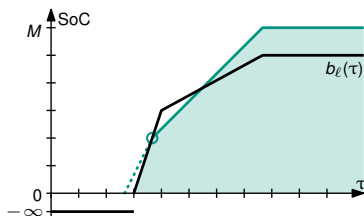
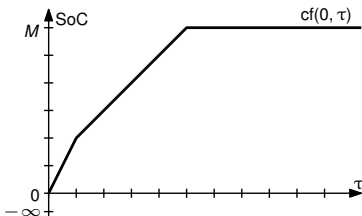
## Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation  $\Rightarrow$  Bestimme  $\tau_C$  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





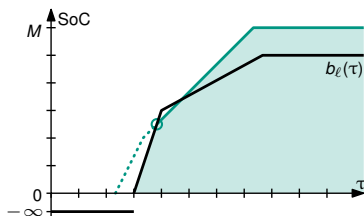
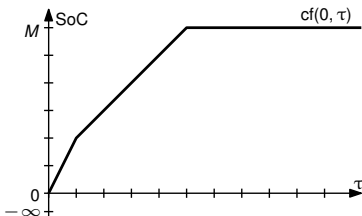
## Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation  $\Rightarrow$  Bestimme  $\tau_C$  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



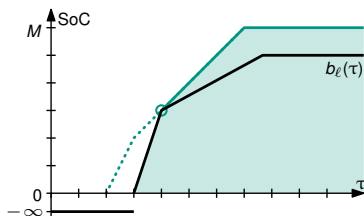
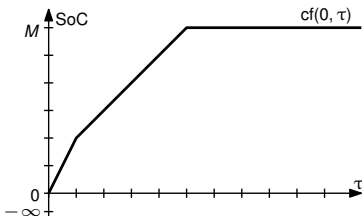
## Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation  $\Rightarrow$  Bestimme  $\tau_C$  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



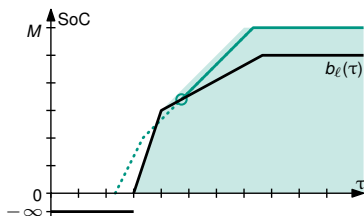
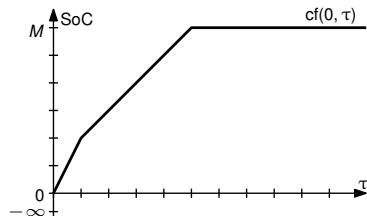
## Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation  $\Rightarrow$  Bestimme  $\tau_C$  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



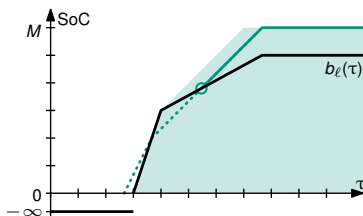
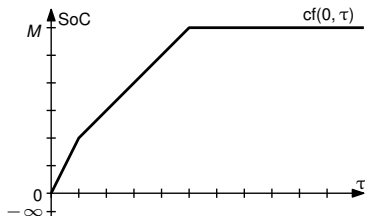
## Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation  $\Rightarrow$  Bestimme  $\tau_C$  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



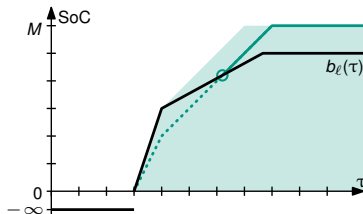
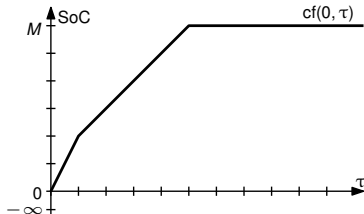
## Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation  $\Rightarrow$  Bestimme  $\tau_C$  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



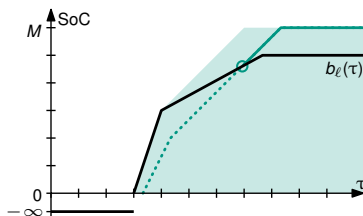
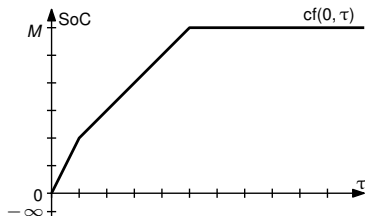
## Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation  $\Rightarrow$  Bestimme  $\tau_C$  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



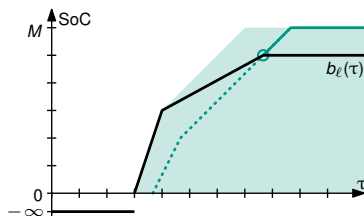
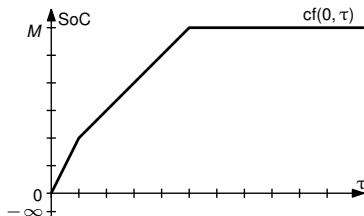
## Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation  $\Rightarrow$  Bestimme  $\tau_C$  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



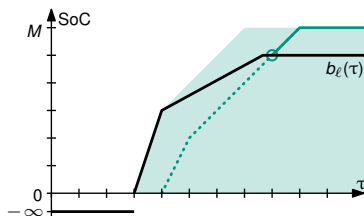
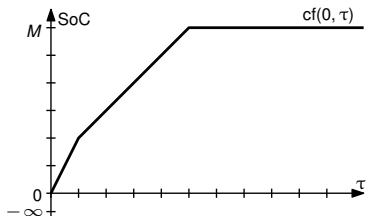
## Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation  $\Rightarrow$  Bestimme  $\tau_C$  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





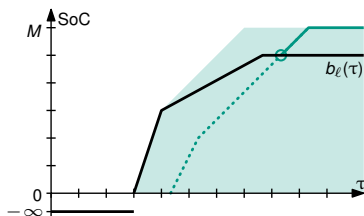
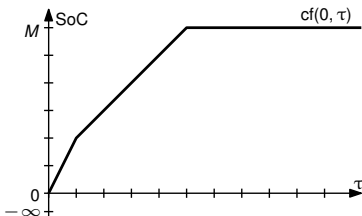
## Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation  $\Rightarrow$  Bestimme  $\tau_C$  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



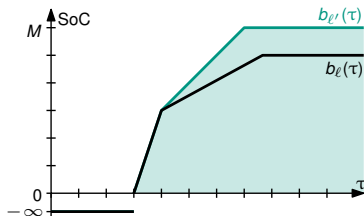
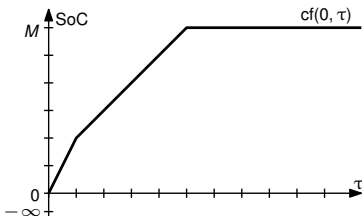
## Settling von Ladestationen

- Nur die letzte Ladestation wird im Label gespeichert
- Erreichen einer Ladestation  $\Rightarrow$  Bestimme  $\tau_C$  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  $\tau_C$  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  $b_\ell(\tau)$

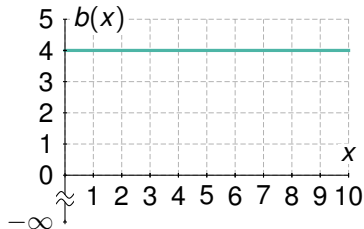
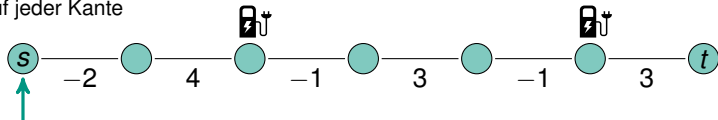
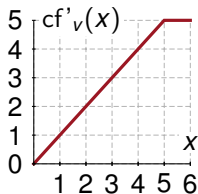
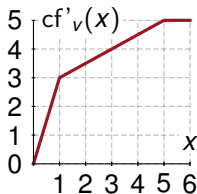
## Gegeben:

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

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

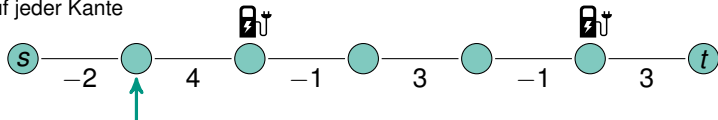
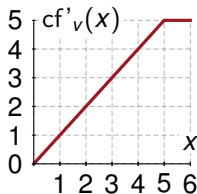
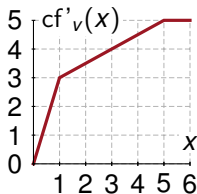
# Beispiel

Annahme:  
Fahrzeit 1  
auf jeder Kante



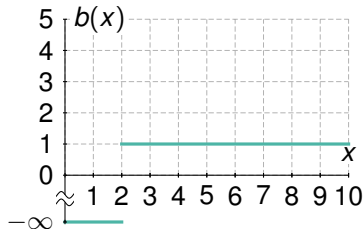
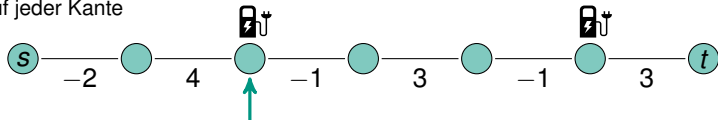
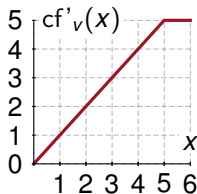
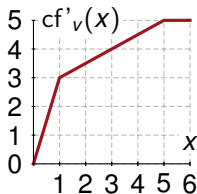
# Beispiel

Annahme:  
Fahrzeit 1  
auf jeder Kante



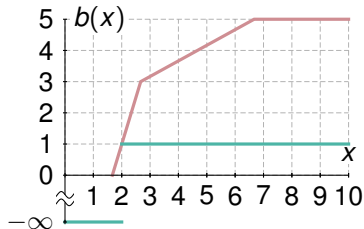
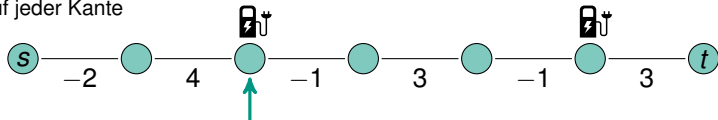
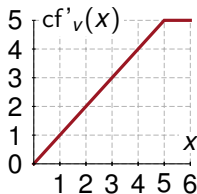
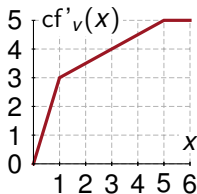
# Beispiel

Annahme:  
Fahrzeit 1  
auf jeder Kante



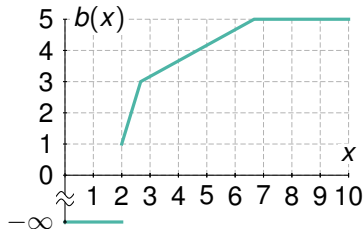
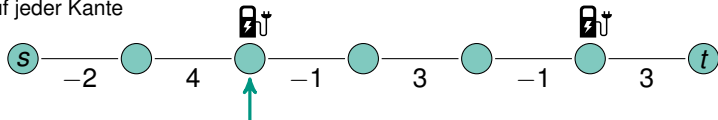
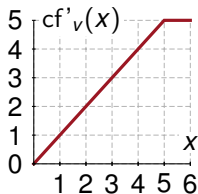
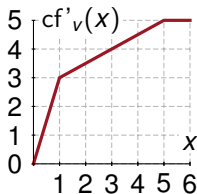
# Beispiel

Annahme:  
Fahrzeit 1  
auf jeder Kante



# Beispiel

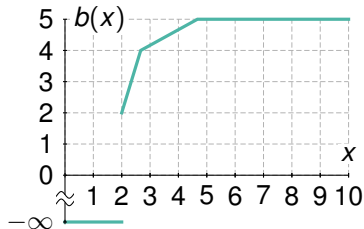
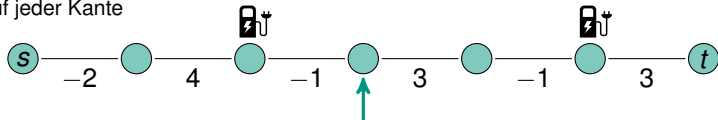
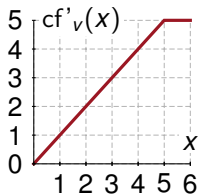
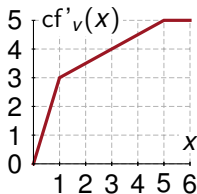
Annahme:  
Fahrzeit 1  
auf jeder Kante





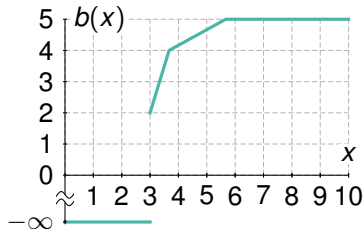
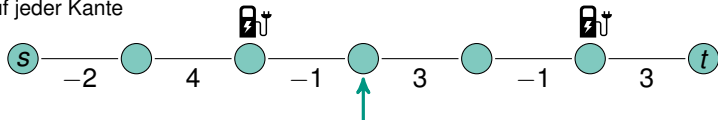
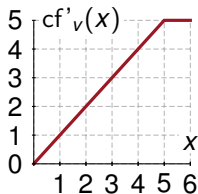
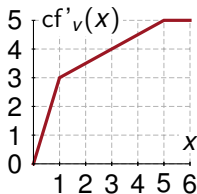
# Beispiel

Annahme:  
Fahrzeit 1  
auf jeder Kante



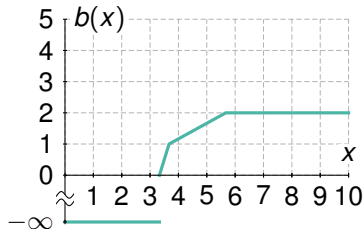
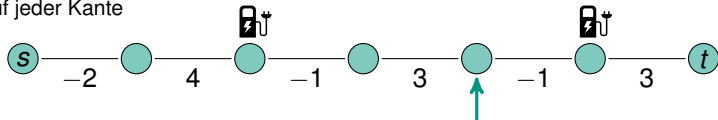
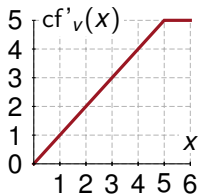
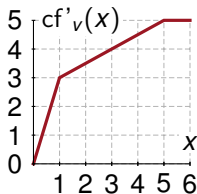
# Beispiel

Annahme:  
Fahrzeit 1  
auf jeder Kante



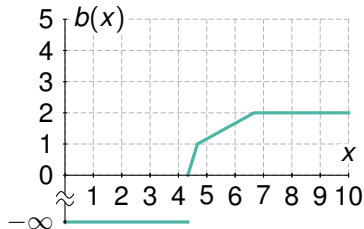
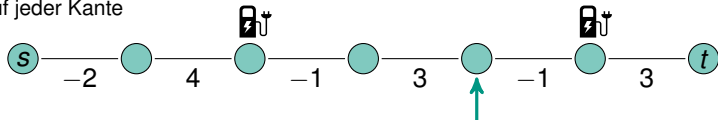
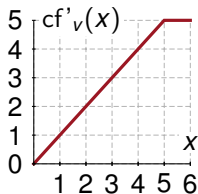
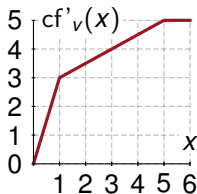
# Beispiel

Annahme:  
Fahrzeit 1  
auf jeder Kante



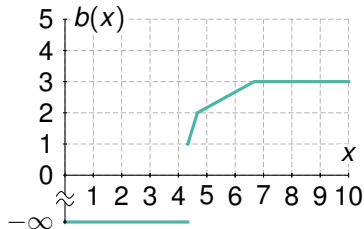
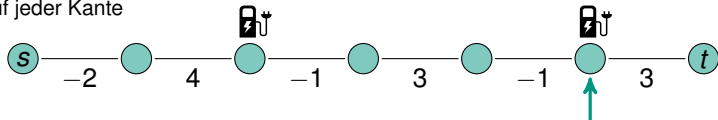
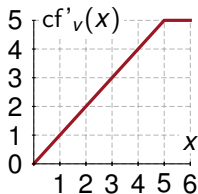
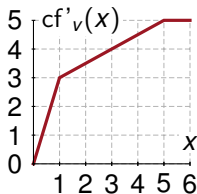
# Beispiel

Annahme:  
Fahrzeit 1  
auf jeder Kante



# Beispiel

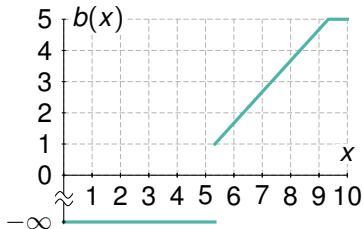
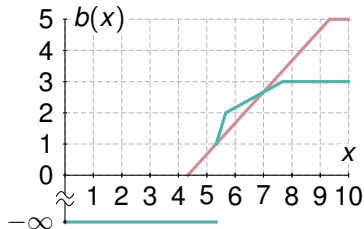
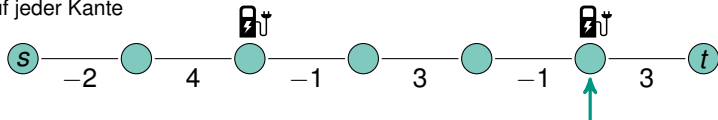
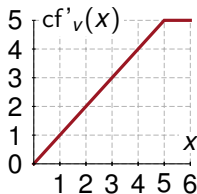
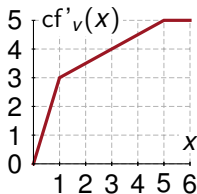
Annahme:  
Fahrzeit 1  
auf jeder Kante





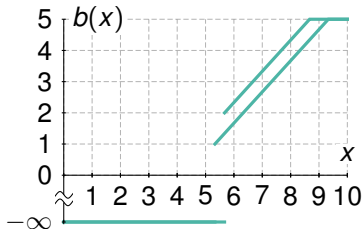
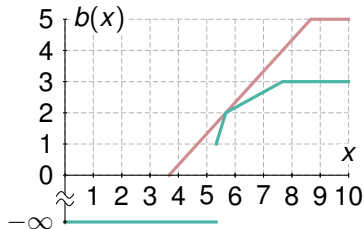
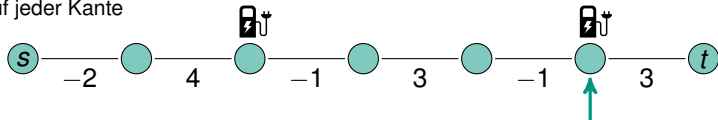
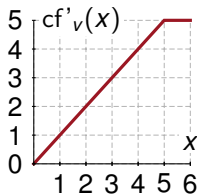
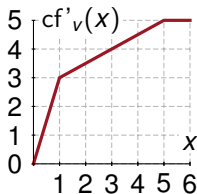
# Beispiel

Annahme:  
Fahrzeit 1  
auf jeder Kante



# Beispiel

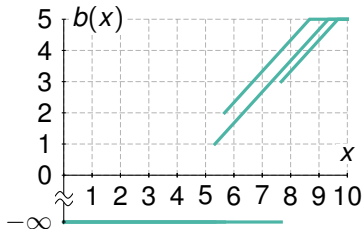
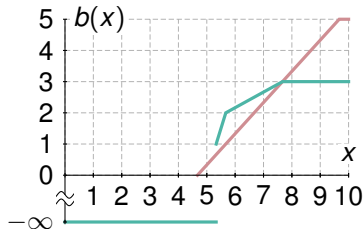
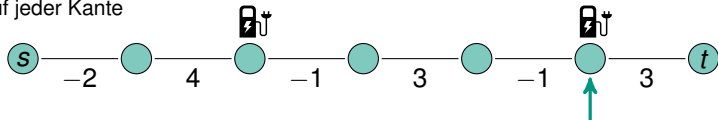
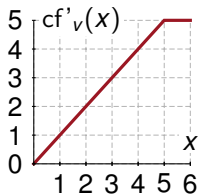
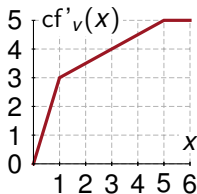
Annahme:  
Fahrzeit 1  
auf jeder Kante





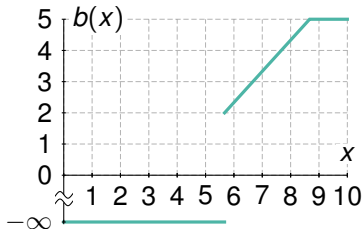
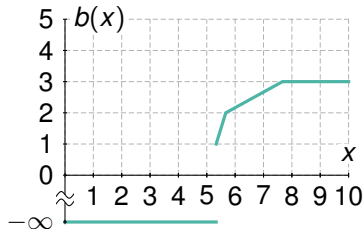
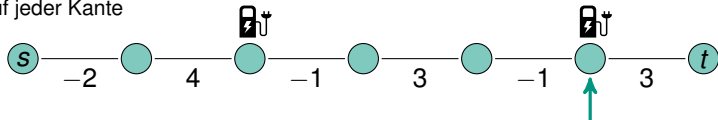
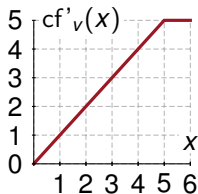
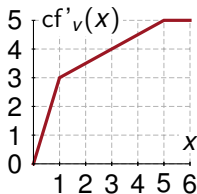
# Beispiel

Annahme:  
Fahrzeit 1  
auf jeder Kante



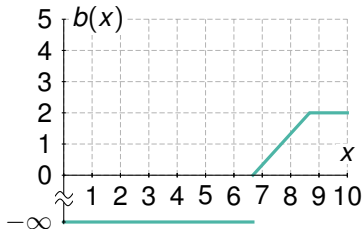
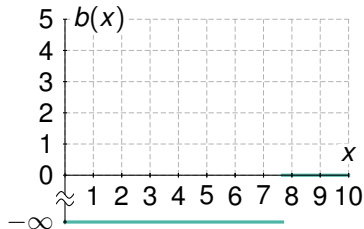
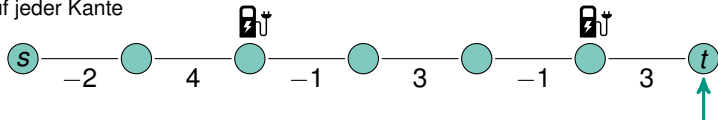
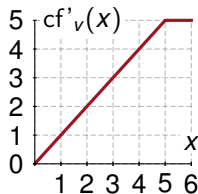
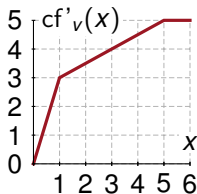
# Beispiel

Annahme:  
Fahrzeit 1  
auf jeder Kante



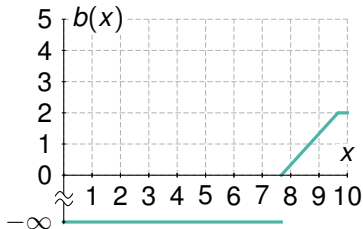
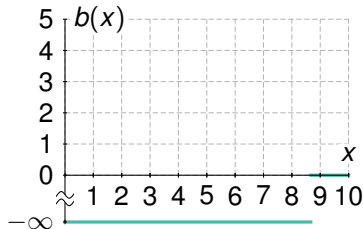
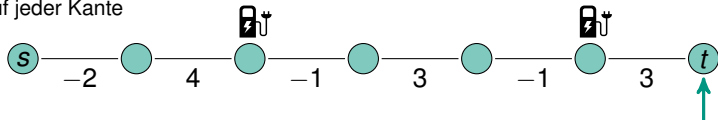
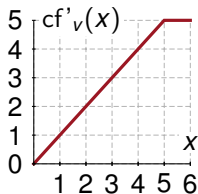
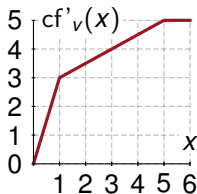
# Beispiel

Annahme:  
Fahrzeit 1  
auf jeder Kante

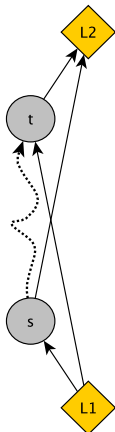


# Beispiel

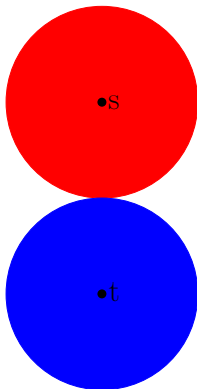
Annahme:  
Fahrzeit 1  
auf jeder Kante



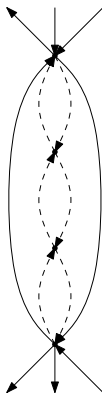
## Landmarken



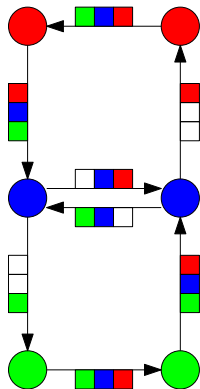
## Bidirektionale Suche



## Kontraktion



## Arc-Flags



## Probleme:

- Kontraktion muss kürzeste Wege Distanzen erhalten
  - SoC während Vorberechnung unbekannt
  - Battery Constraints müssen beachtet werden
  - Ladefunktionen müssen berücksichtigt werden

## Probleme:

- Kontraktion muss kürzeste Wege Distanzen erhalten
  - SoC während Vorberechnung unbekannt
  - Battery Constraints müssen beachtet werden
  - Ladefunktionen müssen berücksichtigt werden

## Lösung:

- Benutze Verbrauchsfunktionen  
(Battery Constraints in Kantengewichten enthalten)
- Ladestation per Definition wichtig  
(Oben Halten, nicht kontrahieren  $\Rightarrow$  Core Graph)

## Probleme:

- Kontraktion muss kürzeste Wege Distanzen erhalten
  - SoC während Vorberechnung unbekannt
  - Battery Constraints müssen beachtet werden
  - Ladefunktionen müssen berücksichtigt werden

## Lösung:

- Benutze Verbrauchsfunktionen  
(Battery Constraints in Kantengewichten enthalten)
- Ladestation per Definition wichtig  
(Oben Halten, nicht kontrahieren  $\Rightarrow$  Core Graph)

## Aber:

- Shortcuts repräsentieren jeweils Pareto-Mengen (Fahrzeit, Verbrauch)
- Pareto-Mengen werden exponentiell groß
- Breche Vorberechnung ab  $\Rightarrow$  unkontrahierter Core-Graph ( $\sim 0.5\%$ )



## Erinnerung:

- A\* benutzt Knotenpotential um Suche zum Ziel zu leiten
- Potential gibt untere Schranke für Fahrzeit zu  $t$ , pro Knoten
- Gute Technik für schwere/komplizierte Suchprobleme
- Klassischer Ansatz: Rückwärtssuche von  $t$

## Beobachtung:

- Fahrzeit zu  $t$  hängt auch vom SoC ab
- Metriken beeinflussen sich gegenseitig

## Erinnerung:

- A\* benutzt Knotenpotential um Suche zum Ziel zu leiten
- Potential gibt untere Schranke für Fahrzeit zu  $t$ , pro Knoten
- Gute Technik für schwere/komplizierte Suchprobleme
- Klassischer Ansatz: Rückwärtssuche von  $t$

## Beobachtung:

- Fahrzeit zu  $t$  hängt auch vom SoC ab
- Metriken beeinflussen sich gegenseitig

## Idee:

- **Fahrzeit** zu  $t$  abhängig von aktueller **Position** und **SoC**
- Nutze Potential  $\pi: V \times [0, M] \rightarrow \mathbb{R}_{\geq 0}$ , welches beides berücksichtigt

## Gesucht:

- Potential  $\pi: V \times [0, M] \rightarrow \mathbb{R}_{\geq 0}$ , bildet (Knoten, SoC) auf Zeit zu  $t$  ab

## Beobachtung:

- Ladestationen erlauben "Umwandlung" von Zeit in SoC

## Gesucht:

- Potential  $\pi: V \times [0, M] \rightarrow \mathbb{R}_{\geq 0}$ , bildet (Knoten, SoC) auf Zeit zu  $t$  ab

## Beobachtung:

- Ladestationen erlauben "Umwandlung" von Zeit in SoC
- Benutze dafür neue Metrik
- Sei dazu  $c_{\max}$  die maximale Ladegeschwindigkeit  
(Maximum über die Steigung aller Ladefunktionen)
- Neue Metrik  $\omega$ :

$$\omega(e) := \tau_d(e) + \frac{\gamma(e)}{c_{\max}}$$

- Beschreibt min. Fahrzeit, falls alle Energie geladen werden muss

**Algorithmus:** Läuft in 2 Phasen:

- 1: Rückwertssuche von  $t$  berechnet Potential  $\pi$
- 2: Vorwärtssuche von  $s$  nach  $t$ , beschleunigt durch  $\pi$

**Potential Berechnung:**

- Drei *unikriterielle* Dijkstra Suchen von  $t$  aus:
  - Auf Metrik  $\tau_d$ : Berechnet min. Fahrzeit  $\pi_\tau$  zu  $t$  (ohne Energieverbrauch)
  - Auf Metrik  $\gamma$ : Berechnet min Energieverbrauch  $\pi_\gamma$
  - Auf Metrik  $\omega$ : Berechnet min. Fahrzeit  $\pi_\omega$ , falls  $b_s = 0$

**Algorithmus:** Läuft in 2 Phasen:

- 1: Rückwertssuche von  $t$  berechnet Potential  $\pi$
- 2: Vorwärtssuche von  $s$  nach  $t$ , beschleunigt durch  $\pi$

**Potential Berechnung:**

- Drei *unikriterielle* Dijkstra Suchen von  $t$  aus:
  - Auf Metrik  $\tau_d$ : Berechnet min. Fahrzeit  $\pi_\tau$  zu  $t$  (ohne Energieverbrauch)
  - Auf Metrik  $\gamma$ : Berechnet min Energieverbrauch  $\pi_\gamma$
  - Auf Metrik  $\omega$ : Berechnet min. Fahrzeit  $\pi_\omega$ , falls  $b_s = 0$
- Setze dann:

$$\pi(v, b) := \begin{cases} \pi_\tau(v) & , \text{ falls } b \geq \pi_\gamma(v) \\ \pi_\omega(v) - \frac{b}{c_{\max}} & , \text{ sonst} \end{cases}$$

## Algorithmus:

- Vorbereitung CH:
  - Hält Ladestationen oben
  - Lässt kleinen Core unkontrahiert
- Zur Query
  - CH Aufwärtssuchen von  $s$  und  $t$  bis Core erreicht  
(Normaler Energie-CSP-Algorithmus, da keine Ladestationen)
  - Anschließend A\* eingeschränkt auf den Core Graphen

## Algorithmus:

- Vorbereitung CH:
  - Hält Ladestationen oben
  - Lässt kleinen Core unkontrahiert
- Zur Query
  - CH Aufwärtssuchen von  $s$  und  $t$  bis Core erreicht  
(Normaler Energie-CSP-Algorithmus, da keine Ladestationen)
  - Anschließend A\* eingeschränkt auf den Core Graphen

## Heuristiken:

- Weitere Beschleunigung durch Heuristiken möglich
- Pfade die bezüglich  $\omega$ -Metrik minimal sind, sind oft optimal
- Relaxiere pro Shortcut nur  $\omega$ -minimale Pareto Punkte



## Straßen Netzwerk:

- Europa (Eur) & Deutschland (Ger)

## Energieverbrauch:

- PHEM – Entwickelt von der TU Graz [Hausberger et al. '09]
- SRTM Höhendaten (Shuttle Radar Topography Mission)
- Ladestations Positionen von ChargeMap

Instanzen	# Knoten	# Kanten	# Kanten mit $\gamma < 0$	# S
Ger	4 692 091	10 805 429	1 119 710 (10.36%)	1 966
Eur	22 198 628	51 088 095	6 060 648 (11.86%)	13 810
Osg	5 588 146	11 711 088	1 142 391 (9.75%)	643

## CH Vorbereitung:

- Auswirkung der Core Größe auf die Vorbereitung

Core Größe		Vorbereitung	Query [ms]	
Avg. deg.	# Knoten	[h:m:s]	CS: only BSS	CS: realistic
8	344 066 (7.33%)	2:58	1 474.1	47 979.9
16	116 917 (2.49%)	4:01	536.5	1 669.0
32	65 375 (1.39%)	5:03	436.1	1 356.8
64	43 036 (0.91%)	7:07	449.8	1 408.8
128	30 526 (0.65%)	11:16	509.6	1 585.4
256	22 592 (0.48%)	20:22	647.5	2 098.5
512	17 431 (0.37%)	37:11	880.7	2 739.9
1024	13 942 (0.29%)	1:05:51	1 264.6	3 934.2
2048	11 542 (0.24%)	2:00:27	1 822.6	5 670.1
4096	9 842 (0.20%)	4:17:36	2 706.6	8 420.1

Instanz	$M$	Preproc.	Exact Query		Heuristic Query		
			Feas.	CHarge	$H_\omega$	$H_\omega^A$	
<b>Only BSS</b>	Ger-c1966	16 kWh	5:03	100	1 398	436	21
	Ger-c1966	85 kWh	4:59	100	1 013	48	28
	Eur-c13810	16 kWh	30:32	63	10 786	9943	207
	Eur-c13810	85 kWh	30:16	100	47 921	1022	41
<b>Mixed CS</b>	Ger-c1966	16 kWh	5:03	100	8 629	1 357	155
	Ger-c1966	85 kWh	4:59	100	2 614	342	34
	Eur-c13810	16 kWh	30:32	63	24 148	17 630	2 694
	Eur-c13810	85 kWh	30:16	100	86 193	26 867	600

Vorberechnungszeiten in Minuten:Sekunden, Query Zeiten in Millisekunden

## 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

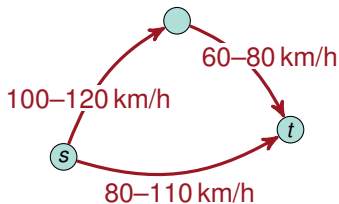
# Geschwindigkeitsanpassung

## Problem EV CONSTRAINED SHORTEST PATH

- Input:**
- Graph  $G = (V, E)$
  - Min./Max. Geschwindigkeit pro Kante
  - Start  $s$ , Ziel  $t$ , initialer SoC

- Output:**  $s-t$  Pfad + Geschwindigkeit pro Kante, so dass
- Battery Constraints eingehalten
  - Fahrzeit minimiert

- Geschwindigkeit anpassen um Energie zu sparen
- Min./Max. Geschw.: Verkehrsfluss, Geschwindigkeitsbeschränkungen ...
- $\mathcal{NP}$ -schweres Problem

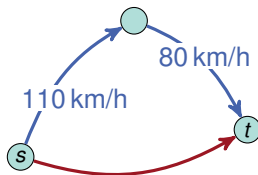


## Problem EV CONSTRAINED SHORTEST PATH

- Input:**
- Graph  $G = (V, E)$
  - Min./Max. Geschwindigkeit pro Kante
  - Start  $s$ , Ziel  $t$ , initialer SoC

- Output:**  $s-t$  Pfad + Geschwindigkeit pro Kante, so dass
- Battery Constraints eingehalten
  - Fahrzeit minimiert

- Geschwindigkeit anpassen um Energie zu sparen
- Min./Max. Geschw.: Verkehrsfluss, Geschwindigkeitsbeschränkungen ...
- $\mathcal{NP}$ -schweres Problem



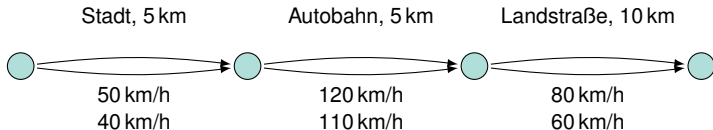
**Ziel:** Energie sparen durch Anpassung der Geschwindigkeit

- Pro Straßensegment: Interval mit min./max. Geschwindigkeit

**Erster Ansatz:**

- Diskretisiere mögliche Geschwindigkeiten pro Segment
- Eine Kante pro Geschwindigkeit

Routing auf **Multigraph**





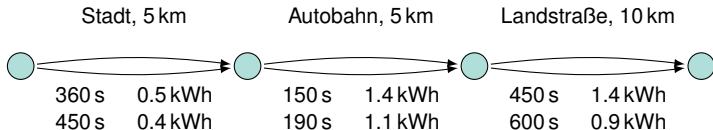
**Ziel:** Energie sparen durch Anpassung der Geschwindigkeit

- Pro Straßensegment: Interval mit min./max. Geschwindigkeit

**Erster Ansatz:**

- Diskretisiere mögliche Geschwindigkeiten pro Segment
- Eine Kante pro Geschwindigkeit

Routing auf **Multigraph**



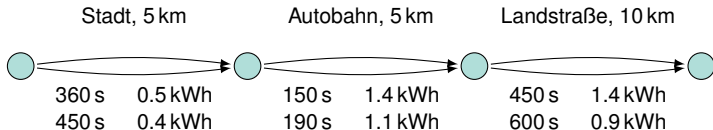
**Ziel:** Energie sparen durch Anpassung der Geschwindigkeit

- Pro Straßensegment: Interval mit min./max. Geschwindigkeit

**Erster Ansatz:**

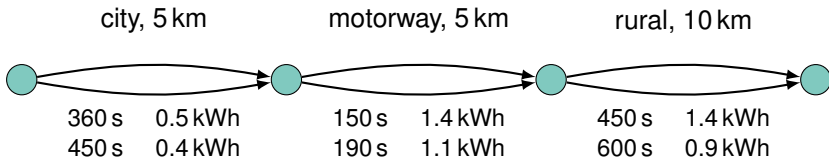
- Diskretisiere mögliche Geschwindigkeiten pro Segment
- Eine Kante pro Geschwindigkeit

Routing auf **Multigraph**



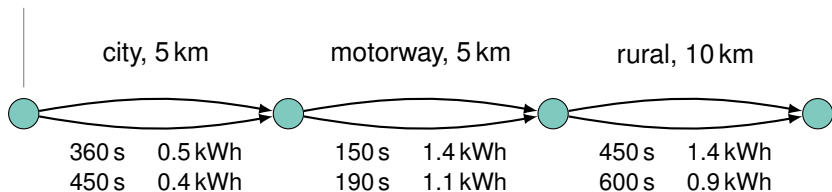
Einfache Umsetzung, aber hohe Lösungskomplexität

Verwende MCD zur Routenberechnung

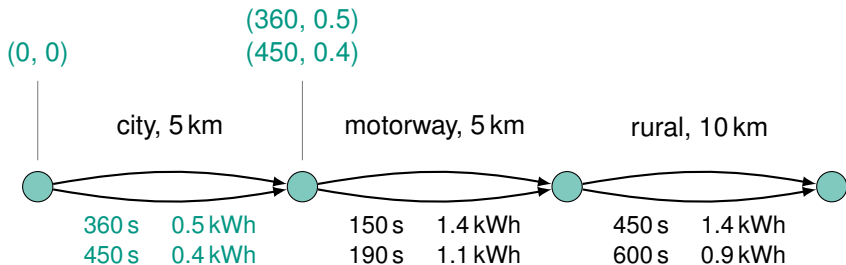


Verwende MCD zur Routenberechnung

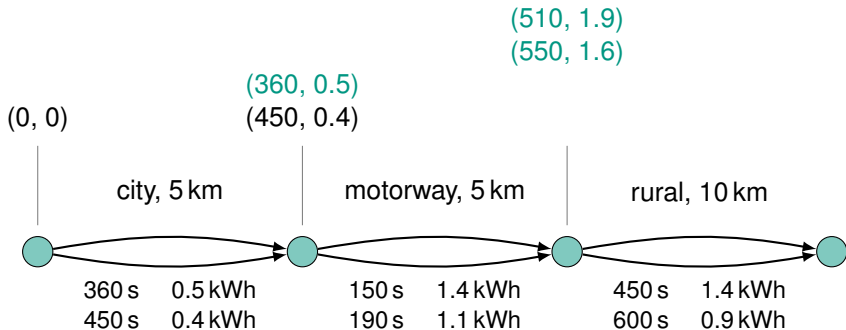
(0, 0)



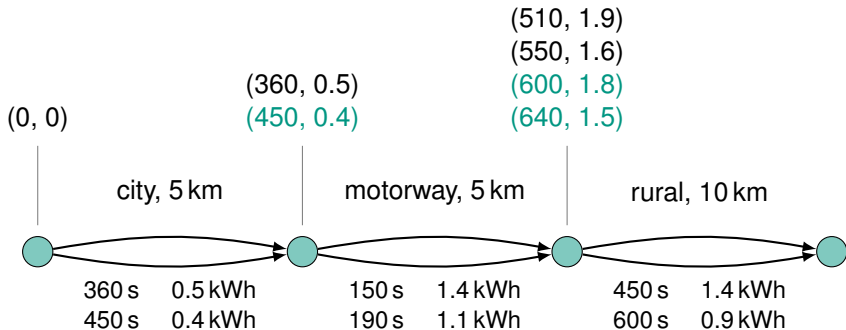
Verwende MCD zur Routenberechnung



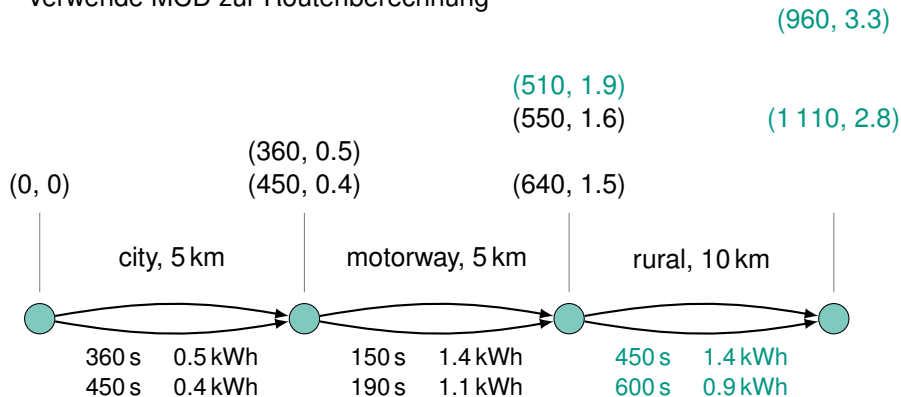
Verwende MCD zur Routenberechnung



Verwende MCD zur Routenberechnung

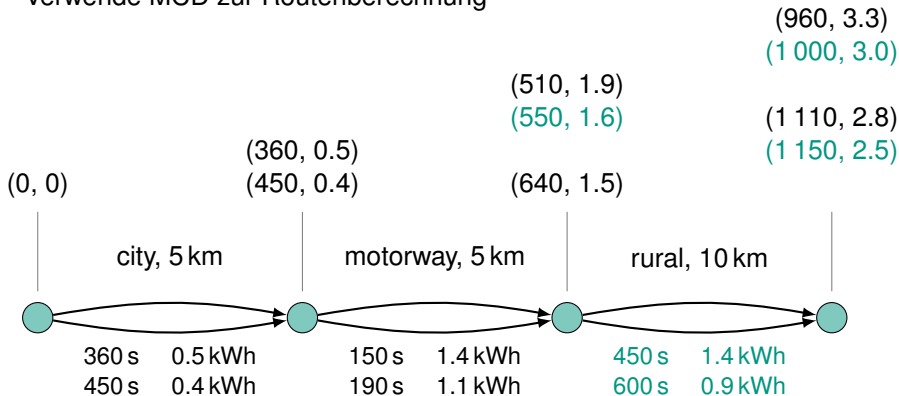


Verwende MCD zur Routenberechnung

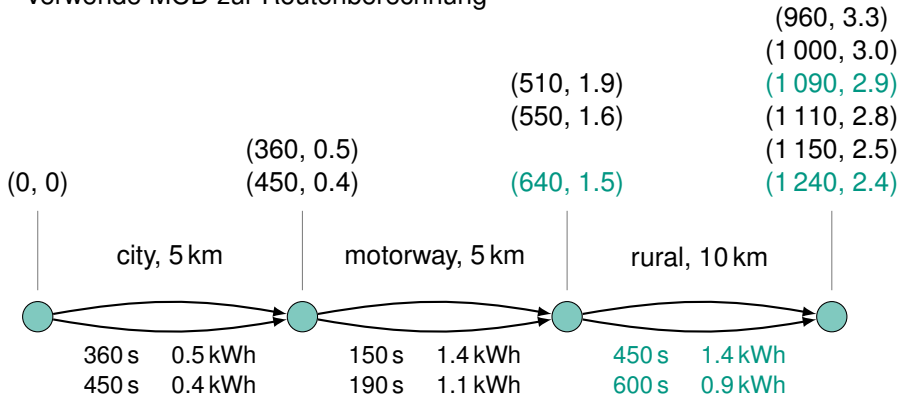




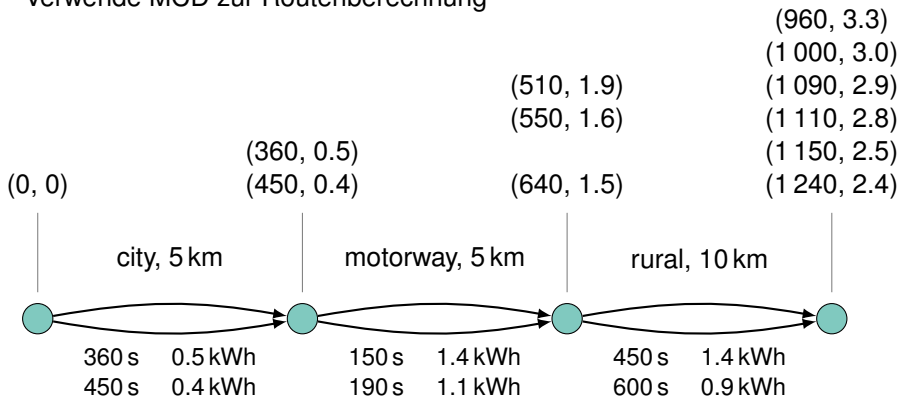
Verwende MCD zur Routenberechnung



Verwende MCD zur Routenberechnung



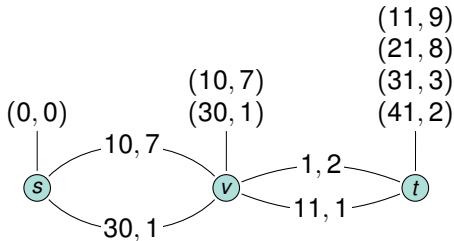
Verwende MCD zur Routenberechnung



**Worst Case:**  $n$  Knoten mit je  $k$  parallelen Kanten  $\Rightarrow \Theta(k^n)$  Labels

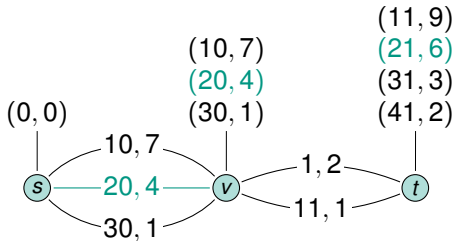
- + einfach zu implementieren
- exponentiell viele Labels auf einer Route
- nicht alle Lösungen abgebildet
- unattraktive Lösungen
- Entfernen/Einsetzen von Knoten beeinflusst Lösung

⇒ Genauigkeit vs Rechenaufwand

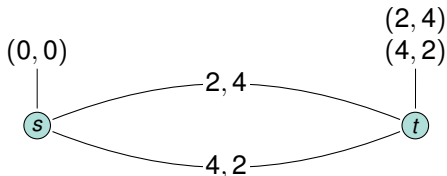


- + einfach zu implementieren
- exponentiell viele Labels auf einer Route
- nicht alle Lösungen abgebildet
- unattraktive Lösungen
- Entfernen/Einsetzen von Knoten beeinflusst Lösung

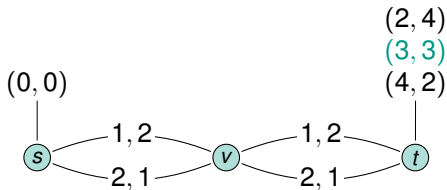
⇒ Genauigkeit vs Rechenaufwand



- + einfach zu implementieren
  - exponentiell viele Labels auf einer Route
  - nicht alle Lösungen abgebildet
  - unattraktive Lösungen
  - Entfernen/Einsetzen von Knoten beeinflusst Lösung
- ⇒ Genauigkeit vs Rechenaufwand

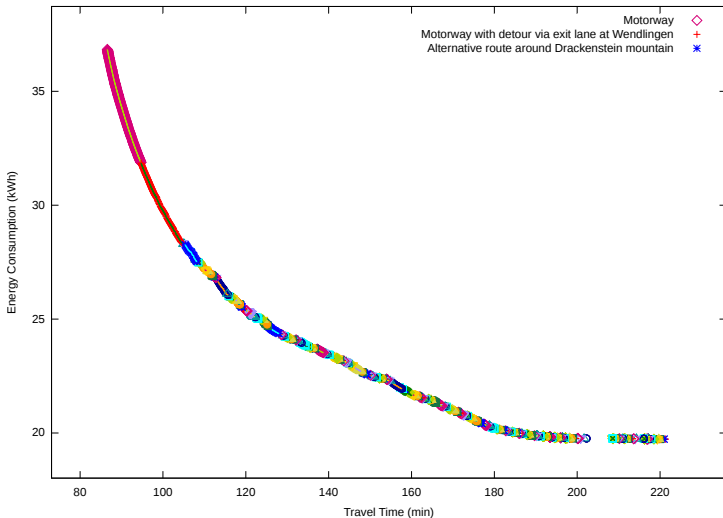


- + einfach zu implementieren
  - exponentiell viele Labels auf einer Route
  - nicht alle Lösungen abgebildet
  - unattraktive Lösungen
  - Entfernen/Einsetzen von Knoten beeinflusst Lösung
- ⇒ Genauigkeit vs Rechenaufwand



# Einfaches Modell – Beispiel Pareto Menge

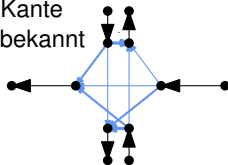
Pareto front for query from Karlsruhe (49.0169,8.41784) to Ulm (48.4002,9.98479), ETP HR A\* EA, Similarity / 12, 40 kWh





*Wie bekommen wir Verbräuche auf den Kanten im Straßengraphen?*

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



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

$v$ : Geschwindigkeit

$s$ : Steigung

$\lambda_1, \lambda_2, \lambda_3$ : (nichtnegative) Konstanten

Verbrauch auf einer Kante:  $\lambda_1 v^2 + \lambda_2 s + \lambda_3$ .

$v$ : Geschwindigkeit

$s$ : Steigung

■ Wir wollen **Fahrzeit** auf **Verbrauch** abbilden

■ Es gilt  $v = \ell/x$

$\ell$ : Kantenlänge

$x$ : Fahrzeit

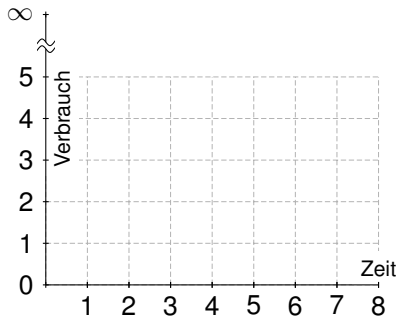
■ Setze  $\alpha := \lambda_1 \ell^2$

und  $\gamma := \lambda_2 s + \lambda_3$

■ Löse auf nach  $x$

⇒ ergibt **Tradeoff Funktion**

$$c(x) = \frac{\alpha}{(x - \beta)^2} + \gamma$$



■ Parameter  $\beta$  für Verbrauchsfunktionen von **Pfaden** benötigt

■ Minimale Fahrzeit  $\underline{x}$  und maximale Fahrzeit  $\bar{x}$  pro Kante

Verbrauch auf einer Kante:  $\lambda_1 v^2 + \lambda_2 s + \lambda_3$ .

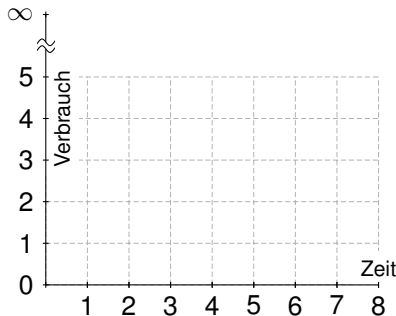
$v$ : Geschwindigkeit

$s$ : Steigung

- Tradeoff Funktion

$$c(x) = \frac{\alpha}{(x - \beta)^2} + \gamma$$

- Bildet Fahrzeit auf Verbrauch ab
- Minimale Fahrzeit  $\underline{x}$
- Maximale Fahrzeit  $\bar{x}$



Minimaler Verbrauch entlang eines Pfades?

Verbrauch auf einer Kante:  $\lambda_1 v^2 + \lambda_2 s + \lambda_3$ .

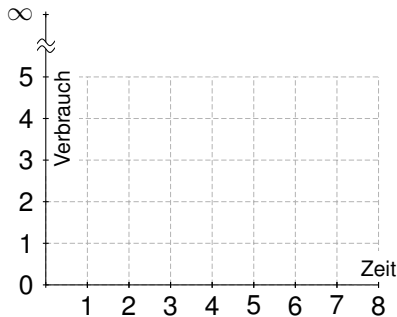
$v$ : Geschwindigkeit

$s$ : Steigung

- Tradeoff Funktion

$$c(x) = \frac{\alpha}{(x - \beta)^2} + \gamma$$

- Bildet Fahrzeit auf Verbrauch ab
- Minimale Fahrzeit  $\underline{x}$
- Maximale Fahrzeit  $\bar{x}$



Minimaler Verbrauch entlang eines Pfades?

Verbrauch auf einer Kante:  $\lambda_1 v^2 + \lambda_2 s + \lambda_3$ .

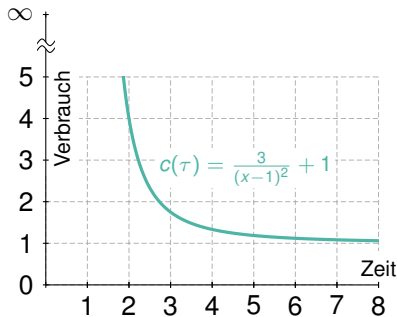
$v$ : Geschwindigkeit

$s$ : Steigung

## Tradeoff Funktion

$$c(x) = \frac{\alpha}{(x - \beta)^2} + \gamma$$

- Bildet **Fahrzeit** auf **Verbrauch** ab
- Minimale Fahrzeit  $\underline{x}$
- Maximale Fahrzeit  $\bar{x}$



Minimaler Verbrauch entlang eines Pfades?

Verbrauch auf einer Kante:  $\lambda_1 v^2 + \lambda_2 s + \lambda_3$ .

$v$ : Geschwindigkeit

$s$ : Steigung

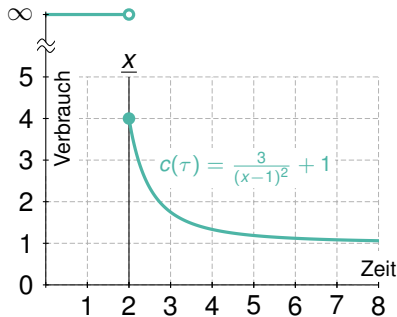
## Tradeoff Funktion

$$c(x) = \frac{\alpha}{(x - \beta)^2} + \gamma$$

## Bildet Fahrzeit auf Verbrauch ab

## Minimale Fahrzeit $\underline{x}$

## Maximale Fahrzeit $\bar{x}$



Minimaler Verbrauch entlang eines Pfades?

Verbrauch auf einer Kante:  $\lambda_1 v^2 + \lambda_2 s + \lambda_3$ .

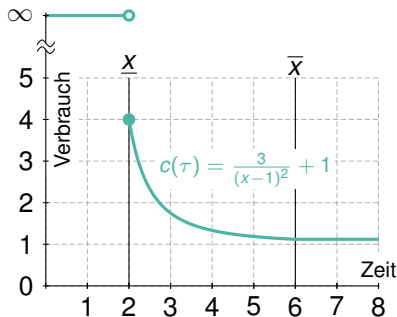
$v$ : Geschwindigkeit

$s$ : Steigung

## Tradeoff Funktion

$$c(x) = \frac{\alpha}{(x - \beta)^2} + \gamma$$

- Bildet Fahrzeit auf Verbrauch ab
- Minimale Fahrzeit  $\underline{x}$
- Maximale Fahrzeit  $\bar{x}$



Minimaler Verbrauch entlang eines Pfades?

Verbrauch auf einer Kante:  $\lambda_1 v^2 + \lambda_2 s + \lambda_3$ .

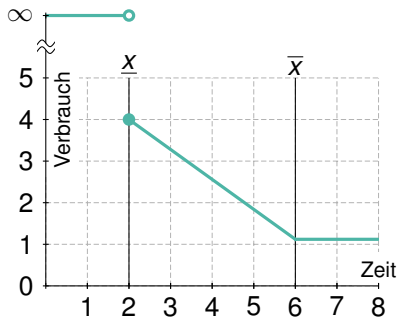
$v$ : Geschwindigkeit

$s$ : Steigung

- Tradeoff Funktion

$$c(x) = \alpha x + \beta$$

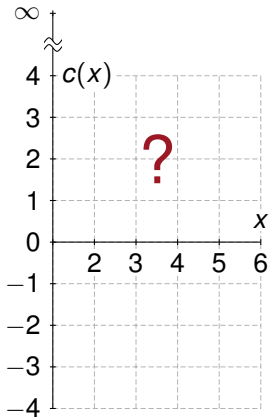
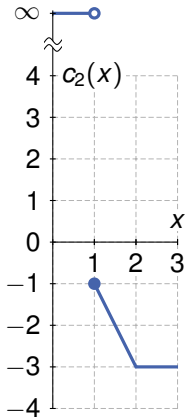
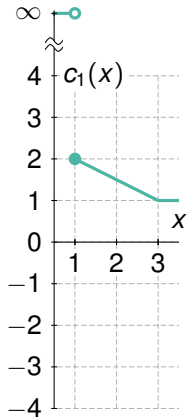
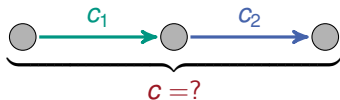
- Bildet Fahrzeit auf Verbrauch ab
- Minimale Fahrzeit  $\underline{x}$
- Maximale Fahrzeit  $\bar{x}$



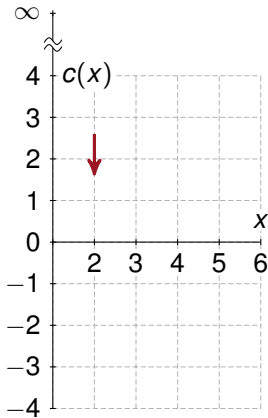
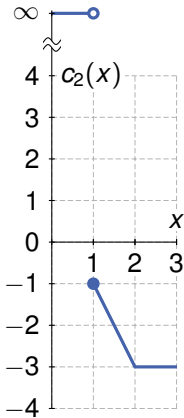
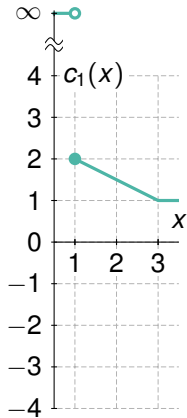
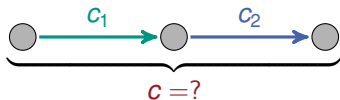
Minimaler Verbrauch entlang eines Pfades?



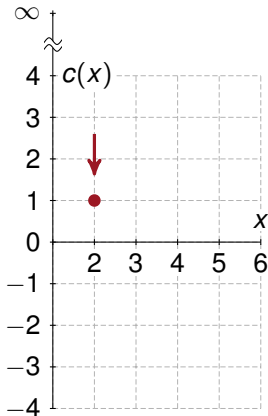
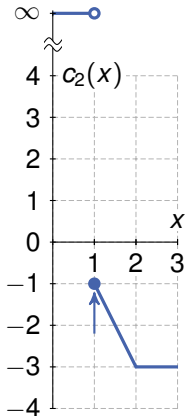
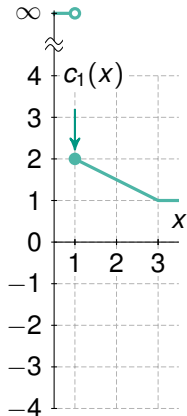
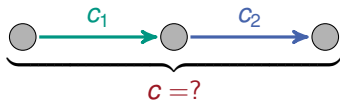
# Tradeoff-Funktion eines Pfades



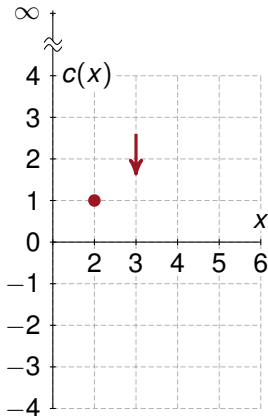
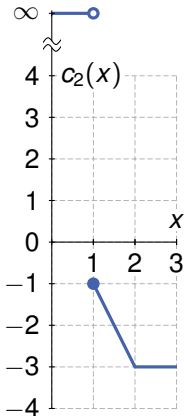
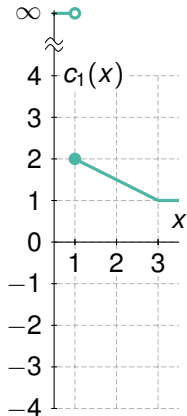
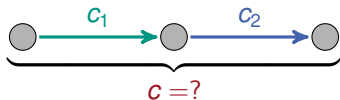
# Tradeoff-Funktion eines Pfades



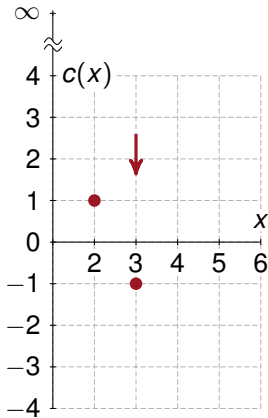
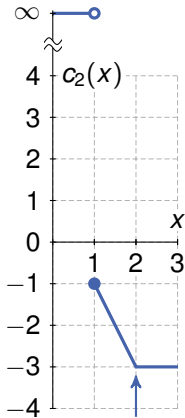
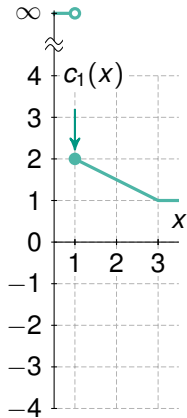
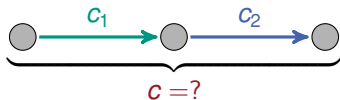
# Tradeoff-Funktion eines Pfades



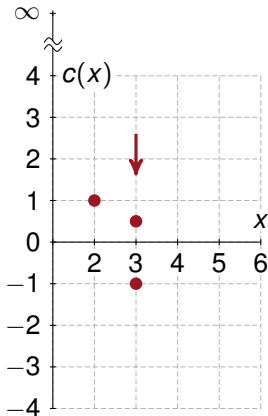
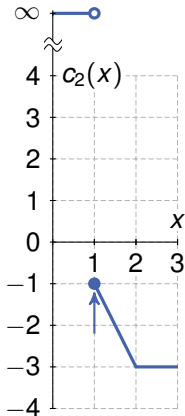
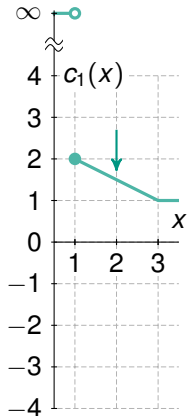
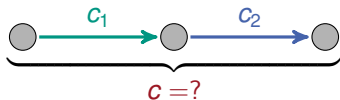
# Tradeoff-Funktion eines Pfades



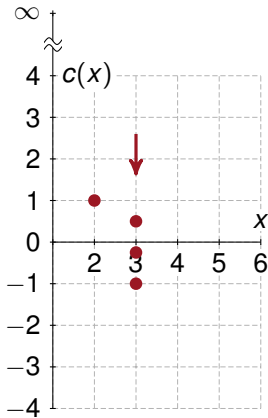
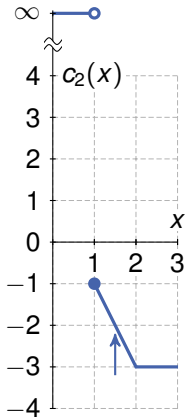
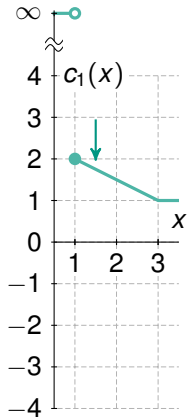
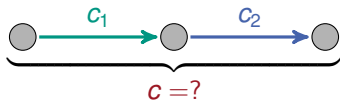
# Tradeoff-Funktion eines Pfades



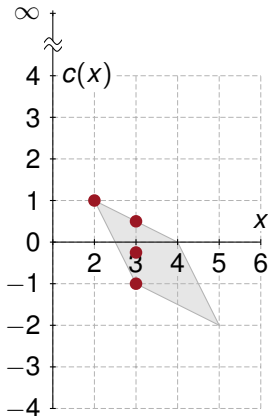
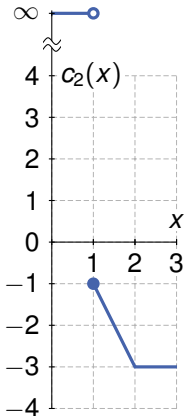
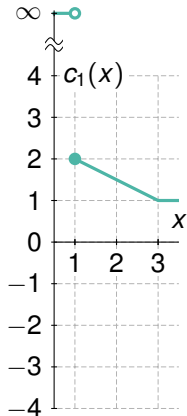
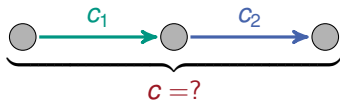
# Tradeoff-Funktion eines Pfades



# Tradeoff-Funktion eines Pfades

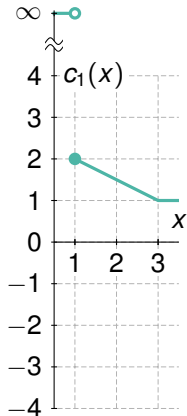
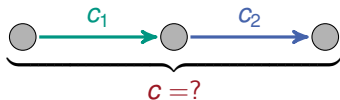


# Tradeoff-Funktion eines Pfades

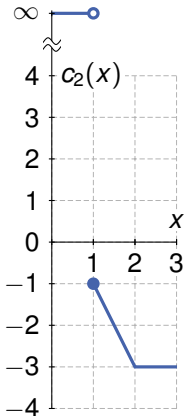




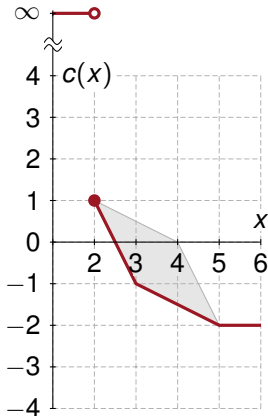
# Tradeoff-Funktion eines Pfades



$\oplus$



=

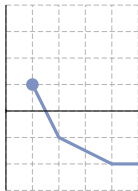


**Ziel:** Minimaler Verbrauch entlang Pfad

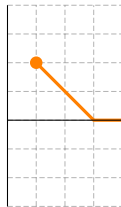
**Formal:** Geg. zwei Tradeoff-Funktionen  $c_1$  und  $c_2$ , berechne

$$(c_1 \oplus c_2)(\tau) := \min_{\Delta} c_1(\Delta) + c_2(\tau - \Delta)$$

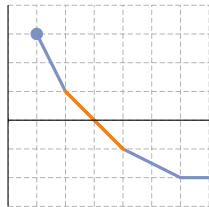
- Stückweise linear
- Konvex
- Linearzeit-Algorithmus



$\oplus$



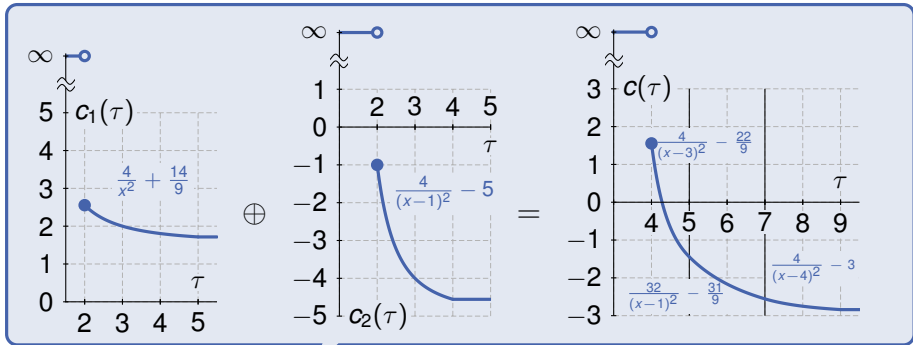
=



Ähnlich für das realistische Modell

# Linken von Tradeoff-Funktionen

Ziel: Minimaler Verbrauch entlang Pfad



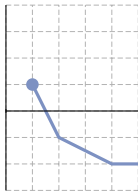
Ähnlich für das realistische Modell

**Ziel:** Minimaler Verbrauch entlang Pfad

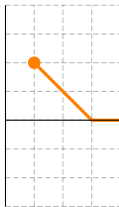
**Formal:** Geg. zwei Tradeoff-Funktionen  $c_1$  und  $c_2$ , berechne

$$(c_1 \oplus c_2)(\tau) := \min_{\Delta} c_1(\Delta) + c_2(\tau - \Delta)$$

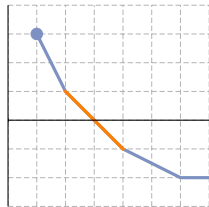
- Stückweise linear
- Konvex
- Linearzeit-Algorithmus



$\oplus$



=



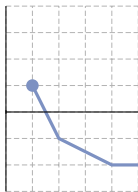
Ähnlich für das realistische Modell

**Ziel:** Minimaler Verbrauch entlang Pfad

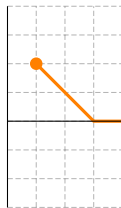
**Formal:** Geg. zwei Tradeoff-Funktionen  $c_1$  und  $c_2$ , berechne

$$(c_1 \oplus c_2)(\tau) := \min_{\Delta} c_1(\Delta) + c_2(\tau - \Delta)$$

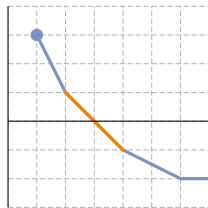
- Stückweise linear
- Konvex
- Linearzeit-Algorithmus



$\oplus$



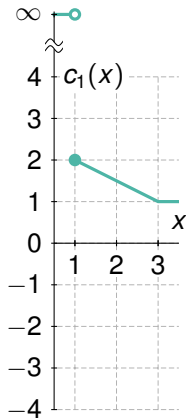
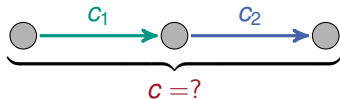
=



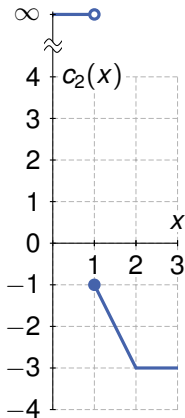
Ähnlich für das realistische Modell

**Aber:** Battery Constraints nicht berücksichtigt

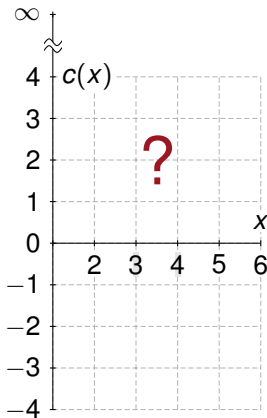
# Battery Constraints



$\oplus$

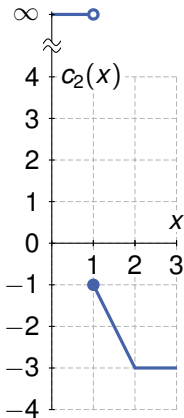
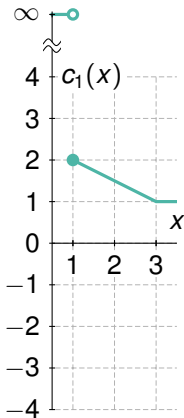
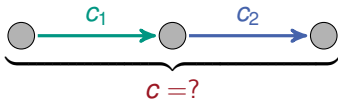


=

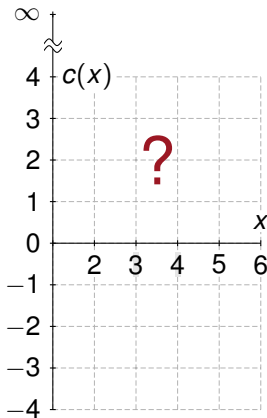


# Battery Constraints

Min. state of charge: 0  
Max. state of charge: 4  
Initial state of charge: 1.5

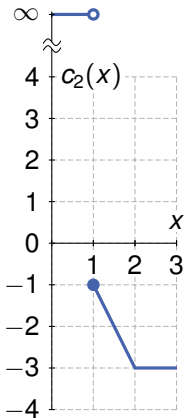
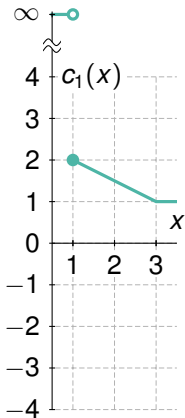
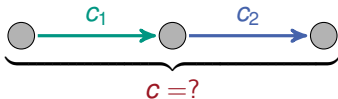


=

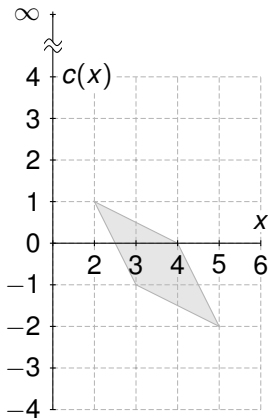


# Battery Constraints

Min. state of charge: 0  
Max. state of charge: 4  
Initial state of charge: 1.5



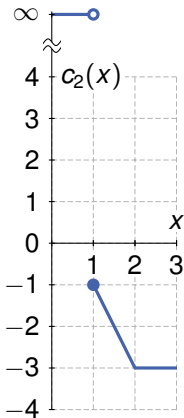
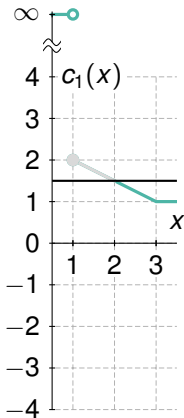
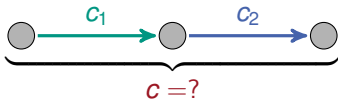
=



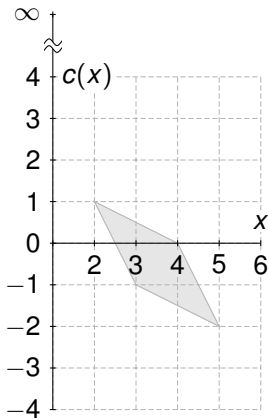


# Battery Constraints

Min. state of charge: 0  
Max. state of charge: 4  
Initial state of charge: 1.5

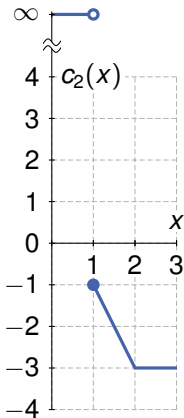
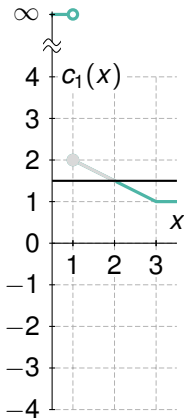
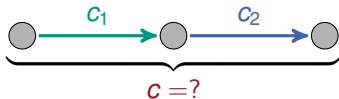


=

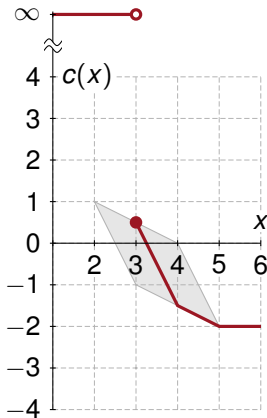


# Battery Constraints

Min. state of charge: 0  
Max. state of charge: 4  
Initial state of charge: 1.5



=

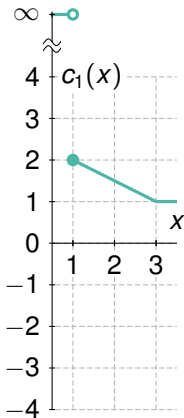
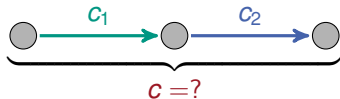


# Battery Constraints

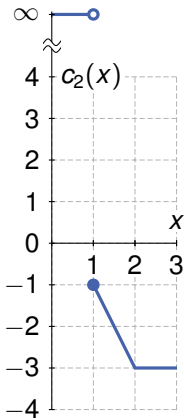
Min. state of charge: 0

Max. state of charge: 4

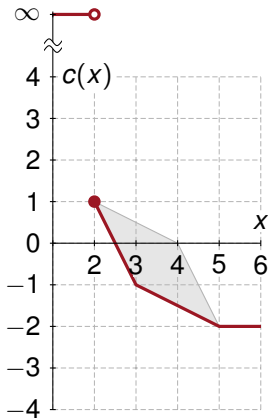
Initial state of charge: 4



$\oplus$



=

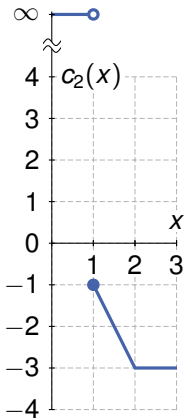
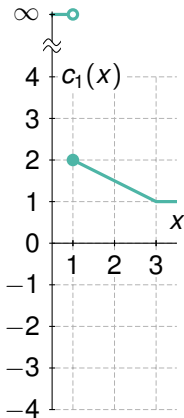
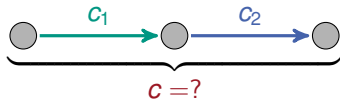


# Battery Constraints

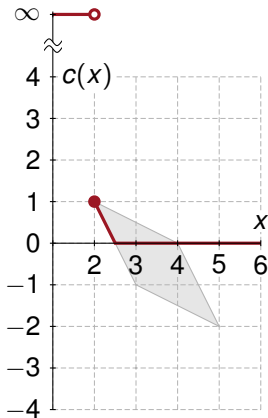
Min. state of charge: 0

Max. state of charge: 4

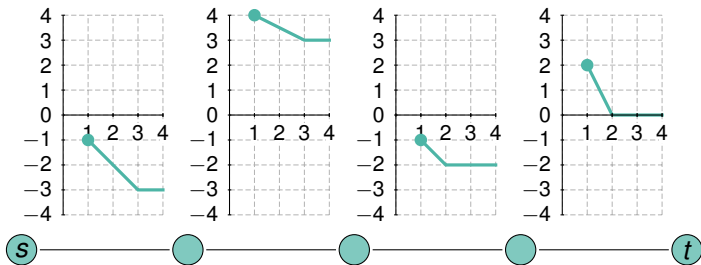
Initial state of charge: 4



=

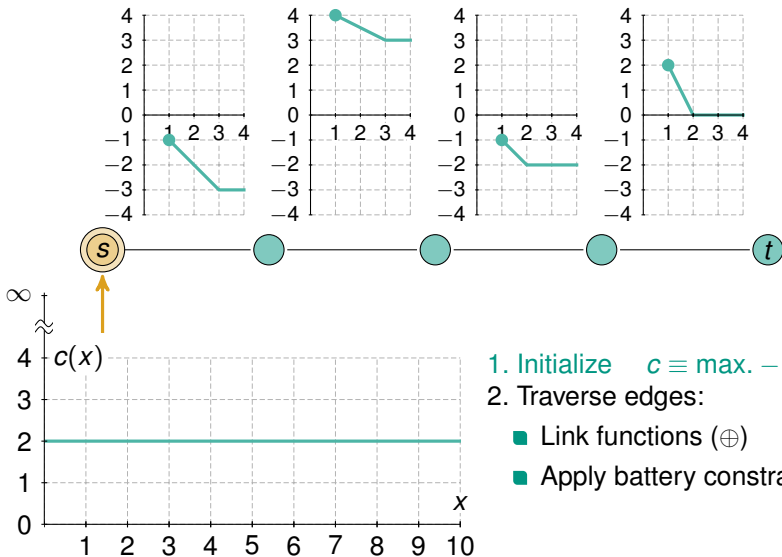


# Tradeoff Function Propagation (TFP)

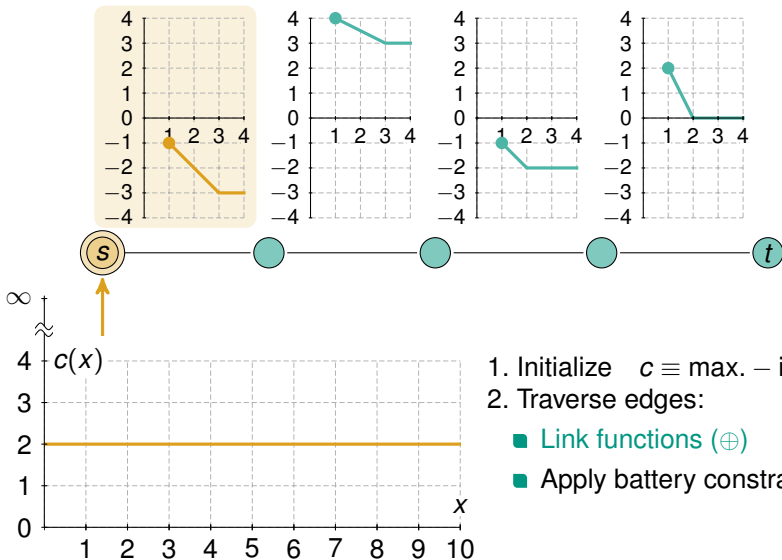


1. Initialize  $c \equiv \max.$  – init. SoC
2. Traverse edges:
  - Link functions ( $\oplus$ )
  - Apply battery constraints

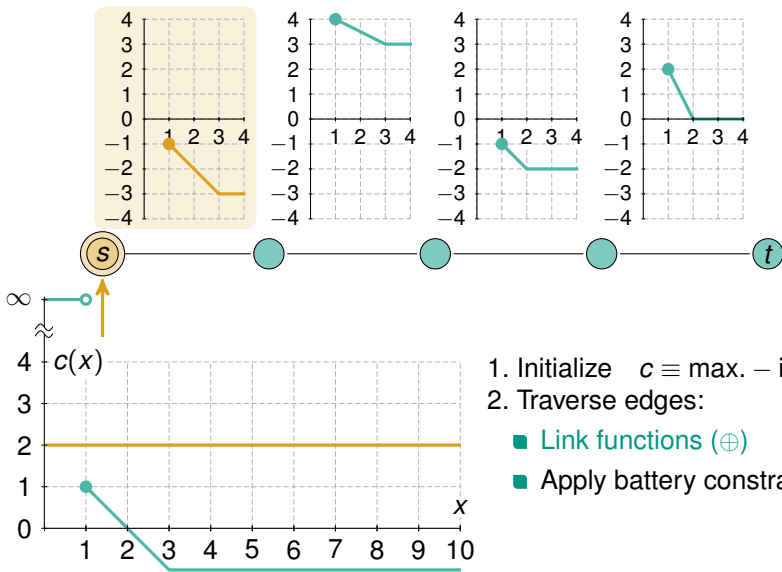
# Tradeoff Function Propagation (TFP)



# Tradeoff Function Propagation (TFP)



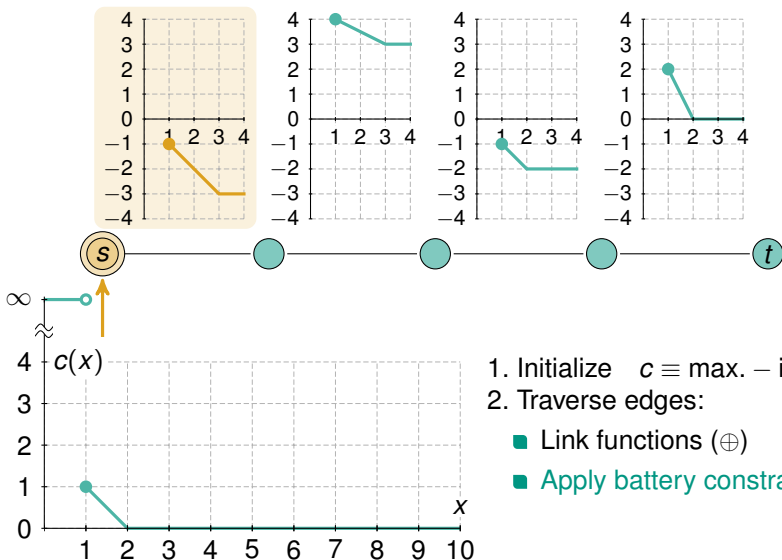
# Tradeoff Function Propagation (TFP)



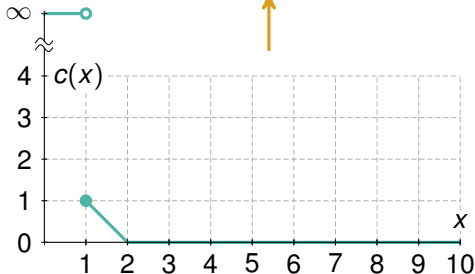
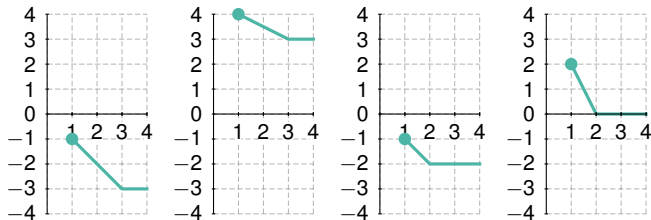
1. Initialize  $c \equiv \max.$  – init. SoC
2. Traverse edges:
  - Link functions ( $\oplus$ )
  - Apply battery constraints



# Tradeoff Function Propagation (TFP)

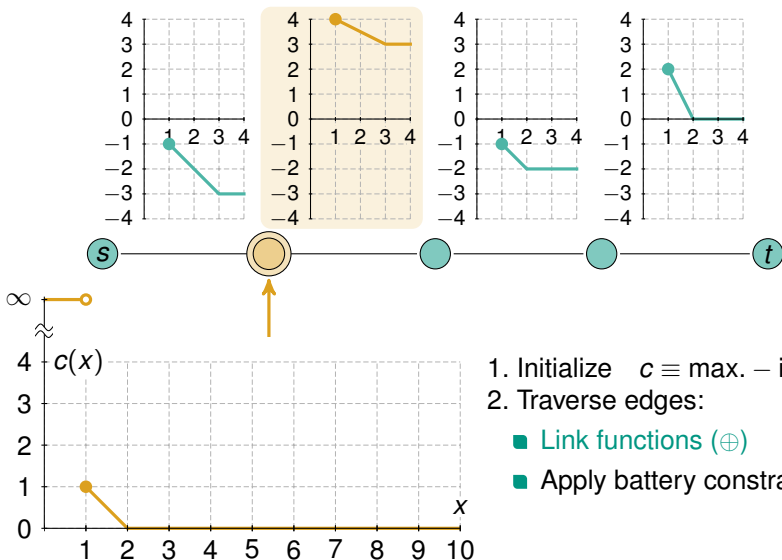


# Tradeoff Function Propagation (TFP)

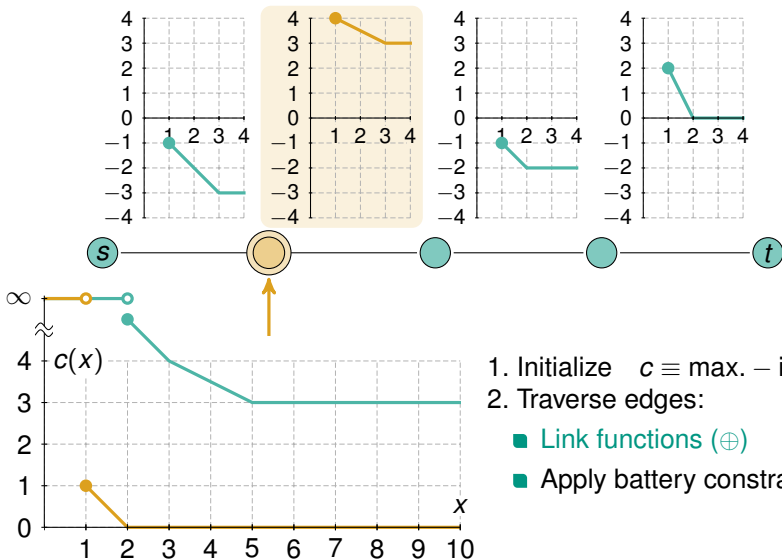


1. Initialize  $c \equiv \max.$  – init. SoC
2. Traverse edges:
  - Link functions ( $\oplus$ )
  - Apply battery constraints

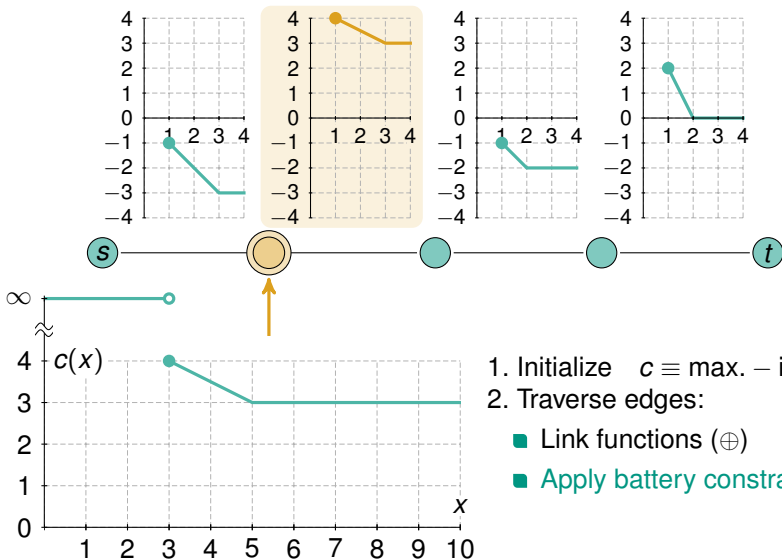
# Tradeoff Function Propagation (TFP)



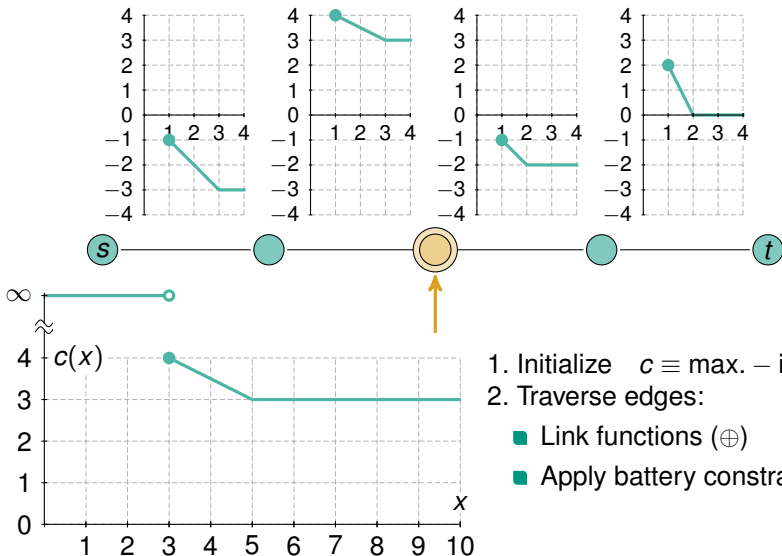
# Tradeoff Function Propagation (TFP)



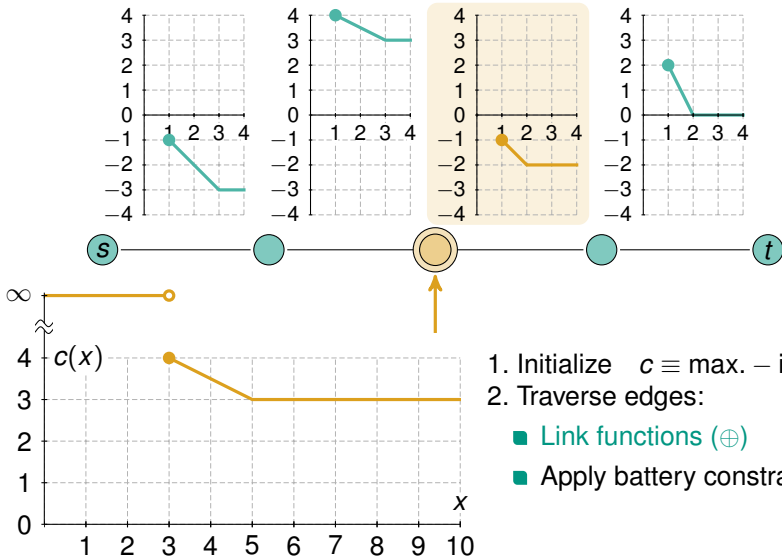
# Tradeoff Function Propagation (TFP)



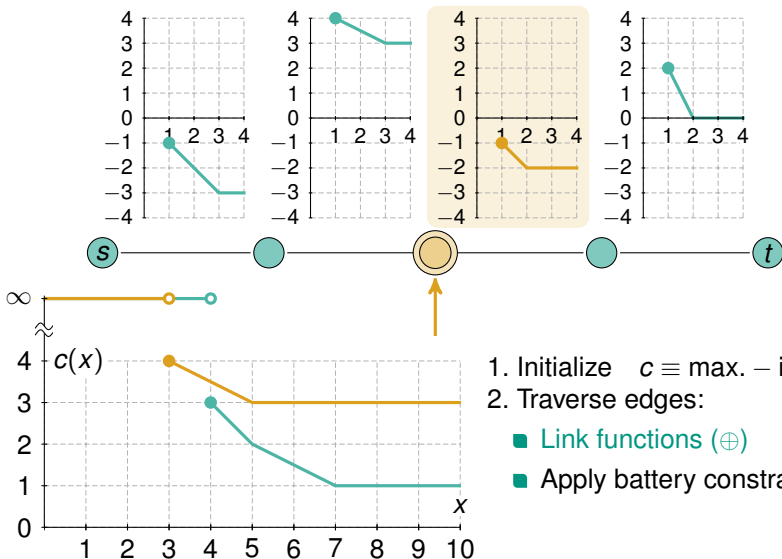
# Tradeoff Function Propagation (TFP)



# Tradeoff Function Propagation (TFP)



# Tradeoff Function Propagation (TFP)

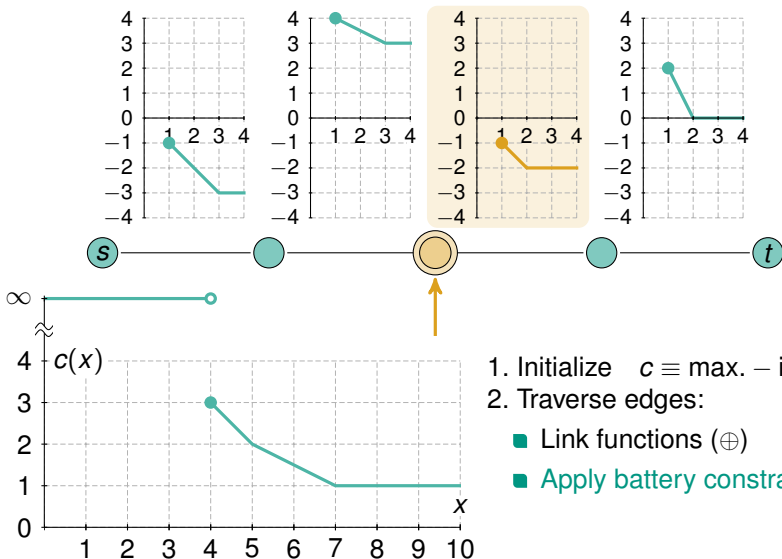


1. Initialize  $c \equiv \max.$  – init. SoC
2. Traverse edges:

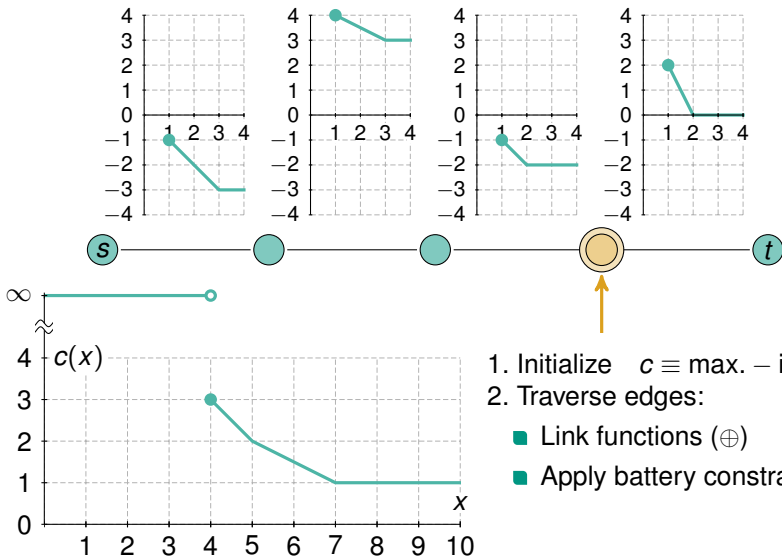
- Link functions ( $\oplus$ )
- Apply battery constraints



# Tradeoff Function Propagation (TFP)

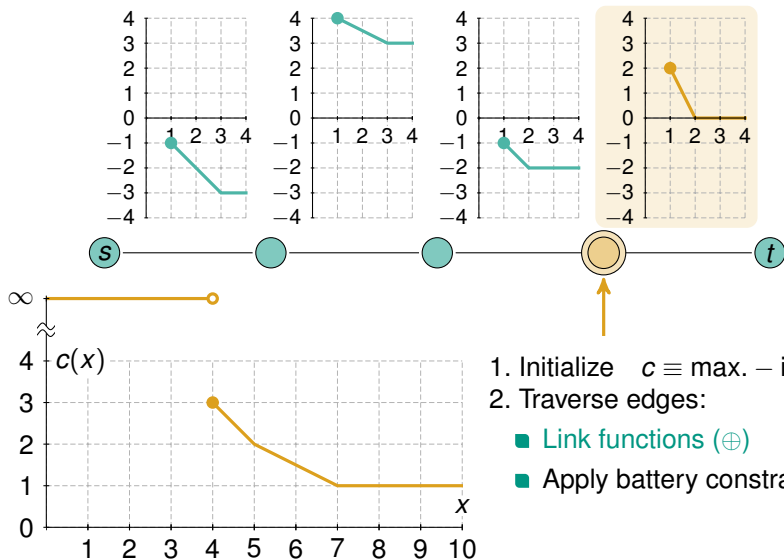


# Tradeoff Function Propagation (TFP)

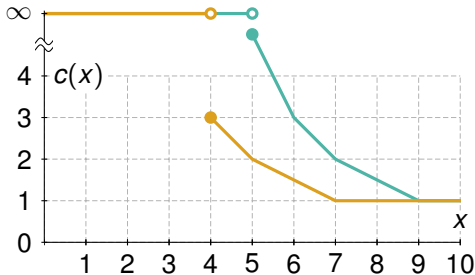
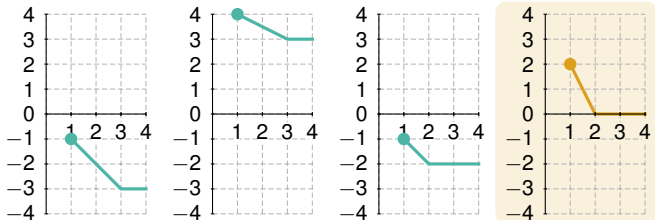


1. Initialize  $c \equiv \max.$  – init. SoC
2. Traverse edges:
  - Link functions ( $\oplus$ )
  - Apply battery constraints

# Tradeoff Function Propagation (TFP)

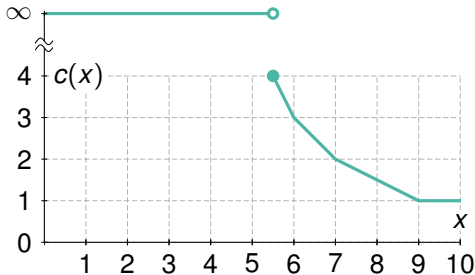
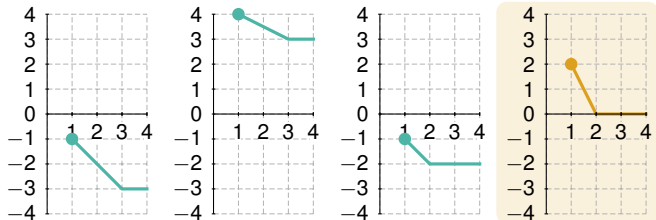


# Tradeoff Function Propagation (TFP)



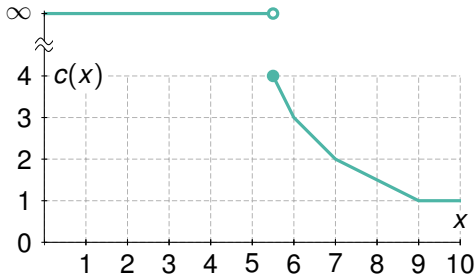
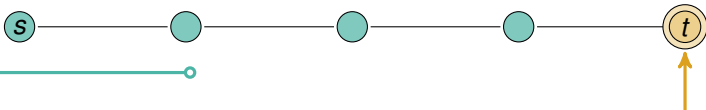
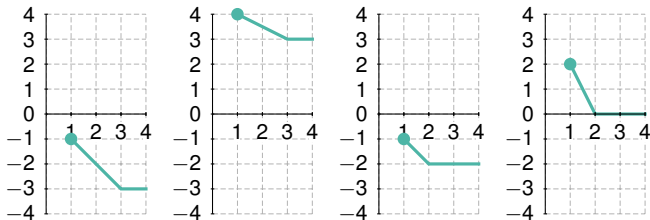
1. Initialize  $c \equiv \max.$  – init. SoC
2. Traverse edges:
  - Link functions ( $\oplus$ )
  - Apply battery constraints

# Tradeoff Function Propagation (TFP)



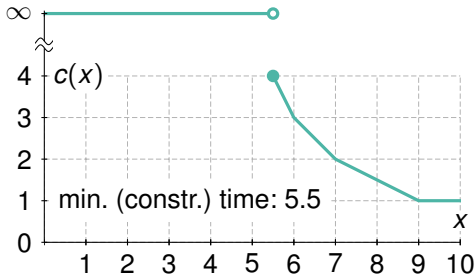
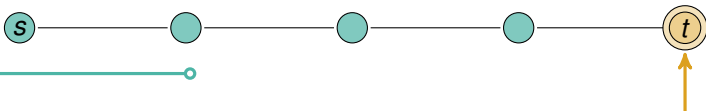
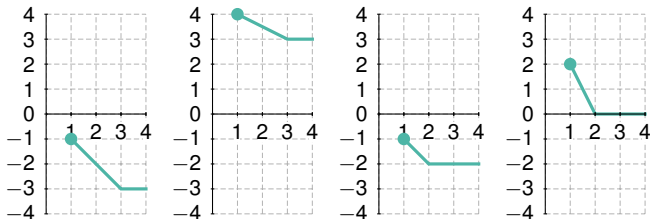
1. Initialize  $c \equiv \max.$  – init. SoC
2. Traverse edges:
  - Link functions ( $\oplus$ )
  - Apply battery constraints

# Tradeoff Function Propagation (TFP)



1. Initialize  $c \equiv \max.$  – init. SoC
2. Traverse edges:
  - Link functions ( $\oplus$ )
  - Apply battery constraints

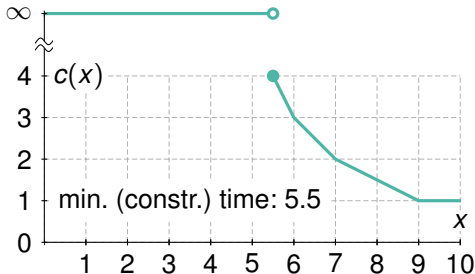
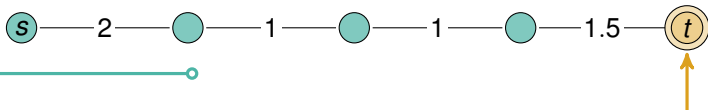
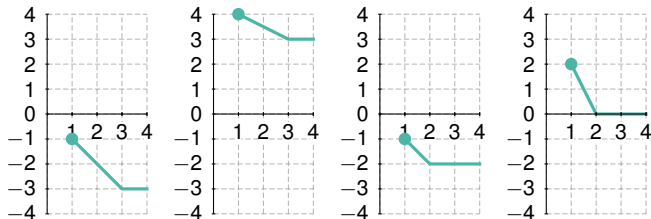
# Tradeoff Function Propagation (TFP)



1. Initialize  $c \equiv \max.$  – init. SoC
2. Traverse edges:

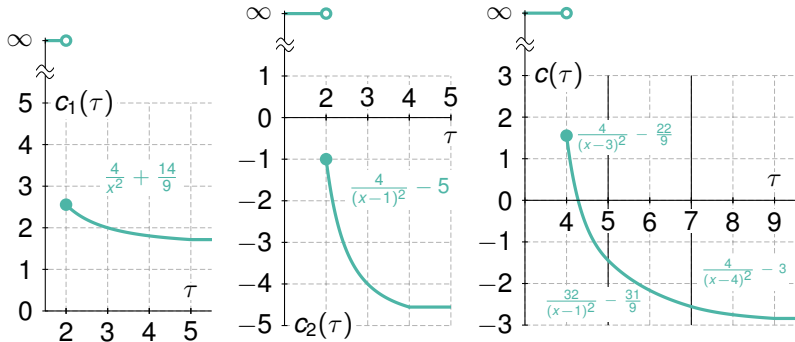
- Link functions ( $\oplus$ )
- Apply battery constraints

# Tradeoff Function Propagation (TFP)



1. Initialize  $c \equiv \max.$  – init. SoC
2. Traverse edges:
  - Link functions ( $\oplus$ )
  - Apply battery constraints

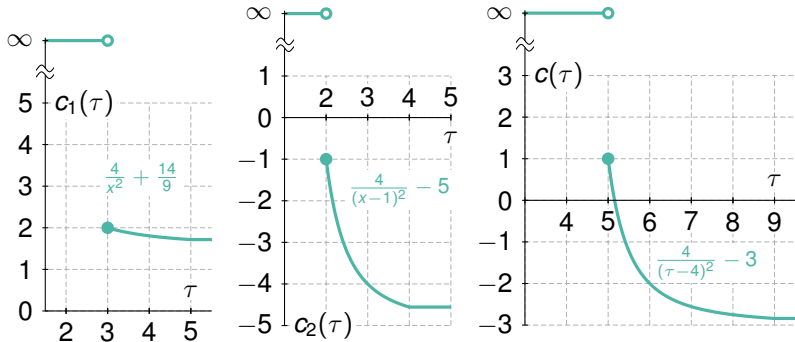




Initialer SoC (an  $c_1$ ): 4, min. SoC: 0, max. SoC: 8

Battery Constraints beeinflussen Resultat zusätzlich:

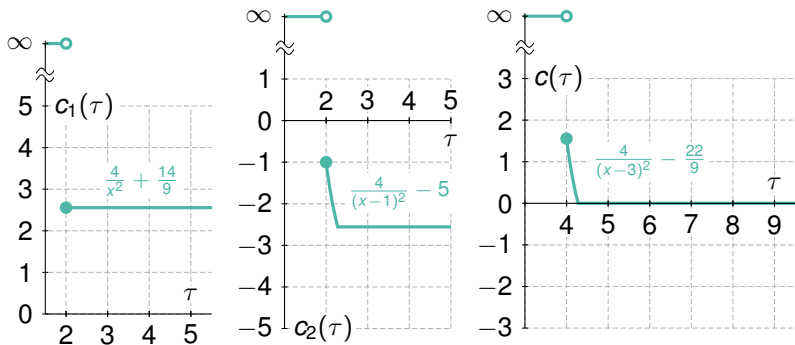
- Akku fast leer: Geschwindigkeit muss auf  $c_1$  reduziert werden
- Akku voll: Langsam fahren auf  $c_2$  lohnt nicht



Initialer SoC (an  $c_1$ ): 2, min. SoC: 0, max. SoC: 8

Battery Constraints beeinflussen Resultat zusätzlich:

- Akku fast leer: Geschwindigkeit muss auf  $c_1$  reduziert werden
- Akku voll: Langsam fahren auf  $c_2$  lohnt nicht



Initialer SoC (an  $c_1$ ): 8, min. SoC: 0, max. SoC: 8

Battery Constraints beeinflussen Resultat zusätzlich:

- Akku fast leer: Geschwindigkeit muss auf  $c_1$  reduziert werden
- Akku voll: Langsam fahren auf  $c_2$  lohnt nicht

## Herausforderungen:

- Tradeoff-Funktionen in Suche integrieren
- Battery Constraints “on-the-fly” anwenden
- Label Sets verwalten (mehrere Fkt. pro Knoten!)

## Experimente:

Straßennetz Europa, PHEM Verbräuche

- EV mit 2 kWh Akku (Reichweite  $\sim 10$  km)

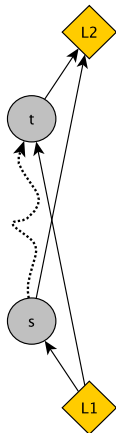
Algorithm	# Labels	# Cmp.	Time [ms]
Multi-graph	30 990 k	21 301 M	47 755
Tradeoff fct.	103 k	4 M	444

(Hardware: Intel Xeon E5-1630v3, 3.7 GHz, 128 GiB RAM)

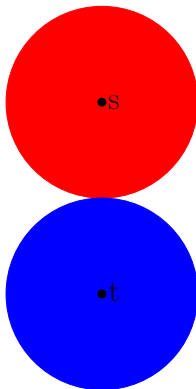
- Pfade auf Multigraph sind länger (diskretisierte Verbräuche)!

Realistisches Modell zahlt sich aus

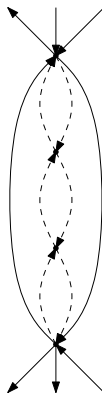
## Landmarken



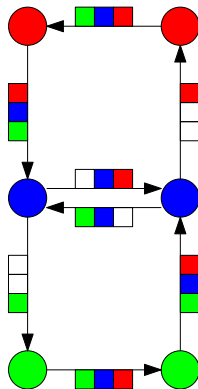
## Bidirektionale Suche



## Kontraktion

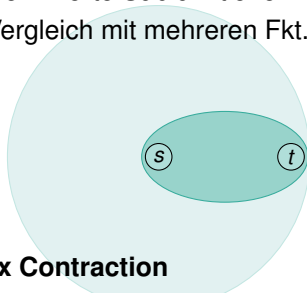


## Arc-Flags



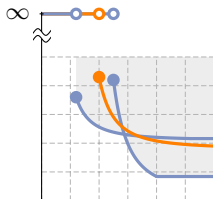
## Verbesserte Dominanz-Checks

- Dominierte Subfunktionen
- Vergleich mit mehreren Fkt.



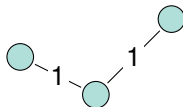
## Vertex Contraction

- Shortcuts entspr. Pfaden
- Verbrauch auf Shortcut?



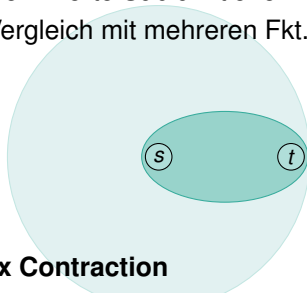
## A\* Suche

- Ähnlich wie bei CFP
- Berücksichtige SoC



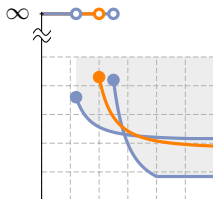
## Verbesserte Dominanz-Checks

- Dominierte Subfunktionen
- Vergleich mit mehreren Fkt.



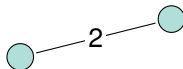
## Vertex Contraction

- Shortcuts entspr. Pfaden
- Verbrauch auf Shortcut?



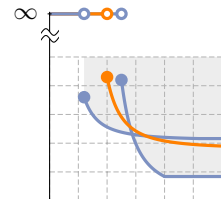
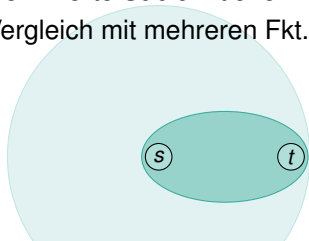
## A\* Suche

- Ähnlich wie bei CFP
- Berücksichtige SoC



## Verbesserte Dominanz-Checks

- Dominierte Subfunktionen
- Vergleich mit mehreren Fkt.

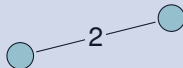


## A\* Suche

- Ähnlich wie bei CFP
- Berücksichtige SoC

## Vertex Contraction

- Shortcuts entspr. Pfaden
- Verbrauch auf Shortcut?





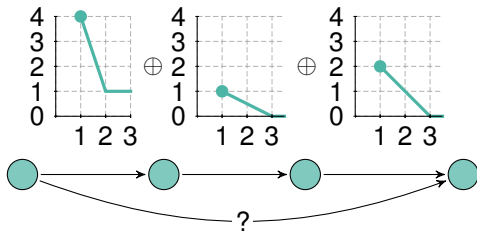
# Konstruktion von Shortcuts

**Idee:** Shortcuts repräsentieren einzelne Pfade

**Aber:** Energieverbrauch abh. von Geschwindigkeit **und** SoC

- Alle Tradeoff-Funktionen explizit speichern?
- Speichere **bivariate** Funktionen  $f(x, b)$ ?

Betrachte Sonderfälle:



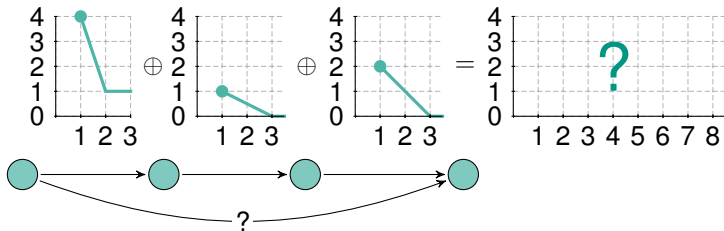
# Konstruktion von Shortcuts

**Idee:** Shortcuts repräsentieren einzelne Pfade

**Aber:** Energieverbrauch abh. von Geschwindigkeit **und** SoC

- Alle Tradeoff-Funktionen explizit speichern?
- Speichere **bivariate** Funktionen  $f(x, b)$ ?

Betrachte Sonderfälle:



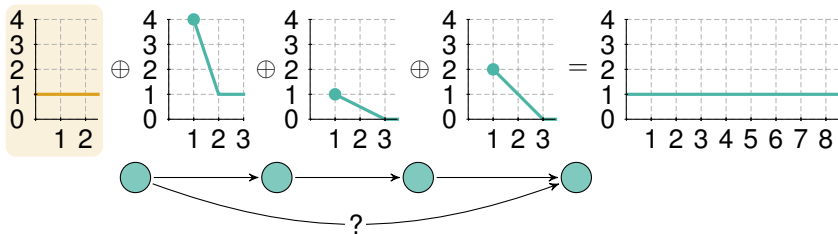
# Konstruktion von Shortcuts

**Idee:** Shortcuts repräsentieren einzelne Pfade

**Aber:** Energieverbrauch abh. von Geschwindigkeit **und** SoC

- Alle Tradeoff-Funktionen explizit speichern?
- Speichere **bivariate** Funktionen  $f(x, b)$ ?

Betrachte Sonderfälle:



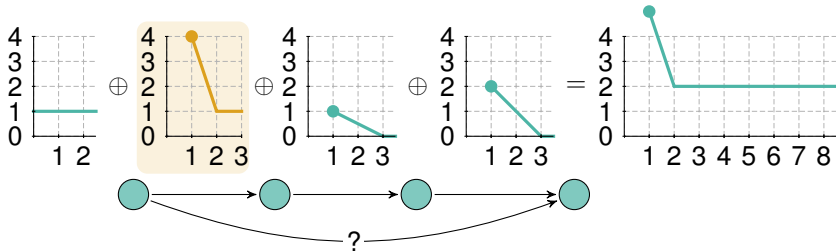
# Konstruktion von Shortcuts

**Idee:** Shortcuts repräsentieren einzelne Pfade

**Aber:** Energieverbrauch abh. von Geschwindigkeit **und** SoC

- Alle Tradeoff-Funktionen explizit speichern?
- Speichere **bivariate** Funktionen  $f(x, b)$ ?

Betrachte Sonderfälle:



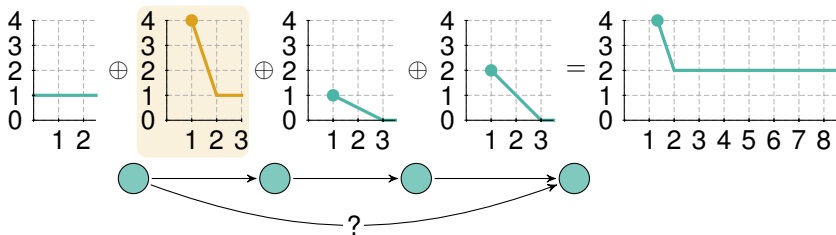
# Konstruktion von Shortcuts

**Idee:** Shortcuts repräsentieren einzelne Pfade

**Aber:** Energieverbrauch abh. von Geschwindigkeit **und** SoC

- Alle Tradeoff-Funktionen explizit speichern?
- Speichere **bivariate** Funktionen  $f(x, b)$ ?

Betrachte Sonderfälle:



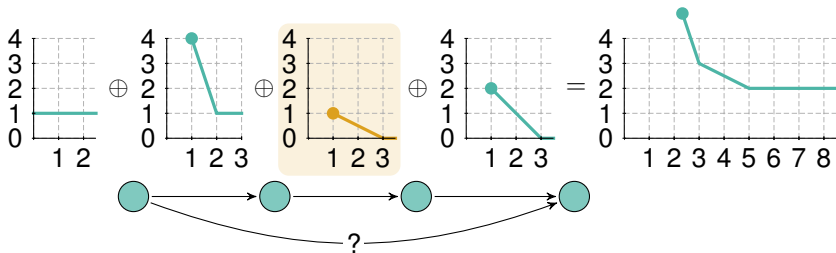
# Konstruktion von Shortcuts

**Idee:** Shortcuts repräsentieren einzelne Pfade

**Aber:** Energieverbrauch abh. von Geschwindigkeit **und** SoC

- Alle Tradeoff-Funktionen explizit speichern?
- Speichere **bivariate** Funktionen  $f(x, b)$ ?

Betrachte Sonderfälle:



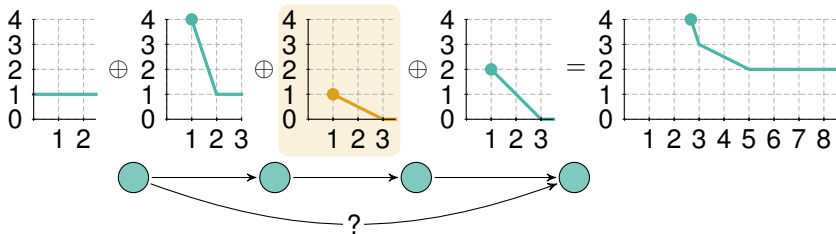
# Konstruktion von Shortcuts

**Idee:** Shortcuts repräsentieren einzelne Pfade

**Aber:** Energieverbrauch abh. von Geschwindigkeit **und** SoC

- Alle Tradeoff-Funktionen explizit speichern?
- Speichere **bivariate** Funktionen  $f(x, b)$ ?

Betrachte Sonderfälle:



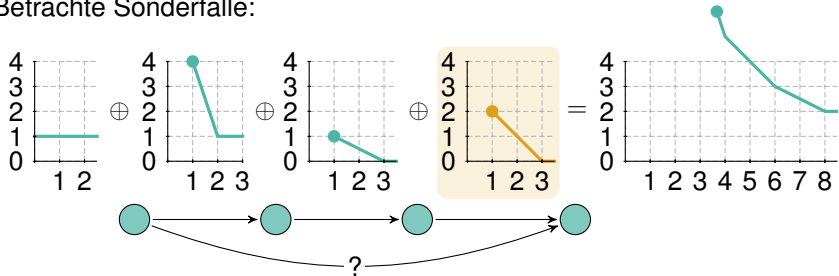
# Konstruktion von Shortcuts

**Idee:** Shortcuts repräsentieren einzelne Pfade

**Aber:** Energieverbrauch abh. von Geschwindigkeit **und** SoC

- Alle Tradeoff-Funktionen explizit speichern?
- Speichere **bivariate** Funktionen  $f(x, b)$ ?

Betrachte Sonderfälle:





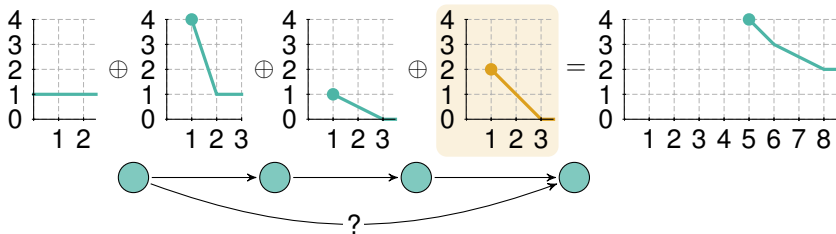
# Konstruktion von Shortcuts

**Idee:** Shortcuts repräsentieren einzelne Pfade

**Aber:** Energieverbrauch abh. von Geschwindigkeit **und** SoC

- Alle Tradeoff-Funktionen explizit speichern?
- Speichere **bivariate** Funktionen  $f(x, b)$ ?

Betrachte Sonderfälle:



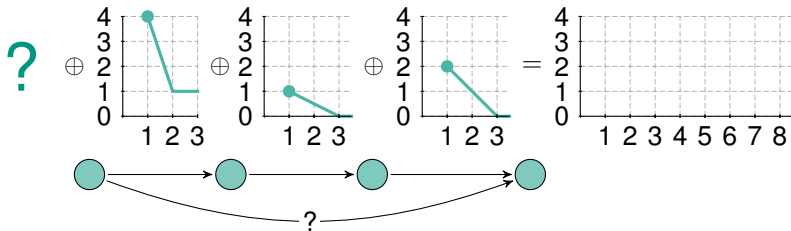
# Konstruktion von Shortcuts

**Idee:** Shortcuts repräsentieren einzelne Pfade

**Aber:** Energieverbrauch abh. von Geschwindigkeit **und** SoC

- Alle Tradeoff-Funktionen explizit speichern?
- Speichere **bivariate** Funktionen  $f(x, b)$ ?

Betrachte Sonderfälle:



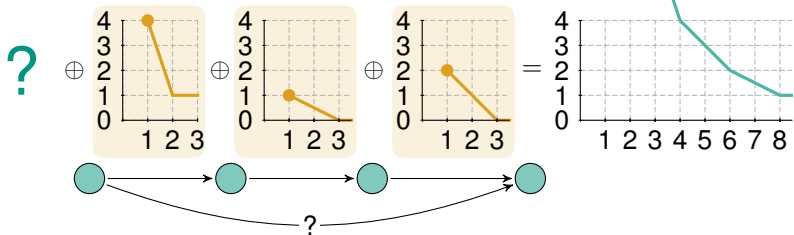
# Konstruktion von Shortcuts

**Idee:** Shortcuts repräsentieren einzelne Pfade

**Aber:** Energieverbrauch abh. von Geschwindigkeit **und** SoC

- Alle Tradeoff-Funktionen explizit speichern?
- Speichere **bivariate** Funktionen  $f(x, b)$ ?

Betrachte Sonderfälle:



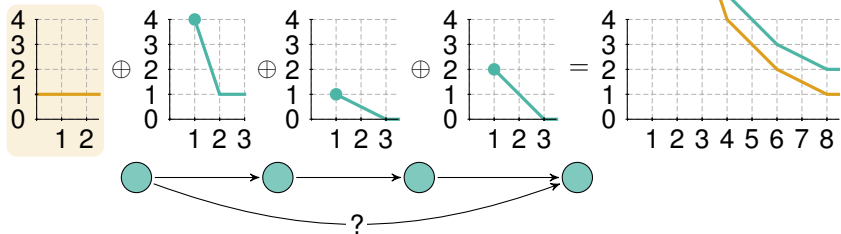
# Konstruktion von Shortcuts

**Idee:** Shortcuts repräsentieren einzelne Pfade

**Aber:** Energieverbrauch abh. von Geschwindigkeit **und** SoC

- Alle Tradeoff-Funktionen explizit speichern?
- Speichere **bivariate** Funktionen  $f(x, b)$ ?

Betrachte Sonderfälle:



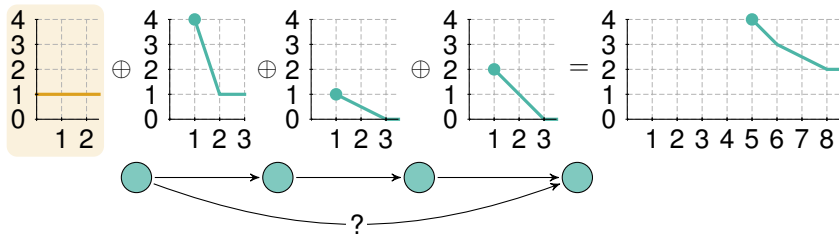
# Konstruktion von Shortcuts

**Idee:** Shortcuts repräsentieren einzelne Pfade

**Aber:** Energieverbrauch abh. von Geschwindigkeit **und** SoC

- Alle Tradeoff-Funktionen explizit speichern?
- Speichere **bivariate** Funktionen  $f(x, b)$ ?

Betrachte Sonderfälle:



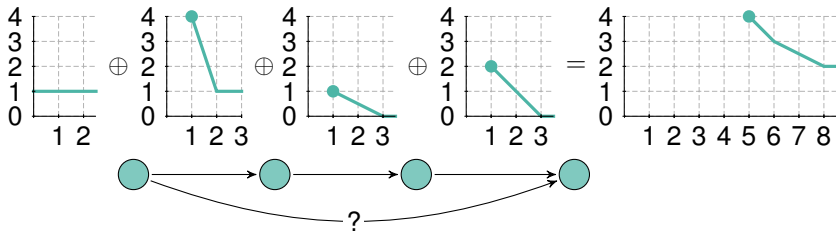
# Konstruktion von Shortcuts

**Idee:** Shortcuts repräsentieren einzelne Pfade

**Aber:** Energieverbrauch abh. von Geschwindigkeit **und** SoC

- Alle Tradeoff-Funktionen explizit speichern?
- Speichere **bivariate** Funktionen  $f(x, b)$ ?

Betrachte Sonderfälle:

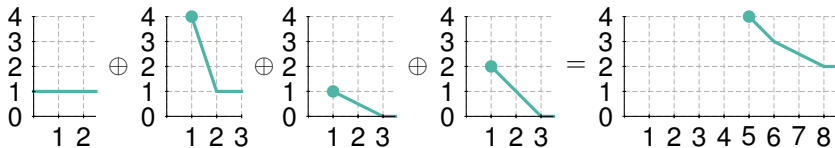


**Idee:** Shortcuts repräsentieren einzelne Pfade

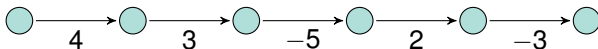
**Aber:** Energieverbrauch abh. von Geschwindigkeit **und** SoC

- Alle Tradeoff-Funktionen explizit speichern?
- Speichere **bivariate** Funktionen  $f(x, b)$ ?

Betrachte Sonderfälle:



⇒ Kontrahiere nur falls alle Verbräuche selbes Vorzeichen haben

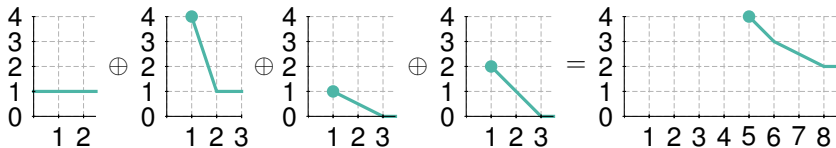


**Idee:** Shortcuts repräsentieren einzelne Pfade

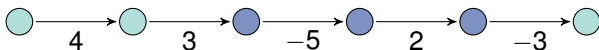
**Aber:** Energieverbrauch abh. von Geschwindigkeit **und** SoC

- Alle Tradeoff-Funktionen explizit speichern?
- Speichere **bivariate** Funktionen  $f(x, b)$ ?

Betrachte Sonderfälle:



⇒ Kontrahiere nur falls alle Verbräuche selbes Vorzeichen haben





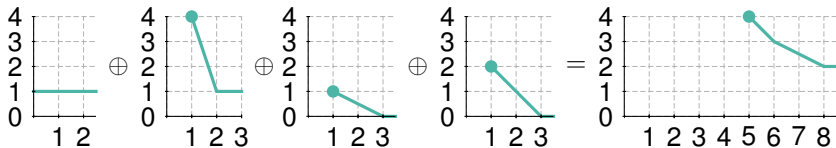
# Konstruktion von Shortcuts

**Idee:** Shortcuts repräsentieren einzelne Pfade

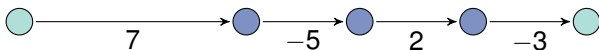
**Aber:** Energieverbrauch abh. von Geschwindigkeit **und** SoC

- Alle Tradeoff-Funktionen explizit speichern?
- Speichere **bivariate** Funktionen  $f(x, b)$ ?

Betrachte Sonderfälle:



$\Rightarrow$  Kontrahiere nur falls alle Verbräuche selbes Vorzeichen haben



# Entladende Pfade

## Kontrahiere mehr Knoten?

Entladende Pfade: Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen

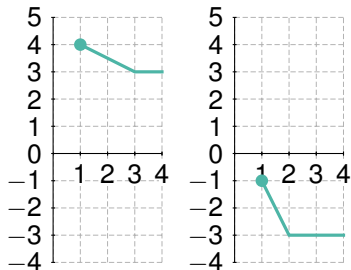


# Entladende Pfade

## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen

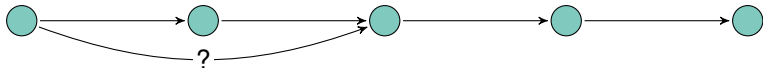
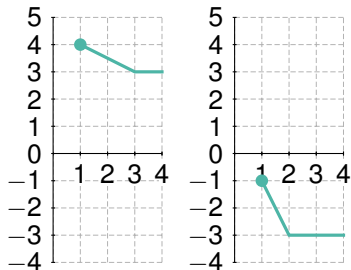


# Entladende Pfade

## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen

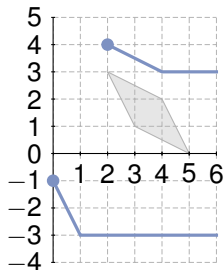


# Entladende Pfade

## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen

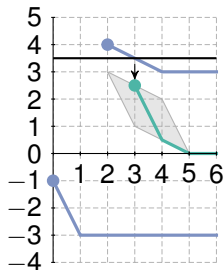


# Entladende Pfade

## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen

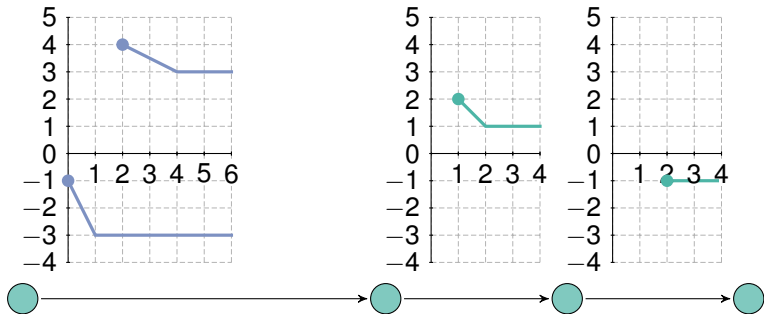


# Entladende Pfade

## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen

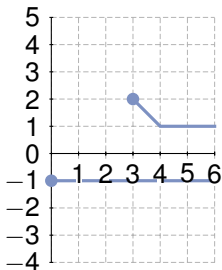
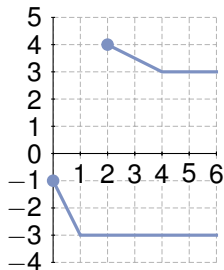


# Entladende Pfade

## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen



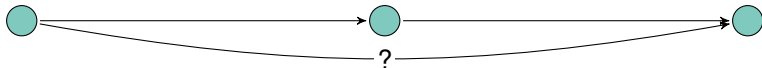
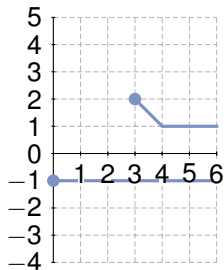
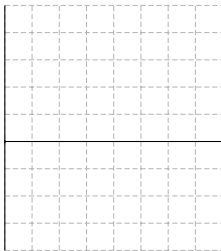
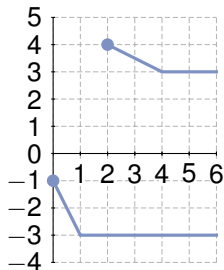


# Entladende Pfade

## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen

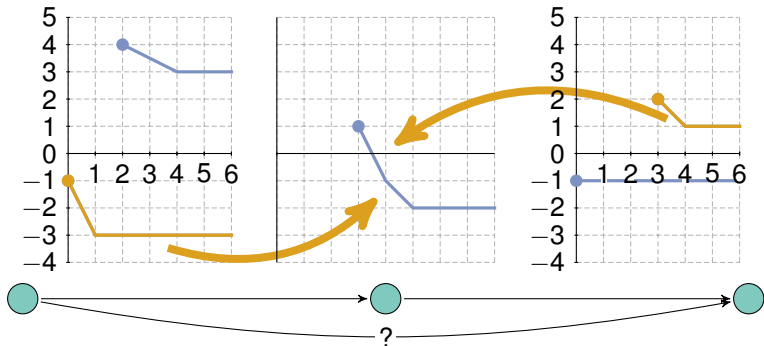


# Entladende Pfade

## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen

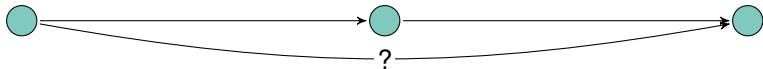
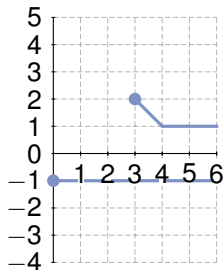
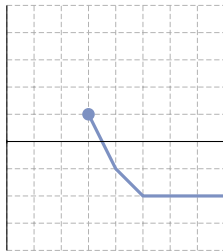
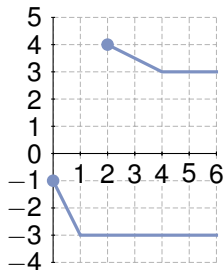


# Entladende Pfade

## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen

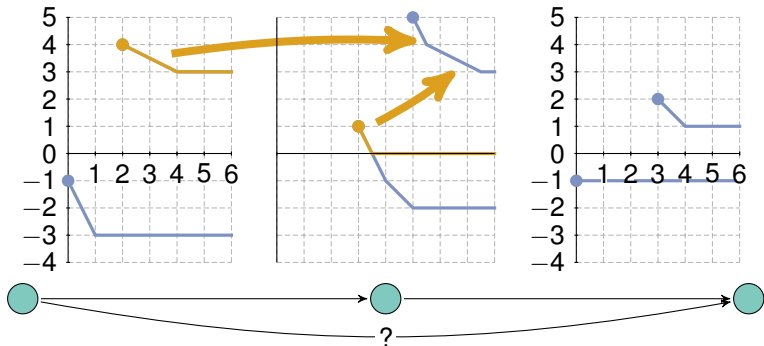


# Entladende Pfade

## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen

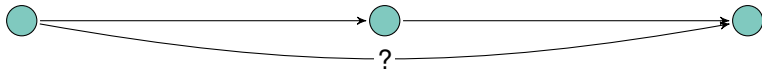
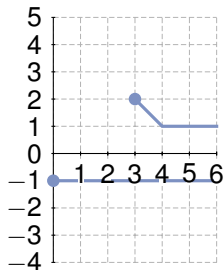
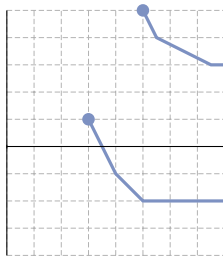
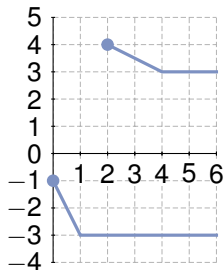


# Entladende Pfade

## Kontrahiere mehr Knoten?

Entladende Pfade: Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen

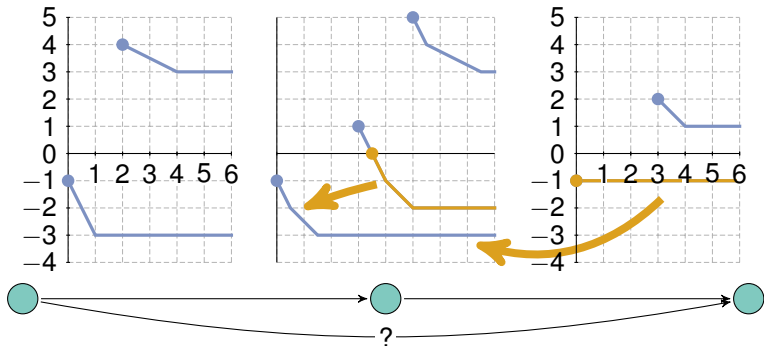


# Entladende Pfade

## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen

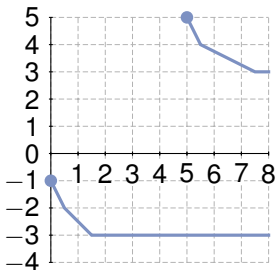


# Entladende Pfade

## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen

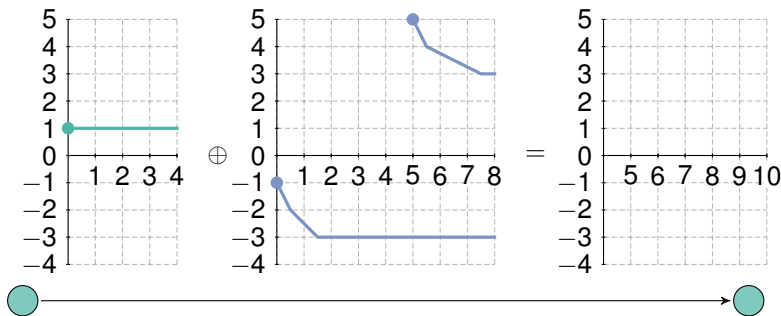


# Entladende Pfade

## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen



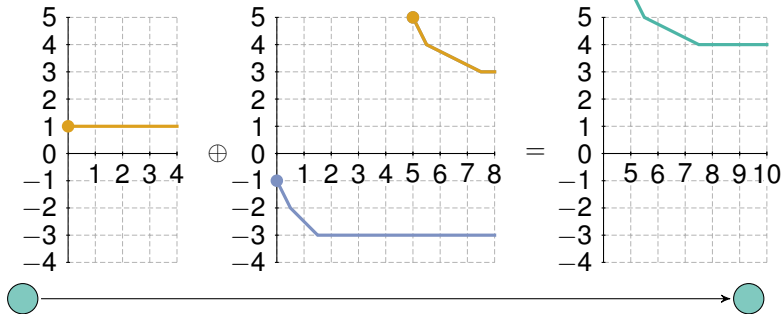


# Entladende Pfade

## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen

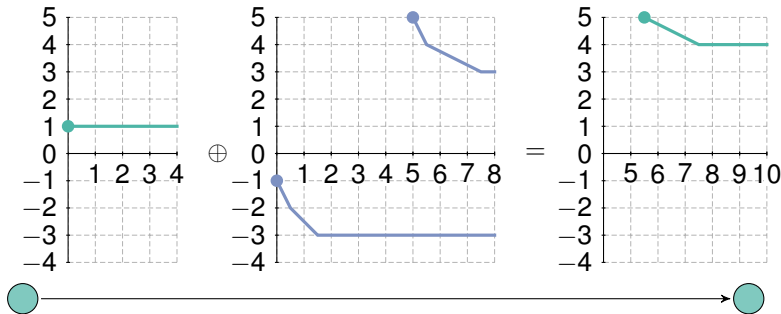


# Entladende Pfade

## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen

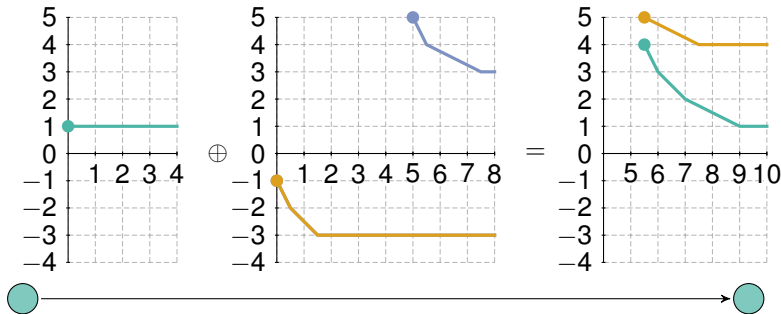


# Entladende Pfade

## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen

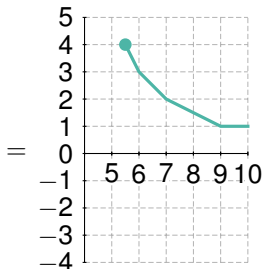
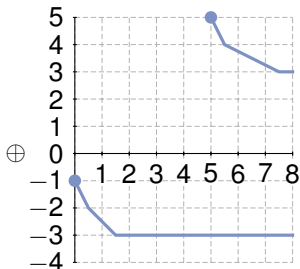
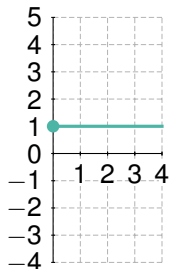


# Entladende Pfade

## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen



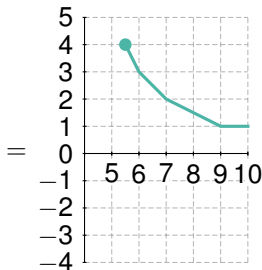
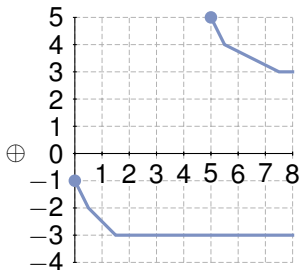
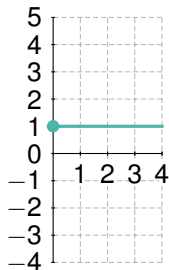
⇒ Kontrahiere Knoten inzident zu entladenden Shortcuts

# Entladende Pfade

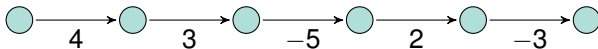
## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen



⇒ Kontrahiere Knoten inzident zu entladenden Shortcuts

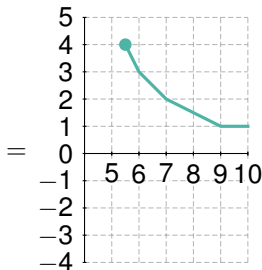
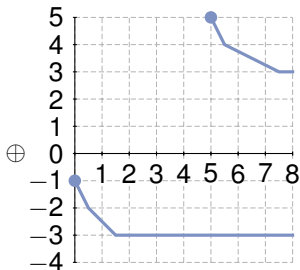
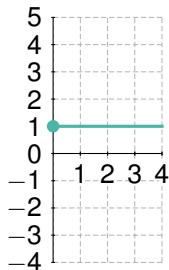


# Entladende Pfade

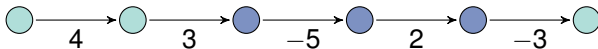
## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen



⇒ Kontrahiere Knoten inzident zu entladenden Shortcuts

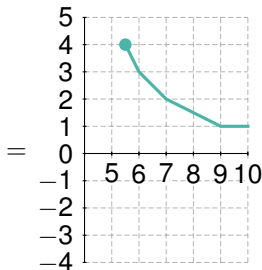
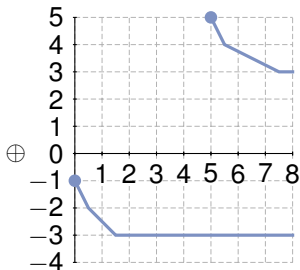
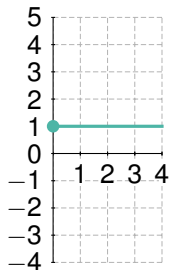


# Entladende Pfade

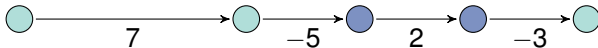
## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen



⇒ Kontrahiere Knoten inzident zu entladenden Shortcuts

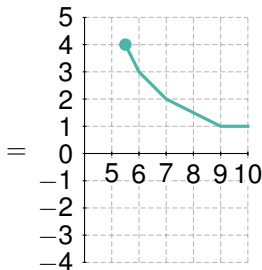
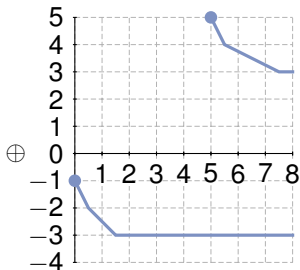
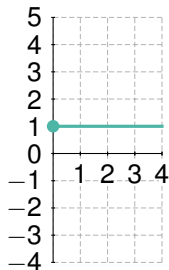


# Entladende Pfade

## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen



⇒ Kontrahiere Knoten inzident zu entladenden Shortcuts



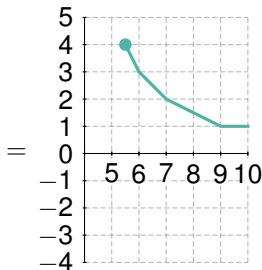
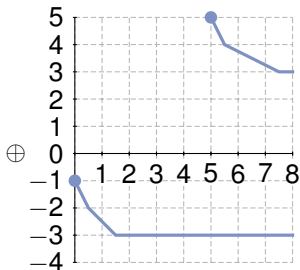
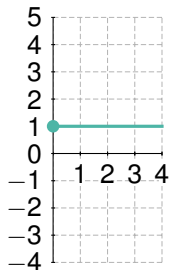


# Entladende Pfade

## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen



⇒ Kontrahiere Knoten inzident zu entladenden Shortcuts

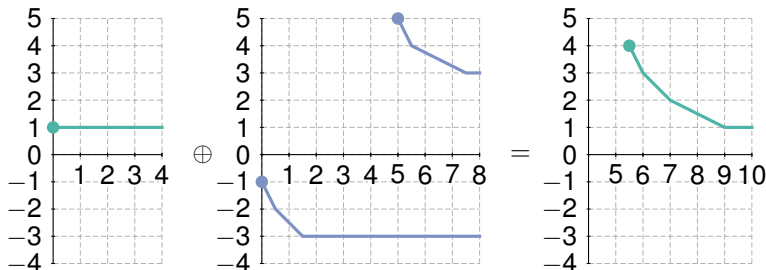


# Entladende Pfade

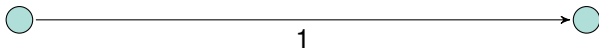
## Kontrahiere mehr Knoten?

**Entladende Pfade:** Verbrauch zu jedem Zeitpunkt nichtnegativ

- Es muss nur getestet werden, ob Akku leer
- Effizientere Operationen

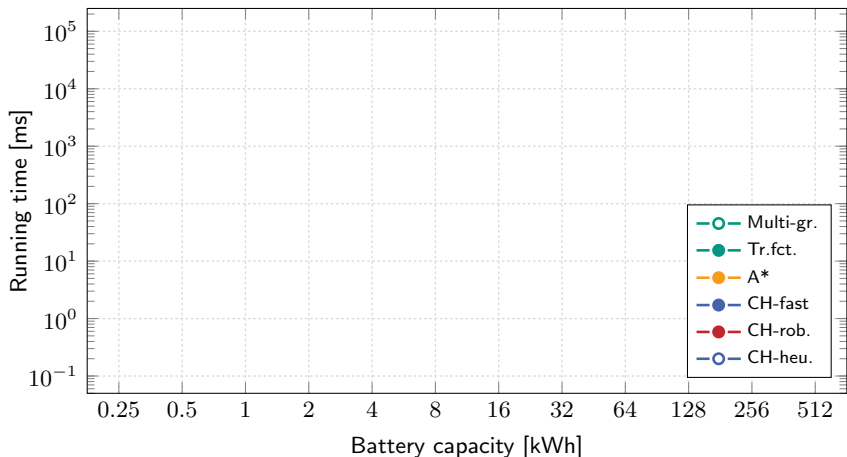


⇒ Kontrahiere Knoten inzident zu entladenden Shortcuts



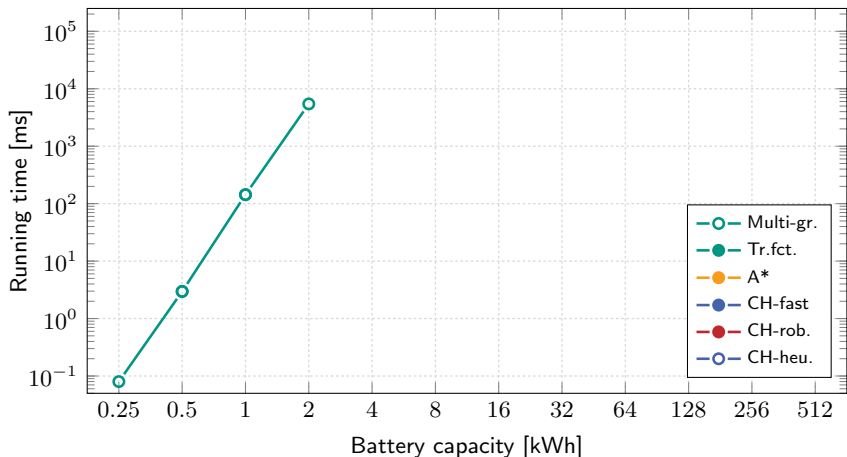
- Road network of Europe; max. running time: 60 min.

Hardware: Intel Xeon E5-1630v3, 3.7 GHz, 128 GiB RAM



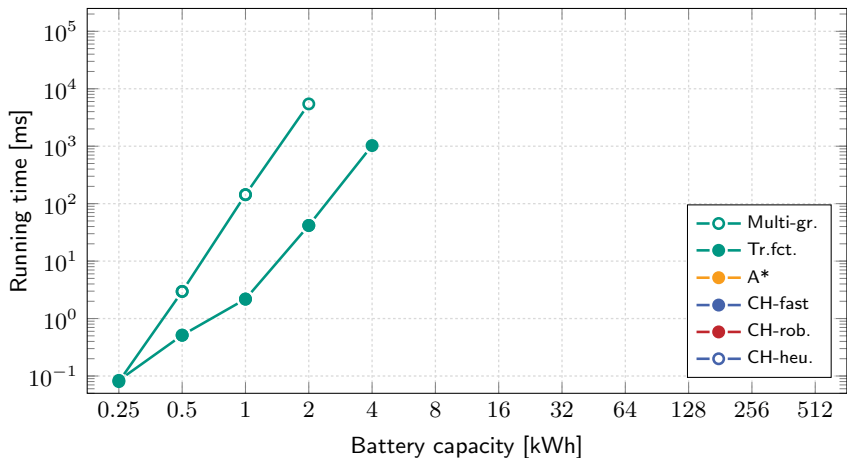
- Road network of Europe; max. running time: 60 min.

Hardware: Intel Xeon E5-1630v3, 3.7 GHz, 128 GiB RAM



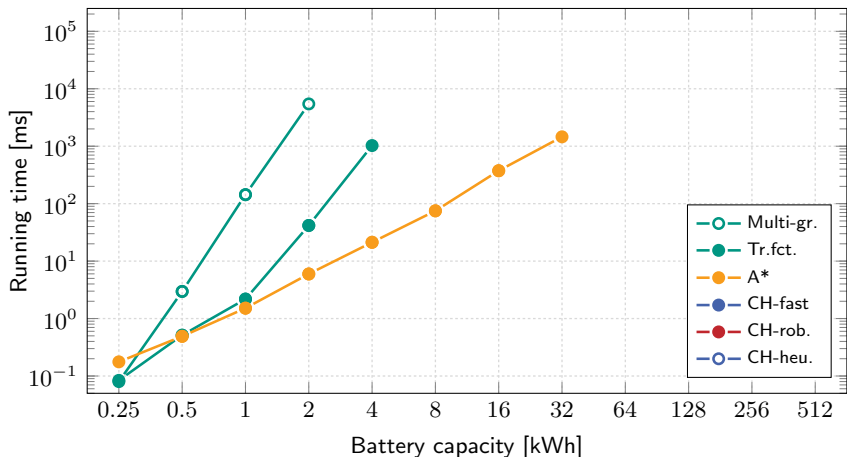
- Road network of Europe; max. running time: 60 min.

Hardware: Intel Xeon E5-1630v3, 3.7 GHz, 128 GiB RAM



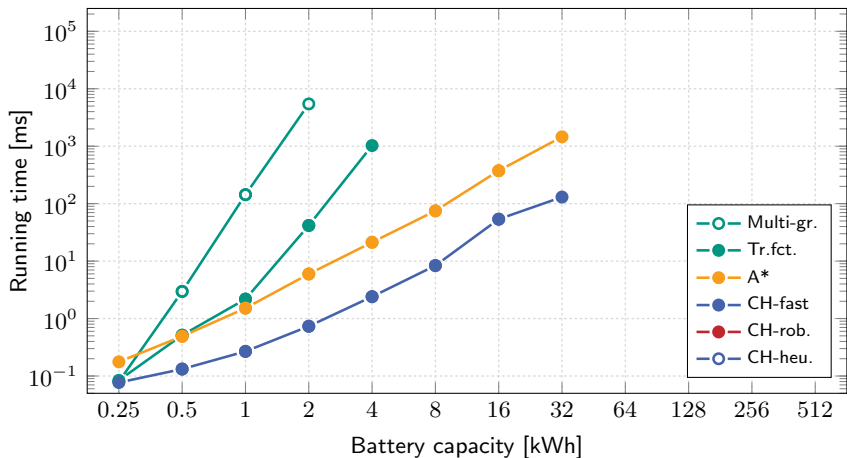
- Road network of Europe; max. running time: 60 min.

Hardware: Intel Xeon E5-1630v3, 3.7 GHz, 128 GiB RAM



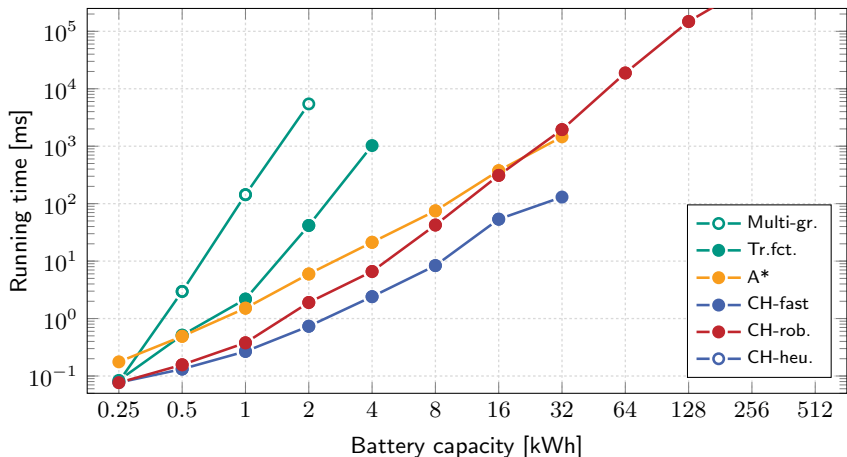
- Road network of Europe; max. running time: 60 min.

Hardware: Intel Xeon E5-1630v3, 3.7 GHz, 128 GiB RAM



- Road network of Europe; max. running time: 60 min.

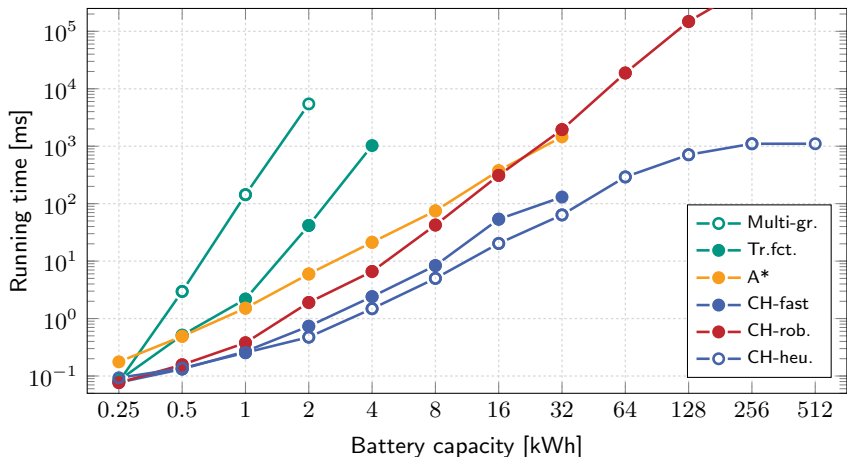
Hardware: Intel Xeon E5-1630v3, 3.7 GHz, 128 GiB RAM





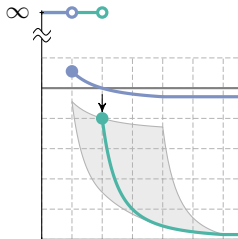
- Road network of Europe; max. running time: 60 min.

Hardware: Intel Xeon E5-1630v3, 3.7 GHz, 128 GiB RAM

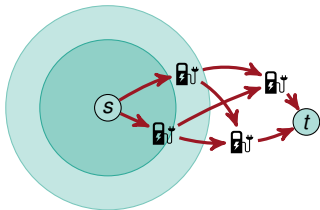


## Constrained Shortest Paths für EVs

- $\mathcal{NP}$ -schwere Probleme
- Realistische Modelle
- Engineering, Speedup-Techniken



Optimale Resultate in wenigen Sekunden für realistische Anfragen



## Offene Punkte

- Laden + Geschwindigkeit
- Turn costs
- Mehr Heuristiken
- Zeitabhängigkeit



Moritz Baum, Julian Dibbelt, Andreas Gemsa, Dorothea Wagner, and Tobias Zündorf.

Shortest feasible paths with charging stops for battery electric vehicles.

In *Proceedings of the 23rd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 44:1–44:10. ACM Press, 2015.



Moritz Baum, Julian Dibbelt, Lorenz Hübschle-Schneider, Thomas Pajor, and Dorothea Wagner.

Speed-consumption tradeoff for electric vehicle route planning.

In *Proceedings of the 14th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'14)*, volume 42 of *OpenAccess Series in Informatics (OASICs)*, pages 138–151. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.



Moritz Baum, Julian Dibbelt, Dorothea Wagner, and Tobias Zündorf.

Modeling and engineering constrained shortest path algorithms for battery electric vehicles.

In *Proceedings of the 25th Annual European Symposium on Algorithms (ESA'17)*, volume 87 of *Leibniz International Proceedings in Informatics*, pages 11:1–11:16, 2017.