Das Rundreiseproblem und Stabilität von Approximationsalgorithmen

Florian Forster

21. Februar 2008

Zusammenfassung

Diese Seminararbeit stellt das Rundreiseproblem und das ΔTSP vor und führt kurz in die Approximationsalgorithmen ein. Anschließend werden die Approximationsalgorithmen APPR2 und Christofides vorstellt und ihre Approximationsgüten abgeschätzt. Im zweiten Teil der Arbeit wird der Begriff der „Stabilität“ von Approximationsalgorithmen eingeführt und anhand der beiden Algorithmen besprochen. Dabei werden zwei Abstandsfunktionen, $\textsc{Dist}_{\text{three}}$ und $\textsc{Dist}_{\text{path}}$, verwendet, wobei beide Algorithmen instabil und quasistabil bezüglich $\textsc{Dist}_{\text{three}}$ und stabil bezüglich $\textsc{Dist}_{\text{path}}$ sind.

Inhaltsverzeichnis

1 Einleitung

Das **Rundreiseproblem** ist ein bekanntes und beliebtes Problem der Informatik, weil es einfach erklärt ist, aber trotz der vermeintlichen Einfachheit sind keine effizienten Algorithmen bekannt. Die Problemstellung ist die kürzeste Rundreise zu finden, so dass ein Handlungsreisender auf dieser Reise alle Städte einmal besucht. Nach dieser Idee heißt das Problem auf englisch „Traveling salesperson problem“ oder kurz **TSP**.

Nachdem in der Informatik Städte eine eher untergeordnete Rolle spielen, wird das Problem meist formaler definiert:

Definition 1 *Gegeben ist ein vollständiger, gewichteter Graph $G = (V, E)$. Das Rundreiseproblem ist, den leichtesten Hamiltonkreis in G zu finden.*

Diese Definition klingt zwar sehr abstrakt, das Problem stellt sich aber ganz natürlich in der Praxis: Ein Roboterarm, der Löcher in eine Platine bohrt, sollte dabei die kürzeste Strecke zurücklegen müssen um möglichst effizient zu arbeiten. Schulbusse müssen Kinder an bestimmten Haltestellen aufnehmen und nach der Schule wieder absetzen.

Dass dieses Problem wirklich schwer ist, ist nicht auf den ersten Blick klar: Zum Einen ist in einem vollständigen Graphen jede Reihenfolge der Knoten ein Hamiltonkreis – **irgendeine** Rundreise zu finden ist also trivial. Zum Anderen sind Algorithmen, die in einem gewichteten Graphen den leichtesten Weg von einem Knoten zu einem (oder allen) Anderen berechnen, seit langem bekannt. Bekannte Algorithmen sind zum Beispiel der Ford-Bellman-Algorithmus und der Algorithmus von Dijkstra.

Wie die meisten interessanten Probleme befindet sich TSP aber in der Komplexitätsklasse NP [3]. Für die Probleme in dieser Komplexitätsklasse sind trotz intensiver Forschung keine effizienten Algorithmen bekannt. Es ist auch nicht bekannt, ob es effiziente Algorithmen überhaupt geben kann. Eine häufig geäußerte Vermutung ist aber, dass es für Probleme in NP keine effizienten Algorithmen gibt.

2 Das ΔTSP und seine Approximation

Für viele Probleme in NP sind sogenannte **Approximationsalgorithmen** bekannt. Diese Algorithmen sind zwar effizient, finden aber nicht „die beste“ Lösung, sondern nur eine „gute“. Das heißt: Die Lösungen, die diese Algorithmen berechnen, werden nicht beliebig schlecht. Ein häufiger Fall ist, dass man einen konstanten Faktor angeben kann, so dass die Werte der approximierten Lösung höchstens um diesen Faktor von dem Wert der optimalen Lösung abweichen:

Definition 2 *Sei Φ ein Problem aus NP und I eine Instanz dieses Problems. Ein Approximationsalgorithmus A_Φ besitzt eine relative Approximationsgüte r, wenn für alle Instanzen der Quotient von Gewicht der approximierten Lösung und Gewicht der Optimalen Lösung OPT_Φ kleiner oder gleich r ist:*

$$r \geqq \frac{weight(A_\Phi(I))}{weight(OPT_\Phi(I))}$$

Wenn man das TSP als Optimierungsproblem formuliert[1] handelt es sich um ein Minimierungsproblem, da das Gewicht der Rundreise minimiert werden soll. Bei Maximierungsproblemen muss man entweder den Bruch umdrehen oder das „$\geqq$“ durch ein „$\leqq$“ ersetzen, da hier die suboptimalen Lösungen kleinere Werte als die optimale Lösung haben.

Für das TSP könnte das zum Beispiel heißen, dass die Lösung eines Approximationsalgorithmus höchstens das 100-fache Gewicht der optimalen Lösung hat. Würde allerdings ein solcher Algorithmus existieren, könnte man trickreich beliebige Graphen zu vollständigen Graphen erweitern und anhand der berechneten Lösung das Hamiltonkreis-Problem, also die Frage,

[1] Probleme in NP sind per Definition Entscheidungsprobleme.

ob der Graph eine Rundreise besitzt, die alle Knoten genau einmal besucht, beantworten. Das Hamiltonkreis-Problem ist ebenfalls in NP – die Existenz eines solchen Algorithmus würde also $P = NP$ beweisen. Eine Suche nach einem entsprechenden Approximationsalgorithmus für TSP ist also nahezu aussichtslos. Das selbe gilt natürlich für alle Faktoren, 100 ist nur ein wild aus der Luft gegriffenes Beispiel.

Dieses Ergebnis muss aber nicht bedeuten, dass es keinen Sinn macht sich mit dem Problem zu beschäftigen. Bei genauerem Hinsehen fällt auf, dass beide Beispiele, die in der Einleitung angeführt wurden, eigentlich Spezialfälle sind: Die Gewichte der Kanten sind nämlich nicht beliebig, sondern durch den Abstand definiert. Das heißt, dass die Gewichte die **Dreiecksungleichung** erfüllen:

$$\text{weight}(u, w) \leqq \text{weight}(u, v) + \text{weight}(v, w)$$

Ein Umweg über v ist also immer mindestens so teuer wie der direkte Weg.

Diese Randbedingung stellt eine Einschränkung des Problems dar, die stark genug ist um Approximationsalgorithmen mit relativer Güte, so wie oben definiert, zu erlauben. Im folgenden werden gleich zwei Algorithmen vorgestellt, die eine relative Approximationsgüte garantieren.

2.1 Der APPR2-Algorithmus

Der APPR2-Algorithmus arbeitet in relativ einfachen Schritten:

Algorithmus 1 APPR2

Eingabe: Vollständiger, gewichteter Graph G, dessen Gewichte die Dreiecksungleichung erfüllen.
Berechne den leichtesten Spannbaum H von G.
Dupliziere alle Kanten in H.
Berechne eine Eulertour $t = (v_{t_0}, v_{t_1}, \ldots)$ auf H.
Eliminiere doppelt vorkommende Knoten aus t.
Gib t aus.

In Abbildung 1 befindet sich eine grafische Veranschaulichung der Funktionsweise des APPR2-Algorithmus.

Das Duplizieren der Kanten hat den Zweck, dass alle Knoten einen geraden Grad haben, denn nur auf solchen Graphen existieren Eulertouren. Das Gewicht dieser Eulertour, die auch immer ein Hamiltonkreis ist, ist nicht beliebig: Wenn man aus der optimalen Rundreise OPT die schwerste Kante entfernt, erhält man einen Spannbaum von G. Das Gewicht dieses Spannbaums ist aber mindestens so groß wie das Gewicht der leichtesten Spannbaums H. Es gilt also:

$$\text{weight}(H) \lneqq \text{weight}(\text{OPT}(G))$$

Durch das duplizieren der Kanten wird das Gewicht von H verdoppelt. Durch das Überspringen von Knoten wird die Tour nicht schwerer, weil die Kantengewichte von G die Dreiecksungleichung erfüllen. Für die Ausgabe t gilt also:

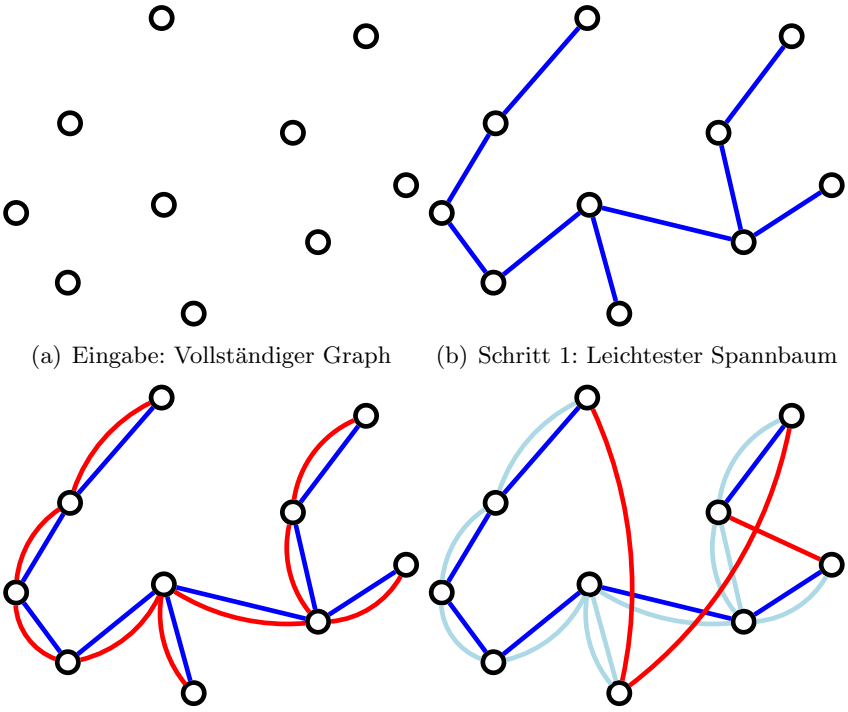
$$\text{weight}(t) \lneqq 2 \cdot \text{weight}(\text{OPT}(G))$$

Die relative Approximationsgüte des APPR2-Algorithmus ist kleiner als 2:

$$r \lneqq \frac{2 \cdot \text{weight}(\text{OPT}(G))}{\text{weight}(\text{OPT}(G))} = 2$$

Die Rundreise, die der APPR2-Algorithmus berechnet, ist also höchstens doppelt so schwer wie die optimale Rundreise.

Wichtig: Der APPR2-Algorithmus selbst verwendet die Dreiecksungleichung nicht, lediglich die Analyse, also die Abschätzung der Approximationsgüte, macht davon Gebrauch!



(a) Eingabe: Vollständiger Graph (b) Schritt 1: Leichtester Spannbaum

(c) Schritt 2: Duplizieren der Kanten (d) Schritt 3: Eulertour ohne Duplikate

Abbildung 1: Der APPR2-Algorithmus: Zunächst wird der leichteste Spannbaum berechnet und dessen Kanten verdoppelt. Auf diesem Graphen wird eine Eulertour berechnet und doppelte Knoten eliminiert.

2.2 Der Christofides-Algorithmus

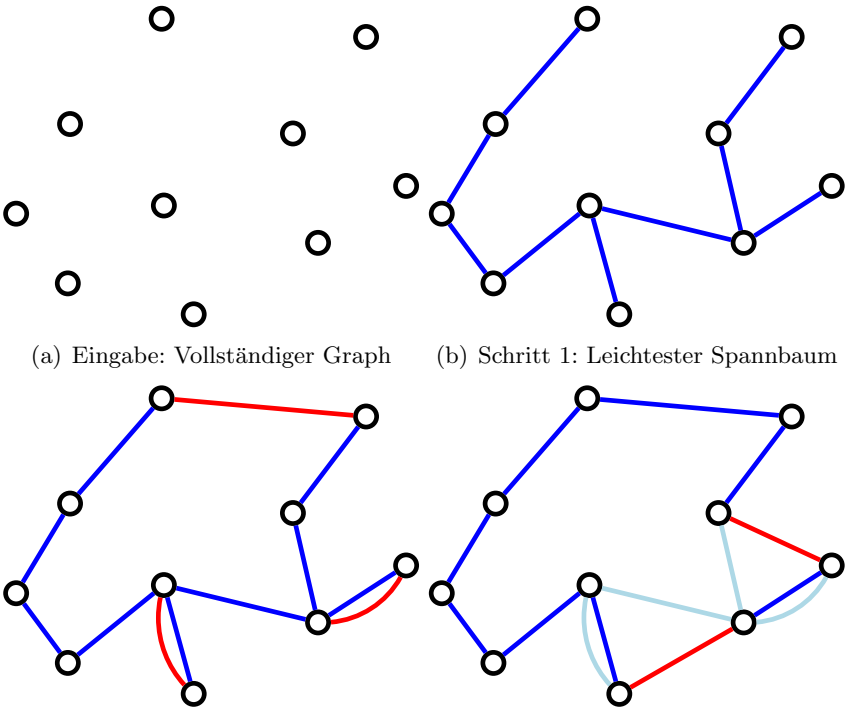
Der Christofides-Algorithmus [2] verbessert das Ergebnis des APPR2-Algorithmus durch einen Trick. Der APPR2-Algorithmus dupliziert alle Kanten des minimalen Spannbaums. Das hat den Zweck, dass alle Knoten des resultierenden Graphen einen geraden Grad besitzen, denn nur auf solchen Graphen existiert eine Eulertour. Der Christofides-Algorithmus ersetzt diesen Schritt. Statt alle Kanten zu duplizieren berechnet er ein leichtestes perfektes Matching auf allen Knoten des leichtesten Spannbaums, die einen ungeraden Grad haben. Sich davon zu überzeugen, dass die Anzahl dieser Knoten immer gerade sein muss, bleibt dem Leser überlassen. Anschließend haben wieder alle Knoten einen geraden Grad, so dass wieder eine Eulertour berechnet werden kann. Die Duplikate werden wieder eliminiert.

Algorithmus 2 Christofides

Eingabe: Vollständiger, gewichteter Graph G, dessen Gewichte die Dreiecksungleichung erfüllen.
Berechne den leichtesten Spannbaum H von G.
Berechne das leichteste perfekte Matching M aller Knoten aus H, die ungeraden Grad haben.
Berechne eine Eulertour $t = (v_{t_0}, v_{t_1}, \ldots)$ auf $H \uplus M$.
Eliminiere doppelt vorkommende Knoten aus t.
Gib t aus.

In Abbildung 2 befindet sich eine grafische Veranschaulichung der Funktionsweise des Christofides-Algorithmus.

Beim Gewicht des leichtesten Spannbaums können wir genauso argumentieren wie beim APPR2-Algorithmus. Das leichteste perfekte Matching müssen wir aber noch Abschätzen. Dazu überlegen wir uns zwei weitere Matchings, M_0 und M_1: Entsprechend ihrer Reihenfolge in



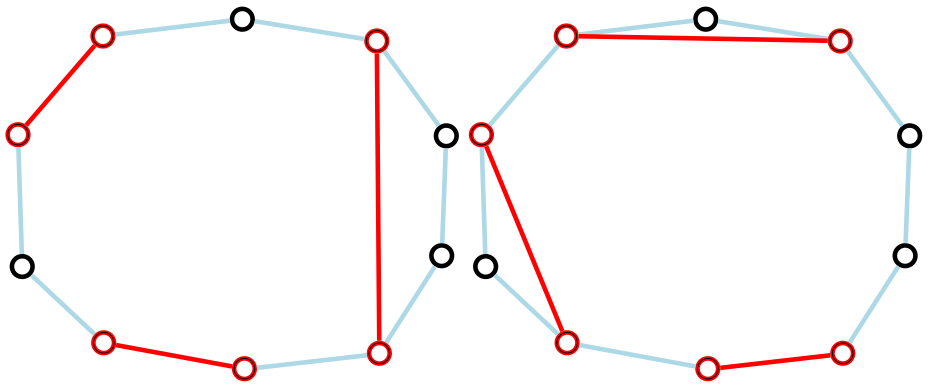
(a) Eingabe: Vollständiger Graph

(b) Schritt 1: Leichtester Spannbaum

(c) Schritt 2: Leichtestes perfektes Matching auf Knoten mit ungeradem Grad

(d) Schritt 3: Eulertour ohne Duplikate

Abbildung 2: Der Christofides-Algorithmus: Zunächst wird der leichteste Spannbaum berechnet. Anschließend wird ein leichtestes perfektes Matching auf allen Knoten mit ungeradem Grad berechnet. Auf der Vereinigung von Spannbaum und Matching wird eine Eulertour berechnet und doppelte Knoten eliminiert.



(a) Matching M_0 (b) Matching M_1

Abbildung 3: Schematische Darstellung der Matchings M_0 und M_1: Die Knoten mit ungeradem Grad werden entsprechend ihrer Reihenfolge in OPT(G) sortiert und die Kante, die aufeinander folgende Knoten verbindet, in das Matching aufgenommen.

einer optimalen Rundreise wird immer die Kante zwischen zwei aufeinander folgenden Knoten mit ungeradem Grad in das Matching aufgenommen. Abbildung 3 zeigt das vorgehen schematisch. Auf diese Art und Weise erhält man zwei perfekte Matchings. Da die Matchings entlang der optimalen Rundreise „abkürzen“, und da die Dreiecksungleichung gilt, ist die Summer der Gewichte kleiner oder gleich dem Gewicht einer optimalen Rundreise:

$$\text{weight}(M_0) + \text{weight}(M_1) \leqq \text{weight}(\text{OPT}(G))$$

Das heißt, dass das Gewicht des leichteren Matchings von M_0 und M_1 höchstens $\frac{1}{2}\cdot\text{weight}(\text{OPT}(G))$ sein kann. Da M aber ein **leichtestes** perfektes Matching ist, gilt diese Obergrenze auch für dessen Gewicht:

$$\text{weight}(M) \leqq \frac{1}{2}\text{weight}(\text{OPT}(G))$$

Das Gewicht der von Christofides gefundenen Lösung kann also wie folgt abgeschätzt werden:

$$\text{weight}(t) = \text{weight}(H) + \text{weight}(M) \leq \frac{3}{2}\text{weight}(\text{OPT}(G))$$

Die Approximationsgüte des Christofides-Algorithmus ist also $r = 1{,}5$.

Wichtig: Auch der Christofides-Algorithmus verwendet die Dreiecksungleichung nicht! Wir haben sie wieder verwendet um zu begründen, dass die Lösungen des Algorithmus nicht beliebig schlecht werden. Zur Berechnung der Lösung wurde dieser Umstand aber nicht verwendet.

3 Stabilität von Approximationsalgorithmen

Wie wir gesehen haben lässt sich das (allgemeine) TSP gar nicht gut approximieren, es sei denn dass $P = NP$ gilt, also für alle Probleme in NP effiziente Algorithmen existieren. Das ΔTSPhingegen lässt sich recht gut approximieren. Was ist aber mit Probleminstanzen, die die Dreiecksungleichung nur ein wenig verletzen? Dazu müssen wir erstmal definieren, was „ein wenig verletzen“ heißt:

Definition 3 *Sei L_Φ die Spezialisierung eines Optimierungsproblems L. $A_\Phi(x)$ sei ein Approximationsalgorithmus für L_Φ mit relativer Güte δ_Φ. $h_\Phi(x)$ sei eine Abstandsfunktion, für die gilt:*

- $h_\Phi(x) \gneqq 0$ *für* $x \in L \setminus L_\Phi$

- $h_\Phi(x) = 0$ *für* $x \in L_\Phi$
- $h_\Phi(x)$ *effizient berechenbar*

Sei $L_{\phi,h,r}$ *die Menge aller Probleminstanzen, deren Abstand (bezüglich* h_ϕ*) kleiner oder gleich* r *ist:*

$$L_{\phi,h,r} = \{x \in L : h_\phi(x) \leqq r\}$$

Wir haben jetzt also eine Abstandsfunktion, mit der wir den Abstand einer Instanz zur Menge der speziellen Instanzen angeben können. Im Fall von TSP können wir damit also ausdrücken wie sehr eine Instanz die Dreiecksungleichung verletzt. Mit Hilfe dieser Abstandsfunktion können wir Algorithmen jetzt einteilen, je nachdem wie gut sie mit eigentlich „ungültigen“ Eingaben zurechtkommen [1]:

Definition 4 *Stabilität von Approximationsalgorithmen*

- $A_\phi(x)$ *heißt* p**-stabil bezüglich** h_ϕ*, wenn für jedes* $r, 0 \lneqq r \leqq p$ *ein* $\delta_{\phi,r} \in \mathbb{R}^{>1}$ *existiert, so dass* $A_\phi(x)$ *ein Approximationsalgorithmus für* $L_{\phi,h,r}$ *mir relativer Güte* $\delta_{\phi,r}$ *ist.*
- $A_\phi(x)$ *heißt* **stabil bezüglich** h_ϕ*, wenn* $A_\phi(x)$ *für* **alle** $p \in \mathbb{R}^+$ *p-stabil bezüglich* h_ϕ *ist.*
- $A_\phi(x)$ *heißt* **instabil bezüglich** h_ϕ*, wenn* $A_\phi(x)$ *für* **kein** $p \in \mathbb{R}^+$ *p-stabil bezüglich* h_ϕ *ist.*
- $A_\phi(x)$ *heißt* $(r, f_r(n))$**-quasistabil bezüglich** h_ϕ*, wenn* $A_\phi(x)$ *ein Approximationsalgorithmus für* $L_{\phi,h,r}$ *mir relativer Güte* $f_r(n)$ *ist.*

Hier ist wichtig zu beachten, das bei **stabilen** Algorithmen die Größe der Eingabe keine Auswirkungen auf die Approximationsgüte hat, während das bei **quasistabilen** Algorithmen durchaus der Fall ist.

3.1 Abstandsfunktionen für das ΔTSP

Um diese Definitionen auch zu verwenden wollen wir im Folgenden zwei Abstandsfunktionen für das ΔTSP definieren und untersuchen, ob der APPR2- und der Christofides-Algorithmus stabil bezüglich dieser Funktionen sind.

Definition 5 *Abstandsfunktionen für das* Δ*TSP*

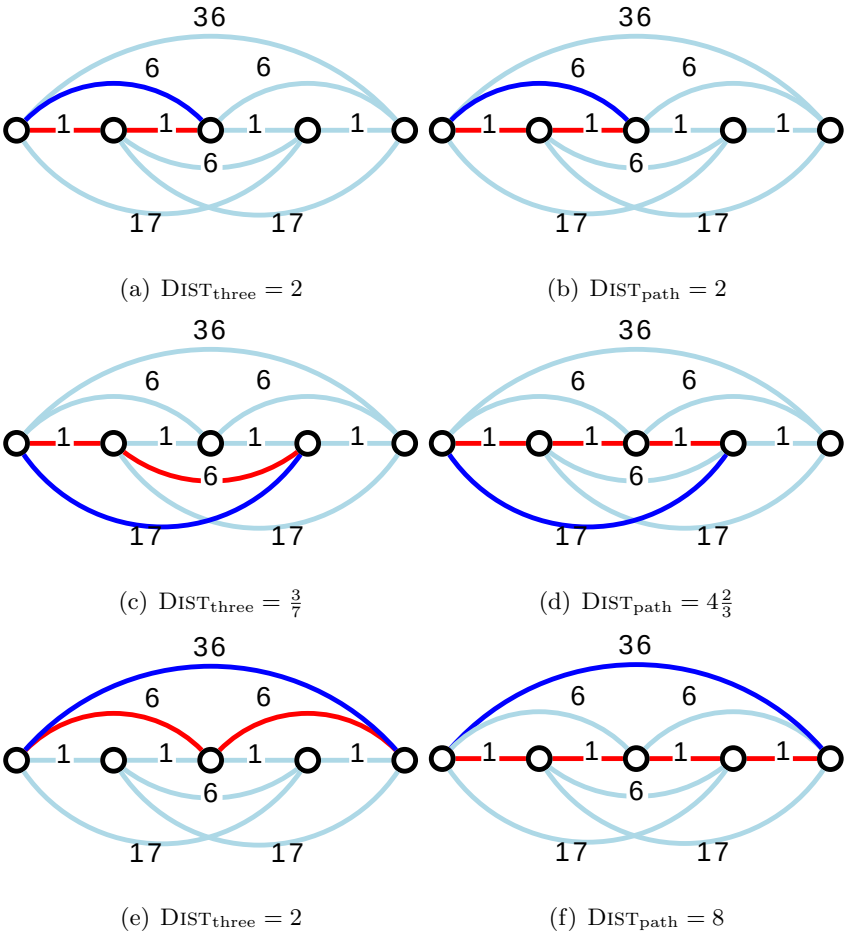
- *Funktion* $\textsc{Dist}_{three} =$

$$max\left\{0, max\left\{\frac{weight(u,v)}{weight(u,w) + weight(w,v)} - 1, u, v, w \in V\right\}\right\}$$

- *Sei* $W_{u,v} = \{w_1 = u, w_2, \ldots, w_m = v\}$ *ein einfacher Pfad von* u *nach* v*.*
  Funktion $\textsc{Dist}_{path} =$

$$max\left\{0, max\left\{\frac{weight(u,v)}{weight(W_{u,v})} - 1, u, v \in V\right\}\right\}$$

Die Abstandsfunktion $\textsc{Dist}_{\text{three}}$ definiert ein sehr intuitives Abstandsmaß: Von alles Dreiecken wird der Quotient aus dem Gewicht des direkten Weges und Gewicht des Umwegs berechnet und eins abgezogen. Da dieser Quotient bei Graphen, die die Dreiecksungleichung erfüllen, immer kleiner als Eins ist, ist das Endergebnis hier Null. Bei Graphen die die Dreiecksungleichung verletzen ist der Umweg leichter als der direkte Weg und der Quotient größer als Eins. Das Endergebnis ist dann ungleich Null.



(a) $\textsc{Dist}_{\text{three}} = 2$ (b) $\textsc{Dist}_{\text{path}} = 2$

(c) $\textsc{Dist}_{\text{three}} = \frac{3}{7}$ (d) $\textsc{Dist}_{\text{path}} = 4\frac{2}{3}$

(e) $\textsc{Dist}_{\text{three}} = 2$ (f) $\textsc{Dist}_{\text{path}} = 8$

Abbildung 4: Vergleich von $\textsc{Dist}_{\text{three}}$ (links) und $\textsc{Dist}_{\text{path}}$ (rechts).

Die Abstandsfunktion $\textsc{Dist}_{\text{path}}$ sieht etwas genauer hin: Statt nur Umwege mit zwei Kanten zu beachten, verwendet sie einen Pfad beliebiger Länge. Dadurch wird der tatsächlich leichteste Weg zwischen zwei Knoten verwendet um den Abstand zu berechnen, was den Abstand unter Umständen deutlich schneller in die Höhe treibt. Diesen Unterschied sieht man an einem konkreten Beispiel in Abbildung 4.

3.2 Stabilität bezüglich $\textsc{Dist}_{\text{path}}$

Sowohl der 2APPR-Algorithmus als auch der Christofides-Algorithmus sind **stabil** bezüglich $\textsc{Dist}_{\text{path}}$. Die Ausgabe der beiden Algorithmen kann man als Pfad schreiben:

$$t = v_0 \to v_1 \to \ldots \to v_n \to v_0$$

Statt der direkten Kante von einem Knoten zum nächsten, $v_i \to v_{i+1}$, kann man auch den leichtesten Pfad verwenden:

$$t = v_0 \to_p v_1 \to_p \ldots \to_p v_n \to_p v_0$$

Dabei soll $u \to_p v$ bezeichnen, dass nicht die Kante zwischen u und v sondern der leichteste Pfad zwischen u und v gemeint ist.

Da das Gewicht jeder Kante $u \to v$ höchstens das $(r+1)$-fache des Gewichts des Pfades $u \to_p v$ ist, wird die Lösung durch das Überspringen von Knoten höchstens um das $(r+1)$-fache schwerer. Der APPR2-Algorithmus besitzt die relative Approximationsgüte $2(r+1)$, der Christofides-Algorithmus besitzt die relative Approximationsgüte $\frac{3}{2}(r+1)$.

3.3 Stabilität bezüglich $\text{Dist}_{\text{three}}$

Ganz so einfach wie die Argumentation bei der Funktion $\text{DIST}_{\text{path}}$ ist die Angelegenheit bei $\text{DIST}_{\text{three}}$ nicht, da ja mehrere Knoten hintereinander übersprungen werden können. Dazu betrachten wir die Kante (u, v) sowie w, den leichtesten Pfad von u nach v mit n Knoten:

$$w = (w_0 = u) \to w_1 \to w_2 \to \ldots \to w_{n-1} \to (w_n = v)$$

Wenn wir den Pfad betrachten können wir die Anzahl der Kanten halbieren, indem wir jeden zweiten Knoten überspringen. Das Gewicht der neuen Kanten ist möglicherweise größer als die Summe der Gewichte der ersetzten Kanten:

$$\text{weight}(w_i \to w_{i+2}) \leqq (r+1) \cdot \text{weight}(w_i \to w_{i+1} \to w_{i+2})$$

Das heißt, dass sich durch dieses Ersetzen das Gewicht des gesamten Pfades höchstens um $(r+1)$ erhöht hat. Da bei diesem Ersetzen die Anzahl der Kanten immer halbiert wird, kann das Verfahren $\lceil \log_2 m \rceil$-mal angewandt werden. Daher ist das Gewicht der Kante zwischen u und v höchstens:

$$\text{weight}(u \to v) \leqq (1+r)^{\lceil \log_2 m \rceil} \cdot \text{weight}(w)$$

Da jeder Pfad p mit m Kanten in G mit einer einzigen Kante e mit dem Gewicht

$$\text{weight}(e) \leqq (1+r)^{\lceil \log_2 m \rceil} \cdot \text{weight}(p)$$

ersetzt werden kann, ist die Approximationsgüte nicht mehr unabhängig von der Größe der Eingabe. Ein Pfad p kann bis zu $|V|-1$ Kanten beinhalten, und diese Zahl wächst natürlich mit der Länge der Eingabe. Deshalb sind der APPR2-Algorithmus und der Christofides-Algorithmus **instabil** bezüglich $\text{DIST}_{\text{three}}$.

Allerdings kann man die Approximationsgüten der Algorithmen durch je eine Funktion angeben, welche abhängig von der Länge der Eingabe n ist:

- Der APPR2-Algorithmus ist $\left(r, 2\,(1+r)^{\lceil \log_2 n \rceil}\right)$-quasistabil bezüglich $\text{DIST}_{\text{three}}$
- Der Christofides-Algorithmus ist $\left(r, \frac{3}{2}\,(1+r)^{\lceil \log_2 n \rceil}\right)$-quasistabil bezüglich $\text{DIST}_{\text{three}}$

Die Autoren von [1] stellen in dieser Arbeit den PMC-Algorithmus vor, welcher stabil bezüglich $\text{DIST}_{\text{three}}$ ist. Der Algorithmus ist dem Christofides-Algorithmus ähnlich, verwendet aber „Pfad-Matchings“ und eine clevere Routine (und Argumentation) um den Eulerkreis zu berechnen. Die angegebene Approximationsgüte ist $\frac{3}{2}(1+r)^2$, die Laufzeit wird mit $O(n^3)$ angegeben. Es existieren also auch Algorithmen für das TSP, die stabil bezüglich $\text{DIST}_{\text{three}}$ sind.

Literatur

[1] Hans-Joachim Böckenhauer, Juraj Hromkovic, Ralf Klasing, Sebastian Seibert, and Walter Unger. Towards the notion of stability of approximation for hard optimization tasks and the traveling salesman problem. In *CIAC '00: Proceedings of the 4th Italian Conference on Algorithms and Complexity*, pages 72–86, London, UK, 2000. Springer-Verlag.

[2] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. In *Algorithms and Complexity: New Directions and Recent Results*, page 441. Academic Press, 1976.

[3] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.