Technische Universität München
Zentrum Mathematik

Diskrete Optimierung: Fallstudien aus der Praxis

Barbara Wilhelm | Michael Ritter

Der Nearest-Insert-Algorithmus für TSP

Im Folgenden finden Sie eine formale Beschreibung des gewählten Approximationsalgorithmus. Arbeiten Sie die Beschreibung durch, machen Sie sich Notizen. Versuchen Sie zunächst, die Schritte des Algorithmus mit Hilfe von kleinen Skizzen nachzuvollziehen und die Aussage des Lemmas (ohne Beweis) zu verstehen. Machen Sie sich außerdem klar, wie das Lemma angewendet wird, um die Approximationsgüte des Algorithmus zu beweisen (Satz 2 mit Beweis). Befassen Sie sich erst dann mit dem Beweis des Lemmas. Überlegen Sie sich, wie Sie die Erklärung des Algorithmus gestalten (Struktur, wichtige Ideen, Resultate). Wenn Sie Fragen haben, wenden Sie sich an Ihre Betreuer.

Problem: Traveling Salesman
Input: Ein Graph $G = (V, E)$ mit einer Distanzfunktion $d : E \to \mathbb{Q}_{\geq 0}$.

Aufgabe: Finde eine Tour, die alle Knoten des Graphen G genau einmal besucht und deren Länge bezüglich d möglichst klein ist, oder stelle fest, dass keine solche Tour existiert.

Eine *Tour* in einem Graphen ist ein Kreis, der jeden Knoten des Graphen genau einmal enthält. Das Traveling Salesman-Problem fragt also nach einer möglichst kurzen „Rundreise“, die jeden Knoten genau einmal besucht. Wie Sie bereits wissen, gibt es für allgemeines TSP Nichtapproximierbarkeitsaussagen: Die Existenz einer Approximation mit konstantem Approximationsfaktor würde $\mathcal{P} = \mathcal{NP}$ implizieren. Für eine Reihe praktisch interessanter Probleme kann man aber eine eingeschränkte Version des TSP betrachten. Von besonderer Bedeutung ist hier das *metrische TSP*.

Problem: Metrisches TSP
Input: Ein *vollständiger* Graph $G = (V, E)$ mit einer Distanzfunktion $d : E \to \mathbb{Q}_{\geq 0}$, so dass die Dreiecksungleichung gilt, d. h., für alle $i, j, k \in V$ ist

$$d(\{i, j\}) \leq d(\{i, k\}) + d(\{k, j\}).$$

Aufgabe: Finde eine Tour, die alle Knoten des Graphen G genau einmal besucht und deren Länge bezüglich d möglichst klein ist.

Im metrischen TSP ist also einerseits der zu Grunde liegende Graph vollständig (und damit ist auch die Existenz einer TSP-Tour gesichert), andererseits gilt für die Distanzen eine Dreiecksungleichung. Für alle praktischen Probleme, bei denen die Entfernungen auf einer Metrik beruhen, ergibt sich also ein solches TSP. Es lässt sich zeigen, dass auch metrisches TSP ein $\mathcal{NP}$-schweres Problem ist, die Einschränkung verändert die Komplexität also nicht. Immerhin lassen sich mit Hilfe der Dreiecksungleichung aber effiziente Approximationsalgorithmen konstruieren und mit einem solchen wollen wir uns hier beschäftigen.

Die Nearest-Insert-Heuristik baut rekursiv eine TSP-Tour in G auf. Dazu beginnt der Algorithmus mit einer kleinstmöglichen Subtour (eine Subtour ist eine TSP-Tour auf einer Teilmenge der Knoten von G), also einem einzelnen Knoten. Die Rekursion fügt einer bereits vorhandenen Subtour einen weiteren Knoten hinzu, so dass die Subtour so lange anwächst, bis schließlich der komplette Graph G durchlaufen wird. Wir verwenden folgende Notationen:

- Eine Subtour T_i wird je nach Kontext als Tour mit Reihenfolge der Form $(v_1, v_2, \ldots, v_k, v_1)$ aufgefasst oder als Menge der Form $\{v_1, v_2, \ldots, v_k\}$.
- Für eine Subtour T_i bezeichnet $V \setminus T_i$ die Menge aller Knoten in V, die nicht in der Subtour T_i enthalten sind.
- Für eine Subtour T_i und einen Knoten $v \in V \setminus T_i$ bezeichnet $d(T_i, v)$ die Entfernung zwischen der Tour und dem Knoten v, d. h.,

$$d(T_i, v) := \min \{d(\{u, v\}) : u \in T_i\}.$$

- Für eine Subtour T_i und einen Knoten $v \in V \setminus T_i$ bezeichne $\Delta(T_i, v)$ die Kosten, die durch Einfügen von v in die Tour T_i entstehen. Dazu wird eine Kante $\{u, w\}$ in T_i gewählt (u und w sind also Knoten, die in T_i aufeinander folgen) und durch die beiden Kanten $\{u, v\}$ und $\{v, w\}$ ersetzt. Die Wahl der Kante $\{u, w\}$ erfolgt so, dass die Kosten dieses Einfüge-Vorgangs minimal sind. Wir definieren daher $\Delta(T_1, v) := d(T_1, v)$ und für $i \geq 2$

$$\Delta(T_i, v) := \min \Big\{ \big[d(T_i) - d(\{u, w\}) + d(\{u, v\}) + d(\{v, w\})\big] - d(T_i) : \\ u, w \in T_i \text{ sind in } T_i \text{ aufeinander folgende Knoten} \Big\}$$

Im Detail funktioniert Nearest-Insert wie folgt:

1. Wähle einen beliebigen Knoten $v_1 \in V$ und starte mit der Subtour $T_1 := (v_1)$.
2. Wähle einen Knoten $v_2 \in V \setminus T_1$, so dass $d(T_1, v_2)$ minimal wird, und setze $T_2 := (v_1, v_2, v_1)$.
3. Für $i = 2, \ldots, n-1$:
   a) Wähle einen Knoten $v_{i+1} \in V \setminus T_i$, so dass $d(T_i, v_{i+1})$ minimal wird.
   b) Bestimme $\Delta(T_i, v_{i+1})$ sowie zwei in T_i aufeinander folgende Knoten u_i und w_i, für die das Minimum angenommen wird.
   c) Erzeuge T_{i+1} aus T_i durch Löschen der Kante $\{u_i, w_i\}$ und Einfügen der Kanten $\{u_i, v_{i+1}\}$ und $\{v_{i+1}, w_i\}$.

Für den Beweis der Approximationsgüte verwenden wir eine Hilfsaussage, die ein hinreichendes Kriterium für eine 2-Approximation liefert.

Lemma 1
Sei K_n ein vollständiger Graph auf $n \in \mathbb{N}$ Knoten und $d : E \to \mathbb{Q}_{\geq 0}$ eine Distanzfunktion, welche die Dreiecksungleichung erfüllt. Seien $T_1, \ldots, T_n$ die Subtouren aus dem Nearest-Insert-Algorithmus und $v_2, \ldots, v_n$ die Knoten, die der Reihe nach durch den Algorithmus hinzugefügt

werden (es gilt also $v_{i+1} = T_{i+1} \setminus T_i$). Weiter gelte für alle $i \in \{1, \ldots, n-1\}$ folgende Bedingung:

$$\forall\, p \in T_i \text{ und } q \notin T_i : \quad \Delta(T_i, v_{i+1}) \leq 2d(\{p, q\})$$

(Das Einfügen des Knoten v_{i+1} ist also „billiger“ als das Doppelte jeder beliebigen Kante, die „aus T_i herausführt“.)

Ist T die vom Nearest-Insert-Algorithmus erzeugte TSP-Tour, so gilt für jeden minimalen Spannbaum T^* in (K_n, d):

$$d(T) \leq 2d(T^*) \leq 2d(\tau_{\text{opt}}),$$

wobei τ_{opt} eine optimale TSP-Tour in (K_n, d) ist.

Beweis. Mit den Bezeichnungen aus dem Lemma konstruieren wir eine Bijektion

$$\phi : \{1, \ldots, n-1\} \to E(T^*),$$

die jedem Rekursionsschritt des Algorithmus eine Kante von T^* zuordnet, d. h., für alle $i \in \{1, \ldots, n-1\}$ gilt: $\phi(i) = \{p_i, q_i\}$ für passend gewählte $p_i \in T_i$ und $q_i \notin T_i$. Damit folgt die Behauptung dann wie folgt:

$$d(T) = \sum_{i=1}^{n-1} \underbrace{\Delta(T_i, v_{i+1})}_{\leq 2d(\{p_i, q_i\})} \leq 2 \sum_{i=1}^{n-1} d(\{p_i, q_i\}) = 2d(T^*)$$

Die letzte Gleichung gilt, weil $\{p_i, q_i\}$ jeweils so gewählt ist, dass sie eine Kante in T^* ist; da ϕ eine Bijektion auf $E(T^*)$ ist, durchläuft die Summe also alle Kanten des Spannbaums. Da weiter jede TSP-Tour einen Spannbaum beinhaltet (Weglassen einer beliebigen Kante ergibt einen Spannbaum), ist $d(\tau_{\text{opt}}) \geq d(T^*)$, das zeigt die Behauptung.

Wir zeigen jetzt, wie die Bijektion ϕ zu konstruieren ist. Für beliebiges $i \in \{1, \ldots, n-1\}$ sei $j \in \{1, \ldots, n-1\}$ maximal mit den beiden Eigenschaften

1. $j < i + 1$,
2. alle Knoten auf dem eindeutigen v_{i+1}-v_j-Weg in T^* mit Ausnahme des Knotens v_j selbst haben einen Index $\geq i + 1$.

Wir definieren dann $\phi(i) = e_i$ als die letzte Kante auf dem eindeutigen v_{i+1}-v_j-Weg. (Überlegen Sie sich, warum ϕ damit wohldefiniert ist!) Die ausgewählte Kante $\{p_i, q_i\} = e_i$ erfüllt dann auch $p_i = v_j \in T_i$ (da $j < i + 1$ und v_j deshalb schon in die Tour eingebunden ist) und $q_i \notin T_i$ (da der Index von q_i größer als i ist).

Es bleibt zu zeigen, dass ϕ damit auch wirklich bijektiv ist. Da Definitions- und Bildbereich von ϕ endlich sind, genügt es, dafür die Injektivität zu zeigen. Angenommen, es gäbe $i_1, i_2 \in \{1, \ldots, n-1\}$ mit $i_1 < i_2$, aber $e := \phi(i_1) = \phi(i_2)$. Seien j_1, j_2 die zugehörigen größten Indizes in $\{1, \ldots, n-1\}$, die Bedingungen 1 und 2 erfüllen. Wir unterscheiden zwei Fälle:

Fall 1: $v_{j_1} = v_{j_2}$, d. h., die Pfade v_{i_1+1}-v_{j_1} und v_{i_2+1}-v_{j_2} (die ja beide e als letzte Kante haben) durchlaufen e in gleicher Richtung. Wir betrachten dann den Teilpfad P des v_{i_1+1}-v_{j_1}-Pfades, der in v_{i_1+1} beginnt und bis zum ersten gemeinsamen Knoten mit dem

v_{i_2+1}-v_{j_2}-Pfad reicht. (Machen Sie sich eine Skizze!) Weil $i_1 < i_2$ ist, gibt es einen Index m (mit $v_m \in P$) mit den Eigenschaften

- $m \leq i_2 + 1$ und
- alle Knoten auf dem v_m-v_{i_2+1}-Pfad außer m selbst haben einen Index $\geq i_2 + 1$.

Das bedeutet aber

$$i_1 + 1 \leq m \leq j_2 = j_1 < i_1 + 1,$$

ein Widerspruch. Zur zweiten Ungleichung: Wäre $m > j_2$, so gäbe es einen Pfad von v_{i_2+1} nach v_m, für den die Bedingungen 1 und 2 erfüllt wären. Dann wäre aber j_2 nicht der *maximale* Index mit diesen Eigenschaften.

Fall 2: $v_{j_1} \neq v_{j_2}$, d. h., die Kante e wird von den beiden Pfaden in verschiedener Richtung durchlaufen. Dann gilt

$$j_1 < i_1 + 1 < i_2 + 1 \leq j_1,$$

da j_1 im „Inneren“ des v_{i_2+1}-v_{j_2}-Pfades liegt, erneut ein Widerspruch. □

Wir überlegen uns nun, warum aus diesem Lemma die gewünschte 2-Approximation für Nearest-Insert folgt.

Satz 2
Nearest-Insert ist eine 2-Approximation für metrisches TSP.

Beweis. Seien wieder $T_1, \ldots, T_n$ die Subtouren aus dem Algorithmus und T die vom Algorithmus erzeugte TSP-Tour. Sei $i \in \{2, \ldots, n-1\}$ und $v_{i+1} \in T_{i+1} \setminus T_i$ der vom Algorithmus gewählte Knoten, der in T_i eingefügt werden soll, d. h., v_{i+1} minimiert $d(T_i, v_{i+1})$. Dann gibt es einen Knoten $u \in T_i$, so dass $\min\{d(\{u, w\}) : u \in T_i, w \in V \setminus T_i\} = d(\{u, v_{i+1}\})$. Sei w der Nachfolger von u in der Subtour T_i, dann gilt

$$\Delta(T_i, v_{i+1}) \leq d(\{u, v_{i+1}\}) + d(\{v_{i+1}, w\}) - d(\{u, w\}) \leq 2d(\{u, v_{i+1}\}),$$

da wegen der Dreiecksungleichung $d(\{v_{i+1}, w\}) \leq d(\{u, v_{i+1}\}) + d(\{u, w\})$.

Weil $u \in T_i$ und $v_{i+1} \notin T_i$ gerade den Ausdruck $d(\{u, v_{i+1}\})$ minimieren, sind damit die Voraussetzungen des Lemmas erfüllt. Sei nun T^* ein minimaler Spannbaum in (K_n, d) und τ_{opt} eine optimale TSP-Tour, dann folgt mit dem Lemma

$$d(T) \leq 2d(T^*) \leq 2d(\tau_{\text{opt}}).$$ □