FPTAS für das *Restricted Shortest Path*-Problem

Rasmus Diederichsen Sebastian Höffner

7. Dezember 2015

1 Problemstellung

Das *Shortest Path*-Problem besteht darin, in einem gewichteten Graphen den kürzesten Weg von einem Start- zum Zielknoten auszurechnen. Wir kennen schon das *Single Source Shortest Path*-Problem, das darin besteht vom Startknoten die kürzesten Wege zu allen anderen Knoten auszurechnen. Das Problem lässt sich mit Dijkstras Algorithmus lösen. Weiterhin gibt es das *All Pairs Shortest Path*-Problem, für das wir Floyds Algorithmus kennengelernt haben. Hierbei werden alle kürzesten Wege zwischen je zwei Knoten berechnet.

Unser Problem wandelt die obigen insofern ab, als Kanten nicht mehr nur Gewicht c_{ij} haben, sonden nun noch eine Verzögerung (*delay* oder *transit time*) t_{ij} zusätzlich haben.

Wir möchten den nach Gewichten kürzesten Pfad von Knoten 1 nach n berechnen, dessen Verzögerung $\leq T$ ist.

2 Exakte Lösung

Durch dynamische Programmierung kann das Problem gelöst werden. Sei $g_j(c)$ die Gesamtverzögerund des schnellsten Pfades von 1 zu j, der nicht länger als c ist. Eine vereinfachende Annahme ist, dass der Graph azyklisch ist, sodass $(i,j) \in E \Rightarrow i < j$. Angeblich kann man dies leicht auf zyklische Graphen erweitern (wie auch immer).

2.1 Algorithmus B

$$
\begin{aligned}
g_1(c) &= 0, && c = 0, \dots, OPT,\\
g_j(0) &= \infty, && j = 2, \dots, n,\\
g_j(c) &= \min\left\{ g_j(c-1), \min_{k \mid c_{kj} \le c} \{g_k(c - c_{kj}) + t_{kj}\} \right\}, && j = 2, \dots, n;\ c = 1, \dots, OPT
\end{aligned}
$$

Abbildung 1: Algorithmus B

Die Laufzeit ist hier $\mathcal{O}(OPT \cdot n \cdot \text{Aufwand pro } (c,j))$. Für jedes $g_j(c)$ kann ein Aufwand von $\mathcal{O}(n)$ anfallen, da alle Vorgängerknoten durchsucht werden müssen. Es ergibt sich eine Laufzeit von $\mathcal{O}\left(n^2 OPT\right) = \mathcal{O}(|E| OPT)$.

2.2 Wann terminiert der Algorithmus?

Man kennt zwar OPT nicht, aber man weiß, dass $OPT = \min\{c \mid g_n(c) \le T\}$, also setzt man OPT, sobald man das erste c gefunden hat mit $g_n(c) \le T$, denn schneller kann es per Definition von g nicht werden.

Das Problem ist, dass der Algorithmus nur pseudopolynomiell ist (vgl. Rucksack). die Kantengewichte und damit OPT können aber exponentiell in der Eingabe(bit)länge sein.

Gemäß (Lorenz et al., 1999) gilt die Komplexität nur für azyklische Graphen. Für beliebige (auch mit Kantengewichten 0) sei die Laufzeit $\mathcal{O}(|E||V| OPT)$.

3 Die Test-Prozedur

Um ein FPTAS für das Problem zu konstruieren, wird zunächst ein Verfahren gesucht, das untere und obere Schranken für OPT berechnet. Hassin bemerkt, dass ein wesentliches Problem bei der Entwicklung von FPTAS war, dass man keine guten Grenzen kannte, deren Quotient polynomiell in der Eingabegröße ist. Wir haben allerdings keine Ahnung, warum so etwas hilfreich wäre.

Man wünscht sich einen polynomiellen $TEST(k)$, sodass

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

denn mit diesem könnte man durch binäre Suche auf $\{0, \dots, UB\}$ das Problem exakt lösen. Es ist aber NP-schwer, daher wird ein Test gesucht, sodass

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

Der Test skaliert und rundet alle Kantengewichte als $\hat{c}_{ij} = \lfloor \frac{c_{ij}(n-1)}{k\epsilon} \rfloor$ und wendet dann Algorithmus B an, bis $g_n(c) \leq T$ gefunden ist für $c < \frac{n-1}{\epsilon}$ oder $c \geq \frac{n-1}{\epsilon}$. Macht man die Division rückgängig, so wird klar, dass die Kosten einer Kante um höchstens $\frac{k\epsilon}{n-1}$ reduziert wird, die Kosten eines Pfades über $n-1$ Kanten also um maximal $k\epsilon$.

Falls $c < \frac{n-1}{\epsilon}$, dann gilt für den gefundenen Pfad im Originalgraphen

$$\begin{aligned} k &\leq k \\ \frac{k\epsilon}{n-1}\frac{n-1}{\epsilon} &\leq k \end{aligned}$$

Es folgt

$$\begin{aligned} \frac{k\epsilon}{n-1}c &< k \\ \frac{k\epsilon}{n-1}c + k\epsilon &< k + k\epsilon \\ \frac{k\epsilon}{n-1}c + k\epsilon &< k(1+\epsilon) \end{aligned}$$

Falls stattdessen $c \geq \frac{n-1}{\epsilon}$, so sind die Kosten im Originalgraphen größer gleich

$$\begin{aligned} c &\geq \frac{n-1}{\epsilon} \\ c\frac{k\epsilon}{n-1} &\geq \frac{k\epsilon}{n-1}\frac{n-1}{\epsilon} \\ c\frac{k\epsilon}{n-1} &\geq k \end{aligned}$$

Zudem ist $g_n(c)$ der billigste T-Pfad, alle anderen sind also höchstens so günstig. Somit leistet der Test das Geforderte.

```
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
```

Durch binäre Suche kann man nach oben beschränkte Zahlen runden, und alle größer als k werden hier rausgeworfen. Dafür fällt Laufzeit $\mathcal{O}\left(|E| \log \frac{n-1}{\epsilon}\right)$ an.

Algorithmus B führt insgesamt weniger als $\frac{n-1}{\epsilon}$ Iterationen durch, für die je schlimmstenfalls $\mathcal{O}(|E|)$ Zeit anfällt. Die Gesamtlaufzeit des Tests ist daher $\mathcal{O}\left(|E| \log \frac{n-1}{\epsilon} + |E| \frac{n-1}{\epsilon}\right) = \mathcal{O}\left(|E| \frac{n-1}{\epsilon}\right)$

4 Das FPTAS

Das Schema setzt obere und untere Schranken $LB \leq OPT \leq UB$ voraus. Initiale Werte können z.B. $LB = 1$ oder $LB =$ kürzester Pfad nach Kosten und $UB = \sum n-1$ längste Kanten oder $UB =$ Kosten des schnellsten Pfades von 1 nach n.

```
1  $UB := \sum(n-1)$ größte Kosten
2  $LB := 1$
3
4  Falls $UB \leq 2LB$, Goto Zeile 11
5  Sonst:
6     $k := \sqrt{UB \cdot LB}$
7     Falls $TEST(k)$ == true, $LB := k$
8     Sonst $UB := k(1+\epsilon)$
9     Goto Zeile 4
10
11    Setze $c_{ij} \leftarrow c_{ij}(n-1)/LB\epsilon$
12    Berechne mit Algorithmus B die optimale Lösung
```

Falls $UB \leq (1+\epsilon)LB$, so hat man mit UB schon eine ϵ-Approximation für OPT. Also sei $UB > (1+\epsilon)$. Man testet nun mittels $TEST(k)$ für Werte $LB < k < UB(1+\epsilon)$, um die Schranken zu verengen, da man entweder $UB = k(1+\epsilon)$ oder $LB = k$ setzen kann. Durch eine Art binäre Suche lassen sich die Grenzen schnell einengen. Man tut dies, bis $\frac{UB}{LB} \leq 2$ oder eine andere Konstante. Die Werte für k werden als $\sqrt{UB \cdot LB}$ berechnet (warum?).

Um das Problem approximativ zu lösen, wendet man nun Algorithmus B auf eine skalierte und gerundete Instanz mit Gewichten $c_{ij}(n-1)/LB\epsilon$ an. Die Abweichung vom optimalen Pfad kann (wie für $TEST$ gezeigt) nicht größer als $k\epsilon$, hier also $LB\epsilon < OPT\epsilon$ sein.

Durch Rundung der Kantenkosten reduziert sich die Laufzeit der letzten Anwendung von Algorithmus B auf $\mathcal{O}\left(|E| OPT \frac{(n-1)}{LB\epsilon}\right)$. Da $OPT \leq 2LB$, ist dies eine Teilmenge von $\mathcal{O}(|E| 2LB \frac{(n-1)}{LB\epsilon}) = \mathcal{O}(|E| 2 \frac{(n-1)}{\epsilon}) = \mathcal{O}\left(|E| \frac{n-1}{\epsilon}\right)$. Laut Hassin muss man $\log\log \frac{UB}{LB}$ Tests durchführen, bis der Quotient auf 2 sinkt.

Die Wurzelbestimmung kann teuer sein, aber es genügt, ein k zu finden, sodass $\sqrt[2]{\frac{UB}{LB}} < k < \sqrt{\frac{UB}{LB}}$. Dies geht offenbar in $\log\log \frac{UB}{LB}$ (Tippfehler im Paper?). Die einzelnen Aufrufe von $TEST$ benötigen wie oben $\mathcal{O}\left(|E| \frac{n-1}{\epsilon}\right)$. Insgesamt ergibt sich eine Laufzeit von $\mathcal{O}(\log\log \frac{UB}{LB} \cdot (|E| \frac{n-1}{\epsilon} + \log\log \frac{UB}{LB}))$, was polynomiell in n und ϵ.