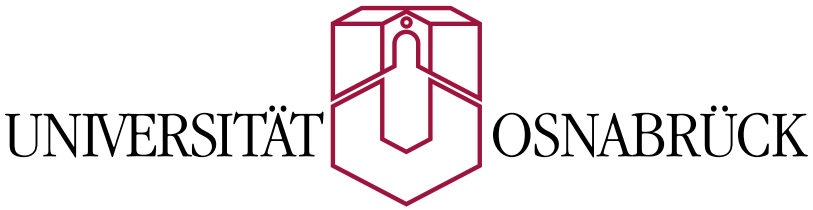
FPTAS für das Restricted Shortest Path-Problem

Rasmus Diederichsen Sebastian Höffner

Universität Osnabrück

6. Dezember 2015



Inhalt

1. Das Problem
2. Exakte Lösung
   - Algorithmus
   - Laufzeit
   - Terminierung
   - Beispiel
3. Das FPTAS
   - Test für Grenzen von OPT
   - Laufzeit

Problemstellung

Gegeben

- azyklischer Graph $G = (V, E)$
- $(u, v) \in E$ hat Gewicht c und Verzögerung t

Single Source Shortest Path

Berechne vom Startknoten aus alle nach Kosten kürzesten Wege zu allen anderen ▸ Dijkstra

All Pairs Shortest Path

Kürzeste Wege zwischen allen Knotenpaaren ▸ Floyd

Das Problem

Gegeben

- azyklischer Graph $G = (V, E)$
- $(u, v) \in E$ hat gewicht c und Verzögerung t

Restricted Shortest Path

Finde nach Kosten kürzesten Weg von a nach b mit Verzögerung $\leq T$. **NP**-schwer.

Exakte Lösung

Algorithmus

Dynamische Programmierung (ähnlich wie Knapsack). Kanten (i, j) mit $i < j$, da azyklisch.

Algorithmus

$$g_1(c) = 0, \text{ Für } c = 0, \ldots, OPT,$$
$$g_j(0) = \infty, \text{ Für } j = 2, \ldots, n,$$
$$g_j(c) = \min \left\{ g_j(c-1), \min_{k | c_{kj} \leq c} \{ g_k(c - c_{kj}) + t_{kj} \} \right\}$$
$$\text{Für } j = 2, \ldots, n; \; c = 1, \ldots, OPT$$

Exakte Lösung

Laufzeit

$$g_1(c) = 0, \text{ Für } c = 0, \ldots, OPT,$$
$$g_j(0) = \infty, \text{ Für } j = 2, \ldots, n,$$
$$g_j(c) = \min \left\{ g_j(c-1), \min_{k | c_{kj} \leq c} \{ g_k(c - c_{kj}) + t_{kj} \} \right\}$$
$$\text{Für } j = 2, \ldots, n; \; c = 1, \ldots, OPT$$

- $\mathcal{O}(OPT \cdot n \cdot \text{Aufwand pro } (c, j))$
  - Pro (c, j) evtl. alle Vorgänger betrachten
  - $\mathcal{O}\left(n^2 OPT\right) = \mathcal{O}\left(|E| OPT\right)$
- Pseudopolynomiell

Exakte Lösung

Terminierung

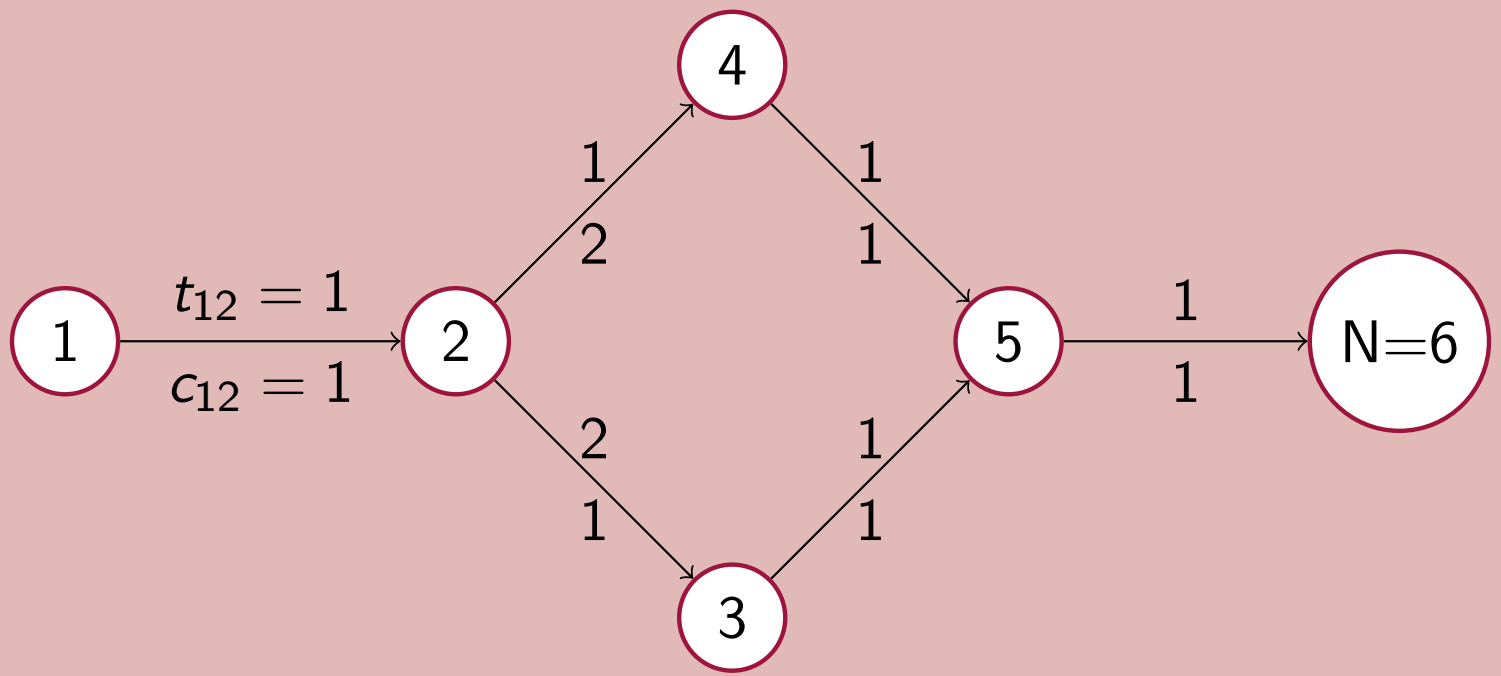
Man weiß $OPT = \min\{c \mid g_n(c) \leq T\}$

- Setze OPT, sobald erstes c mit $g_n(c) \leq T$ gefunden.

Exakte Lösung

Beispiel

Autobahn? Oder doch lieber Landstraße?



Exakte Lösung

Beispiel

$j \backslash c$	0	1	2	3	4	5
1	0	0	0	0	0	0
2	∞	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	∞	∞
6	∞	∞	∞	∞	∞	∞

Exakte Lösung

Beispiel

$j \backslash c$	0	1	2	3	4	5
1	0	0	0	0	0	0
2	∞	1	∞	∞	∞	∞
3	∞	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	∞	∞
6	∞	∞	∞	∞	∞	∞

$$g_2(1) = \min\left\{g_2(0), \min_{k|c_{kj}\leq c}\left\{g_k\left(c - c_{kj}\right) + t_{kj}\right\}\right\}$$

$$g_2(1) = \min\left\{\infty, \min\left\{g_1\left(1-1\right)+1\right\}\right\}$$

$$g_2(1) = 1$$

Exakte Lösung

Beispiel

$j \backslash c$	0	1	2	3	4	5
1	0	0	0	0	0	0
2	∞	1	1	1	1	1
3	∞	∞	3	3	3	3
4	∞	∞	∞	2	2	2
5	∞	∞	∞	4	3	3
6	∞	∞	∞	∞	5	**4**

Das FPTAS

Test für Grenzen von *OPT*

Wir suchen zunächst ein Verfahren, dass untere und obere Schranken für OPT findet.

- Wünsch-dir-was: Polynomieller Algorithmus $TEST(k)$, sodass

$$TEST_{magic}(k) = \begin{cases} 1 & \text{falls } OPT \geq k \\ 0 & \text{falls } OPT < k \end{cases}$$

  - Binäre Suche auf $0, \ldots, UB$
  - Leider **NP**-schwer

Das FPTAS

Test für Grenzen von *OPT*

$TEST_{magic}(k)$ kann nicht existieren, also schwächer:

Eigenschaften von $TEST(k)$

$$TEST(k) = \begin{cases} 1 & \text{falls } OPT \geq k \\ 0 & \text{falls } OPT < k(1+\epsilon) \end{cases}$$

Das FPTAS

Test für Grenzen von OPT

$TEST_{magic}(k)$ kann nicht existieren, also schwächer:

$TEST(k)$

- Skaliere und runde Kantengewichte als $\hat{c}_{ij} = \left\lfloor \frac{c_{ij}(n-1)}{k\epsilon} \right\rfloor$
- Wende exakten Algorithmus an, bis $g_n(c) \leq T$ gefunden ist oder $c \geq \frac{n-1}{\epsilon}$.

Das FPTAS

Test für Grenzen von *OPT*

$TEST(k)$ erfüllt seinen Zweck:

$c < \frac{n-1}{\epsilon}$

$$k \leq k$$

$$\frac{k\epsilon}{n-1}\frac{n-1}{\epsilon} \leq k$$

Durch Einsetzen folgt:

$$\frac{k\epsilon}{n-1}c < k$$

$$\frac{k\epsilon}{n-1}c + k\epsilon < k + k\epsilon$$

$$\frac{k\epsilon}{n-1}c + k\epsilon < k(1+\epsilon)$$

Das FPTAS

Test für Grenzen von *OPT*

TEST(k) erfüllt seinen Zweck:

$c \geq \frac{n-1}{\epsilon}$

$$c \geq \frac{n-1}{\epsilon}$$

$$c\frac{k\epsilon}{n-1} \geq \frac{k\epsilon}{n-1}\frac{n-1}{\epsilon}$$

$$c\frac{k\epsilon}{n-1} \geq k$$

Das FPTAS

Test für Grenzen von *OPT*

Test-Algorithmus

```
1  Setze c ← 0
2  Für alle (i,j) ∈ E:
3      Falls c_ij > k, entferne (i,j)
4      Sonst c_ij ← ⌊c_ij(n−1)/kϵ⌋
5
6  Falls c ≥ (n−1)/ϵ, return true
7  Sonst:
8      Wende Algorithmus B an und berechne g_j(c) für
           j = 2,...,n
9      Falls g_n(c) ≤ T, return false
10     Sonst:
11         Setze c ← c+1
12         Goto Zeile 6
```

Das FPTAS

Laufzeit von *TEST*

- Runden: in $\mathcal{O}(\log n)$ durch binäre Suche, falls nach oben beschränkt ▸ $\mathcal{O}\left(|E| \log \frac{n-1}{\epsilon}\right)$
- Exakter Algorithmus führt $\leq \frac{n-1}{\epsilon}$ Iterationen durch, $\mathcal{O}\left(|E|\right)$ pro Iteration
  - Insgesamt $\mathcal{O}\left(|E| \log \frac{n-1}{\epsilon} + |E| \frac{n-1}{\epsilon}\right) = \mathcal{O}\left(|E| \frac{n-1}{\epsilon}\right)$