Das CSP Problem ist folgendermaßen definiert:

$$\text{(CSP)} \quad \begin{cases} \min \sum_{p \in \mathcal{P}} \tau_p f_p & \\ l_p & \leq L \text{ für } f_p > 0 \\ \sum_{p \in \mathcal{P}} f_p & = d \\ f & \text{zulässiger, nicht negativer Fluss} \end{cases}$$

wobei $l_a, \tau_a \in \mathbb{R}$, $l_p = \sum_{a \in p} l_a$ und $\tau_p = \sum_{a \in p} \tau_a$.

4.1. **Komplexität.** Bevor wir uns das CSP genauer anschauen noch eine kurze Wiederholung zweier Definitionen. Ein NP-vollständiges Problem heißt *stark NP-vollständig*, falls es auch dann noch NP-vollständig ist, wenn die Eingabe nur aus Zahlen besteht, deren Größe polynomiell in der Eingabelänge beschränkt ist. Andernfalls heißt das Problem *schwach NP-vollständig*. Für stark NP-vollständige Problem kann es - unter der Annahme NP ungleich P - noch nicht einmal so genannte *pseudopolynomielle Algorithmen* geben. Das sind Algorithmen, deren Laufzeit polynomiell ist, wenn die Größe aller in der Eingabe vorkommenden Zahlen polynomiell in der Eingabelänge beschränkt sind.

Proposition 4.1. *Das CSP ist schwach NP-vollständig.*

Beweis. Um zu zeigen, dass das CSP schwach NP-vollständig ist, reduzieren wir es auf das PARTITION Problem, das schwach NP-vollständig ist. Im PARTITION Problem sind n Zahlen $a_1, ..., a_n$ gegeben mit $\sum_{i=1}^{n} a_i = b$. Gesucht ist eine Teilmenge $I \subset \{1, ..., n\}$ mit $\sum_{i \in I} a_i = b$. Sei I eine Instanz von PARTITION. Dann kann man folgendermaßen eine Instanz I' des CSP konstruieren (Abb. 28), wobei der τ_a dem ersten Wert und l_a dem zweiten Wert der Kantenbewertung entspricht.

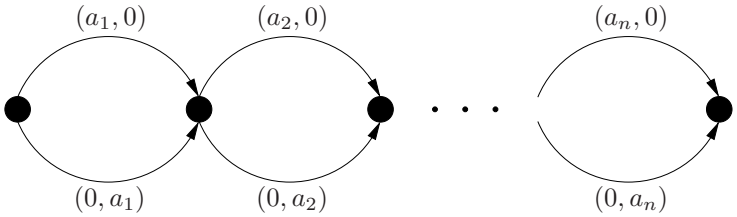


Abbildung 28. Aus der Instanz I der PARTITION wird die Instanz I' des CSP konstruiert.

Beim CSP wird jetzt ein kürzester Weg p bzgl. der Zeit gesucht, mit $\sum_{a \in P} l_a \leq L$. Setzt man $L = b$, so ermöglicht das lösen der Instanz die folgende Frage zu beantworten: existiert ein $s - t$ Weg p mit $\tau(p) \leq b$ und $l(p) \leq b$? Die Antwort lautet offensichtlich genau dann „Ja“, wenn eine Indexmenge I mit $\sum_{i \in I} a_i = b$ existiert. □

Proposition 4.2. *Das CSP kann in pseudopolynomieller Zeit gelöst werden.*

Beweis. analog zu SUBSET SUM aus AMD I. □

4.2. **Lösungsansätze.** Im Folgenden werden wir drei Ideen zur Lösung des CSP betrachten. Die erste geht über die Lagrange-Relaxation mit Branch & Bound (B&B) und wurde 1989 von Beasly und Christofides vorgestellt. In der zweiten Idee wir ein Labeling Algorithmus entwickelt, der den Dijkstra so erweitert, dass er auch für Pareto-optimale Wege nutzbar ist. Als letztes betrachten wir in Abschnitt 4.4 einen geometrischen Ansatz von Mehlhorn und Zieglemann aus dem Jahr 2000.

4.2.1. *Der Ansatz von Beasly und Christofides.* Der Ansatz basiert auf einem Branch & Bound bzgl. der Wahl von Kanten ähnlich wie beim Problem der Telekom aus ADM II. Wir formulieren also ein IP und legen bestimmte Kanten fest. Zur Bestimmung von unteren Schranken benutzen wir die Lagrange- Relaxation der IP Formulierung.

Sei

$$\text{IP} \begin{cases} \min \tau(p) & \\ l(p) & \leq L \\ p & \in \mathcal{P}_{s,t} \end{cases}$$

das IP, das zu lösen ist. Daraus ergibt sich das Lagrange-Relaxierte Problem (LR_μ)

$$(\text{LR})_\mu \begin{cases} \min \Lambda(\mu): & = \tau(p) + \mu(l(p) - L) \\ \mu & \geq 0 \\ p & \in \mathcal{P}_{s,t} \end{cases}$$

Für jeden Wert p erhalten wir also neue Kosten

$$\tau(p) + \mu(l(p) + L) = \sum_{a \in p} (\tau_a + \mu l_a) - \mu L,$$

wobei μL für festes μ konstant ist. Um (LR_μ) zu lösen, müssen wir ein kürzeste Wege Problem bzgl. der Kantenbewertung $\tau_a + \mu l_a$ lösen. Allgemein können wir dann für eine Lösung des Problems (LR_μ) folgende Aussagen treffen:

Sei p^* optimale Lösung von CSP und $\tau^* := \tau(p^*)$. Dann gilt die folgenden drei Eigenschaften:

1. $\Lambda(\mu) \leq \tau^*$ für alle $\mu \geq 0$.
2. $\Lambda^* := \max_{\mu \geq 0} \Lambda(\mu) \leq \tau^*$
3. Ist p optimale Lösung von $\Lambda(\mu)$ mit $l(p) \leq L$, dann ist p optimal für CSP.

Die Differenz $\tau^* - \Lambda^*$ heißt *Dualitätslücke*.

Um Λ^* zu berechnen benutzt man das Subgradientenverfahren (s. ADM II, Kapitel 7.4). Sei p_{μ_0}ein kürzester Weg in (LR_{μ_0}), so ist die verletzte Nebenbedingung $l(p_0) - L$ ein Subgradient in μ_0 von $\Lambda(\mu)$. Gehe dann einen Schritt in diese Richtung $\mu^{\text{neu}} := \mu^{\text{alt}} + \theta(l(p_0) - L)$. Daraus erhält man eine neue Kantenbewertung $\tau_a + \mu^{\text{neu}} l_a$. Der Unterschied in den Kantenbewertungen ist dann gerade $(\mu^{\text{neu}} - \mu^{\text{alt}})l_a$.

Der l_a- Anteil aller Kanten wird um denselben Faktor $\mu^{\text{neu}} - \mu^{\text{alt}}$ verändert, wir erhalten also wenig Variation. Ein typischer, unbefriedigender Verlauf ist dann (Abb. 29).

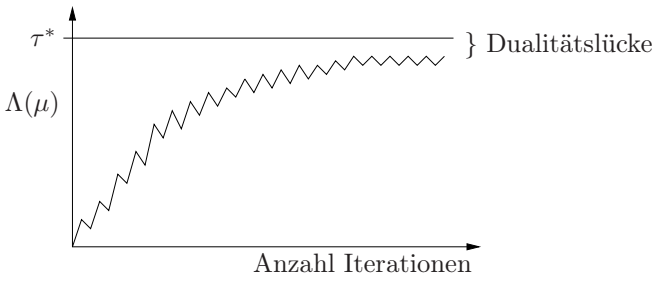


Abbildung 29. Typischer Verlauf von Branch & Bound.

4.2.2. *Pareto-optimale Wege.* Uns sei das folgende Problem gegeben: Sei $G = (V, A)$ ein Graph mit $s, t \in V$. Für jede Kante a gibt es einen Bewertungsvektor $\lambda(a)$ mit $\lambda_i(a) \geq 0$, $i = 1, ..., r$. Die *Bewertung eines Weges* p ist dann

$$\lambda(p) := \sum_{a \in p} \lambda(a) = \begin{pmatrix} \sum_{a \in p} \lambda_1(a) \\ \vdots \\ \sum_{a \in p} \lambda_r(a) \end{pmatrix} =: \begin{pmatrix} \lambda_1(p) \\ \vdots \\ \lambda_r(p) \end{pmatrix}.$$

Note 4.3. Es sind auch verschiedene Operatoren ($\sum$, max, ...) pro Komponente möglich.

Gesucht sind dann alle Pareto-optimalen s, t- Wege.

Definition 4.4. Ein Weg p heißt *Pareto-optimal*, wenn kein Weg p' mit $\lambda(p') \leq \lambda(p)$ und $\lambda_i(p') < \lambda_i(p)$ für ein i existiert. D.h. es gibt kein p', dass p dominiert.

Geometrisch können wir Pareto- Optimalität bei zwei Bewertungesparametern τ_a und l_a einer Kante folgendermaßen veranschaulichen. Ein Weg p kann bzgl. seiner Bewertung in ein Koordinatensystem eingetragen werden. Ein Weg p ist genau dann Pareto-optimal, wenn sich in dem Quadrat $[0, l(p)] \times [0, \tau(p)]$ kein anderer Weg p' befindet (Abb. 30).

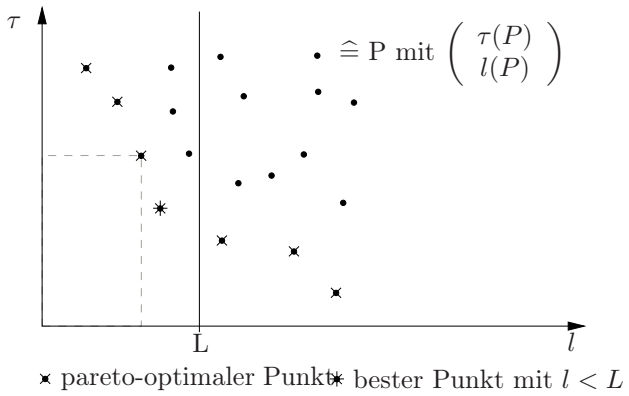


Abbildung 30. Pareto-optimale Wege.

Das mehrere Bewertungsparameter in der Praxis häufig vorkommen kann man sich leicht am Beispiel Bahn und Autoverkehr klar machen. Eine Bahnstrecke wird häufig bzgl. der Fahrzeit, den Kosten und der Anzahl der Umstiege bewertet. Bei der Wahl einer Autostrecke ergibt sich als weiteres Kriterium die Vertrautheit der Strecke.

Im folgenden wollen wir den Dijkstra Algorithmus für Pareto- optimale Wege betrachten. Dazu eine kurze Wiederholung des Dijkstra Algorithmus für kürzeste Wege ($\lambda_a \geq 0$) (Algo. 1).

Die Grundideen des Dijkstras sind, dass für jeden Knoten v eine Distanz $d[v]$, die der Länge des bisherigen kürzesten Weges von s zu v entspricht, berechnet wird und ein Knoten Vorgänger[v], der Vorgänger von v auf dem bisher kürzesten Weg von s nach v, gespeichert wird. Die Werte $d[v]$ und Vorgänger[v] sind zunächst vorläufig und werden bei jeder Iteration aktualisiert. Im Laufe des Algorithmus werden Knoten markiert. Für markierte Knoten sind die Werte und damit die kürzesten Wege von s nach v endgültig.

Algorithm 1 Der Dijkstra Algorithmus für kürzeste Wege

1: **Initialisierung**:
2: $d(v) := \begin{cases} 0 & v = s \\ \lambda_{(s,v)} & \text{falls Kante } (s,v) \text{ existiert} \\ \infty & \text{sonst} \end{cases}$
3: $\text{Vorgäner}[v] := \begin{cases} s & \text{falls Kante } (s,v) \text{ existiert} \\ \infty & \text{sonst} \end{cases}$
4: nur s markiert
5:
6: **Hauptschleife**:
7: **while** $\exists$ unmarkierter Knoten **do**
8: wähle unmarkierten Knoten v mit kleinstem $d[v]$
9: markiere v
10: **for** alle Knoten (v,w) mit unmarkierten w **do**
11: **if** $d[v] + \lambda_{(v,w)} < d[w]$ **then**
12: $d[w] := d[v] + \lambda_{(v,w)}$
13: $\text{Vorgänger}[w] := v$
14: **end if**
15: **end for**
16: **end while**

Die Variation des Dijkstras auf Pareto-optimale Wege wurde 1994 von Theune entwickelt. Anstatt von $d[v]$ wird jetzt ein $r-$dimensionaler Vektor $d[v] = (d_1[v], ..., d_r[v])^\top$ mit $d_k[v]$ bzgl. λ_k gespeichert. Im folgenden bezeichen wir dies Vektoren als Label. In einem Knoten können mehrere Label $d[v]$ gespeichert werden. Jedes Label entspricht dann einem bisher ermittelten Pareto-optimalen $s, v-$Weg. Wurde im Dijkstra Algorithmus für kürzeste Wege ein Knoten markiert, so markieren wir dieses mal Label. Als Auswahlregel eines neuen Labels, ergibt sich: Wähle das lexikographisch kleinste unmarkierte Label. Die Aktualisierung der Label funktioniert dann folgendermaßen: betrachte zu dem gewählten Label alle Kanten (v,w) des zum Label gehörigen Knoten v. Nehme den Vektor $d[v] + \lambda_{(v,w)}$ zu den bereits in w abgespeicherten Labeln hinzu, streiche nicht Pareto-optimale und aktualisiere ggf. Vorgängerstruktur, welche jetzt auch Labelbasiert ist (Algo. 2).

Algorithm 2 Der Dijkstra Algorithmus für Pareto-optimale Wege

Initialisierung:

$$d[v] := \begin{cases} (0,...,0)^\top & v = s \\ \lambda_{(s,v)} & \text{falls Kante } (s,v) \text{ existiert} \\ (\infty,...,\infty)^\top & \text{sonst} \end{cases}$$

Anfangs nur $d[s]$ markiert

Hauptschleife:
while $\exists$ unmarkiertes Label **do**
  wähle lexikographisch kleinsten unmarkiertes Label $d[v]$
  markiere $d[v]$
  sei v der zugehörige Knoten
  for alle Kanten (v, w) **do**
    füge dem Knoten w das Label $d[w] := d[v] + \lambda(v, w)$ hinzu
    streiche nicht Pareto-optimale Label bei w
    falls $d[w]$ nicht gestrichen wird, so setzte Vorgänger$[d[w]] := d[v]$
  end for
end while

Betrachten wir im folgenden ein Beipiel, wie der Pareto- Dijkstra funktioniert.

Example 4.5. Anwendung des Pareto- Dijkstras an dem Graphen in Abbildung 31.

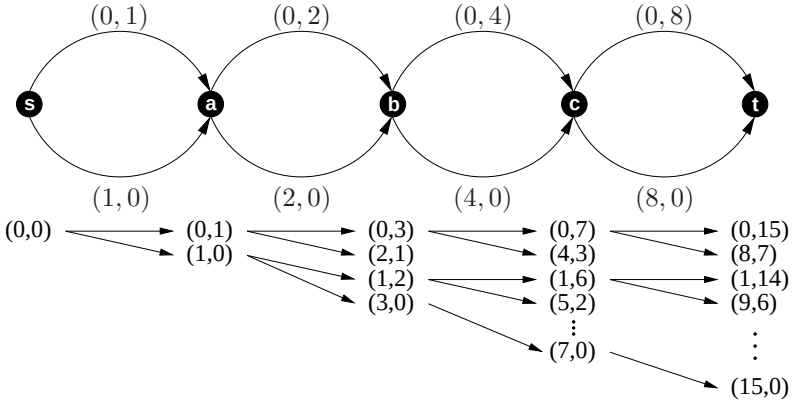


Abbildung 31. Anwendung des Pareto-Dijkstras.

Wir beginnen mit dem Knoten s. Von dort gehen zwei Kanten zum Knoten a mit den Bewertungen $(0, 1)$ und $(1, 0)$. Der Vorgänger von b wird dann auf a gesetzt. Wir erhalten für a also zwei Bewertungsvektoren $(0, 1)$ und $(1, 0)$. Der Lexikographisch kleinere ist der Knoten $(0, 1)$. Mit dem geht es weiter zum Knoten c und wir erhalten dort die Bewertungen $(0, 3)$ und $(2, 1)$. Von allen Bewertungen, außer den schon betrachteten, ist jetzt $d[c] = (0, 3)$ die lexikographisch kleinste. Von dort gehen wir zu Knoten d und danach zum Knoten t. Betrachten wir nun die noch nicht markierten $d[v]$, so ist das lexikographisch kleinste jetzt $d[a] = (1, 0)$. Von dort erhalten wir zwei neue Bewertungen $(1, 2)$ und $(3, 0)$ von b. Den Markierungs- und Bewertungsschritt wiederholen wir jetzt, bis kein $d[v]$ mehr unmarkiert ist. Damit sind wir fertig und haben alles Pareto-optimalen Wege erhalten.

An dem Beispiel sieht man sehr gut, dass die Anzahl der Label exponentiell wachsen kann. Wir haben es also mit einem exponentiellen Algorithmus zu tun. Das er korrekt arbeitet zeigen wir im nächsten Satz.

Theorem 4.6. *Der Algorithmus Pareto-Dijkstra arbeitet korrekt.*

Beweis. Wir zeigen: sobald ein Label markiert wird, gehört es zu den Pareto-Optima für diesen Knoten.

Betrachte den Zeitpunkt, zu dem Label $d[v]$ gewählt wird. Angenommen $d[v]$ ist nicht Pareto-optimal. Dann existiert ein Pareto-optimaler Weg p von s nach v mit $\lambda(p) \leq d[v]$ und es existiert ein i mit $\lambda_i[p] < d_i[v]$. Dabei war $\lambda(p)$ durch $\lambda_i(p) = \sum_{a \in p} \lambda_i(a)$ definiert. Sei $p = \{s = v_1, v_2, ..., v_m = v\}$ mit zugehörigen Pareto-optimalen Bewertungsvektoren $d_1, d_2, ..., d_m$ mit $d_i = d_{i-1} + \lambda(v_{i-1}, v_i)$.

Sei d_k das letzte Label von v, das bereits markiert ist. Ein solches existiert, da $d_1 = d[s]$ markiert und $d[v] = d[v_m]$ nicht markiert ist. Dann ist $k < m$, sonst wäre $d_k[v]$ bereits in der Menge der Label von v enthalten und hätte $d[v]$ weg dominiert. Zum Zeitpunkt der Markierung von d_k wird $d_{k+1} = d_k + \lambda_{(v_k, v_{k+1})}$ zu v_{k+1} hinzugefügt und ist Pareto-optimal, da dieses ein Anfangsstück eines Pareto-optimalen Weges ist. Durch die Wahl von k bleibt dieses Label $d_k[v]$ bis zur Wahl von $d[v]$ unmarkiert. Es gilt aber $d_{k+1} <_{\text{lex}} d[v]$ und wir erhalten einen Widerspruch zur Auswahlregel. Wir haben also gezeigt, dass jedes markierte Label Pareto-optimal ist. Da außerdem gilt, falls p Pareto-optimal ist, sind auch die Anfangsstück von p Pareto-optimal, wird jedes Pareto-Optimum im Algorithmus konstruiert.

Jedes Pareto-Optimum wird konstruiert, da anschaulich begründet, jedes Label, falls es nicht wegdominiert wird, betrachtet wird. Es kann auch kein „Weg“ in G existieren, desen Label nicht erfasst wird, da von jedem Label aus alle ausgehenden Kanten betrachtet werden. □

Wir stellen uns jetzt die Frage, wie groß die maximale Anzahl der Pareto-optimalen Label in einem Knoten werden kann. Dazu nehmen wir an, dass $\lambda_k(a) \in \mathbb{Z}_+$, d.h. die Bewertung einer Kante bzgl. eines Kriterium k ist positiv ganzzahlig, und $L_k(v)$:=maximale Weglänge (elem. Weg) von s nach v bzgl. λ_k. Außerdem gibt uns der Wert pareto(v) die Anzahl der Pareto Optima in v an. Diesen Wert können wir folgendermaßen abschätzen:

$$\text{pareto}(v) \leq L_1(v) \cdot L_2(v) \cdot ... \cdot L_r(v).$$

So grob diese Abschätzung auch ist, können wir trotzdem daraus folgern, dass pareto(v) polynomiell in n ist, falls r konstant ist (z.B. $r = 2$) und $L_k(v)$ polynomiell in n sind. Damit ist dann auch der Algorithmus polynomiell. Gilt z.B. $\lambda_k(a) \leq n$, so erhalten wir $L_k(v) \leq (n-1)n \leq n^2$ und damit pareto$(v) \leq n^{2r}$. In der Praxis ergeben sich polynomielle $L_k(v)$ z.B. bei der Hop Length, die die Anzahl der Kanten angibt. Viele Straßennetze erfüllen die Bedingung, dass die Kantenlänge $\lambda_k(a) \leq L_0$ (z.B. $L_0 = 1.000$ m oder 3min) beschränkt ist. Also ist in diesem Fall pareto$(v) \leq ((n-1)L_0)^r$.

Im Jahr 1983 wenden Aneja, Assarwald und Nair den Pareto Dijkstra auf das CSP

$$\text{CSP} \begin{cases} \min \tau(p) & \\ \ell(p) & \leq L \\ \text{Kantenbewertung} & (\tau_a, \ell_a) \end{cases}$$

an. Jedes Label $d[v]$ eines Knoten v besteht aus dem Tupel (τ, ℓ), τ für die Zeit und l für die Länge. Ein Label $d[v] = (\tau, \ell)$ verwerfen wir, wenn $l > L$ gilt oder es von einem anderen dominiert wird. Der Algorithmus liefert also Pareto-Optima mit $l \leq L$. Aus der praktischen Erfahrung erhält man, dass der Aufwand für zwei

Subgradientenschritte ungefähr dem Aufwand für den Pareto Dijkstra entsprechen. Man kann den Pareto Dijkstra noch ordentlich beschleunigen, indem man die beiden Methoden (Subgradienten und Pareto Dijkstra) verheiratet. Bevor wir dies angehen, betrachten wir noch Approximationen des Problems.

4.3. **Approximation Pareto-optimaler Wege.** Um den Pareto-optimalen Weg eines Problems zu approximieren, gibt es mehrere Ideen und Methoden. Wir werden im Folgenden gewichtete Summen der Kriterien, einfache Skalierung der Längen und den ϵ- Schubfach Dijkstra betrachten.

Als erstes betrachten wir die **gewichtete Summe der Kriterien**, um bzgl. dieser die kürzesten Wege zu berechnen. Dies macht man häufig bei mehreren Zielkriterien. Diese wurde 1984 von Joffe untersucht. Er betrachtet das folgende Zulässigkeitsproblem: Gegeben seien Schranken L_i, $i = 1, ..., r$. Finde Weg p mit $\lambda_i(p) \leq L_i$. Dieses kann man in das folgende Problem transformieren: Berechne den kürzesten Weg bzgl. $\lambda'(a) = \alpha_1\lambda_1(a) + ... + \alpha_r\lambda_r(a)$. Die α_i sind Gewichte, die von den L_i abhängen. Im Allgemeinen liefert dies eine schlechte Approximation. Dies zeigt sich an dem folgenden Beispiel. Sei $L_1 < L_2$ und damit $\alpha_1 < \alpha_2$. Im Beispiel 4.5 würde so $(0, 15)$ berechnet werden, was schlecht für $L_1 = 7$ und $L_2 = 8$ ist.

Allgemein kann man zu den gewichteten Summen der Kriterien sagen, dass

$$\max_{k=1,...,r} \lambda'_k(p) \leq r \cdot \max_{k=1,...,r} L_k$$

gilt. Diese Ungleichung ist scharf.

Die zweite Idee ist die **einfache Skalierung der Längen**. Die Methode hat Teune 1995 untersucht. Dabei geht es darum, den Vektor $\lambda(a) = (\lambda_1(a), ..., \lambda_r(a))^\top$ auf den weniger wichtigen Kriterien zu skalieren und somit eine neue Kantenbewertung $\lambda'(a) \in \mathbb{R}^r$ mit $\lambda'_1(a) = \lambda_1(a)$ und $\lambda'_i(a) = \left\lfloor \frac{\lambda_i(a)}{\alpha} \right\rfloor$ für $i \in \{2, ..., r\}$ mit $\alpha > 1$ zu erhalten. Bei der Skalierung gehen wir davon aus, dass das erste Kriterium das wichtigste ist. Durch die neue Kantenbewertung wird pareto(v) kleiner und die lexikographische Ordnung bleibt erhalten. Wir berechnen also Pareto-optimale Wege bzgl. λ', die dann auch Pareto-optimale Wege bzgl. λ sind.

Example 4.7. Betrachten wir noch mal das vorherige Beispiel. Die neue Bewertung mit $\alpha = 4$ ergibt dann (Abb. 32).

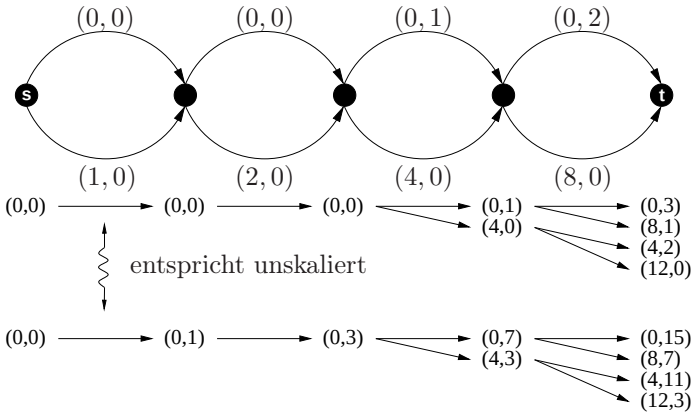


Abbildung 32. Die einfache Skalierung der Längen mit $\alpha = 4$.

Bei der Betrachtung der ersten beiden Kanten erhalten wir nur die Label $(0, 0)$, die alle anderen Werte weg dominieren. Erst ab der Kante (c, d) erhalten wir wieder mehr Label.

Jedem Label am Knoten t können wir genau ein Label aus dem ursprünglichen Problem bzgl. λ zuordnen. Wir erhalten also $Y := \{\text{Pareto-optimale Wege bzgl. } \lambda'\}$, eine Teilmenge der Menge $X := \{\text{Pareto-optimale Wege bzgl. } \lambda\}$.

Jetzt stellt sich die Frage, wie gut diese Teilmenge die gesamt Menge approximiert. Dazu kann man folgendes sagen: zu einem Pareto-optimalen Weg p mit Bewertung $d \in X$ existiert ein Pareto-optimaler Weg p' mit einer Bewertung $d' \in Y$ mit

$$|d_k - d'_k| \leq l(\alpha - 1),$$

wobei l die Anzahl der Kanten von p' ist. Diese Abschätzung ist scharf, jedoch oft zu grob.

Example 4.8. Betrachten wir in unserem Beispiel 4.5 das Label $d = (9, 6)$, dann finden wir bei das Label $d' = (8, 7)$ aus dem Beispiel 4.7, das der Abschätzung genügt. Dem Label $d = (13, 2)$ können wir das Label $d' = (12, 3)$ zuordnen.

Die dritte Idee, ein **voll-polynomiales Approximations Schema** für das CSP zu betrachten, stammt von Hassin aus dem Jahr 1992.

Als erstes nehmen wir an, dass wir den Optimalwert τ^* eines längenbeschränkten s, t Weges kennen. Dann betrachten wir die folgende Idee: Lege in jedem Knoten Schubfächer der Länge $\frac{\tau^*}{(n-1)/\epsilon}$ für Labels an, die das Intervall $[0, (1+\epsilon)\tau^*]$ in bis auf das letzte äquidistante Teilintervalle unterteilen. In das Schubfach i kommen alle Label von v, deren $\tau-$ Wert im Intervall $](i-1)\frac{\tau^*}{(n-1)/\epsilon}, i \cdot \frac{\tau^*}{(n-1)/\epsilon}]$ liegt, wobei n der Anzahl der Knoten im Graphen G entspricht.

Der $\epsilon-$Schubfach Dijkstra funktioniert dann folgendermaßen: Gehe vor wie beim Pareto-Dijkstra. Allerdings werden anstatt aller Label immer nur ein Label pro Knoten pro Schubfach gespeichert und weiterverfolgt. Genauer: sei $d[v]$, das Label, das markiert ist. Berechne alle „neuen“ Label $d[w]$ mit $(v, w) \in V$, wobei ein Label mit $l > L$ sofort gelöscht wird. Die restlichen Label werden in die Schubfächer i aufgeteilt. Entstehen für ein Schubfach mehrere Label wird das beste Label bzgl. l gespeichert und sein τ- Wert auf die obere Grenze des Schubfaches gerundet. Damit ist ausgehend von $d[v]$ pro Schubfach nur noch ein Label zu betrachten. Wähle am Ende des Algorithmus den Weg mit dem kleinsten $\tau-$Wert bei t.

Achtung: Der Algorithmus kann nicht abgebrochen werden, sobald der ein Label den Zielknoten erreicht hat, auch wenn die Lexikographische Regel angewandt wird.

Example 4.9. Am folgenden Graphen (Abb. 33) sieht man leicht, dass der Algorithmus nicht abgebrochen werden darf, bevor nicht alle Label bearbeitet wurden.

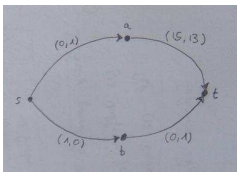


Abbildung 33. Der Algorithmus liefert keinen pareto-optimalen Weg, falls wir nach erreichen des Zielknotens den Algorithmus beenden.

Der Pareto- Dijkstra beginnt an der Quelle s und generiert ein Label $d[a] = (0, 1)$ am Knoten a und ein Label $d[b] = (1, 0)$ am Knoten b. Da $d[a]$ lexikographisch kleiner als $d[b]$ ist, wird als nächstes ein Label $d[t] = (15, 14)$ für den Zielknoten t erstellt. Der paretro optimale Weg hätte aber die Länge $(1, 1)$, indem er von s über b nach t geht. Das Label für diesen Weg wird aber erst nach dem Label $d[t]$ erstellt.

Zur Verdeutlichung des $\epsilon-$Schubfach Dijkstra wenden wir diesen noch mal auf das Beispiel 4.5 an.

Example 4.10. Betrachten wir noch erneut das Beispiel 4.5 bzgl. des ϵ- Schubfach Dijkstra. Wähle $L = 7$ als Längenschranke für das zweite Kriterium. Der Optimalwert für $L = 7$ ist $\tau^* = 8$, was wir aus dem vorherigen Beispiel wissen. Wähle $\epsilon = 1$, dann ergibt sich mit $n = 5$ die Intervalllänge $\frac{\tau^*}{(n-1)/\epsilon} = \frac{8}{4 \cdot 1} = 2$. Das Gesamtintervall ist dann $[0, 16]$, das wir in 8 Schubfächer der Länge 2 unterteilen. Mit der Anwendung des Schubfach-Dijkstras erhalten wir also den folgenden Ablauf (Abb. 34).

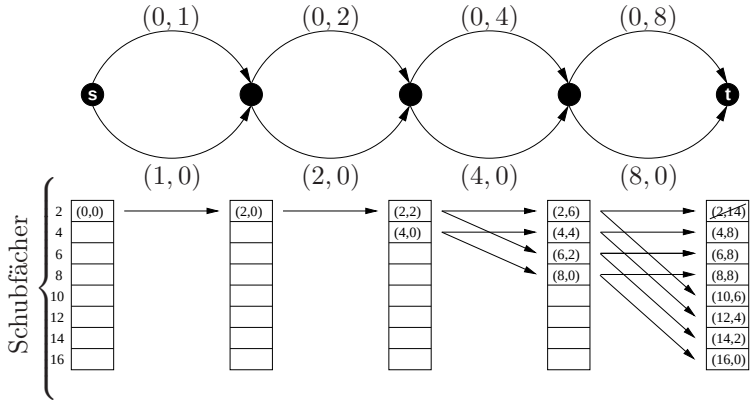


Abbildung 34. Schubfach-Dijkstra.

Lemma 4.11. *Der vom ϵ- Schubfach Dijkstra gefundene Weg ist längenbeschränkt und hat maximal einen um den Faktor $(1 + \epsilon)$ höheren $\tau-$Wert als der kürzeste längenbeschränkte Weg.*

Beweis. Sei p_ϵ der gefundene Weg. Durch die Konstruktion des ϵ- Schubfach Dijkstras ist klar, das der Weg p_ϵ längenbeschränkt ist.

Außerdem ist p_ϵ elementar, weil der Dijkstra - Algorithmus nur elementare Wege findet. Er enthält also höchstens $n - 1$ Kanten. Pro Kante macht man einen Rundungsfehler im τ-Wert, der höchstens der Schubfachlänge $(= \frac{\tau^*}{(n-1)/\epsilon})$ beträgt. Der Rundungsfehler ist Additiv, weswegen der Gesamtfehler höchstens $(n-1)\frac{\tau^*}{(n-1)/\epsilon} = \tau^*\epsilon$ beträgt. Die Länge des optimalen Weges $\tau(p_\epsilon)$ im $\epsilon-$Schubfachdijkstra kann dann höchsten die Länge des optimalen Weges, berechnet durch den normalen Dijkstra, und des Gesamtfehlers betragen. Wir erhalten also

$$\tau(p_\epsilon) \leq \tau^* + \epsilon\tau^* = (1 + \epsilon)\tau^*.$$

Daraus folgt die Behauptung. □

Das Problem bei dem Ansatz ist, dass wir τ^* nicht kennen. Also versuchen wir τ^* mit Hilfe von binärer Suche zu approximieren. Dazu berechnen wir Schranken von τ^*. Eine grobe Abschätzung wäre $\text{UB} = (n-1) \cdot \tau_{\max}$ und $\text{LB} = \min_p \tau(p)$. Beginne mit $\tau = \left\lfloor \frac{\text{UB}-\text{LB}}{2} \right\rfloor$ im ϵ- Schubfach Algorithmus. Falls kein längenbeschränkter Weg für aktuelles τ gefunden wir, dann suchen wir „rechts“ weiter. Falls ein längenbeschränkter Weg gefunden wir mit $\tau-$Wert kleiner als τ wählen wir ein neuen τ Wert „links“ vom alten. Insgesamt erhalten wir maximal $\log((n-1)\tau_{\max})$ viele Aufrufe des ϵ- Schubfach Dijkstra. Die Laufzeit des $\epsilon-$Schubfach Dijkstras ist polynomial in $\frac{1}{\epsilon}$ und n, da die Anzahl der Schubfächer $\frac{(1+\epsilon)\tau}{\frac{\tau}{(n-1)/\epsilon}} = \frac{(1+\epsilon)(n-1)}{\epsilon}$ beträgt. Sobald ein Label markiert wurde, wird wegen der lexikographischen Auswahlregel das dazugehörige Schubfach nicht mehr verändert. Ein Schubfach kann durch

höchstens m Berechnungen bis zu seiner Markierung verändert werden. Insgesamt erhalten wir also einen in n und $\frac{1}{\epsilon}$ polynomialen Algorithmus.

4.4. **Die 2-Schritt Methode von Mehlhorn und Ziegelmann.** Diese Methode benutzt eine wegbasierte IP-Formulierung und die LP Relaxation zur Erzeugung von einer oberen und einer unteren Schranke für die τ- Werte.

Um die wegbasierte IP-Formulierung des Problems zu betrachten, führen wir $0,1$- Variablen x_p für jeden s,t- Weg und $\tau_p = \sum_{a\in p} \tau_a$ und $l_p = \sum_{a\in p} l_a$ ein. Das Primale IP hat dann die Form:

$$(P)\ \begin{cases} \min \sum_p \tau_p x_p & \\ \sum_p l_p x_p & \leq L \\ \sum_p x_p & = 1 \\ x_p & \in \{0,1\}. \end{cases}$$

Um das Programm richtig zu verstehen, muss man es von hinten nach vorne lesen. Die letzten beiden Nebenbedingungen $\sum_p x_p = 1$ und $x_p \in \{0,1\}$ garantieren, dass genau ein Weg ausgesucht wird. Die Nebenbedingung $\sum_p l_p x_p \leq L$ stellt sicher, dass die Längenschranke vom ausgesuchten Wege eingehalten wird. Durch die Zielfunktion $\sum \tau_p x_p$ wird dann derjenige Weg ausgesucht, dessen τ- Wert minimal ist. D.h. das Programm liefert ein optimalen Weg für unser CSP Problem.

Die LP Relaxation erhalten wir mit $x_p \geq 0$:

$$(\mathrm{LP}_{\mathrm{relax}})\ \begin{cases} \min \sum \tau_p x_p & \\ -\sum l_p x_p & \geq -L \\ \sum x_p & = 1 \\ x_p & \geq 0. \end{cases}$$

Wir brauchen die Variablen x_p nicht nach oben beschränken, da dieses schon in der Bedingung $\sum x_p = 1$ enthalten ist. Die LP Relaxation gibt eine untere Schranke des IP's an und somit eine untere Schranke des CSP.

Betrachten wir das duale Programm (D). Da wir nur zwei Nebenbedingungen im primalen Programm haben, hat das duale Programm nur zwei Variablen, die wir später geometrisch interpretieren werden. Sei v die Variable, die zur Nebenbedingung $-\sum l_p x_p \geq -L$, und u diejenige, die zur Nebenbedingung $\sum x_p = 1$ gehört. Die Formulierung des dualen Programms ist:

$$(D)\ \begin{cases} \max u - Lv & \\ u - l_p v & \leq \tau_p\ \forall p \\ v \geq 0, & u \text{ beliebig.} \end{cases}$$

Ersetzten wir v durch $-v$, dann erhalten wir

$$\begin{aligned} &\max u + Lv \\ &\quad u + l_p v \leq \tau_p\ \forall p \\ &\quad v \leq 0,\ u \text{ beliebig.} \end{aligned}$$

Betrachten wir nun die geometrische Standartinterpretation des dualen Programms. Danach definieren die Werte u und v in einem Koordinatensystem und jede Nebenbedingung definiert eine Halbebene an zulässigen $(u,v)-$Punkten. Als Schnitt aller Halbebenen ergibt sich der zulässige Bereich des Problems (Abb. 35).

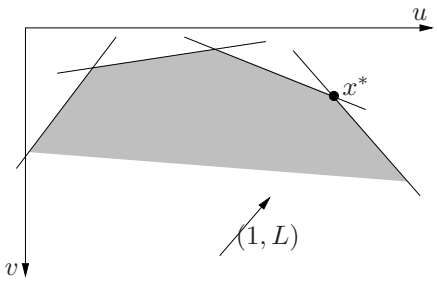


ABBILDUNG 35. Geometrische Standartinterpretation des Dualen Programms. Der Punkt x^* ist der optimale Punkt von (D) bzgl. der Richtung $(1, L)$, der Zielfunktion.

Die Zielfunktion stellt ebenfalls als Gerade dar, die parallel verschoben wird, bis sie ihren Maximalwert an einer Ecke des zulässigen Bereiches annimmt. Diese Interpretation betrachten wir allerdings nicht weiter, sondern benutzten eine geometrisch duale Herangehenesweise.

Geometrisch dual bedeutet, dass man die Rolle von Gerade und Punkt vertauscht. Ein Punkt (u, v) wird jetzt zu einer Geraden $\tau = v \cdot l + u$ in der (l, τ)-Ebene. Eine Nebenbedingung $u + l_p v \leq \tau_p$, der ein Pfad p zugrunde liegt, entspricht jetzt einem Punkt (l_p, τ_p). Die Ungleichung $\overline{u} + l_p \overline{v} \leq \tau_p$ ist jetzt genau dann für die Werte $\overline{u}, \overline{v}$ erfüllt, wenn (l_p, τ_p) oberhalb der Gerade $\tau = \overline{v} l + \overline{u}$ liegt, d.h. $(l_p, \tau) \leq (l_p, \tau_p)$.

Example 4.12. Das folgende Beispiel veranschaulicht die letzte Behauptung.

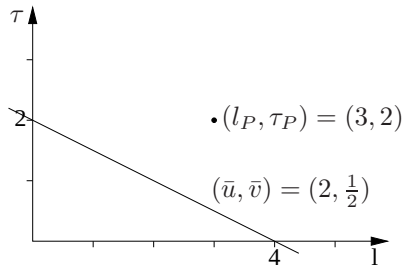


ABBILDUNG 36. Der Punkt $(3, 2)$ liegt oberhalb von $\tau = -\frac{1}{2} l + 2$.

Sei durch den Weg p die Nebenbedingung $u + l_p v \leq \tau_p$ mit $l_p = 3$ und $\tau_p = 2$ gegeben. Betracht die Werte $\overline{u} = 2$ und $\overline{v} = -\frac{1}{2}$. Sie erfüllen die Nebenbedingung, da $2 + 3 \cdot (-\frac{1}{2}) = 0.5 \leq 2$ gilt.

In der geometrisch dualen Interpretation erhalten wir für $\overline{u}$ und $\overline{v}$ die Geradengleichung $\tau = \overline{v} l + \overline{u} = -\frac{1}{2} l + 2$. Der Punkt (l_p, τ_p) liegt dann oberhalb der Geraden τ.

Die Optimierung, also die Lösung des dualen LP's, in dieser Interpretation entspricht dann der Suche eines Paares u^*, v^* bzw. einer Geraden $\tau = v^* l + u^*$ mit nicht- positiver Steigung, da $v^* \leq 0$ sein musste, so dass folgende zwei Dinge erfüllt sind:

1. alle Punkte (l_p, τ_p), wobei p ein s, t- Weg ist, liegen oberhalb von dieser gesuchten Geraden und
2. der Schnittpunkt der gesuchten Geraden mit der Geraden $l = L$ ist so groß wie möglich (Abb. 37).

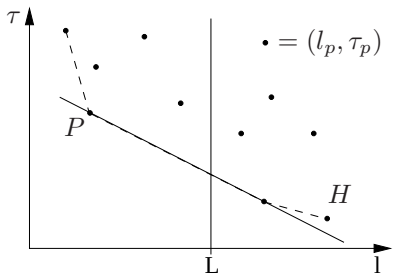


ABBILDUNG 37. Die Punkte (l_p, τ_p) liegen oberhalb der gesuchten Gerade P, deren Schnittpunkt mit $l = L$ maximal ist. Die Gerade H stellt die untere Einhüllende der Wege dar.

Aus der folgenden Betrachtung geht hervor, dass eine solche Gerade genau der optimalen Lösung des dualen Programms entspricht. Sei $\tau = v^* l + u^*$ die gesuchte Gerade. Setzten wir hier $l = L$ ein, so erhalten wir unsere duale Zielfunktion, die wir maximieren wollen. Also muss der Schnittpunkt von $\tau = v^* l + u^*$ und $l = L$ möglichst hoch liegen. Dadurch, dass alle Punkte (l_p, τ_p) oberhalb der Geraden liegen, erfüllen v^* und u^* auch alle Nebenbedingungen des dualen Programms.

Bisher haben wir das LP nur interpretiert. Jetzt wollen wir die optimale Gerade konstruieren. Die Konstruktion dieser Gerade erfolgt durch den „Hüllenansatz“, d.h. Konstruktion der unteren Einhüllenden bei $l = L$.

Im ersten Schritt berechnen wir zwei Wege. Einerseits $P_{l_{\min}} =: P_l$, den kürzesten Weg bzgl. l, sowie $P_{\tau_{\min}} =: P_\tau$, den kürzesten Weg bzgl. τ. Gilt $l(P_l) > L$, dann gibt es keine zulässige Lösung, da kein Weg P die Bedingung $l(P) \leq L$ erfüllt. Gilt hingegen $l(P_\tau) \leq L$, dann haben wir mit der Gerade $\tau = \tau(P_\tau)$ die optimale Lösung gefunden, da $\overline{v} \leq 0$ gelten muss (Abb. 38).

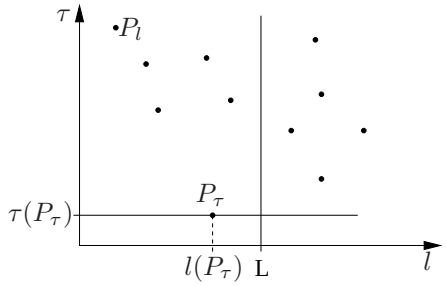


ABBILDUNG 38. Falls $l(P_\tau) \leq L$, dann ist die Gerade mit $\overline{u} = \tau(P_\tau)$ und $\overline{v} = 0$ optimal.

Tragen wir $(l(P_l), \tau(P_l))$ und $(l(P_\tau), \tau(P_\tau))$ als zwei Punkte in das Koordinatensystem, dann erzeugen diese eine erste Gerade $\tau = \overline{v} \cdot l + \overline{u}$ mit der Steigung $\overline{v} = \frac{\tau(P_\tau) - \tau(P_l)}{l(P_\tau) - l(P_l)}$. Damit ist die Initialisierung des Algorithmus abgeschlossen (Abb. 39).

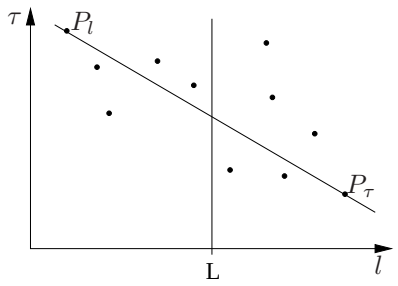


ABBILDUNG 39. Initialisierung des Algorithmus.

Im zweiten Schritt müssen wir prüfen, ob es Punkte unterhalb der Geraden liegen, da in diesem Fall die Gerade noch nicht zulässig bzw. optimal ist. Ein solcher Punkt entspricht einem Weg P mit

$$\overline{v}l_p + \overline{u} > \tau_p \Leftrightarrow \tau_p - \overline{v}l_p < \overline{u}$$

und kann durch kürzeste Wege- Berechnung bzgl. $\tau_a - \overline{v}l_a =: \overline{\tau}_a$ berechnet werden. Es gilt $\overline{\tau}_a \geq 0$, da $\overline{v} \leq 0$ ist.

Graphisch entspricht dies einer Parallelverschiebung der Geraden auf einen extremalen Punkt, der in der graphischen Darstellung dem kürzesten Weg bzgl. $\overline{\tau}_a$ entspricht (Abb. 40).

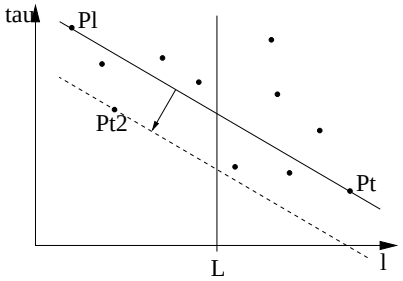


ABBILDUNG 40. Durch die Verschiebung der Geraden erhalten wir den kürzesten Weg $P_{\overline{\tau}_{\min}}$ bzgl. $\overline{\tau}_a$.

Sei nun $P_{\overline{\tau}_{\min}}$ der neue Weg/Punkt bzgl. $\overline{\tau}_a = \tau_a - \overline{v}l_a$. Falls $\tau(P_{\overline{\tau}_{\min}}) = \overline{u}$ gilt, dann liegt $P_{\overline{\tau}_{\min}}$ auf der alten Geraden. Damit ist die alte Gerade optimal und der Schnittpunkt mit $l = L$ ist das Optimum. Ist dies nicht der Fall geht es mit Schritt drei weiter.

Im dritten Schritt aktualisieren wir die alte Gerade durch eine der beiden Geraden durch P_l und $P_{\overline{\tau}_{\min}}$ oder $P_{\overline{\tau}_{\min}}$ und P_τ. Dabei stellt sich die Frage, welche der beiden Geraden besser ist, d.h. welche hat einen höheren Schnittpunkt mit L? (Abb. 41) Dies hängt offenbar mit der Lage des Punktes $P_{\overline{\tau}_{\min}}$ ab. Liegt er links von L, d.h. $\tau(P_{\overline{\tau}_{\min}}) \leq L$, dann ist die Gerade $\overline{P_{\overline{\tau}_{\min}} P_\tau}$ besser, gilt hingegen $\tau(P_{\overline{\tau}_{\min}}) > L$, der Weg liegt also rechts von L, dann sollten wir mit der Gerade $\overline{P_l P_{\overline{\tau}_{\min}}}$ weiter machen. Haben wir eine der beiden Geraden ausgewählt, so machen wir mit Schritt zwei und der neuen Gerade weiter.

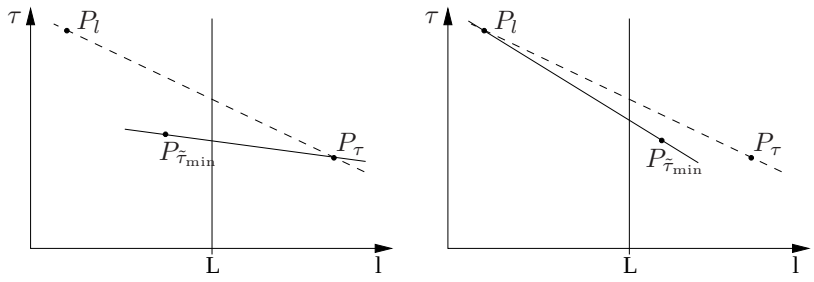


ABBILDUNG 41. Aktualisierung der zulässigen Gerade.

Formaler Dargestellt hat der Algorithmus von Mehlhorn und Ziegelmann (Alg. 3) dann die folgende Form.

Algorithm 3 Der Hüllenansatz-Algorithmus von Mehlhorn & Ziegelmann

Initialisierung:
1. Schritt:
Berechne kürzesten Weg $P_{l_{\min}}$ bzgl. l
if $l(P_{l_{\min}}) > L$ **then**
  return „keine zulässige Lösung“.
else
  UB := $\tau(P_{l_{\min}})$; { obere Schranke für optimalen Weg }
end if
Berechne kürzesten Weg $P_{\tau_{\min}}$ bzgl. τ
if $l(P_{\tau_{\min}})$ **then**
  return $P_{\tau_{\min}}$ { optimal}
else
  $P_\tau := P_{\tau_{\min}}$; $P_l := P_{l_{\min}}${ Initialisierung}
end if

Hauptschleife:
2. Schritt:
Berechne kürzesten Weg $P_{\overline{\tau}_{\min}}$ bzgl. $\tau_a - \overline{v}l_a =: \overline{\tau}_a$
if $\tau(P_{\overline{\tau}_{\min}}) = \overline{u}$ **then**
  return P_l und P_τ { optimal }
end if

3. Schritt:
if $\tau(P_{\overline{\tau}_{\min}}) \leq L$ **then**
  $P_l := P_{\overline{\tau}_{\min}}$ und UB := $\tau(P_{\overline{\tau}_{\min}})$;
else
  $P_\tau := P_{\overline{\tau}_{\min}}$und UB := $\tau(P_{\overline{\tau}_{\min}})$;
end if
Gehe zu Schritt 2

Der Algorithmus liefert uns im Fall, dass $l(P_{\tau_{\min}}) \leq L$ gilt, den Weg $P_{\tau_{\min}}$ aus dem wir mit $\tau(P_{\tau_{\min}})$ eine untere Schranke LB für den Optimalwert des CSP-Problems erhalten. Im anderen Fall erhalten wir zwei Wege P_l und P_τ mit Hilfe derer wir eine untere Schranke berechnen können. Dazu müssen wir die Gerade, die durch die Punkte $(l(P_l), \tau(P_l))$ und $(l(P_\tau), \tau(P_\tau))$ an der Stelle $x = L$ auswerten. Die Gerade τ hat die Form $\tau = \overline{v}(x - l(P_\tau) + \tau(P_\tau)$ mit $\overline{v} = \frac{\tau(P_\tau) - \tau(P_l)}{l(P_\tau) - l(P_l)}$. Also ergibt

sich für die untere Schranke LB

$$\mathrm{LB} = \overline{v}(L - l(P_\tau)) + \tau(P_\tau).$$

Die untere Schranke LB, die wir dadurch erhalten entspricht dem optimal Wert der Lagrange-Relaxation, da in der Lagrange-Relaxation der zulässige Bereich ganzzahlig ist. Also hat auch die LP Relaxation den gleichen Wert wie die Lagrange-Relaxation. Insgesamt bleibt also dieselbe „Dualitätslücke“. Allerdings kann die Laufzeit besser abgeschätzt werden.

Betrachten wir nun den Abbruch und die Anzahl der Schritte.

Theorem 4.13. *[Mehlhorn u. Ziegelmann 00] Der Hüllenansatz arbeitet korrekt und hat eine Laufzeit von* $\mathcal{O}(\log(n \cdot \tau_{\max} \cdot \ell_{\max}) \cdot (n \log n + m))$ *mit* $\tau_{\max} = \max_a \tau_a$ *und* $\ell_{\max} = \max_a \ell_a$.

Beweis. Die Korrektheit des Algorithmus ergibt sich aus den vorausgegangenen Überlegungen, falls er terminiert. Betrachten wir also die Laufzeit und zeigen dabei, dass der Algorithmus terminieren muss.

Der Ansatz der Laufzeitbetrachtung geht über die Größe der „unerforschten“ Gebiete von noch möglichen Punkten unterhalb der momentanen Gerade. Betrachten wir dazu die erste Iteration. (Abb. 42).

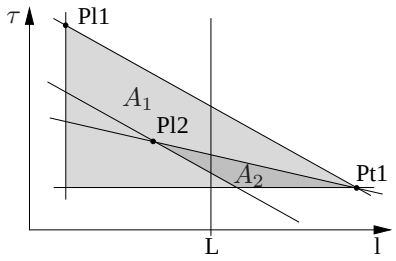


Abbildung 42. Ersten Iteration bezüglich der „unerforschten“ Gebiete.

Zu Beginn können unentdeckte Punkte in Dreieck A_1 liegen. Nach der Parallelverschiebung der Geraden $\overline{P_l^1 P_\tau^l}$ zum Punkt P_l^2 können unentdeckte Punkte nur noch im Dreieck A_2 liegen. In der 2. Iteration wird jetzt P_τ^3 gefunden und die Fläche verkleinert sich weiter (Abb. 43)

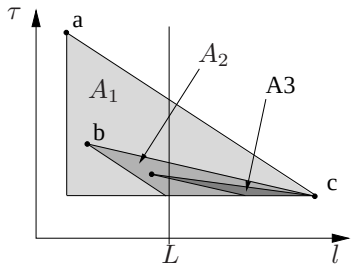


Abbildung 43. Zweite Iteration bezüglich der „unerforschten Gebiete“.

Die maximale Fläche eines Dreiecks bzw. von A_1 kann $\frac{1}{2}(n-1)\tau_{\max}(n-1)l_{\max}$ betragen. Der Flächeninhalt des kleinst möglichen Dreiecks beträgt dann 0.5, da alle Daten bzw. Punkte ganzzahlige sind.Wenn der Algorithmus nicht abbricht, wird die Fläche in jeder Iteration verringert. Genauer können wir zeigen, dass zwischen zwei Iterationen i und $i+1$ des Hüllenansatzes $A_{i+1} \leq \frac{1}{4}A_i$ gilt.

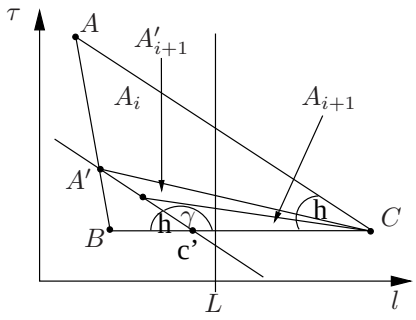


ABBILDUNG 44. Bei jeder Iteration gilt $A_{i+1} \leq \frac{1}{4}A_i$.

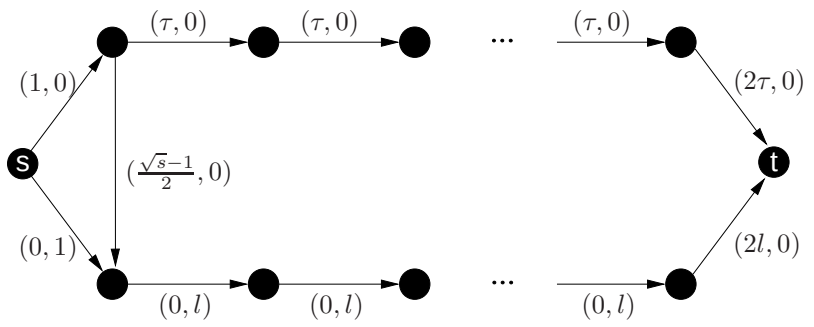
Sei A_i das Dreieck ABC (Abb. 44). Bei der Parallelverschiebung ergibt sich das Dreieck A_{i+1}. Im folgenden betrachten wir allerdings das Dreieck $A'C'C$, das durch die Verschiebung entlang der Parallelen entsteht und dessen Flächenhinhalt größer als das ursprüngliche ist. Aus den Strahlensätzten folgt $\frac{CA'}{CA} = \frac{CC'}{CB}$ und $\frac{CA'}{CA} = \frac{A'C'}{AB}$. Wir definieren $x := \frac{CA'}{CA} < 1$. Also gilt $C'B = (1-x)CB$ und $A'C' = xAB$. Betrachten wir nun den Winkel β. Dann erhalten wir $\gamma = \pi - \beta$ und es gilt $|\sin\gamma| = |\sin\beta|$. Den Flächeninhalt von A_i erhalten wir durch $A_i = \frac{1}{2}AB \cdot CB \cdot \sin\beta$ und den von $A_{i+1} = \frac{1}{2}A'C' \cdot C'B \sin\gamma = \frac{1}{2}x \cdot (1-x) \cdot AB \cdot CB \cdot \sin\beta$. Also erhalten wir $A_{i+1} = x(1-x)A_i$. Mit $x = \frac{1}{2}$ erhalten wir das Maximum und somit $A_{i+1} \leq \frac{1}{4}A_i$.

Aus $A_{i+1} \leq \frac{1}{4}A_i$ folgt dann, dass maximal N Iteration gebraucht werden, bis $\frac{1}{2} \geq (\frac{1}{4})^N \frac{1}{2}n^2\tau_{\max}l_{\max}$ gilt. Also erhalten wir $\mathcal{O}(\log(nl_{\max}\tau_{\max}))$ Iterationen in denen jeweils ein kürzester Weg berechnet wird. Zusammen ergibt sich dann eine Laufzeit von $\mathcal{O}(\log(n\ell_{\max}\tau_{\max}) \cdot (n\log n + m))$. □

Note 4.14. Man kann auch zeigen, dass $\mathcal{O}(m\log^2 m)$ Iterationen reichen, womit wir einen streng polynomieller Algorithmus hätten (Jüttner 03).

Insgesamt bestimmt der Hüllenansatz eine obere Schranke UB und eine untere Schranke LB für das Optimum des CSP. Die Güte der Schranken kann allerdings i.A. beliebig schlecht werden.

Example 4.15. Betrachte das folgende Beispiel zur Güte der Schranken. (Abb. 45)



$s = 1 + \frac{n\tau}{2}$, $n = \#$ Knoten ≥ 4 gerade , $L = \frac{nl}{2}$, $l \geq \tau > 0$

ABBILDUNG 45. Beispiel für schlechte Schranken durch den Hüllenansatz.

Dabei ist $s := 1 + \frac{n}{2}\tau_0$ mit n der Anzahl der Knoten und n sei gerade und größer als 4. Außerdem gelte $L = \frac{nl_0}{2}$ mit $l_0 > \tau_0 > 0$. Der Hüllenansatz findet UB = s

und LB $= \frac{s}{T}$ mit $T := 1 + \frac{n}{2} l_0$. Die optimalen Kosten liegen bei $\frac{\sqrt{s}+1}{2}$. Wir erhalten also nur eine $\Omega(\sqrt{n\tau_0})$ Approximationsgüte.

Das Schließen der Lücke zwischen LB und UB geht z.B. durch den Pareto Dijkstra, wobei Label (τ, l) verworfen werden, wenn $\tau \geq$ UB, $\tau <$ LB oder $l > L$ ist. Damit erhält man für den Pareto Dijkstra durch das hinzufügen der beiden Grenzen eine wesentliche Beschleunigung, kleiner Labellisten, und auch bzgl. Gesamtlaufzeit gehört es mit zu den besten Verfahren.

4.5. **Anwendung auf das eingeschränkte Systemoptimum.** Das eingeschränkte Systemoptimum Problem (ESP) hatte folgende Formulierung:

$$\text{(ESP)} \quad \begin{cases} \min \sum_a \tau_a(x_a(f))x_a(f) & \\ \sum_{p \in \mathcal{P}_k^\epsilon} f_p & = d_k \\ f_p & \geq 0. \end{cases}$$

In Kaptiel 3.3 hatten wir gesehen, dass wir dieses mit Hilfe des Frank-Wolfe Algorithmus lösen konnten, wobei die Berechnung einer Abstiegrichtung uns auf Problem

$$\begin{aligned} \min c_a'(x_a^\ell)x_a & \\ \sum_{p \in \mathcal{P}_k^\epsilon} f_p &= d_k \\ f_p &\geq 0 \end{aligned}$$

geführt hat. Setzten wir $\tau_a := c_a'(x_a^\ell)$ und $\ell_a := c_a'(x_a^\ell)$, so können wir dieses durch die Berechung von k CSP Problemen lösen.

Weitere Ergebnisse sind im Paper „System-Optimal Routing of Traffic Flows with User Constraints in Networks with Congestion“ von Jahn, Möhring, Schulz und Stier-Moses nachzulesen. Die nachfolgenden Abbildungen stellen zwei in dem Paper ermittelte Ergebnisse dar.

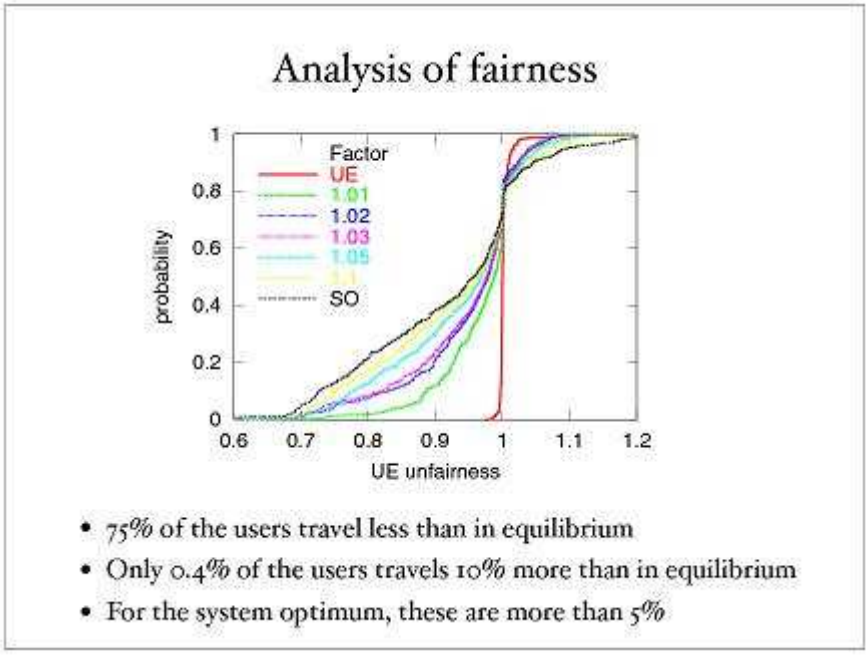


Abbildung 46. Betrachtung der Fairness.

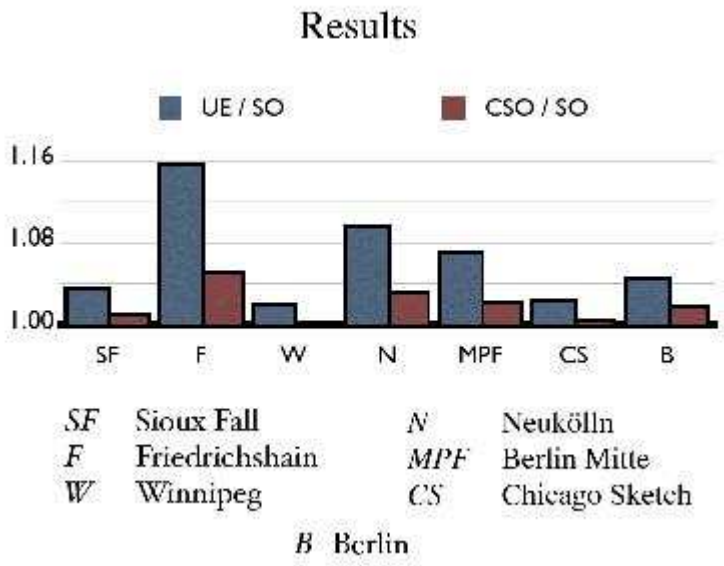


Abbildung 47. Vergleich von UE/SO und CSO/SO.