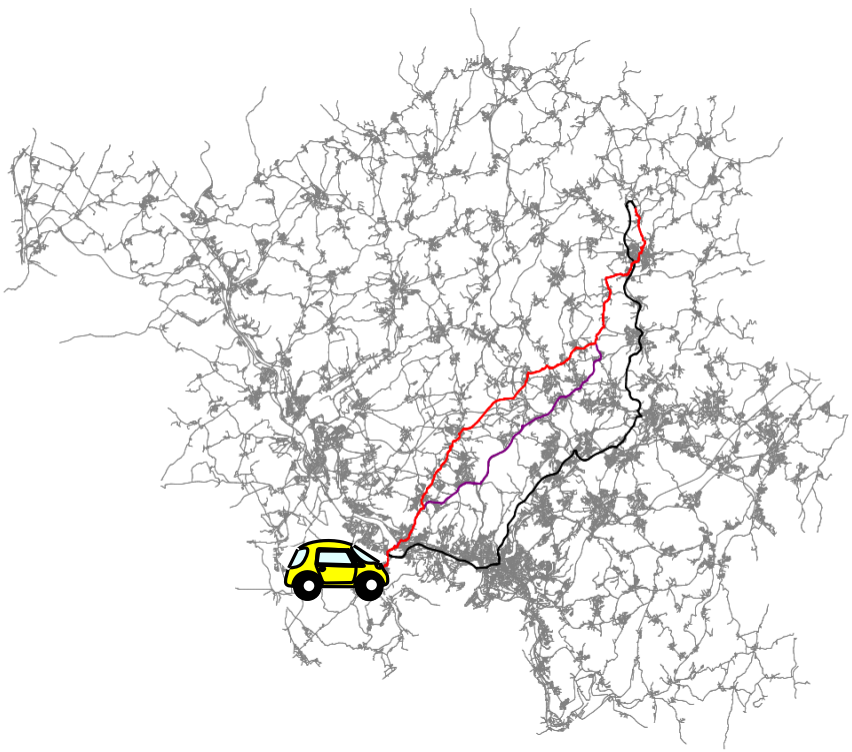
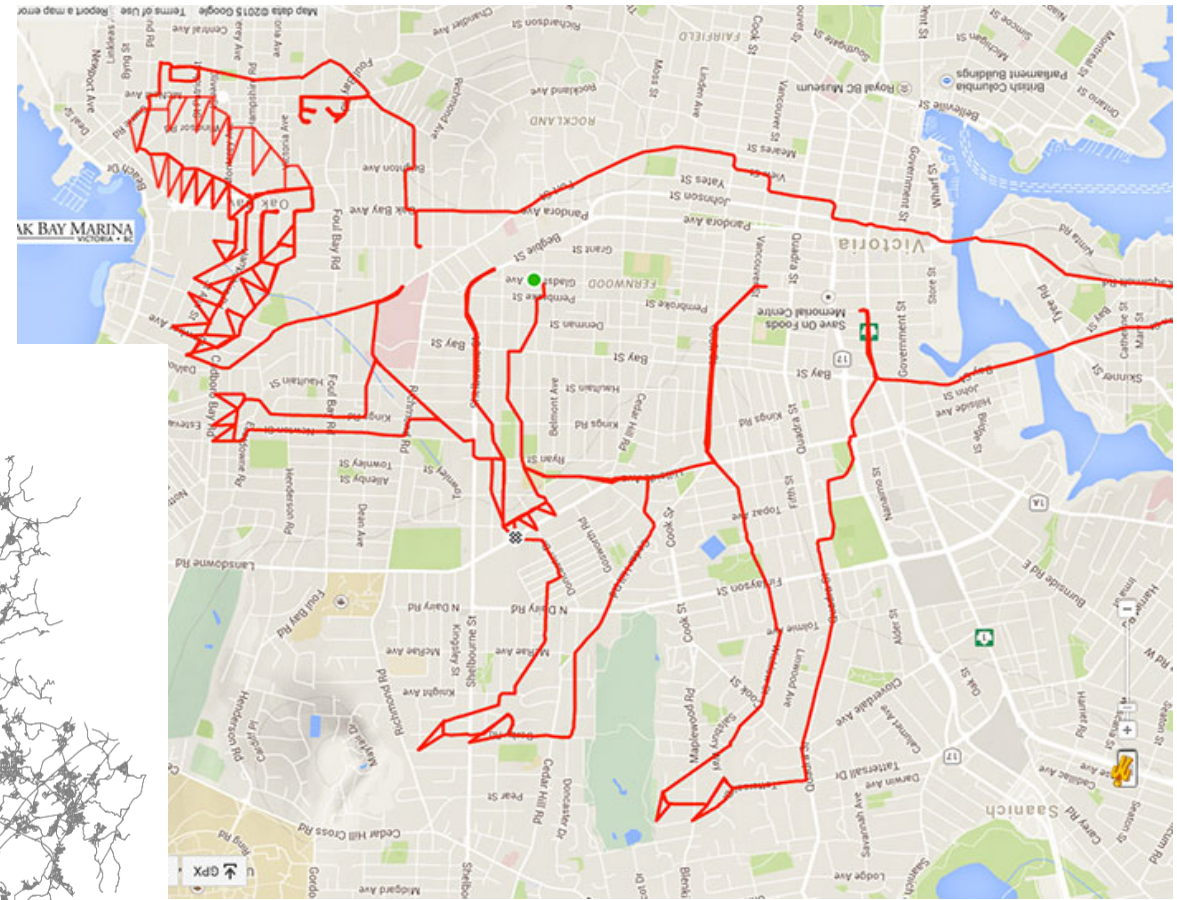
Personalisierte Routenplanung

Vorlesung Routenplanung, KIT, 2016



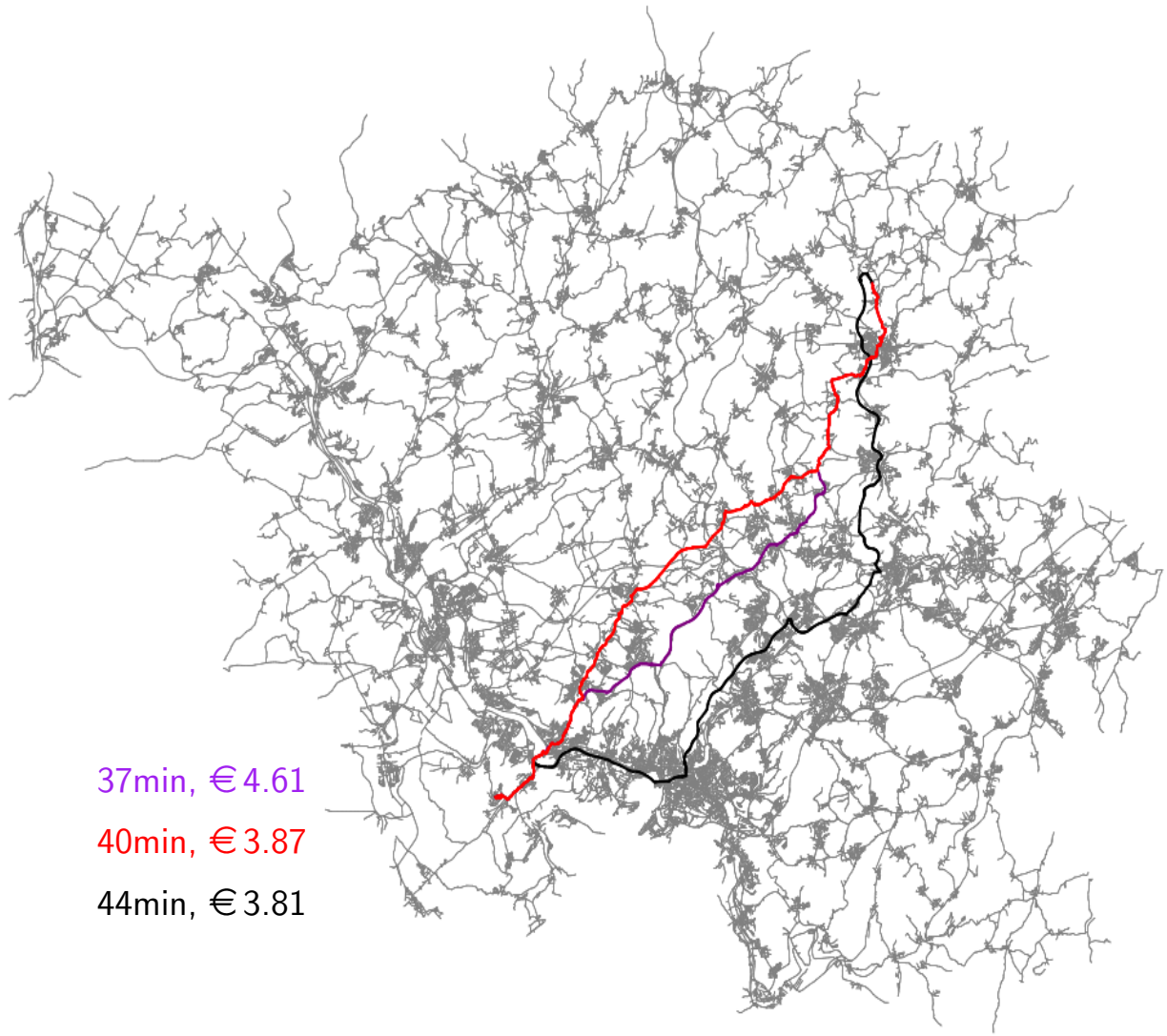


Sabine Storandt, JMU Würzburg

Motivation

Viele Kriterien definieren die 'optimale' Route von A nach B.

- Distanz
- Reisezeit
- Benzinverbrauch
- Staugefahr
- Szenerie
- Energieverbrauch
- Mautkosten
- ...



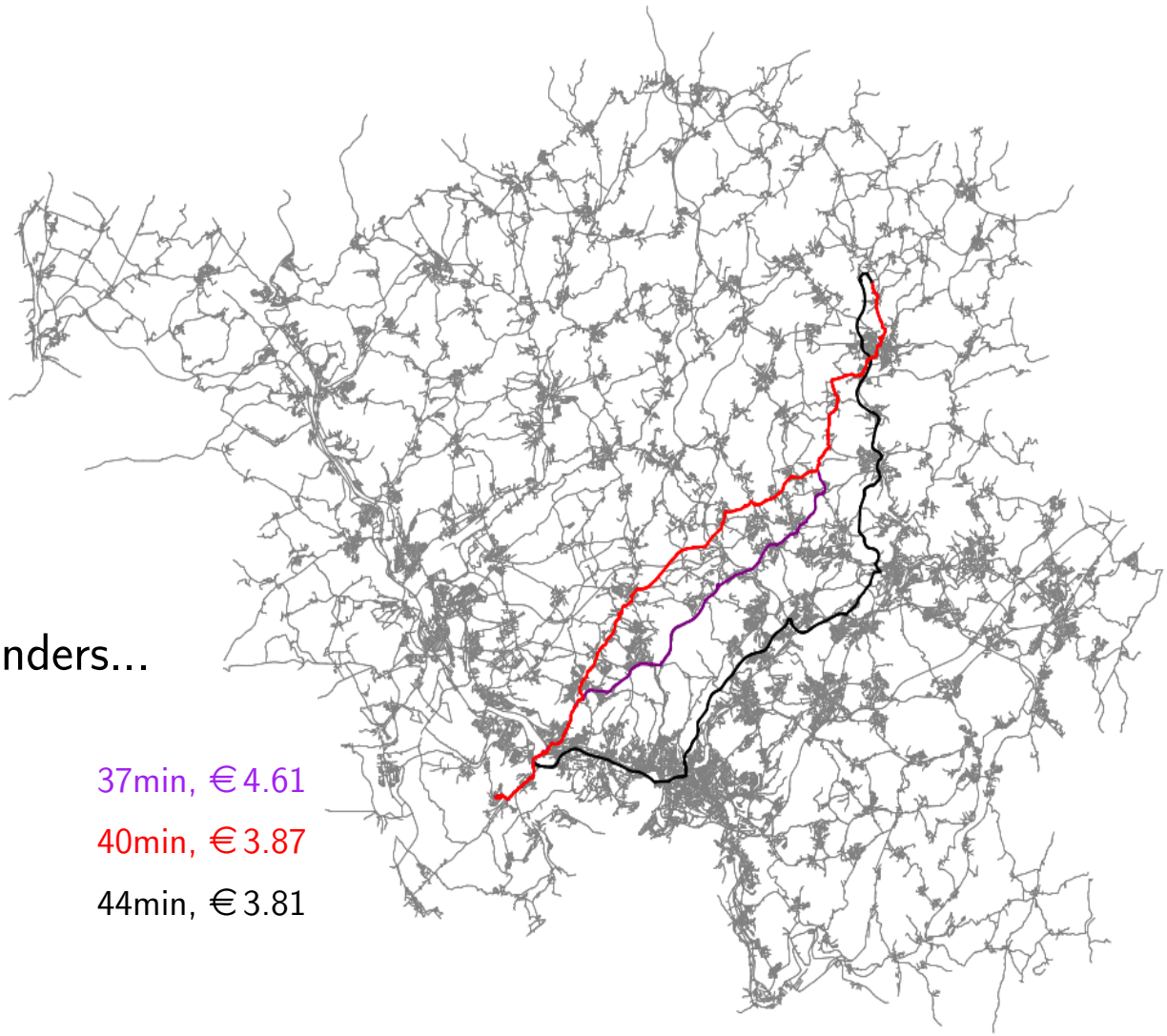
Motivation

Viele Kriterien definieren die 'optimale' Route von A nach B.

- Distanz
- Reisezeit
- Benzinverbrauch
- Staugefahr
- Szenerie
- Energieverbrauch
- Mautkosten
- ...

Gewichtung für jeden Fahrer anders...

Teilweise fahrzeugabhängig.

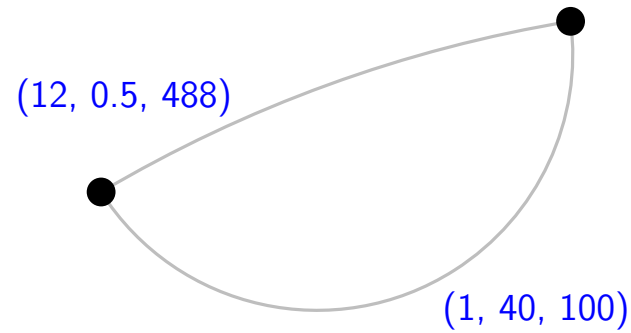


Problemdefinition

Problemdefinition

Gegeben $G(V, E)$ Straßennetzwerk

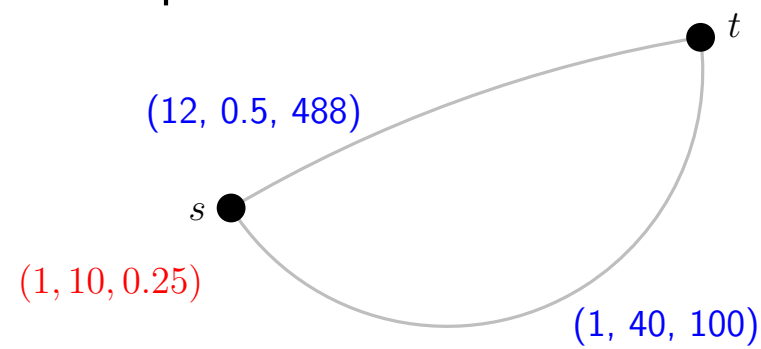
$c : E \to \mathbb{R}^d$ d-dimensionaler Kostenvektor pro Kante



Problemdefinition

Gegeben $G(V, E)$ Straßennetzwerk

$c : E \to \mathbb{R}^d$ d-dimensionaler Kostenvektor pro Kante



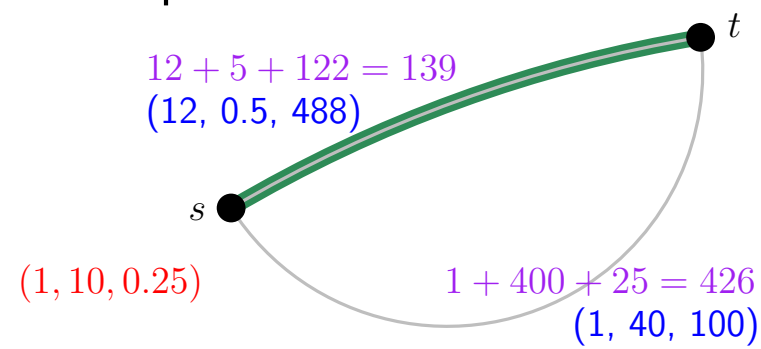
Anfrage $s, t \in V$

$\alpha \in \mathbb{R}^d$ Gewichtsvektor

Problemdefinition

Gegeben $G(V, E)$ Straßennetzwerk

$c : E \to \mathbb{R}^d$ d-dimensionaler Kostenvektor pro Kante



Anfrage $s, t \in V$

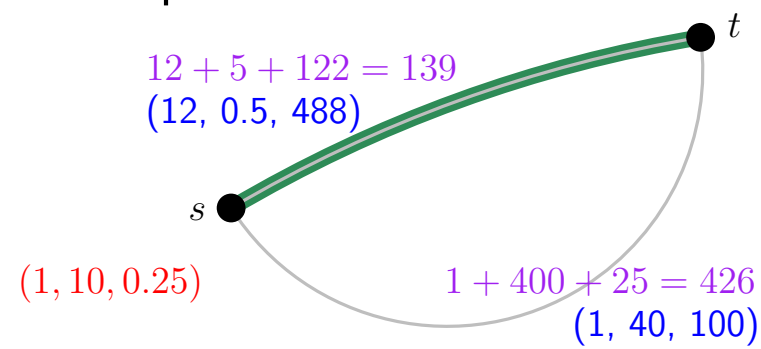
$\alpha \in \mathbb{R}^d$ Gewichtsvektor

Ziel: Finde Pfad p von s nach t mit $\min_{e \in p} \alpha^T c$.

Problemdefinition

Gegeben $G(V, E)$ Straßennetzwerk

$c : E \to \mathbb{R}^d$ d-dimensionaler Kostenvektor pro Kante



Anfrage $s, t \in V$

$\alpha \in \mathbb{R}^d$ Gewichtsvektor

Ziel: Finde Pfad p von s nach t mit $\min_{e \in p} \alpha^T c$.

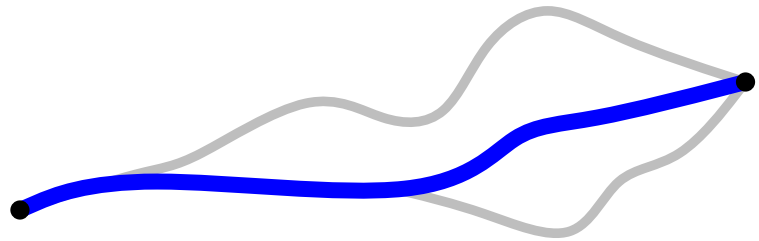
Problem 1: Wie findet man personalisierte optimale Routen effizient?

Problem 2: Kann man α aus Nutzerdaten extrahieren?

Bisherige Ansätze

Bisherige Ansätze

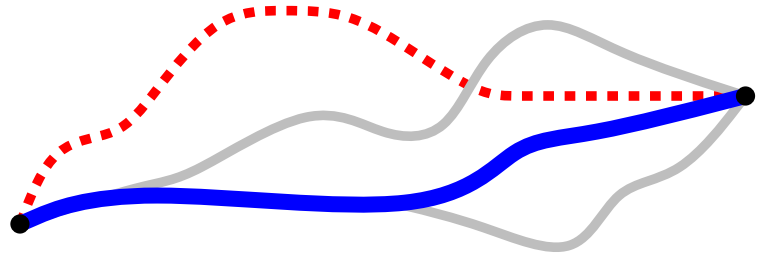
Alternativrouten



Wähle unter den Alternativen die beste gemäß α.

Bisherige Ansätze

Alternativrouten

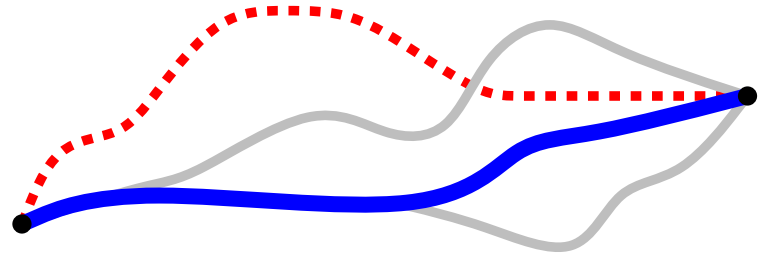


Wähle unter den Alternativen die beste gemäß α.

Optimale α-Route evtl. deutlich billiger...

Bisherige Ansätze

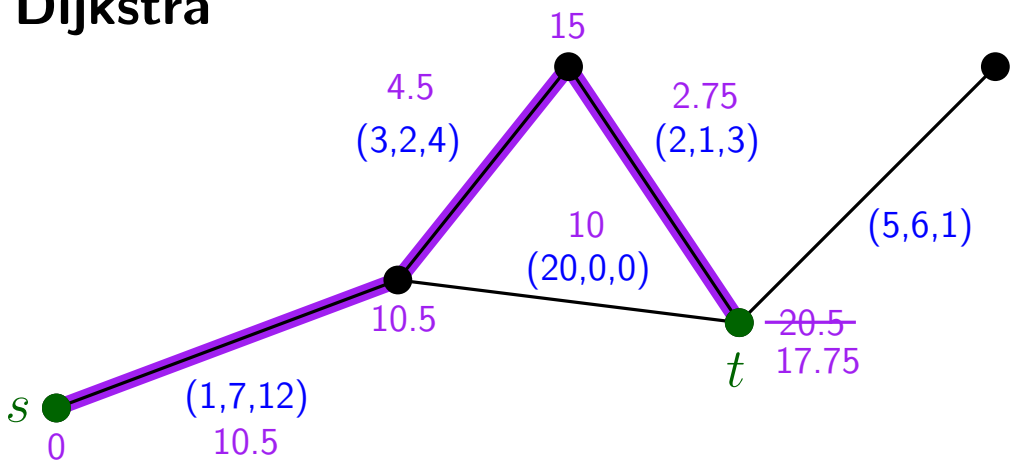
Alternativrouten



Wähle unter den Alternativen die beste gemäß α.

Optimale α-Route evtl. deutlich billiger...

Dijkstra



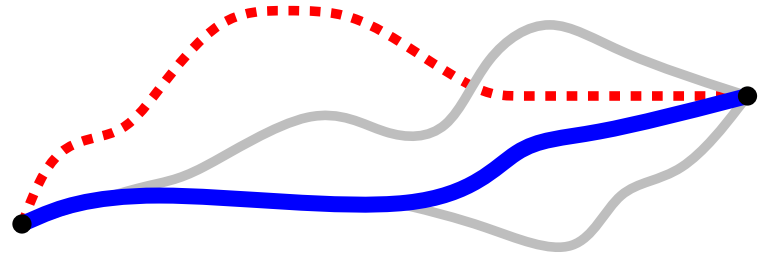
$\alpha = (0.5, 1, 0.25)$

Berechne Kantenkosten bei Relaxierung.

$\mathcal{O}(n \log n + dm)$

Bisherige Ansätze

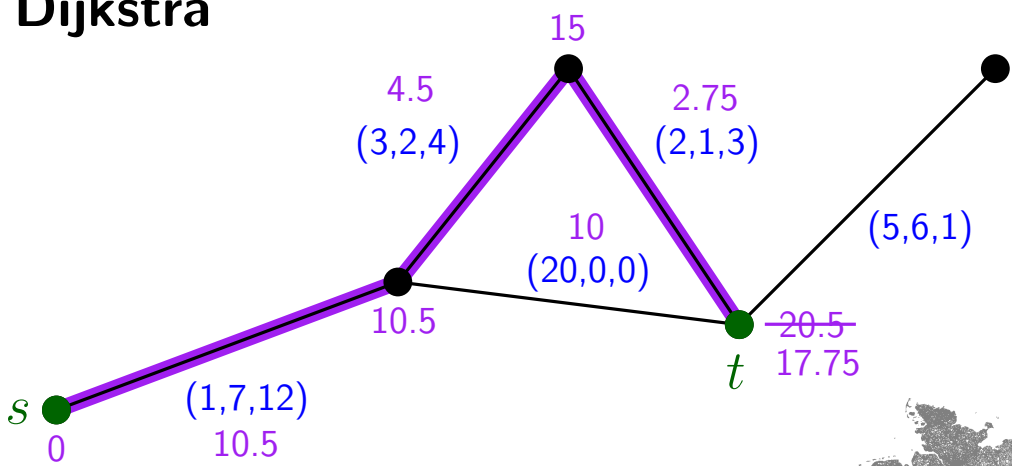
Alternativrouten



Wähle unter den Alternativen die beste gemäß α.

Optimale α-Route evtl. deutlich billiger...

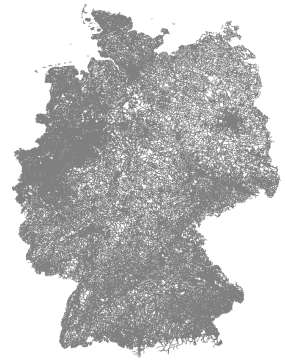
Dijkstra



$\alpha = (0.5, 1, 0.25)$

Berechne Kantenkosten bei Relaxierung.

$\mathcal{O}(n \log n + dm)$



> 7 Sekunden

Bisherige Ansätze

Multikriterielle Routenplanung/CSP

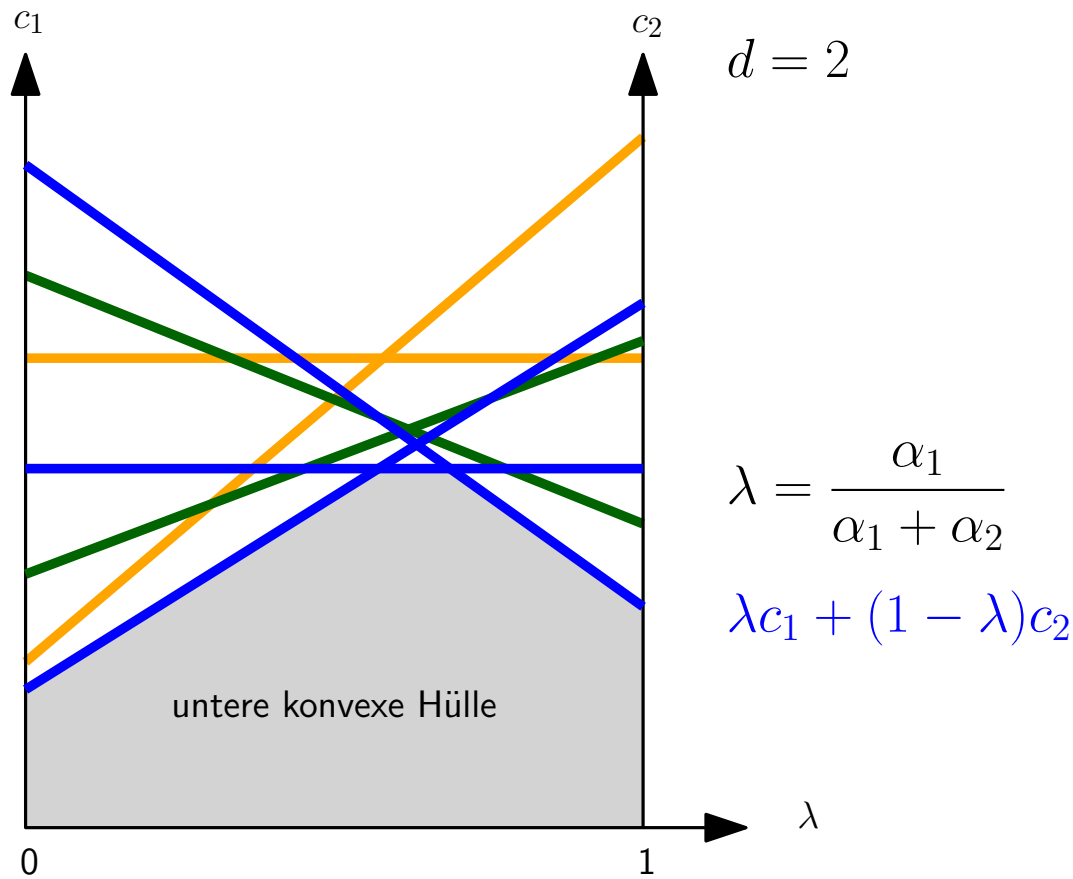
Methoden für personalisierte Routenplanung einsetzbar.

NP-hart, große Pareto-Mengen, auch in der Praxis, Overkill.

dominiert

Pareto-optimal

α-optimal



Bisherige Ansätze

Multikriterielle Routenplanung/CSP

Methoden für personalisierte Routenplanung einsetzbar.

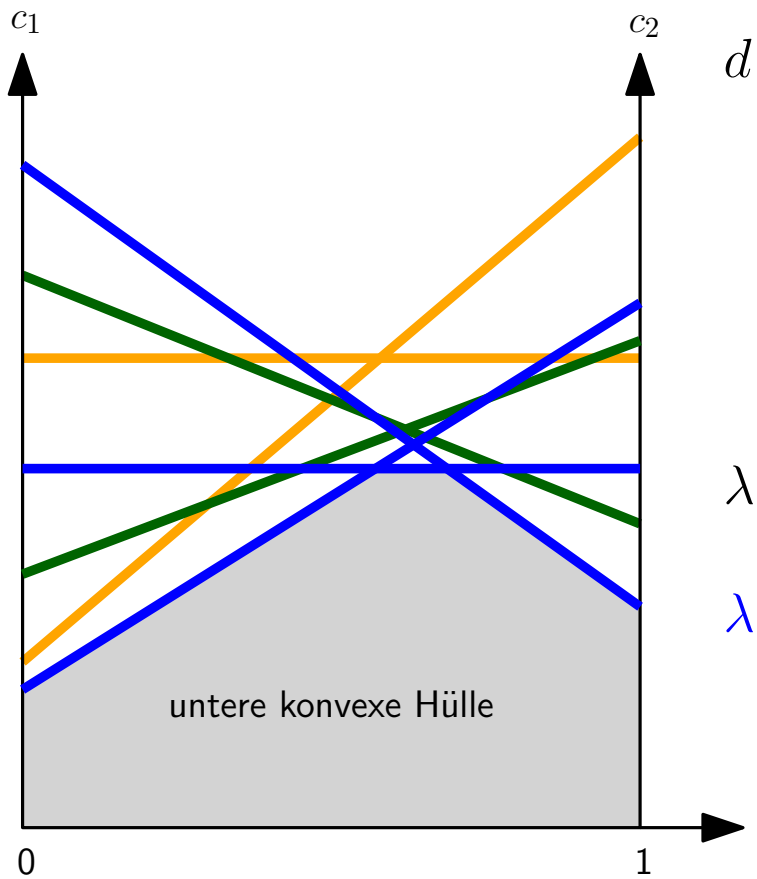
NP-hart, große Pareto-Mengen, auch in der Praxis, Overkill.

dominiert

Pareto-optimal

α-optimal

Pareto-optimale Pfade
Obermenge der
α-optimalen Pfade.

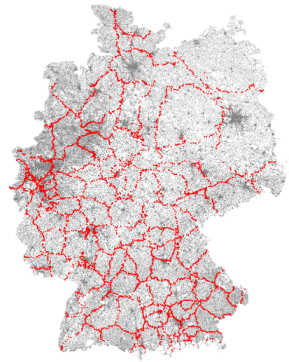


$d = 2$

$$\lambda = \frac{\alpha_1}{\alpha_1 + \alpha_2}$$

$$\lambda c_1 + (1 - \lambda) c_2$$

Bisherige Ansätze

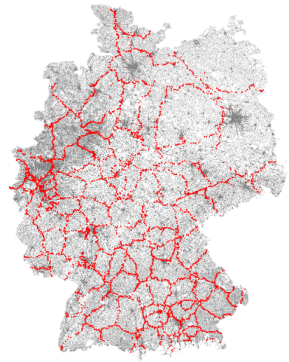


Customization (z.B. CRP, CCH)

3 Phasen: **Zeit**

1. Konstruiere metrik-unabhängigen Overlay Graph. (preprocessing) min
2. Überziehe Overlay mit Metrik. (customization) sec
3. Beantworte Queries im Overlay Graph. (queries) ms

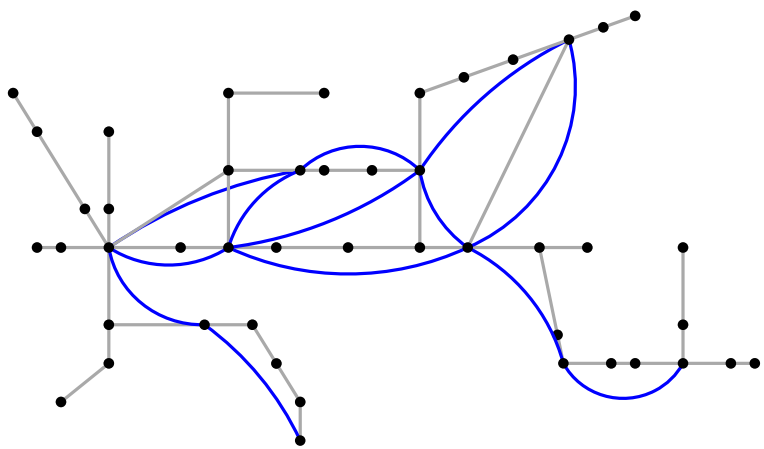
Bisherige Ansätze

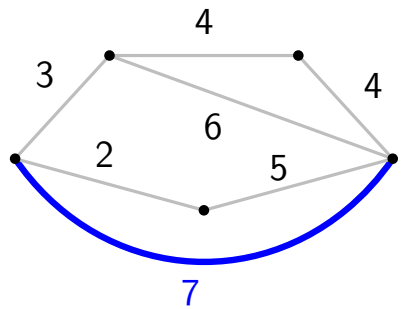


Customization (z.B. CRP, CCH)

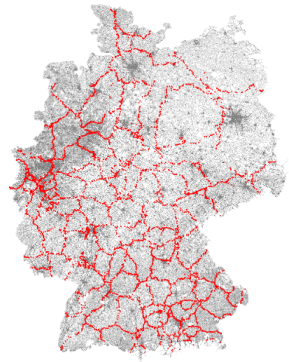
3 Phasen: **Zeit**

1. Konstruiere metrik-unabhängigen Overlay Graph. (preprocessing) min
2. Überziehe Overlay mit Metrik. (customization) sec
3. Beantworte Queries im Overlay Graph. (queries) ms





Bisherige Ansätze



Customization (z.B. CRP, CCH)

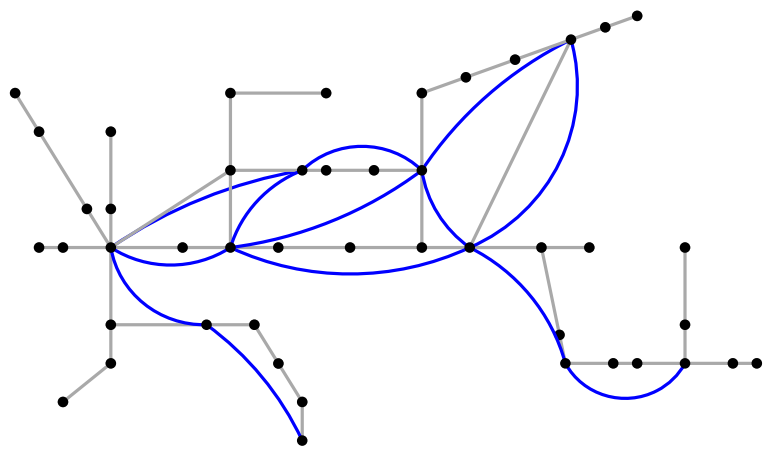
3 Phasen: **Zeit**

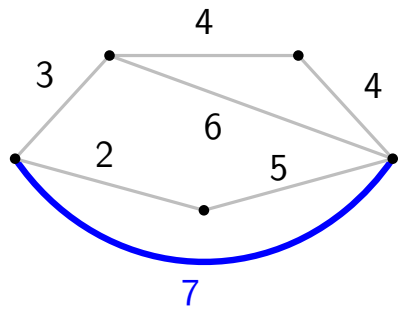
1. Konstruiere metrik-unabhängigen Overlay Graph. (preprocessing) min
2. Überziehe Overlay mit Metrik. (customization) sec
3. Beantworte Queries im Overlay Graph. (queries) ms

pro Anfrage/Query

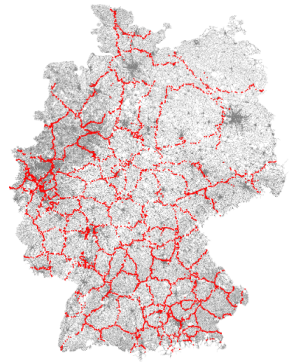
Kann für personalisierte Routenplanung instrumentalisiert werden.

'Personalisiert' den Overlay Graph.





Bisherige Ansätze



Customization (z.B. CRP, CCH)

3 Phasen: **Zeit**

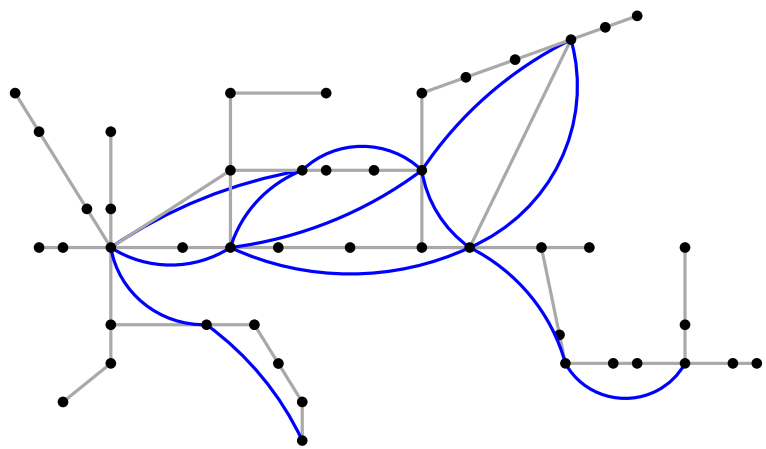
1. Konstruiere metrik-unabhängigen Overlay Graph. (preprocessing) min
2. Überziehe Overlay mit Metrik. (customization) sec
3. Beantworte Queries im Overlay Graph. (queries) ms

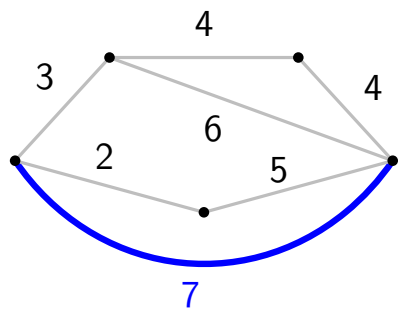
pro Anfrage/Query

Kann für personalisierte Routenplanung instrumentalisiert werden.

'Personalisiert' den Overlay Graph.

Anfragezeiten unparallelisiert kaum schneller als mit Dijkstra (superlinear).



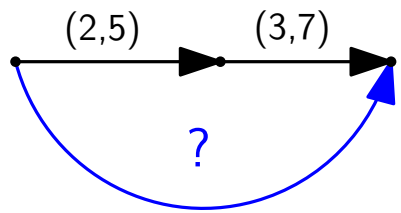


Effiziente Personalisierung

Effiziente Personalisierung

Ansatz 1 Personalisierte CH-Konstruktion.

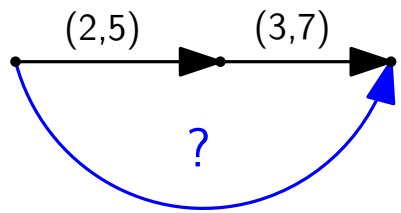
Problem: Welche Shortcuts müssen gesetzt werden?



Effiziente Personalisierung

Ansatz 1 Personalisierte CH-Konstruktion.

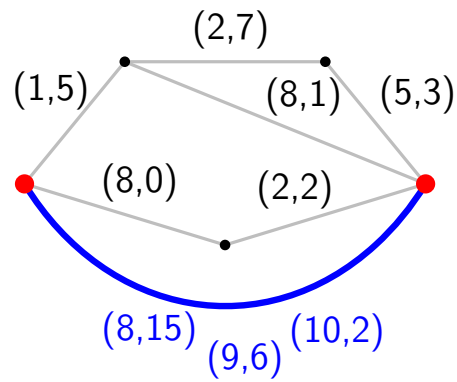
Problem: Welche Shortcuts müssen gesetzt werden?



Ansatz 2 Von Customizable zu Personalizable

1. Konstruiere metrik-unabhängigen Overlay Graph.
2. Überziehe Overlay Graph mit Kostenvektoren.
3. Beantworte personalisierte Anfragen.

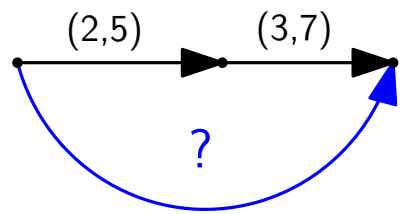
Problem: Wie prunt man Kostenvektormengen?



Effiziente Personalisierung

Ansatz 1 Personalisierte CH-Konstruktion.

Problem: Welche Shortcuts müssen gesetzt werden?



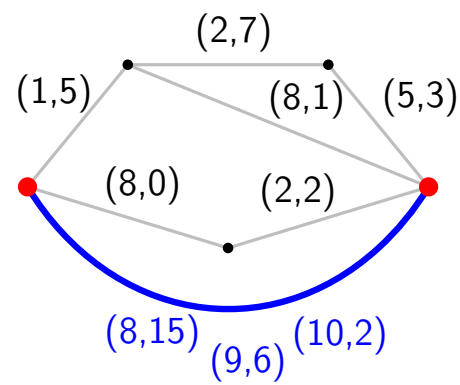
California 11M nodes, 23M edges

Metriken	Reisezeit + Benzinkosten
Dijkstra	2097ms
CH-Dijkstra	2ms
speed-up	>1000

Ansatz 2 Von Customizable zu Personalizable

1. Konstruiere metrik-unabhängigen Overlay Graph.
2. Überziehe Overlay Graph mit Kostenvektoren.
3. Beantworte personalisierte Anfragen.

Problem: Wie prunt man Kostenvektormengen?



Germany 22M nodes, 44M edges

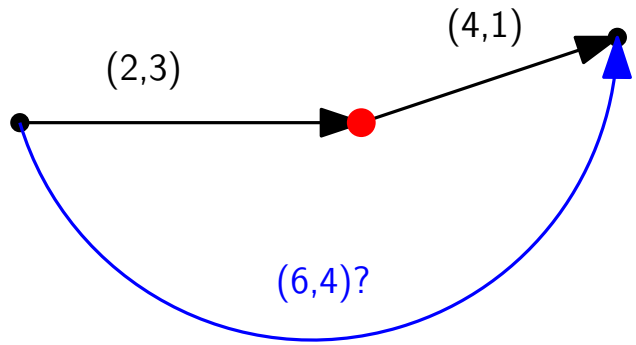
d	# polls	# vectors	time	speed-up
1	$1.5 \cdot 10^5$	$6.9 \cdot 10^5$	71 ms	111
2	$1.7 \cdot 10^5$	$7.3 \cdot 10^5$	78 ms	107
5	$1.7 \cdot 10^5$	$7.7 \cdot 10^5$	82 ms	104
10	$1.6 \cdot 10^5$	$1.3 \cdot 10^6$	135 ms	67
32	$1.6 \cdot 10^5$	$2.5 \cdot 10^6$	287 ms	35
64	$1.8 \cdot 10^5$	$4.9 \cdot 10^6$	612 ms	20

Personalisierte CH

Alle α-optimalen Pfade müssen erhalten werden.

Personalisierte CH

Alle α-optimalen Pfade müssen erhalten werden.

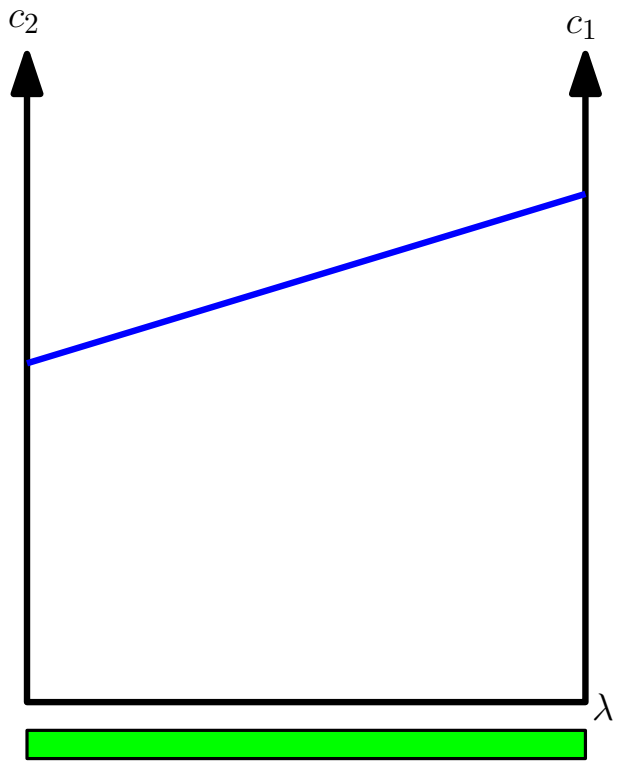


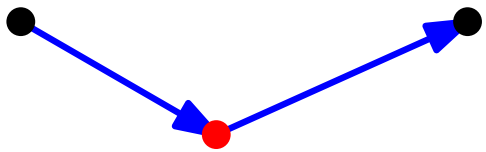
Personalisierte CH

Alle α-optimalen Pfade müssen erhalten werden.

α-optimal = Teil der unteren konvexen Hülle (UKH)

UKH kann exponentiell groß sein.





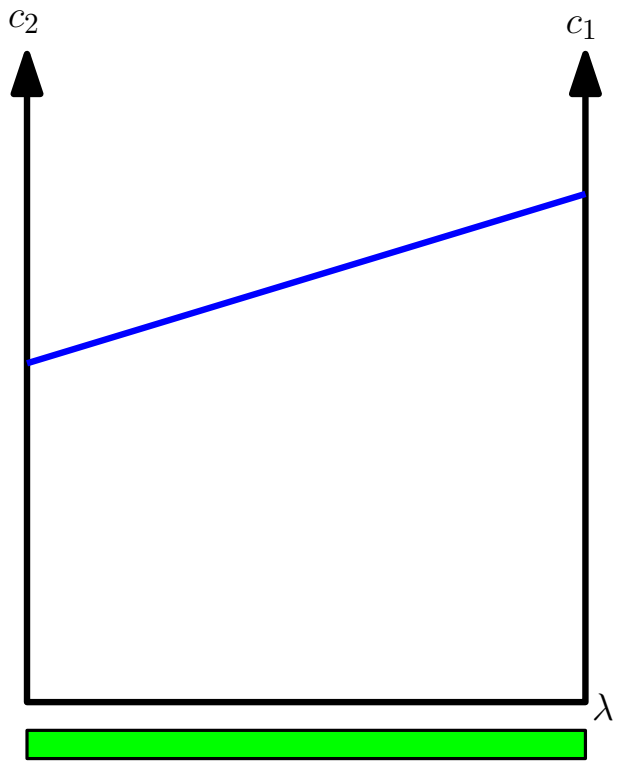
Personalisierte CH

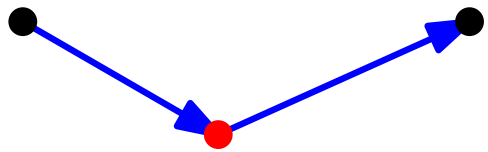
Alle α-optimalen Pfade müssen erhalten werden.

α-optimal = Teil der unteren konvexen Hülle (UKH)

UKH kann exponentiell groß sein.

Check ob Pfad optimal für bestimmtes α = Dijkstra Lauf in G_α





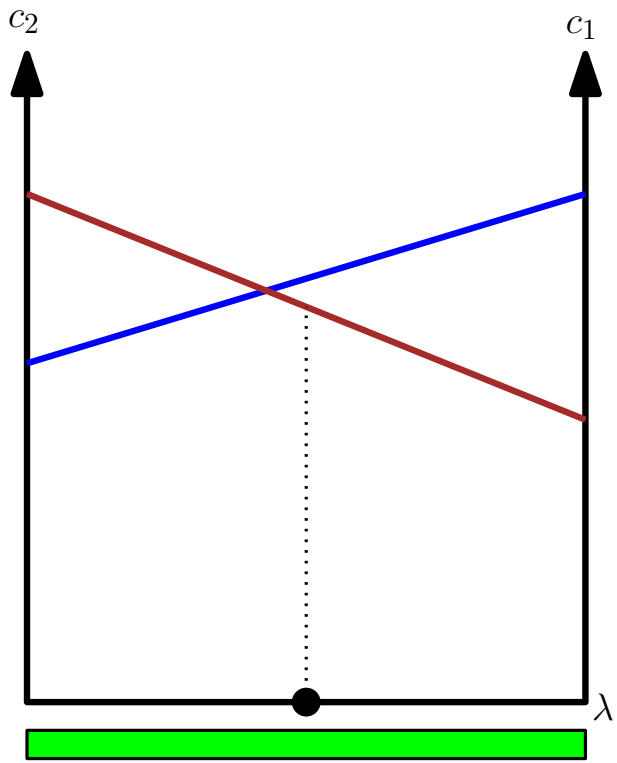
Personalisierte CH

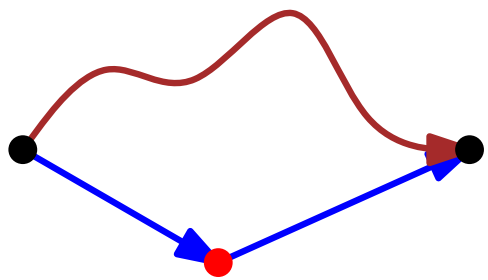
Alle α-optimalen Pfade müssen erhalten werden.

α-optimal = Teil der unteren konvexen Hülle (UKH)

UKH kann exponentiell groß sein.

Check ob Pfad optimal für bestimmtes α = Dijkstra Lauf in G_α





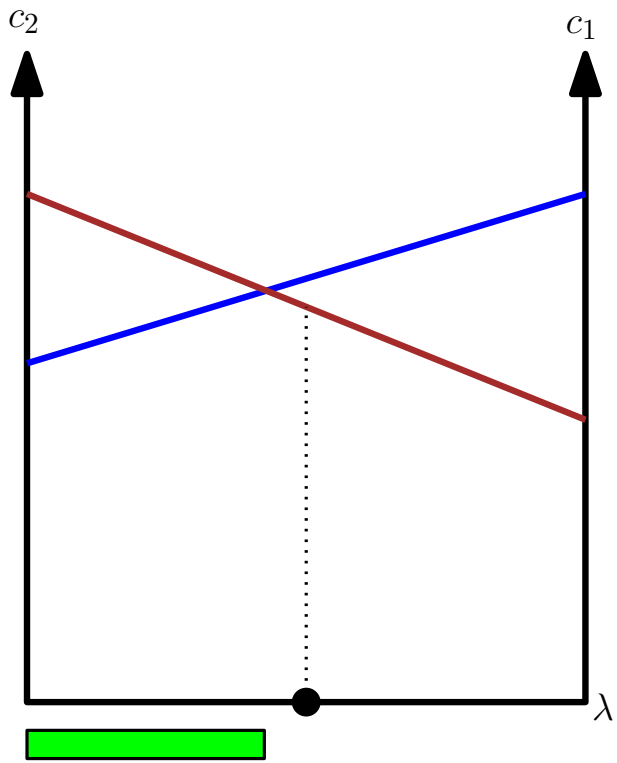
Personalisierte CH

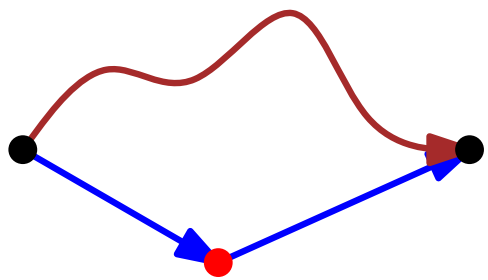
Alle α-optimalen Pfade müssen erhalten werden.

α-optimal = Teil der unteren konvexen Hülle (UKH)

UKH kann exponentiell groß sein.

Check ob Pfad optimal für bestimmtes α = Dijkstra Lauf in G_α





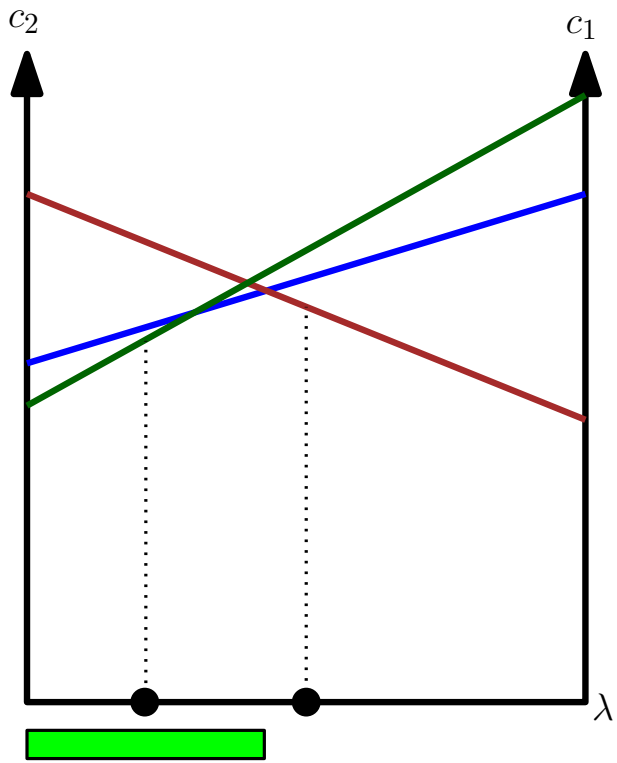
Personalisierte CH

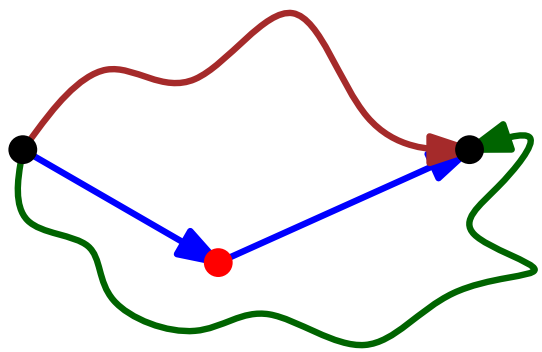
Alle α-optimalen Pfade müssen erhalten werden.

α-optimal = Teil der unteren konvexen Hülle (UKH)

UKH kann exponentiell groß sein.

Check ob Pfad optimal für bestimmtes α = Dijkstra Lauf in G_α





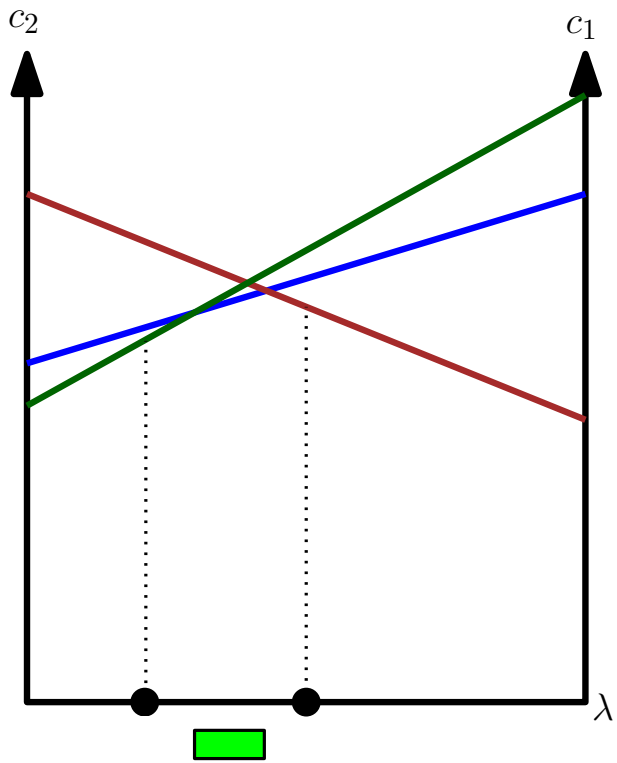
Personalisierte CH

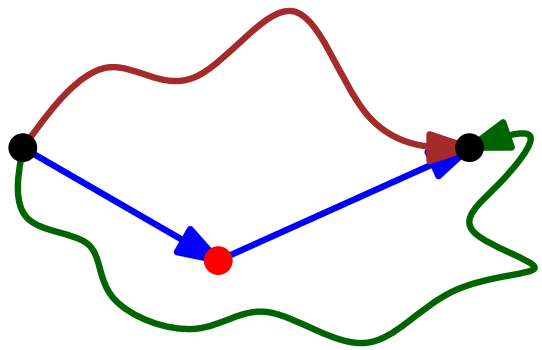
Alle α-optimalen Pfade müssen erhalten werden.

α-optimal = Teil der unteren konvexen Hülle (UKH)

UKH kann exponentiell groß sein.

Check ob Pfad optimal für bestimmtes α = Dijkstra Lauf in G_α





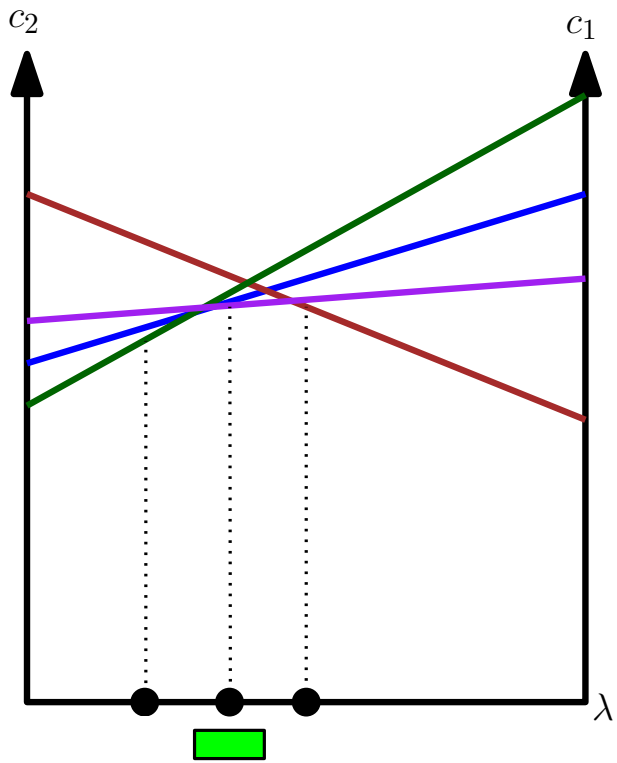
Personalisierte CH

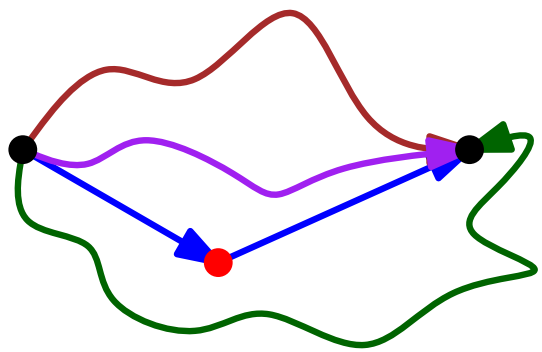
Alle α-optimalen Pfade müssen erhalten werden.

α-optimal = Teil der unteren konvexen Hülle (UKH)

UKH kann exponentiell groß sein.

Check ob Pfad optimal für bestimmtes α = Dijkstra Lauf in G_α





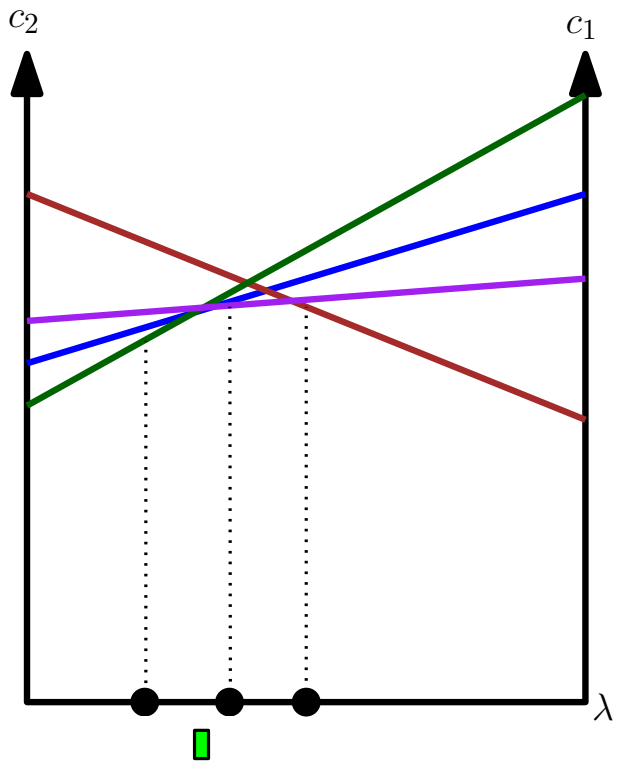
Personalisierte CH

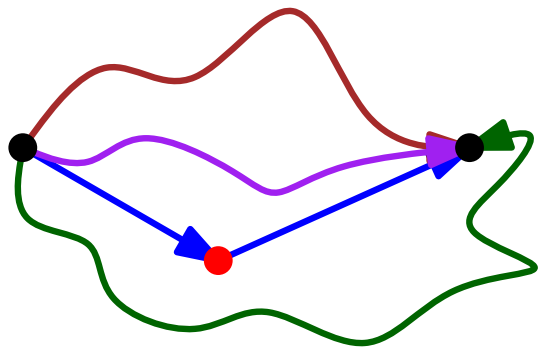
Alle α-optimalen Pfade müssen erhalten werden.

α-optimal = Teil der unteren konvexen Hülle (UKH)

UKH kann exponentiell groß sein.

Check ob Pfad optimal für bestimmtes α = Dijkstra Lauf in G_α





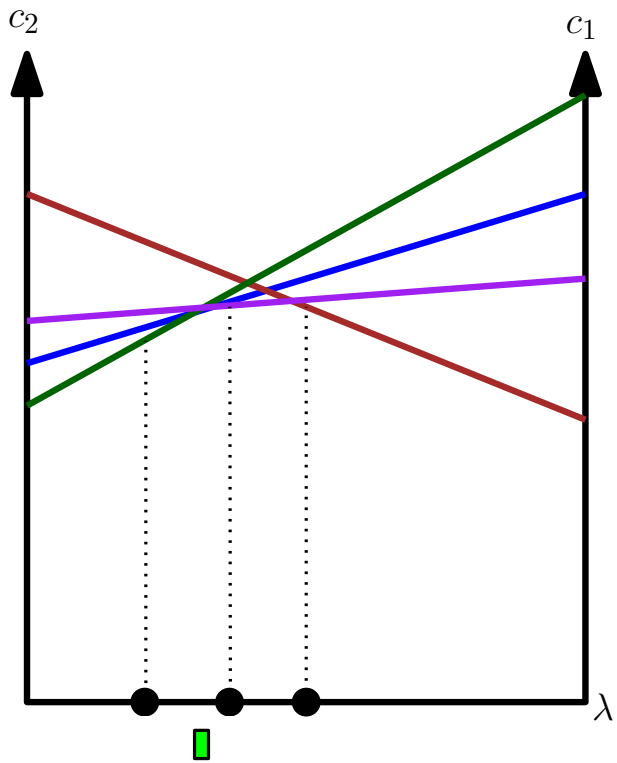
Personalisierte CH

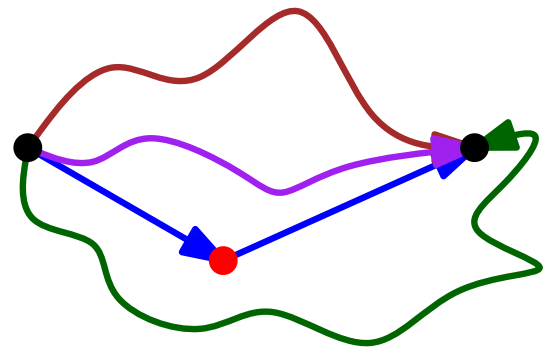
Alle α-optimalen Pfade müssen erhalten werden.

α-optimal = Teil der unteren konvexen Hülle (UKH)

UKH kann exponentiell groß sein.

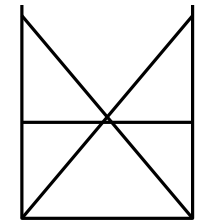
Check ob Pfad optimal für bestimmtes α = Dijkstra Lauf in G_α





$c : E \to \mathbb{N}$

$M = \max_{e \in E} c(e)$



Zwei Eckpunkte der UKH haben Mindestabstand $\Omega(1/(nM)^2)$.

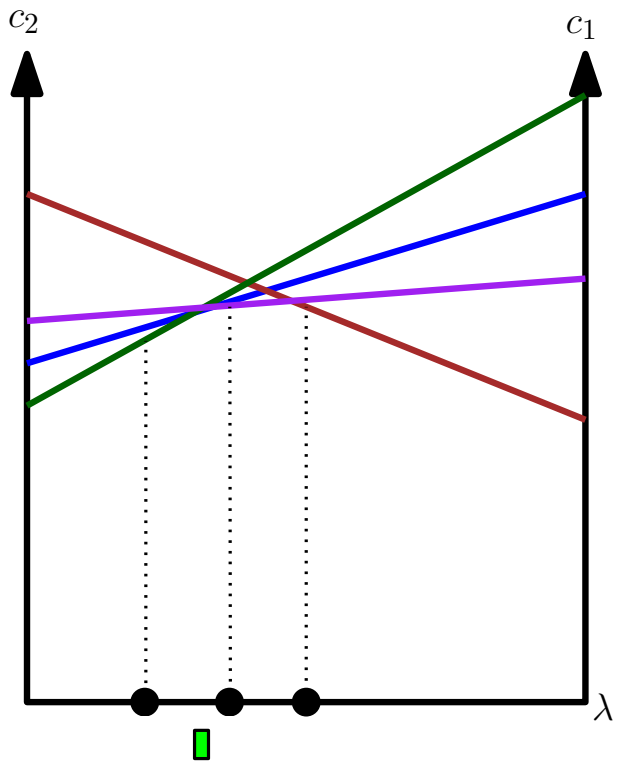
Personalisierte CH

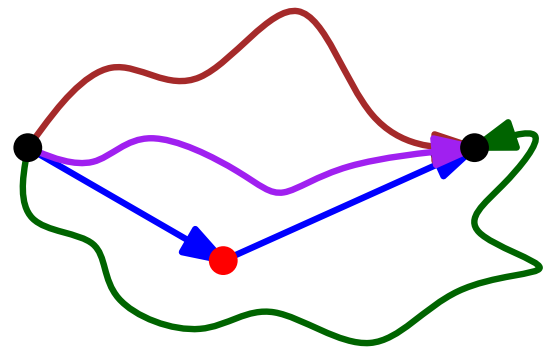
Alle α-optimalen Pfade müssen erhalten werden.

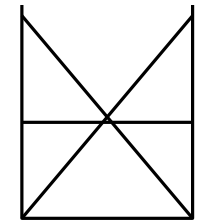
α-optimal = Teil der unteren konvexen Hülle (UKH)

UKH kann exponentiell groß sein.

Check ob Pfad optimal für bestimmtes α = Dijkstra Lauf in G_α







$c : E \to \mathbb{N}$

$M = \max_{e \in E} c(e)$

Zwei Eckpunkte der UKH haben Mindestabstand $\Omega(1/(nM)^2)$.

Intervall wird in jeder Runde halbiert.

Laufzeit $\mathcal{O}(\log(nM)(n \log n + m))$

Personalisierte CH

Alle α-optimalen Pfade müssen erhalten werden.

Generalisierbar für $d > 2$.

Personalisierte CH

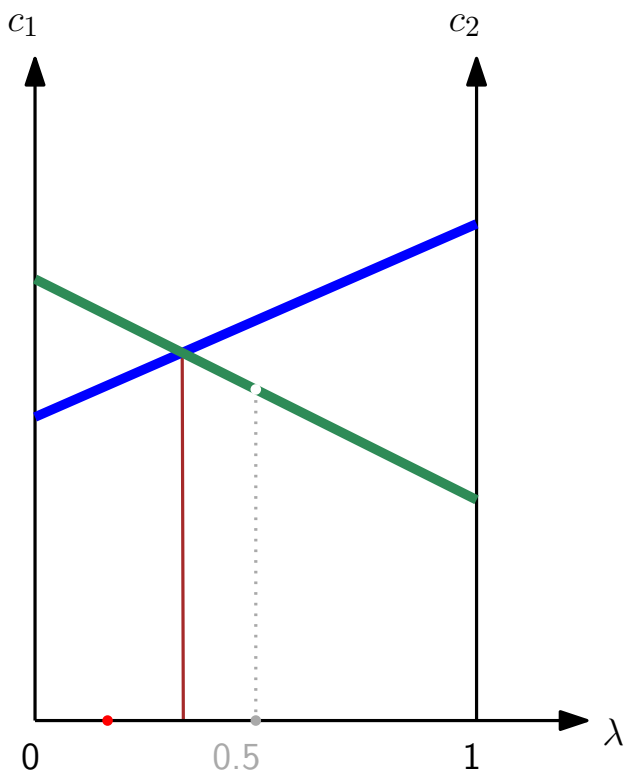
Alle α-optimalen Pfade müssen erhalten werden.

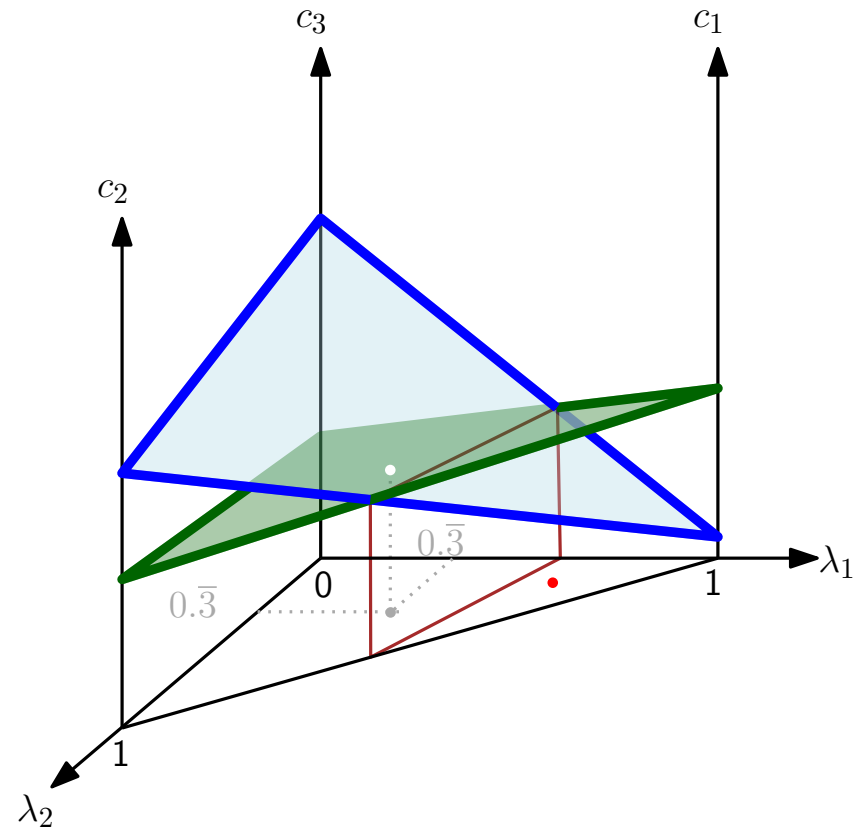
Generalisierbar für $d > 2$.

Kosten $c_1, \cdots, c_d$

UKH $\sum_{i=1}^{d-1} \lambda_i c_i + (1 - \sum_{i=1}^{d-1} \lambda_i) c_d \quad \lambda_i \in [0,1], \sum_{i=1}^{d} \lambda_i = 1$

Linien $\rightarrow$ Hyperebenen





Personalisierte CH

Alle α-optimalen Pfade müssen erhalten werden.

Generalisierbar für $d > 2$.

Erzeugt minimale CH für gegebene Kontraktionsreihenfolge.

Ermöglicht theoretische Laufzeitschranken.

Gute Kontraktionsreihenfolge?

Vorberechnung teuer, da mehrere Dijkstra Läufe notwendig sind für jede Shortcut-Entscheidung.

CAL, d=2 18 min

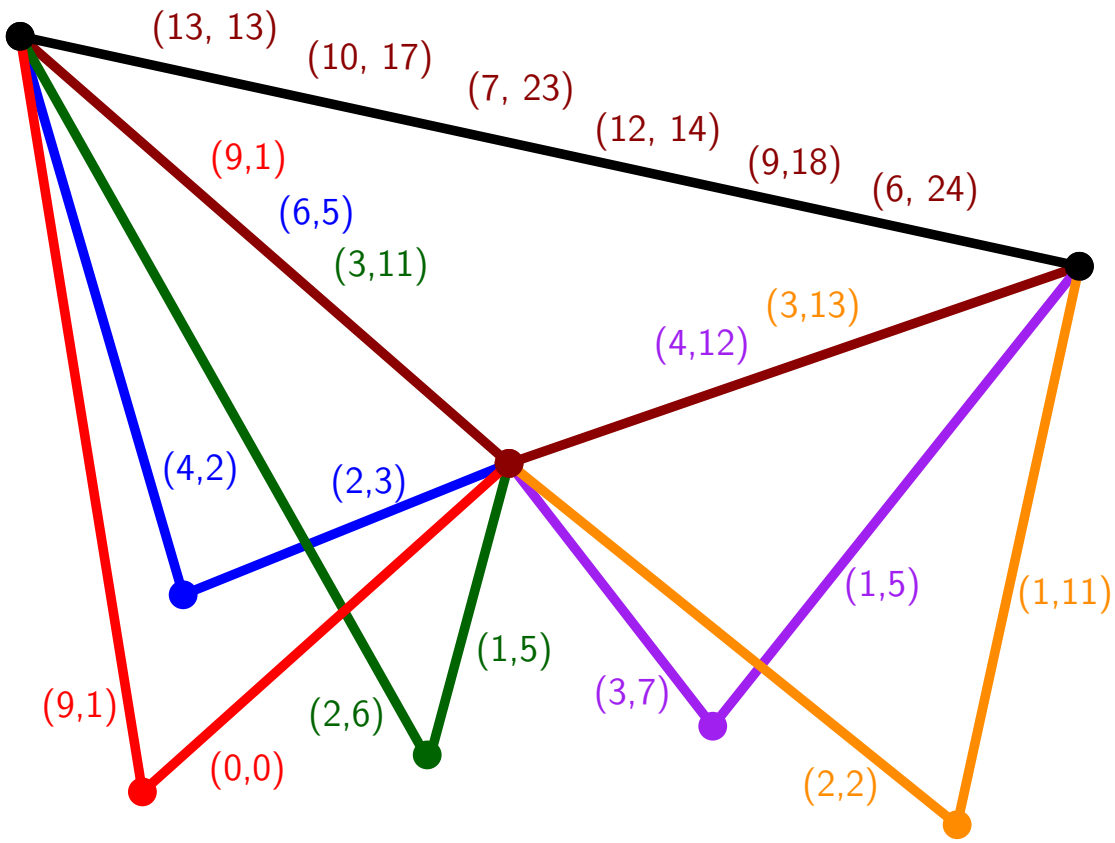
Speed-Up geringer für höheres d.

$$d = 2: \quad \approx 1000, d = 3: \quad \approx 200$$

Von Customizable zu Personalizable

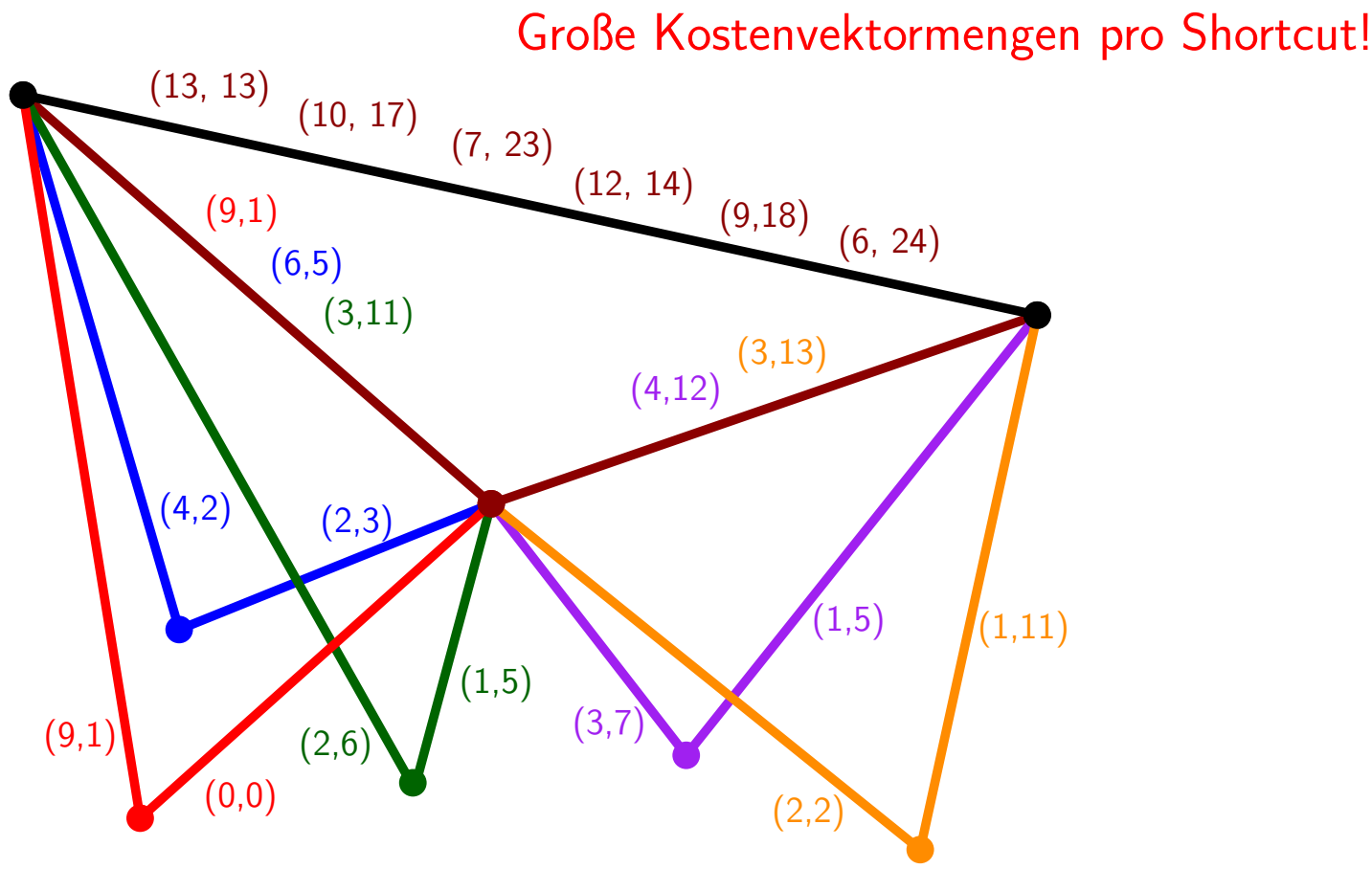
Von Customizable zu Personalizable

Idee konstruiere metrik-unabhängigen Overlay Graph, Customization mit Vektoren statt skalaren Kosten



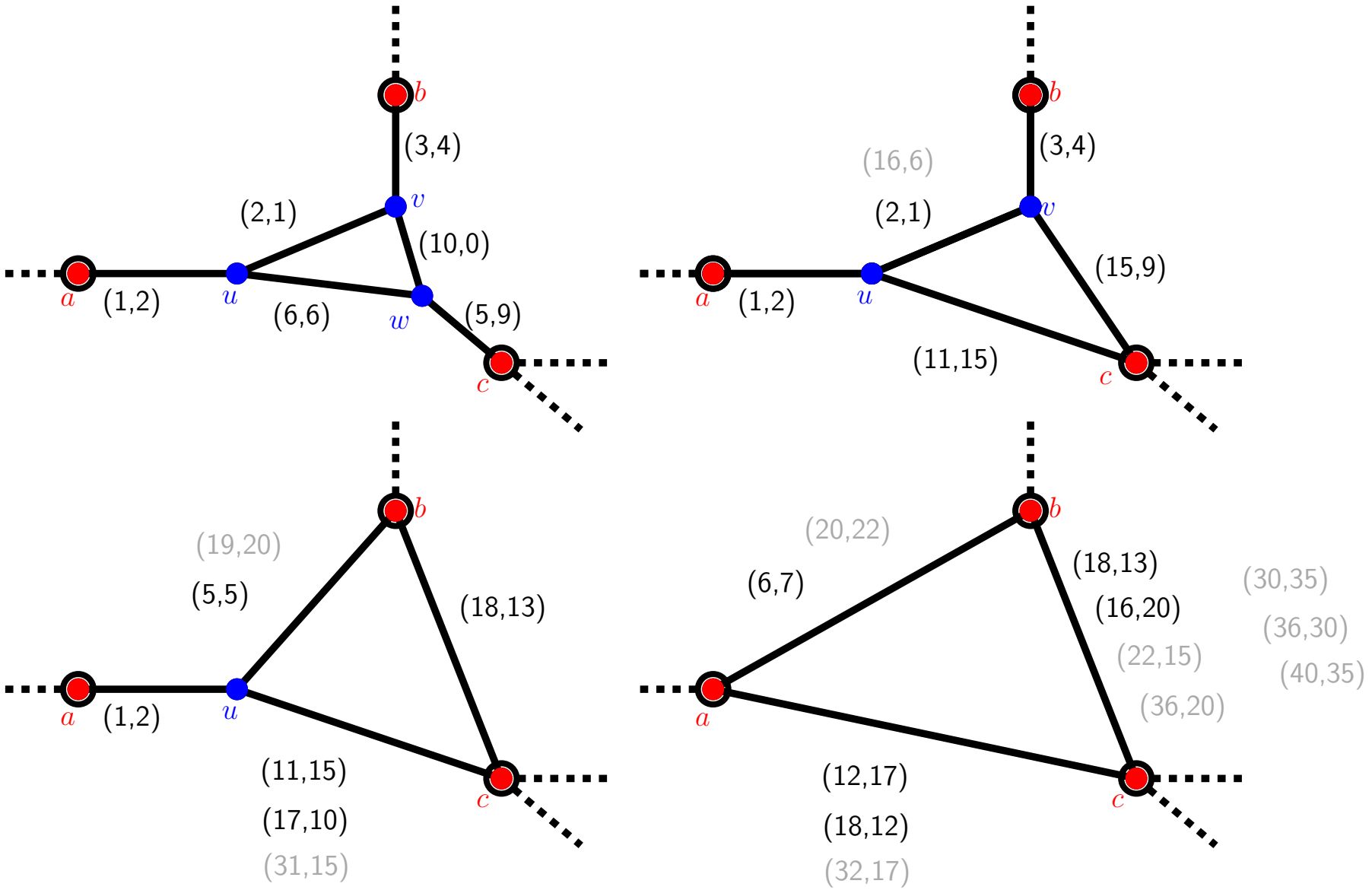
Von Customizable zu Personalizable

Idee konstruiere metrik-unabhängigen Overlay Graph, Customization mit Vektoren statt skalaren Kosten

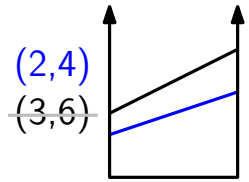


Von Customizable zu Personalizable

Viele Vektoren überflüssig für korrekte Queries...



Von Customizable zu Personalizable

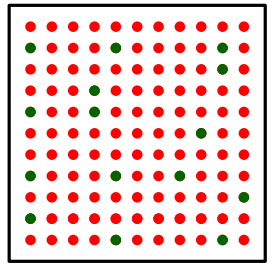


Lösche dominierte Vektoren

ein Vektor $v \in \mathbb{R}^d$ dominiert einen anderen Vektor $w \in \mathbb{R}^d$wenn $v_i \leq w_i$ $\forall i = 1, \ldots, d$ und $v_i < w_i$ für mindest ein $i \in \{1, \ldots, d\}$.

Naiver Ansatz

vergleiche alle Paare von Vektoren $\rightarrow \mathcal{O}(d|S|^2)$

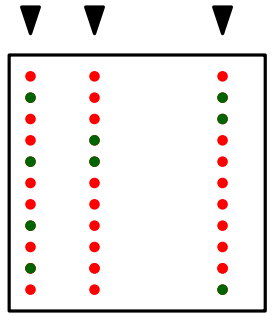


Beschleunigung

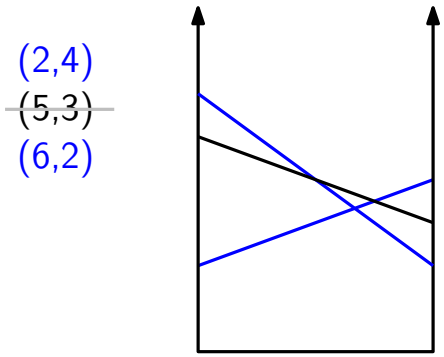
wähle d Kandidaten (z.B. den kostenminimalen pro Dimension)

überprüfe für alle anderen Vektoren ob diese dominiert werden

$\rightarrow$ runtime $\mathcal{O}(d^2|S|)$



Von Customizable zu Personalizable

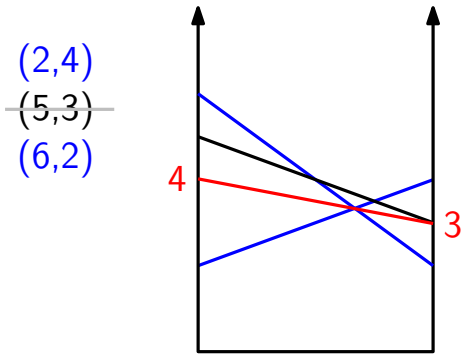


Löschen aufgespannter Vektoren

Lemma

Ein Vektor $v \in \mathbb{R}^d$ kann gelöscht werden wenn eine Konvexkombination von höchstens d anderen Vektoren v dominiert.

Von Customizable zu Personalizable



Löschen aufgespannter Vektoren

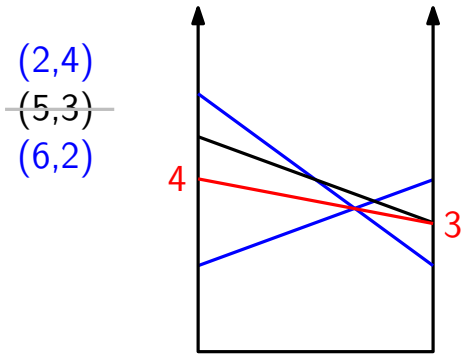
Lemma

Ein Vektor $v \in \mathbb{R}^d$ kann gelöscht werden wenn eine Konvexkombination von höchstens d anderen Vektoren v dominiert.

Annahme: $\exists \alpha : \alpha^T v = z$ minimal und eindeutig

$w_1, \cdots, w_d$ Vektoren die v' aufspannen mit $v' < v$

Von Customizable zu Personalizable



Löschen aufgespannter Vektoren

Lemma

Ein Vektor $v \in \mathbb{R}^d$ kann gelöscht werden wenn eine Konvexkombination von höchstens d anderen Vektoren v dominiert.

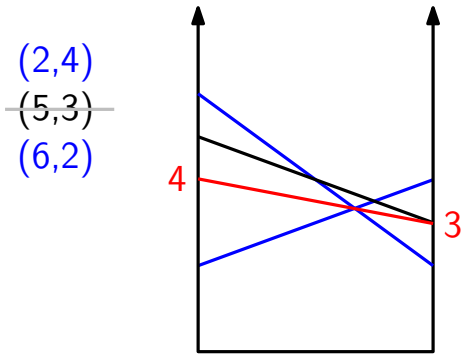
Annahme: $\exists \alpha : \alpha^T v = z$ minimal und eindeutig

$w_1, \cdots, w_d$ Vektoren die v' aufspannen mit $v' < v$

$\sum \gamma_i w_i = v' < v \quad$ mit $\sum \gamma_i = 1$ und $\gamma_i \geq 0$

$\alpha^T \sum \gamma_i w_i = \alpha^T v' < \alpha^T v = z$

Von Customizable zu Personalizable



Löschen aufgespannter Vektoren

Lemma

Ein Vektor $v \in \mathbb{R}^d$ kann gelöscht werden wenn eine Konvexkombination von höchstens d anderen Vektoren v dominiert.

Annahme: $\exists \alpha : \alpha^T v = z$ minimal und eindeutig

$w_1, \cdots, w_d$ Vektoren die v' aufspannen mit $v' < v$

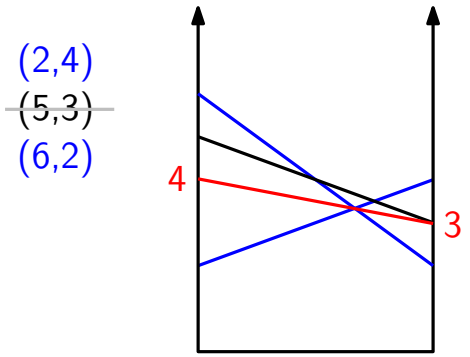
$\sum \gamma_i w_i = v' < v \quad$ mit $\sum \gamma_i = 1$ und $\gamma_i \geq 0$

$\alpha^T \sum \gamma_i w_i = \alpha^T v' < \alpha^T v = z$

$\Rightarrow \sum \gamma_i \alpha^T w_i < z$

$\alpha^T v$ minimal und eindeutig $\Rightarrow \alpha^T w_i > z \Rightarrow \sum \gamma_i \alpha^T w_i > \sum \gamma_i z = z$

Von Customizable zu Personalizable



Löschen aufgespannter Vektoren

Lemma

Ein Vektor $v \in \mathbb{R}^d$ kann gelöscht werden wenn eine Konvexkombination von höchstens d anderen Vektoren v dominiert.

Annahme: $\exists \alpha : \alpha^T v = z$ minimal und eindeutig

$w_1, \cdots, w_d$ Vektoren die v' aufspannen mit $v' < v$

$\sum \gamma_i w_i = v' < v \quad$ mit $\sum \gamma_i = 1$ und $\gamma_i \geq 0$

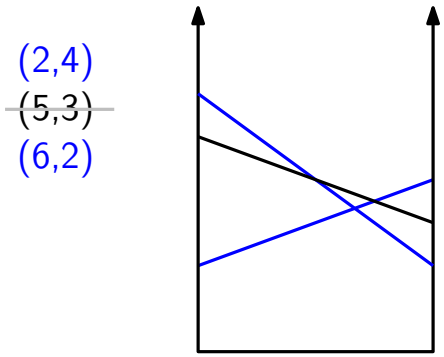
$\alpha^T \sum \gamma_i w_i = \alpha^T v' < \alpha^T v = z$

$\Rightarrow \sum \gamma_i \alpha^T w_i < z$

$\alpha^T v$ minimal und eindeutig $\Rightarrow \alpha^T w_i > z \Rightarrow \sum \gamma_i \alpha^T w_i > \sum \gamma_i z = z$

Widerspruch!

Von Customizable zu Personalizable



Löschen aufgespannter Vektoren

Lemma

Ein Vektor $v \in \mathbb{R}^d$ kann gelöscht werden wenn eine Konvexkombination von höchstens d anderen Vektoren v dominiert.

Pruning Orakel

Überprüfung auf Existenz einer solchen Konvexkombination mit folgendem Gleichungssystem:

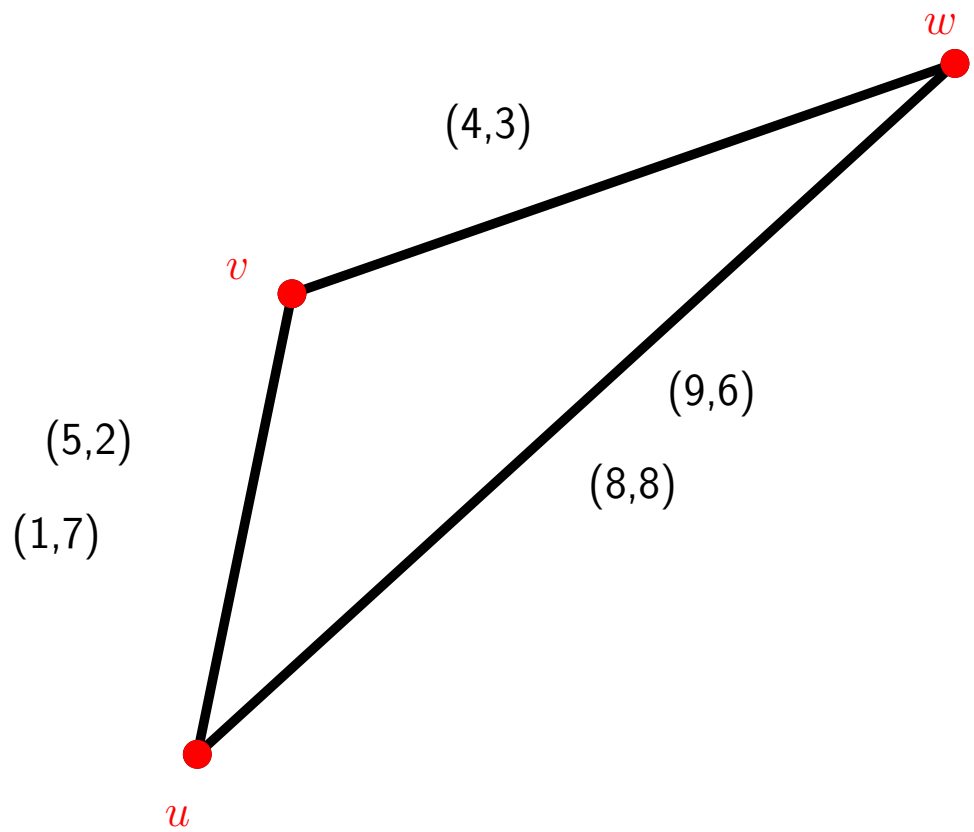
$$\begin{aligned}
\gamma_1 w_{11} + \gamma_2 w_{21} + \cdots + \gamma_d w_{d1} &\leq v_1 \\
\gamma_1 w_{12} + \gamma_2 w_{22} + \cdots + \gamma_d w_{d2} &\leq v_2 \\
&\cdots \\
\gamma_1 w_{1d} + \gamma_2 w_{2d} + \cdots + \gamma_d w_{dd} &\leq v_d \\
\gamma_1 + \gamma_2 + \cdots + \gamma_d &= 1 \\
\gamma_i &\geq 0
\end{aligned}$$

Theoretisch $b = \binom{|S|}{d}$ viele Basen möglich.

Wähle vielversprechende Kandidaten.

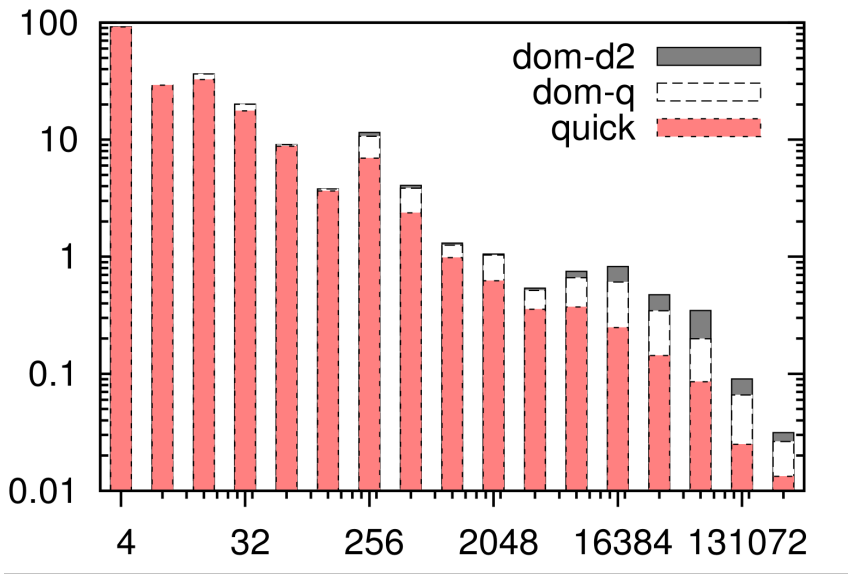
Von Customizable zu Personalizable

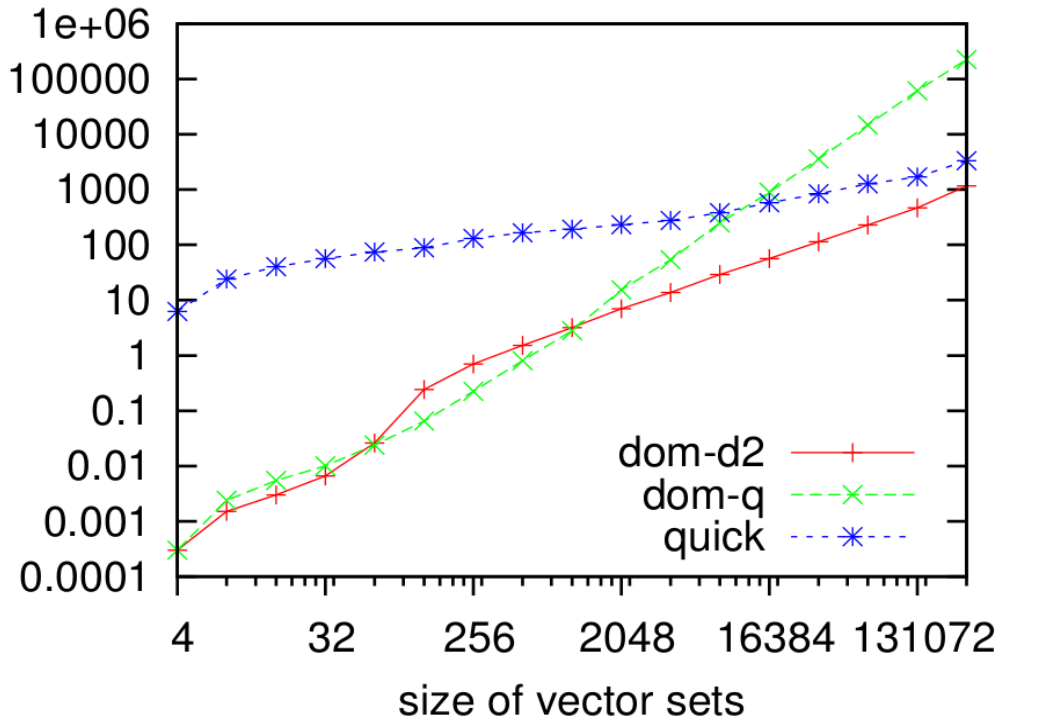
Dreiecks-Pruning



Von Customizable zu Personalizable

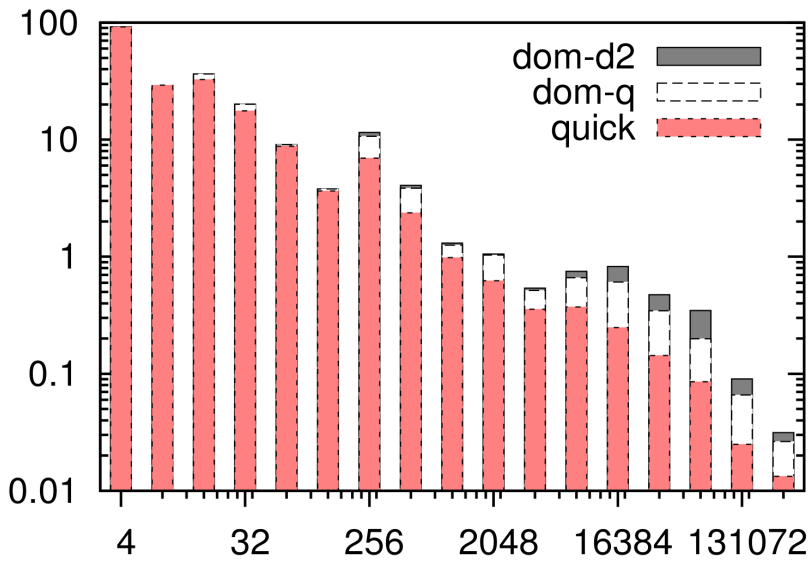
Experimentelle Pruning Ergebnisse

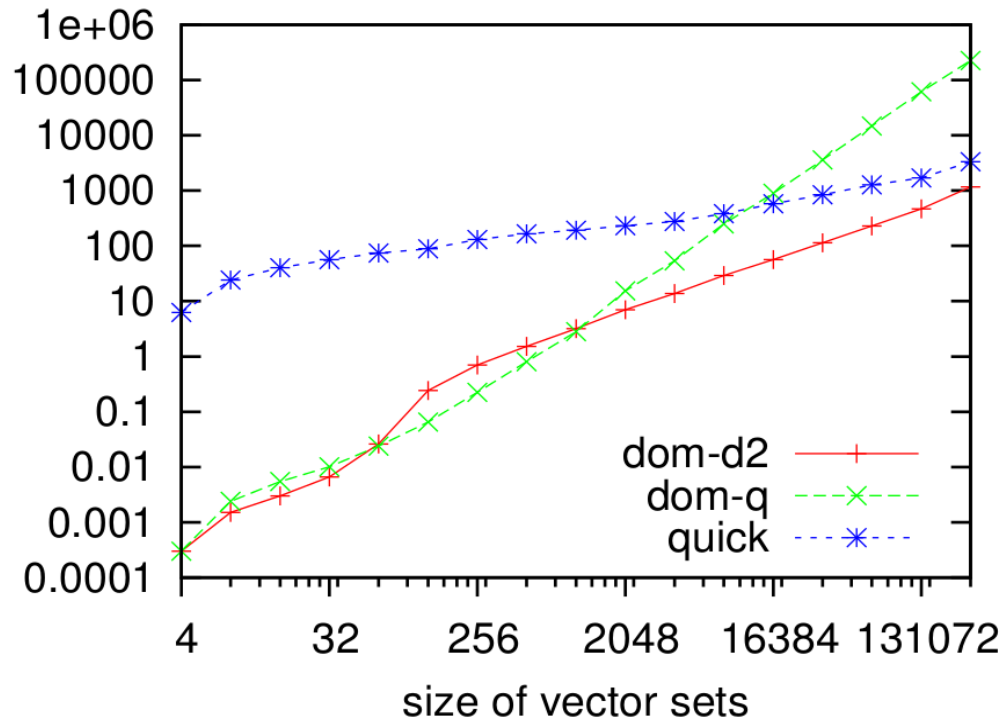




Von Customizable zu Personalizable

Experimentelle Pruning Ergebnisse





Bis $d = 64$ Anfragezeiten unter einer Sekunde.

Automatische α-Deduktion

Automatische α-Deduktion

Problem α manuell zu spezifizieren ist aufwändig und nicht trivial

0.3 Reisezeit, 0.4 Benzinkosten, 0.3 Staugefahr?

0.2 Reisezeit, 0.5 Benzinkosten, 0.3 Staugefahr?

0.2 Reisezeit, 0.2 Benzinkosten, 0.6 Staugefahr?

Automatische α-Deduktion

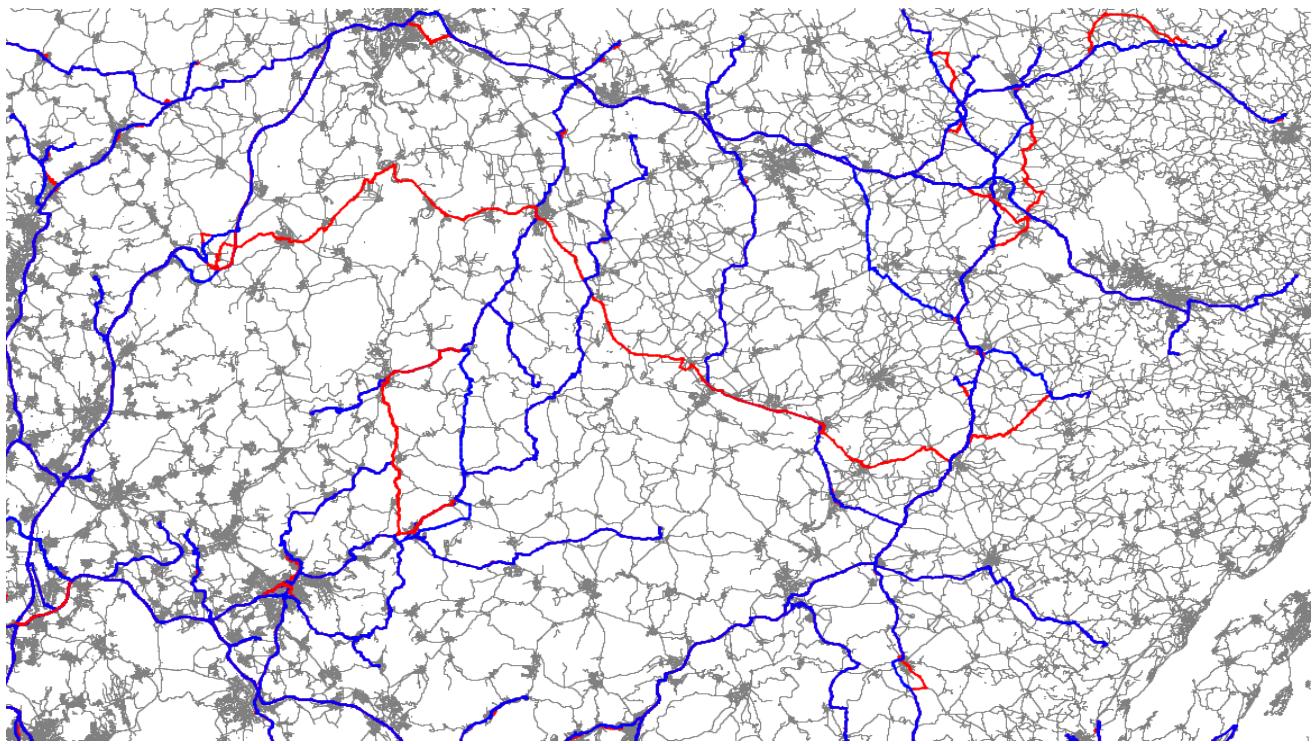
Problem α manuell zu spezifizieren ist aufwändig und nicht trivial

0.3 Reisezeit, 0.4 Benzinkosten, 0.3 Staugefahr?

0.2 Reisezeit, 0.5 Benzinkosten, 0.3 Staugefahr?

0.2 Reisezeit, 0.2 Benzinkosten, 0.6 Staugefahr?

Idee automatische α-Berechnung aus bisherigen Routen eines Nutzers



Automatische α-Deduktion

Eine Menge von Pfaden ist zulässig, wenn es ein α gibt, sodass alle diese Pfade optimal sind für dieses α.

O.B.d.A. $\alpha_i \in [0, 1], \sum_i \alpha_i = 1$

Wie findet man α für eine gegebene Menge P zulässiger Pfade?

Automatische α-Deduktion

Eine Menge von Pfaden ist zulässig, wenn es ein α gibt, sodass alle diese Pfade optimal sind für dieses α.

O.B.d.A. $\alpha_i \in [0,1], \sum_i \alpha_i = 1$

Wie findet man α für eine gegebene Menge P zulässiger Pfade?

LP-Formulierung

$$\begin{aligned} \alpha_1 + \cdots + \alpha_d &= 1 \\ \alpha_1 &\geq 0 \\ &\cdots \\ \alpha_d &\geq 0 \end{aligned}$$

Automatische α-Deduktion

Eine Menge von Pfaden ist zulässig, wenn es ein α gibt, sodass alle diese Pfade optimal sind für dieses α.

O.B.d.A. $\alpha_i \in [0,1], \sum_i \alpha_i = 1$

Wie findet man α für eine gegebene Menge P zulässiger Pfade?

LP-Formulierung

$$\begin{aligned} \alpha_1 + \cdots + \alpha_d &= 1 \\ \alpha_1 &\geq 0 \\ &\cdots \\ \alpha_d &\geq 0 \end{aligned}$$

$\forall p(s,t) \in P$: für alle alternativen $s-t-$Pfade p':

$\alpha^T c(p) - \alpha^T c(p') \leq 0$ alle Alternativpfade müssen teurer sein als p für α

Automatische α-Deduktion

Eine Menge von Pfaden ist zulässig, wenn es ein α gibt, sodass alle diese Pfade optimal sind für dieses α.

O.B.d.A. $\alpha_i \in [0,1], \sum_i \alpha_i = 1$

Wie findet man α für eine gegebene Menge P zulässiger Pfade?

LP-Formulierung

$$\begin{aligned} \alpha_1 + \cdots + \alpha_d &= 1 \\ \alpha_1 &\geq 0 \\ &\cdots \\ \alpha_d &\geq 0 \end{aligned}$$

$\forall p(s,t) \in P$: für alle alternativen $s-t-$Pfade p':

$\alpha^T c(p) - \alpha^T c(p') \leq 0$ alle Alternativpfade müssen teurer sein als p für α

Problem potenziell exponentiell viele Alternativpfade

Automatische α-Deduktion

input : path set P, network $G(V, E)$,
cost vectors $c : E \to \mathbb{R}_+^d$
output: feasible preference α

```
1  begin
       /* initialize LP                                   */
2      LP.add_constraint(∑_{i=1}^{d} α_i = 1);
3      LP.add_constraint(α_1 ≥ 0);
4      ...
5      LP.add_constraint(α_d ≥ 0);
       /* get feasible α ∈ ℝ^d                            */
6      α = LP.solve();
       /* check and refine                                */
7      while true do
8          all_explained = true ;
9          for p_k ∈ P do
10             π =Dijkstra(G, s_k, t_k, α);
11             if c(p_k, α) > c(π, α) then
                   /* path p_k not explained by current
                      α                                    */
12                 all_explained = false;
                   /* add constraint to make current α
                      infeasible                           */
13                 LP.add_constraint(
14                 ∑_{i=1}^{d} (c_i(p_k) − c_i(π)) · α_i ≤ 0);
15         if all_explained then
16             break;
17         else
18             α = LP.solve();
19     return α;
```

Abhilfe

Füge Constraints nach Bedarf hinzu.

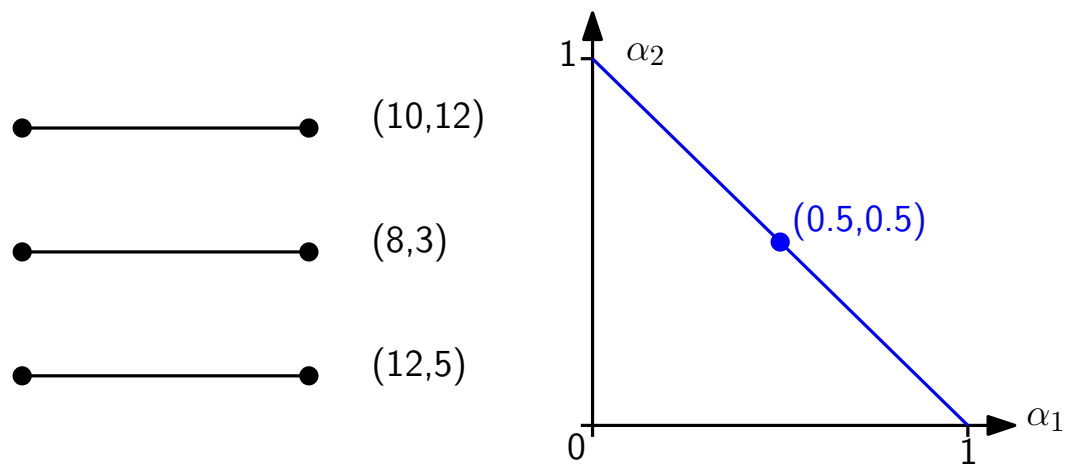
Automatische α-Deduktion

input : path set P, network $G(V, E)$,
cost vectors $c : E \to \mathbb{R}_+^d$
output: feasible preference α

```
1  begin
       /* initialize LP                                   */
2      LP.add_constraint(∑_{i=1}^{d} α_i = 1);
3      LP.add_constraint(α_1 ≥ 0);
4      ...
5      LP.add_constraint(α_d ≥ 0);
       /* get feasible α ∈ ℝ^d                            */
6      α = LP.solve();
       /* check and refine                                */
7      while true do
8          all_explained = true ;
9          for p_k ∈ P do
10             π =Dijkstra(G, s_k, t_k, α);
11             if c(p_k, α) > c(π, α) then
                   /* path p_k not explained by current
                      α                                    */
12                 all_explained = false;
                   /* add constraint to make current α
                      infeasible                           */
13                 LP.add_constraint(
14                 ∑_{i=1}^{d} (c_i(p_k) − c_i(π)) · α_i ≤ 0);
15         if all_explained then
16             break;
17         else
18             α = LP.solve();
19     return α;
```

Abhilfe

Füge Constraints nach Bedarf hinzu.

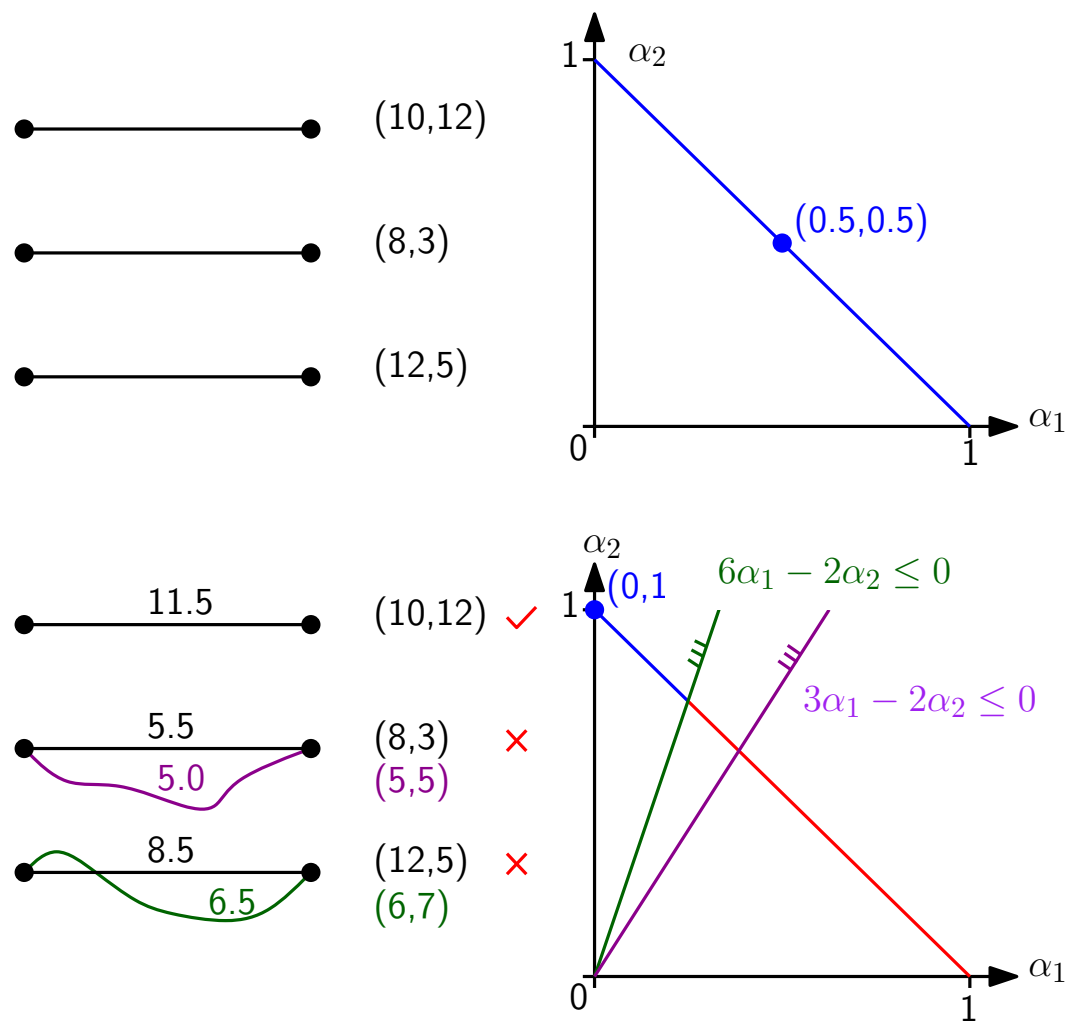


Automatische α-Deduktion

```
input  : path set P, network G(V, E),
         cost vectors c : E → ℝ^d_+
output: feasible preference α
1  begin
       /* initialize LP                                   */
2      LP.add_constraint(∑_{i=1}^{d} α_i = 1);
3      LP.add_constraint(α_1 ≥ 0);
4      ...
5      LP.add_constraint(α_d ≥ 0);
       /* get feasible α ∈ ℝ^d                            */
6      α = LP.solve();
       /* check and refine                                */
7      while true do
8          all_explained = true ;
9          for p_k ∈ P do
10             π =Dijkstra(G, s_k, t_k, α);
11             if c(p_k, α) > c(π, α) then
                   /* path p_k not explained by current
                      α                                    */
12                 all_explained = false;
                   /* add constraint to make current α
                      infeasible                           */
13                 LP.add_constraint(
14                 ∑_{i=1}^{d} (c_i(p_k) − c_i(π)) · α_i ≤ 0);
15         if all_explained then
16             break;
17         else
18             α = LP.solve();
19     return α;
```

Abhilfe

Füge Constraints nach Bedarf hinzu.



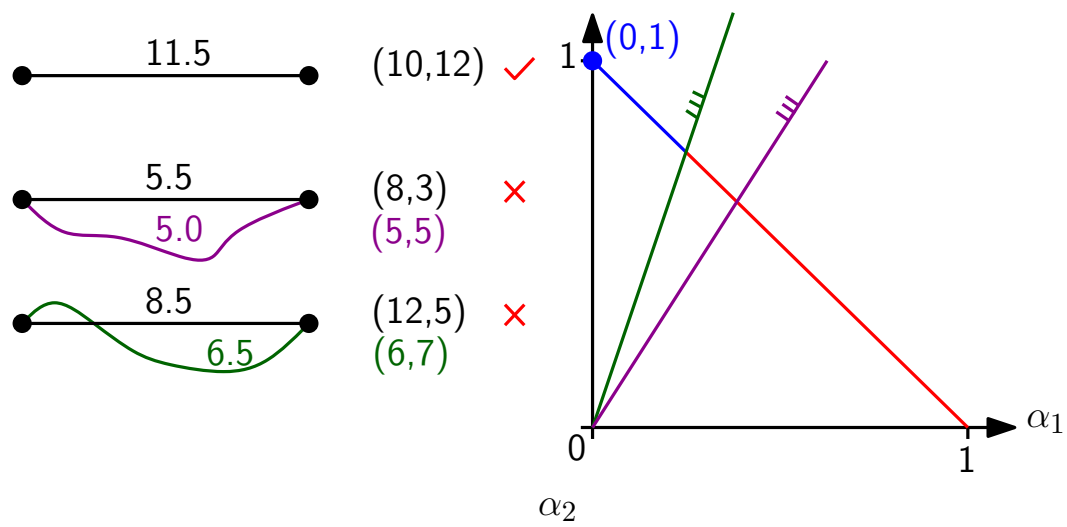
Automatische α-Deduktion

input : path set P, network $G(V, E)$,
cost vectors $c : E \to \mathbb{R}_+^d$
output: feasible preference α

```
1  begin
       /* initialize LP                                   */
2      LP.add_constraint(∑_{i=1}^{d} α_i = 1);
3      LP.add_constraint(α_1 ≥ 0);
4      ...
5      LP.add_constraint(α_d ≥ 0);
       /* get feasible α ∈ ℝ^d                            */
6      α = LP.solve();
       /* check and refine                                */
7      while true do
8          all_explained = true ;
9          for p_k ∈ P do
10             π =Dijkstra(G, s_k, t_k, α);
11             if c(p_k, α) > c(π, α) then
                   /* path p_k not explained by current
                      α                                    */
12                 all_explained = false;
                   /* add constraint to make current α
                      infeasible                           */
13                 LP.add_constraint(
14                 ∑_{i=1}^{d} (c_i(p_k) − c_i(π)) · α_i ≤ 0);
15         if all_explained then
16             break;
17         else
18             α = LP.solve();
19     return α;
```

Abhilfe

Füge Constraints nach Bedarf hinzu.



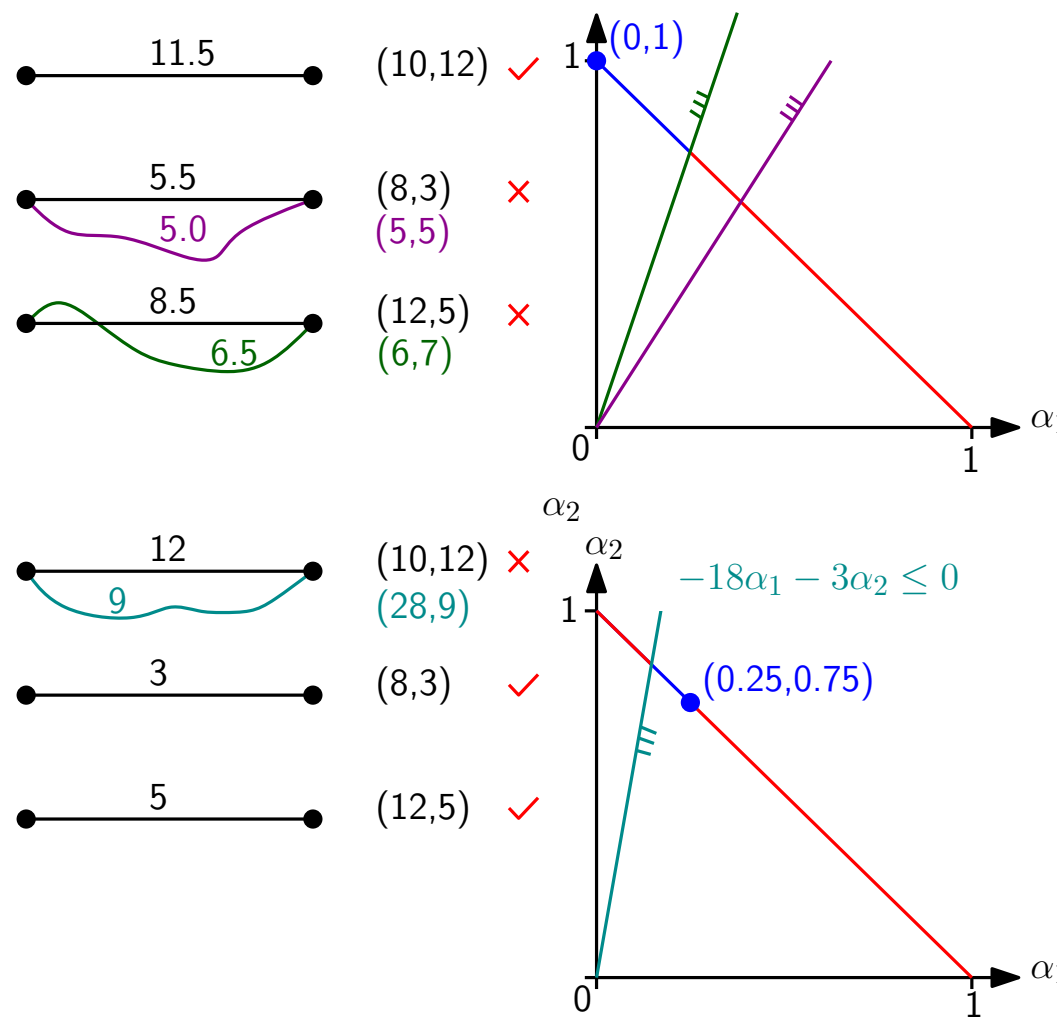
Automatische α-Deduktion

input : path set P, network $G(V, E)$,
cost vectors $c : E \to \mathbb{R}^d_+$
output: feasible preference α

```
1  begin
       /* initialize LP                                   */
2      LP.add_constraint(∑_{i=1}^{d} α_i = 1);
3      LP.add_constraint(α_1 ≥ 0);
4      ...
5      LP.add_constraint(α_d ≥ 0);
       /* get feasible α ∈ R^d                            */
6      α = LP.solve();
       /* check and refine                                */
7      while true do
8          all_explained = true ;
9          for p_k ∈ P do
10             π =Dijkstra(G, s_k, t_k, α);
11             if c(p_k, α) > c(π, α) then
                   /* path p_k not explained by current
                      α                                    */
12                 all_explained = false;
                   /* add constraint to make current α
                      infeasible                           */
13                 LP.add_constraint(
14                 ∑_{i=1}^{d} (c_i(p_k) − c_i(π)) · α_i ≤ 0);
15         if all_explained then
16             break;
17         else
18             α = LP.solve();
19     return α;
```

Abhilfe

Füge Constraints nach Bedarf hinzu.



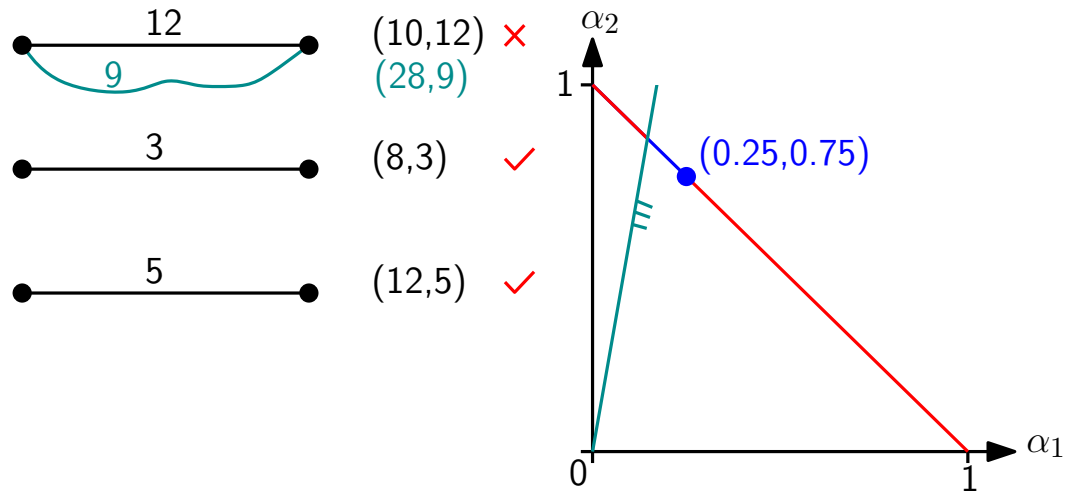
Automatische α-Deduktion

input : path set P, network $G(V, E)$,
cost vectors $c : E \to \mathbb{R}_+^d$
output: feasible preference α

```
1  begin
       /* initialize LP                                   */
2      LP.add_constraint(∑_{i=1}^{d} α_i = 1);
3      LP.add_constraint(α_1 ≥ 0);
4      ...
5      LP.add_constraint(α_d ≥ 0);
       /* get feasible α ∈ ℝ^d                            */
6      α = LP.solve();
       /* check and refine                                */
7      while true do
8          all_explained = true ;
9          for p_k ∈ P do
10             π =Dijkstra(G, s_k, t_k, α);
11             if c(p_k, α) > c(π, α) then
                   /* path p_k not explained by current
                      α                                    */
12                 all_explained = false;
                   /* add constraint to make current α
                      infeasible                           */
13                 LP.add_constraint(
14                 ∑_{i=1}^{d} (c_i(p_k) − c_i(π)) · α_i ≤ 0);
15         if all_explained then
16             break;
17         else
18             α = LP.solve();
19     return α;
```

Abhilfe

Füge Constraints nach Bedarf hinzu.



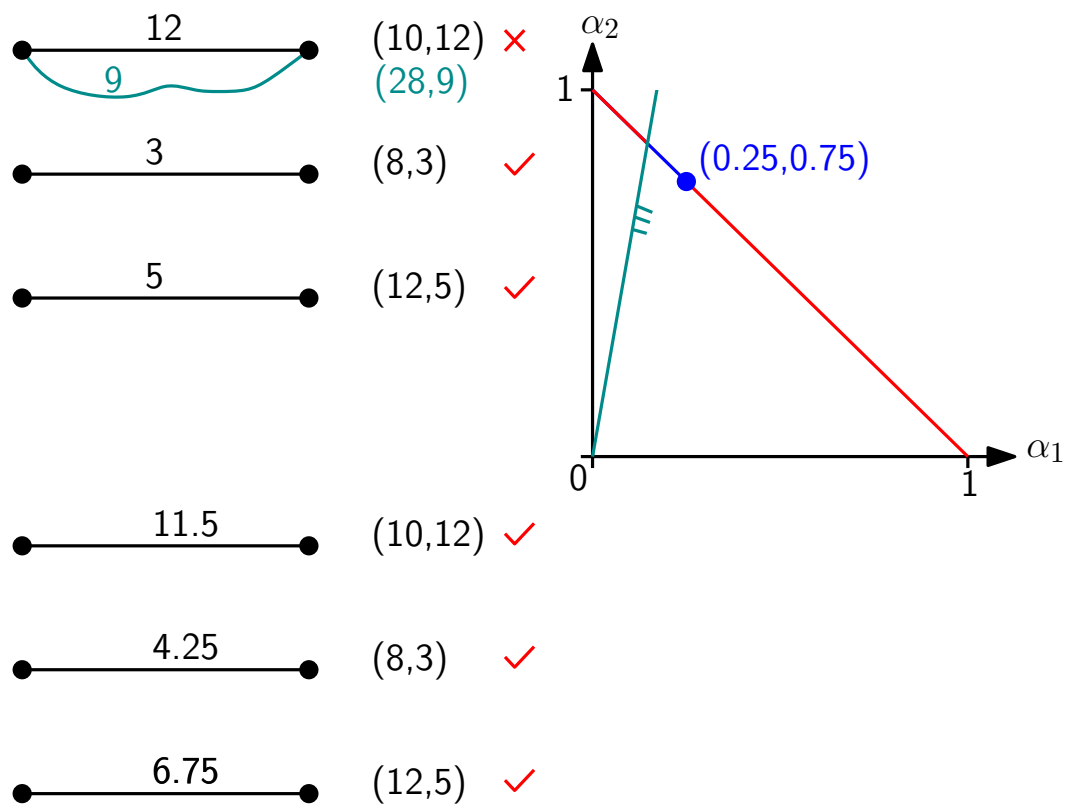
Automatische α-Deduktion

input : path set P, network $G(V, E)$,
cost vectors $c : E \to \mathbb{R}_+^d$
output: feasible preference α

```
1  begin
       /* initialize LP                                   */
2      LP.add_constraint(∑_{i=1}^{d} α_i = 1);
3      LP.add_constraint(α_1 ≥ 0);
4      ...
5      LP.add_constraint(α_d ≥ 0);
       /* get feasible α ∈ ℝ^d                            */
6      α = LP.solve();
       /* check and refine                                */
7      while true do
8          all_explained = true ;
9          for p_k ∈ P do
10             π =Dijkstra(G, s_k, t_k, α);
11             if c(p_k, α) > c(π, α) then
                   /* path p_k not explained by current
                      α                                    */
12                 all_explained = false;
                   /* add constraint to make current α
                      infeasible                           */
13                 LP.add_constraint(
14                 ∑_{i=1}^{d} (c_i(p_k) − c_i(π)) · α_i ≤ 0);
15         if all_explained then
16             break;
17         else
18             α = LP.solve();
19     return α;
```

Abhilfe

Füge Constraints nach Bedarf hinzu.



Automatische α-Deduktion

```
input  : path set P, network G(V, E),
         cost vectors c : E → ℝ^d_+
output: feasible preference α
1  begin
       /* initialize LP                                  */
2      LP.add_constraint(∑_{i=1}^{d} α_i = 1);
3      LP.add_constraint(α_1 ≥ 0);
4      ...
5      LP.add_constraint(α_d ≥ 0);
       /* get feasible α ∈ ℝ^d                           */
6      α = LP.solve();
       /* check and refine                               */
7      while true do
8          all_explained = true ;
9          for p_k ∈ P do
10             π =Dijkstra(G, s_k, t_k, α);
11             if c(p_k, α) > c(π, α) then
                   /* path p_k not explained by current
                      α                                   */
12                 all_explained = false;
                   /* add constraint to make current α
                      infeasible                          */
13                 LP.add_constraint(
14                 ∑_{i=1}^{d} (c_i(p_k) − c_i(π)) · α_i ≤ 0);
15         if all_explained then
16             break;
17         else
18             α = LP.solve();
19     return α;
```

Abhilfe

Füge Constraints nach Bedarf hinzu.

Polyzeitalgorithmus via Ellipsoidmethode.

Automatische α-Deduktion

```
input  : path set P, network G(V, E),
         cost vectors c : E → ℝ^d_+
output: feasible preference α
1  begin
       /* initialize LP                                  */
2      LP.add_constraint(∑_{i=1}^{d} α_i = 1);
3      LP.add_constraint(α_1 ≥ 0);
4      ...
5      LP.add_constraint(α_d ≥ 0);
       /* get feasible α ∈ ℝ^d                           */
6      α = LP.solve();
       /* check and refine                               */
7      while true do
8          all_explained = true ;
9          for p_k ∈ P do
10             π =Dijkstra(G, s_k, t_k, α);
11             if c(p_k, α) > c(π, α) then
                   /* path p_k not explained by current
                      α                                   */
12                 all_explained = false;
                   /* add constraint to make current α
                      infeasible                          */
13                 LP.add_constraint(
14                 ∑_{i=1}^{d} (c_i(p_k) − c_i(π)) · α_i ≤ 0);
15         if all_explained then
16             break;
17         else
18             α = LP.solve();
19     return α;
```

Abhilfe

Füge Constraints nach Bedarf hinzu.

Polyzeitalgorithmus via Ellipsoidmethode.

Beschleunigung in der Praxis:

CCH mit Customization pro while-Durchlauf (innerhalb der Schleife α für alle Queries gleich)

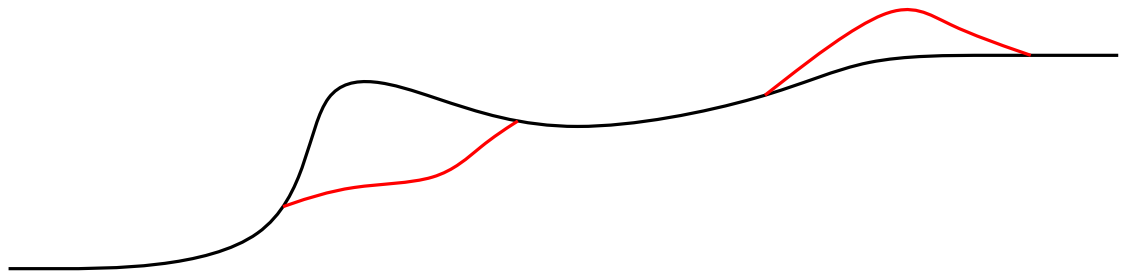
Automatische α-Deduktion

Resultate

$d = 10$

	Stadt			Bundesland			Deutschland		
$\|P\|$	Zeit (s)	Runden	α_{com} A_N	Zeit (s)	Runden	α_{com} A_N	Zeit (s)	Runden	α_{com} A_N
1	0.2	1.8	0.19	33.7	29.0	0.51	141.7	31.4	0.67
10	0.4	6.1	0.17	24.1	14.9	0.21	73.8	16.2	0.23
50	0.4	6.2	0.06	19.9	10.2	0.05	47.3	9.5	0.04
100	0.4	5.8	0.03	20.5	7.7	0.03	32.4	5.2	0.01
1000	0.8	4.2	0.01	85.3	5.3	0.02	103.9	3.1	0.00

A_N misst wie stark sich Pfade unterscheiden, die mit dem echten Nutzer-α und dem von uns berechneten α generiert wurden.



Zusammenfassung

Beschleunigungstechniken in abgewandelter Form für personalisierte Queries anwendbar.

Personalisierte CH.

Notwendigkeit eines Shortcuts entscheidbar in Polyzeit.

Customizable $\rightarrow$ Personalizable.

Schnelles Prunen von hochdimensionalen Vektormengen.

Zusammenfassung

Beschleunigungstechniken in abgewandelter Form für personalisierte Queries anwendbar.

Personalisierte CH.

Notwendigkeit eines Shortcuts entscheidbar in Polyzeit.

Customizable $\rightarrow$ Personalizable.

Schnelles Prunen von hochdimensionalen Vektormengen.

Nutzerpräferenzen können automatisch aus Pfaddaten ermittelt werden.

LP-Formulierung

Ellipsoid-Methode + effizientes Dijkstra-Orakel für Constraints basierend auf CCH

Literatur

Daniel Delling, Andrew V. Goldberg, Moiss Goldszmidt, John Krumm, Kunal Talwar, Renato F. Werneck:
Navigation made personal: inferring driving preferences from GPS traces.
SIGSPATIAL/GIS 2015: 31:1-31:9

Julian Dibbelt, Ben Strasser, Dorothea Wagner:
Fast exact shortest path and distance queries on road networks with parametrized costs.
SIGSPATIAL/GIS 2015: 66:1-66:4

Stefan Funke, Sabine Storandt:
Personalized route planning in road networks.
SIGSPATIAL/GIS 2015: 45:1-45:10

Stefan Funke, Sabine Storandt:
Polynomial-time construction of contraction hierarchies for multi-criteria objectives.
ALENEX 2013: 41-54

Robert Geisberger, Moritz Kobitzsch, Peter Sanders:
Route Planning with Flexible Objective Functions.
ALENEX 2010: 124-137