Programmierpraktikum Diskrete Optimierung
Steiner-Baum-Probleme

Stephan Held
held@or.uni-bonn.de

Sommersemester 2014

1 Problemformulierung

Das Minimum Steiner-Baum-Problem (MST) ist wie folgt definiert. Gegeben eine Menge S von Terminalen wird ein längenminimaler Baum gesucht, der alle Terminale verbindet. Im Rahmen des Programmierpraktikums sind wir vor allem an Steinerbäumen in der Ebene ($S \subset \mathbb{R}^2, |S| < \infty$) mit Manhattan-Distanzen (ℓ_1-Norm), wie sie im VLSI-Design vorherrschend sind (also $(M, d) = (\mathbb{R}^2, \ell_1)$), oder Steinerbäumen in gewichteten Graphen (G, c) und ($S \subseteq V(G)$) interessiert.

Beide Probleme sind NP-schwer, wobei das Steiner-Baum-Problem in $(\mathbb{R}^2, \ell_p)$ beliebig genau approximiert werden kann, während dies für Steinerbäume in Graphen nicht geht, falls P $\neq$ NP. Es gibt jedoch Approximationsalgorithmen mit konstanter Approximationsgüte. Z.B. übersteigt die Länge eines minimalen Spannbaumes einen minimalen Steiner-Baum höchstens um den Faktor 1.5 in der ℓ_1-Metrik, bzw. um einen Faktor 2 in Graphen.

Algorithmen für Graphen können auch auf geometrische Instanzen angewandt werden. Für ℓ_1-Instanzen existiert immer ein minimaler Steiner-Baum im sogenannten Hanan-Gitter, welches dadurch entsteht, dass durch jedes Terminal eine horizontale und vertikale Linie gelegt wird und an den Kreuzungspunkten zweier Linien ein Knoten eingefügt wird.

2 Eingabe-Daten

Das Format für die Steiner-Baum-Instanzen ist das STP Data Format: `http://steinlib.zib.de/format.php`, welches zunächst für Graphen definiert ist, aber auch bei geometrischen Instanzen angewandt werden kann.

Kleine übersichtliche Testinstanzen finden sich auf den Webseiten zum Praktikum `http://www.or.uni-bonn.de/lectures/ss14/praktikum_ss14.html` sowie auf den Seiten zur 11th DIMACS-Challenge `http://dimacs11.cs.princeton.edu/downloads.html`.

Programmiersprache

Die Implementierungen müssen in C/C++ ausgeführt werden und unter Linux mit dem gcc/g++ kompilierbar sein, d.h. sie müssen dem ANSI C99/C++98-Standard genügen. Außer den Standard-Bibliotheken, insbesondere der STL, dürfen keine Hilfsbibliotheken verwendet werden, es sei denn es wird in der Aufgabenstellung ausdrücklich erwähnt.

Der Code muss verständlich programmiert und kommentiert werden. Selbstverständlich müssen die Programme fehlerfrei funktionieren. Insbesondere werden sie mit dem Programm valgrind (http://valgrind.org) überprüft und valgrind-Fehler führen zu Abzügen.

Einführungsaufgabe

Als Teil der Einführungsaufgabe sollte zunächst eine Einleseroutine für Steiner-Bauminstanzen implementiert werden. Soweit nicht explizit in der speziellen Aufgabenstellung angegeben, sollte Anschließend als obere Schranke ein minimaler Spannbaum berechnet werden.

Spätester Abgabetermin für die Einführungsaufgabe ist der 30.04.2014

Es wird jedoch empfohlen, die Implementierung der Gesamtaufgabe schon während der Semesterferien weitestgehend abzuschließen, da in der Vorlesungszeit häufig wenig Zeit bleibt.

3 Aufgaben

3.1 Geometrische Algorithmen für die ℓ_1-Metrik

Aufgabe 1: Der FLUTE-Algorithmus ermöglicht die schnelle Bestimmung optimaler Bäume auf kleinen Instanzen per Table-Lookup. Er soll implementiert werden und Tabellen für bis zu acht Terminalen berechnen. Größere Terminalmengen sollten zunächst in kleinere Mengen partitioniert und dann heuristisch zusammengefügt werden. Eine Variante sollte darin bestehen, iterativ einen minimalen Spannbaum dadurch zu verbessern, dass Teilbäume mit bis zu 8 Knoten optimal verbunden werden.

Chris Chu and Yiu-Chung Wong. FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design. In IEEE Transactions on Computer-Aided Design, vol. 27, no. 1, pages 70-83, January 2008.

Aufgabe 2: Die schnellsten optimalen Algorithmen basieren auf der Beobachtung, dass minimale Steinerbäume immer in sogenannte Full Steiner Trees (FST) zerlegt werden können. Eine hinreichende Obermenge von FSTs, die einen minimalen Steiner-Baum überdeckt, kann in der Praxis effizient enummeriert werden. Anschließend wird zum optimalen Lösen eine optimale Teilmenge an FSTs bestimmt, die alle Terminale überspannt. Dieses entspricht dem NP-schweren *Minimum Spanning Tree Problem in Hypergraphen*, dass mittels ganzzahliger

Optimierung bestimmt werden könnte. In dieser Aufgabe soll das Spannbaumproblem aber mit Hilfe eines Greedy-Algorithmus approximativ gelöst werden.

Martin Zachariasen: Rectilinear full Steiner tree generation. Networks 33(2): 125-143 (1999)

Baudis, G., Gröpl, C., Hougardy, S., Nierhoff, T. and Prömel, H.-J. : Approximating Minimum Spanning Sets in Hypergraphs and Polymatroids. Technical Report (2000).

Aufgabe 2b: Anstelle des Greedy-Algorithmus soll ein Ganzzahliges Programm zum optimalen Lösen der minimalen Spannbaum-Instanzen in Hypergraphen aufgestellt und gelöst werden. Hierzu soll das Problem als ganzzahliges lineares Program forumliert und mit Hilfe eines existierenden Solvers, z.B. CPLEX oder Gurobi, gelöst werden.

3.1.1 Schnelle 1.5-Approximationen

Bei den folgenden 2 Aufgaben geht es darum, in eine 1.5-Approximation innerhalb einer Laufzeit von $O(|S| \log |S|)$ zu finden, was eine Verwendung des Hanan-Gitters der Größe $O(|S|^2)$ ausschließt. Statt dessen soll eine Delaunay-Triangulierung genutzt werden, die linear viele Kanten hat, falls die Terminale in allgemeiner Lage sind, d.h. kein Terminalpaar liegt auf einer Geraden mit der Steigung ± 1. Letzteres kann durch leichte Perturbation erreicht werden.

Aufgabe 3: Berechnen sie einen approximativen Steiner-Baum, dessen Länge durch die Länge eines minimalen Spannbaums beschränkt ist mit Hilfe einer Delaunay-Triangulierung und des MST-H Algorithmus von Liu et al.:

Chih-Hung Liu, Shih-Yi Yuan, Sy-Yen Kuo, and Szu-Chi Wang. 2009. High-performance obstacle-avoiding rectilinear steiner tree construction. ACM Trans. Des. Autom. Electron. Syst. 14, 3, Article 45 (June 2009).

Anschließend soll der Baum nachoptimiert werden, indem L-Strukturen umgeklappt werden, oder Liniensegmente verschoben werden, wenn dies die Länge verkürzt.

Aufgabe 4: Wie Aufgabe 3, aber unter Verwendung des theoretisch schnelleren Algorithmus von Mehlhorn et al. zum Bestimmen eines Steiner-Baums in der Delaunay-Triangulierung.

Kurt Mehlhorn. 1988. A faster approximation algorithm for the Steiner problem in graphs. Inf. Process. Lett. 27, 3

3.2 Algorithmen in Graphen

Aufgabe 5: Der iterative 1-Steiner-Algorithmus setzt iterativ ein oder mehrere Steinerpunkte ein und berechnet einen minimalen Spannbaum auf der modifizierten Terminalmenge. Dadurch

transformiert er einen minimalen Spannbaum sukzessive in einen meist viel kürzeren Steiner-Baum. Die effizienteste Variante hat kubische Laufzeit auf ℓ_1-Instanzen.

Bei der Implementierung ist es wichtig, Spannbäume auf den leicht wechselnden Knotenmengen schnell inkrementell zu berechnen. Durch heuristisches ausdünnen der Steinerpunkkandidatenmenge soll die Laufzeit per Parameter beschränkt werden können, um größere Instanzen zu lösen.

Closing the gap: near-optimal Steiner trees in polynomial time. Jeff Griffith, Gabriel Robins, Jeffrey S. Salowe, and Tongtong Zhang. IEEE Trans. on CAD of Integrated Circuits and Systems 13(11):1351–1365 (1994)

Anschließend soll die Lösung durch lokale Suche verbessert werden: *Eduardo Uchoa, Renato Fonseca F. Werneck: Fast Local Search for Steiner Trees in Graphs. Proceedings of the Twelfth Workshop on Algorithm Engineering and Experiments, ALENEX 2010, Austin, Texas, USA, 2010*

Aufgabe 6: Wie in Aufgabe 5 soll hier lokale Suche verwendet werden, allerdings ausgehend von einem durch Mehlhorns Algorithmus bestimmten 2-optimalen Baum in Graphen.

Kurt Mehlhorn. 1988. A faster approximation algorithm for the Steiner problem in graphs. Inf. Process. Lett. 27, 3.

Eduardo Uchoa, Renato Fonseca F. Werneck: Fast Local Search for Steiner Trees in Graphs. Proceedings of the Twelfth Workshop on Algorithm Engineering and Experiments, ALENEX 2010, Austin, Texas, USA, 2010

Aufgabe 7: Wie in Aufgabe 5 soll hier lokale Suche verwendet werden, ausgehend von einem durch Zelikovskys Algorithmus bestimmen initialen Baum.

A.. Z. Zelikovsky: An 11/6-approximation algorithm for the network steiner problem. Algorithmica, 1993, Volume 9, Issue 5, pp 463–470

Eduardo Uchoa, Renato Fonseca F. Werneck: Fast Local Search for Steiner Trees in Graphs. Proceedings of the Twelfth Workshop on Algorithm Engineering and Experiments, ALENEX 2010, Austin, Texas, USA, 2010

Aufgabe 8: Wie in Aufgabe 5 soll hier lokale Suche verwendet werden, ausgehend von einem durch einen randomisierten Algorithmus bestimmen initialen Baum.

Hans Jürgen Prömel und Angelika Steger: A new approximation algorithm for the Steiner tree problem with performance ratio 5/3, Journal of Algorithms, 36: 89-101.

Eduardo Uchoa, Renato Fonseca F. Werneck: Fast Local Search for Steiner Trees in Graphs. Proceedings of the Twelfth Workshop on Algorithm Engineering and Experiments, ALENEX 2010, Austin, Texas, USA, 2010

Aufgabe 9 (2 Personen): Die in der Praxis schnellsten Algorithmen Steinerbäume in Graphen stammen von Polzin und Daneshmand.

Tobias Polzin, Siavash Vahdati Daneshmand: Improved algorithms for the Steiner problem in networks, Discrete Applied Mathematics, Volume 112, Issues 1–3, 15 September 2001, Pages 263-300.

Ein wesentlicher Schritt ist die Reduktion der Graph-Instanzen. Im Rahmen dieses Projektes sollen untere Schranken mittels Dual-Ascent (Kapitel 2.2.2), die Reduktionen aus Kapitel 3, sowie Steiner-Baum-Heuristiken aus Abschnitt 4 (vor allem 4.3) implementiert werden. Die reduzierten Graphen sollen als SteinerLib-Instanzen herausgeschrieben werden, um sie auch anderen Teilnehmern zur Verfügung zu stellen.

Einführungsaufgabe: Einleseroutine + Dual-Ascent (2.2.2) + Heuristik (4.3)

3.2.1 Dynamische Programmierung

Viele exakte Algorithmus basieren auf dynamischer Programmierung und sind Verbesserungen des sogenannten Dreyfus-Wagner Algorithmus. Um auch große Instanzen zumindest heuristisch lösen zu können, sollten alle auf dynamischer Programmierung basierende Implementierungen in einen minimalen Spannbaum iterativ durch optimales Lösen kleiner Teilbäume zu verbessern (siehe Aufgabe 1)

Aufgabe 10: Implementieren Sie ein dynamisches Programm mit polynomiellen Speicherverbrauch: *Fomin, Fedor V., Fabrizio Grandoni, and Dieter Kratsch. "Faster Steiner tree computation in polynomial-space." Algorithms-ESA 2008. Springer Berlin Heidelberg, 2008. 430-441.*

Aufgabe 11: Implementieren Sie das dynamische Programm von Jens Vygen, welches für bestimmte Instanzklassen der schnellste bekannte Algorithmus ist.

Jens Vygen. 2011. Faster algorithm for optimum Steiner trees. Inf. Process. Lett. 111, 21-22.

Aufgabe 12: Für wenige Terminale ist das dynamische Programm von Fuchs et al. theoretisch schneller als Vygen, welches im Rahmen dieser Aufgabe implementiert werden soll:

B. Fuchs, W. Kern, D. Molle, S. Richter, P. Rossmanith, and X. Wang. 2007. Dynamic Programming for Minimum Steiner Trees. Theor. Comp. Sys. 41, 3 (October 2007), 493-500.

3.3 Backup

Aufgabe 13: Implementieren sie einen exakten Algorithmus basierend auf ganzzahliger Optimierung nach

Koch, T. and Martin, A. (1998), Solving Steiner tree problems in graphs to optimality. Networks, 32: 207–232.

Aufgabe 14: Implementieren Sie den Approximationsalgorithmus für Reach-Aware Steinerbäume nach

Stephan Held and Sophie T. Spirkl: "A Fast Algorithm for Rectilinear Steiner Trees with Length Restrictions on Obstacles", ISPD 2014

Spätester Abgabetermin für das Programmierpraktikum ist der 13.07.2014, 24 Uhr.
Am Ende des Semesters muus jeder Teilnehmer seine Implementierung im Rahmen eines Block-Seminars vorgestellt. Hierbei sollen der Algorithmus, die interessantesten Code-Fragmente, sowie experimentelle Ergebnisse vorgestellt werden.