Der TSP-Approximationsalgorithmus von Rao & Smith

Zusammenfassung zum Seminarvortrag von Fabian Januszewski

Wir betrachten das *Traveling-Salesperson-Problem (TSP)*, das nach einer günstigsten *Traveling-Salesperson-Tour (TST)* fragt. Wir betrachten den konkreten Fall für n Punkte $p_1, \ldots, p_n \in \mathbb{R}^2$ für die euklidische Norm $|\cdot|$. Ist $G = (V, E)$ ein Graph mit $V \subset \mathbb{R}^2$, dann ordnen wir den Kanten $e = \{a, b\} \in E$ das Gewicht $v(e) = |a - b|$ zu, d.h. den euklidischen Abstand der Endpunkte. Einen solchen Graphen nennen wir *euklidisch*. Ist G zusammenhängend, reflexiv und ungerichtet, erhalten wir eine Metrik $d_G : V \times V \to \mathbb{R}$ auf der Menge der Knoten, welche dadurch gegeben ist, daß sie je zwei Knoten $a, b \in V$ die Länge (Kosten) des kürzesten Pfades von a nach b in G zuordnet.

Definition. Sei $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$ eine nicht-leere Menge von Punkten in der euklidischen Ebene. Ein euklidischer Graph $G = (P, E)$ ist für $\epsilon > 0$ ein $(1+\epsilon)$-Spanner, falls $\forall p_i, p_j \in P : \quad d_G(p_i, p_j) \leq (1+\epsilon) \cdot |p_i - p_j|$.

Der vollständige Graph auf P ist für jedes $\epsilon > 0$ ein $(1+\epsilon)$-Spanner, allerdings mit $O(n^2)$ Kanten. Wichtig für uns ist, daß sich ein $(1+\epsilon)$-Spanner mit $O(n)$ Kanten in $O(n \log n)$ berechnen läßt.

Spanner-TST-Lemma. *Sei P eine endliche Menge von Punkten aus $\mathbb{R}^2$ und $\epsilon > 0$ klein. Dann enthält jeder $(1+\epsilon)$-Spanner S von P eine $(1+\epsilon)$-Approximation R einer optimalen Rundreise auf P. Das heißt, sofern $\ell(R)$ die Länge von R im Sinne der euklidischen Gewichtung der Kanten bezeichnet, daß*

$$\ell(R) \leq (1+\epsilon) \cdot OPT(P).$$

Dieses Lemma ist der Schlüssel zum Algorithmus von Rao & Smith. Wir finden einen $(1+\delta)$-Spanner S für ein ausreichend kleines $\delta \leq \epsilon$ und wissen, daß er in jedem Fall eine $(1+\delta)$-TST-Approximation enthält. Diese läßt sich im Allgemeinen nicht direkt effizient berechnen. Deshalb *patchen* wir den Graphen bezüglich einer randomisierten *Quad-Tree-Zerlegung*, indem wir mehrere Schnitte mit einer Quad-Tree-Grenze durch einen einzigen ersetzen, sodaß wir nur noch eine kleine Zahl Schnitte mit Quad-Tree-Box-Grenzen haben. Wie auch Arora dies tut, nutzen wir *dynamisches Programmieren*, um in diesem gepatchten Graphen eine optimale Approximation zu finden. Diese ist im allgemeinen wegen des Patchens keine $(1+\delta)$-Approximation mehr, aber, da δ ausreichnd klein bzgl. ϵ war, immer noch eine $(1+\epsilon)$-Approximation, sofern unser Spanner durch das Patchen nicht zu lang geworden ist.

Ein wesentlicher Unterschied zu Aroras Ansatz ist, daß wir das Patchen tatsächlich *ausführen*, wohingegen Arora diese Idee lediglich dazu nutzte, um nachzuweisen, daß mit Wahrscheinlichkeit $1/2$ eine gute Approximation existiert, welche durch das dynamische Programmieren zutage gefördert werden kann.

Ein weiterer wesentlicher Unterschied ist, daß Arora nicht entscheiden konnte, ob sein Ergebnis eine gültige Approximation war, oder ob nicht. Wir jedoch kennen den Spanner und wissen, daß er eine gültige Approximation enthält. Der Erfolg kann uns nur dadurch verwehrt werden, daß wir diese Lösung nicht effizient finden können, was genau dann der Fall ist, wenn der Spanner durch das Patchen zu lang geworden ist. Das tritt allerdings höchstens in der Hälfte der Fälle ein, sodaß wir die gleiche Erfolgswahrscheinlichkeit wie Arora haben.

Da der Spanner dünn ist, können wir das Patchen in $O(n \log n)$ Zeit bewerkstelligen, genauso wie die Erzeugung des Spanners und des Quad-Trees. Das dynamische Programmieren schlägt bei uns nur mit $O(n)$ Zeit zu Buche, weil wir uns im Gegensatz zu Arora um Portale nicht mehr kümmern müssen und die Anzahl der möglichen Interfaces pro Quad-Tree-Box konstant bzgl. n ist. Damit erhalten wir einen Gesamtzeitaufwand in $O(n \log n)$.

Dieser Algorithmus läßt sich derandomisieren, indem wir eine gute Quad-Tree-Zerlegung finden. Das erreichen wir dadurch, daß wir zuerst einen guten vertikalen Shift a und daraufhin einen guten vertikalen Shift b bestimmen, was in $O(n \log n)$ Zeit möglich ist.

Dieser Algorithmus läßt sich auch auf $d \in \mathbb{N}$ Dimensionen verallgemeinern, die Laufzeit ist dann $2^{(d/\epsilon)^{O(d)}} n + (\frac{d}{\epsilon})^{O(d)} n \log n$, für konstante ϵ und d also immer noch in $O(n \log n)$.