Programmierpraktikum Diskrete Optimierung
Facility-Location-Probleme

Stephan Held
held@or.uni-bonn.de

Sommersemester 2016

1 Problemformulierung

Wir betrachten in diesem Praktikum Standortprobleme (engl.: Facility-Location-Problem) (siehe auch [9], Kapitel 22). Hierbei ist eine endliche Menge von Kunden gegeben, die von einem Standort (z.B. einer Fabrik) aus bedient werden müssen. Dabei fallen Kosten für das Einrichten eines Standorts sowie Kosten für das Anbinden eines Kunden an einen Standort an. Die Standorte und die Kundenanbindungen sollen so gewählt werden, dass die Gesamtkosten minimiert werden.

Es gibt viele Varianten dieses Problems, die sich z.B in der Kostenfunktionen, den erlaubten Standorten oder der Anzahl der Kunden, die ein Standort bedienen kann, unterscheiden.

Wir betrachten Standortprobleme im Chip-Design, wo die Kunden den Registern (1-Bit-Speicherelementen) auf dem Chip entsprechen. Um Strom zu sparen, werden die Register geclustered und von einem komplexen Clock-Buffer getrieben, welche den Standorten entsprechen. Die Register werden möglichst nah an den Clock-Buffer herangezogen und die quadratische Bewegung soll dabei minimiert werden. Ein Clock-Buffer kann nur eine begrenzte Zahl von Registern bedienen und hat selbst einen recht hohen Stromverbrauch.

Mathematisch kann das das Clustern wie folgt modelliert werden.

Instanz: Gegeben sind eine endliche Menge $\mathcal{D} \subset \mathbb{R}^2$ von Kunden, eine Menge $\mathcal{F} = \mathbb{R}^2$ von möglichen Standorten, Fixkosten $f \in \mathbb{R}_+$ für die Bereitstellung eines Standorts, Servicekosten c_{ij} für jedes $i \in \mathcal{F}$ und $j \in \mathcal{D}$ und eine Kapazitätsschranke $u \in \mathbb{R}_+ \cup \{\infty\}$.

Aufgabe: Gesucht sind eine Multi-Menge $I = \{i_1, \dots, i_k\} \subset \mathcal{F}$ von Standorten und eine Zuordnung $x : \mathcal{D} \to I$, welche die Kapazitätsschranke einhält

$$|x^{-1}(i)| \leq u \quad \text{für alle } i \in I, \tag{1}$$

und dabei die Gesamtkosten

$$k \cdot f + \sum_{d \in \mathcal{D}} c_{x(d)d} \tag{2}$$

minimiert.

Multi-Menge bedeutet hier, dass mehrere Standorte auf derselben Lokation in $\mathbb{R}^2$ eröffnet werden dürfen. Wir sprechen auch von SCHWACH-BESCHRÄNKTEN STANDORTPROBLEMEN. Den ersten Teil der Kosten $k \cdot f$ bezeichnen wir auch als *Standortkosten* und den zweiten Teil $\sum_{d \in \mathcal{D}} c_{x(d)d}$ als *Service-Kosten*. Sofern nicht anders erwähnt, betrachten wir bei allen Aufgaben die Kostenfunktion $c_{ij} = \|i - j\|^2$ betrachtet, wobei $\|.\|$ die Euklidische ist. In einer optimalen Lösung liegt der jeder Standort $i \in I$ im Schwerpunkt seiner Kunden $x^{-1}(i)$.

Die meisten Varianten des Standortproblems, insbesondere die hier betrachteten, sind NP-schwer. Das heißt das wir heutzutage nur sehr kleine Instanzen optimal lösen können.

Eine andere populäre Kostenfunktion ist die (nicht-quadrierte) l_1-Norm, bei welcher jeder geöffnete Standort im Median seiner Kunden liegt.

2 Eingabe-Daten

Die Problem-Instanzen liegen im TSPLIB-Format für geometrische Instanzen vor:
`http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/DOC.PS`.

Wir verwenden nur die Kostentypen `EUC_2D`, `CEIL_2D` oder `MAN_2D`. Wir werden unabhängig vom Kostentyp immer mit (quadrierten) Euklidischen Abständen rechnen.

In diesem Format gibt es insbesondere einen Abschnitt `NODE_COORD_SECTION`, in dem die Kundenkoordinaten in Zeilen mit drei Integer-Zahlen der Form:
`<Kunden-Nr.> <x> <y>`
aufgelisted sind. Instanzen werden hier veröffentlicht: `http://www.or.uni-bonn.de/lectures/ss16/praktikum_ss16.html`

3 Programm

Das Programm soll den Dateinamen mit der Instanz sowie die Parameter u und f aus der Kommandozeile lesen. Der Aufruf erfolgt dann wie folgt

`programm <Instanzdateiname> {-f <int>} {-u <int>}.`

Wenn u nicht gesetzt ist, interpretieren wir dies als unbeschränktes Problem $u = \infty$. Wenn f nicht gesetzt ist, wird dies als $f = 2 * \max\{\|j - j'\|^2 : j, j' \in \mathcal{D}\}$ interpretiert.

Das Programm sollte die Lösung in folgendem Format in die Standardausgabe schreiben:

1. Eine Zeile, welche die Gesamtkosten angibt:
   `OBJECTIVE <Gesamtkosten>`
2. Für jeden geöffneten Standort eine Zeile mit der Nummer $(1, \ldots, |I|)$ und den Koordinaten der Facility:
   `FACILITY <Facility-Nr.> <x> <y>`
3. gefolgt von den Kunden, die diesem Standort zugeordnet sind:
   `ASSIGN  <Kunden-Nr.> <Facility-Nr.>`

Programmiersprache

Die Implementierungen müssen in C/C++ ausgeführt werden und unter Linux mit dem clang/gcc/g++ kompilierbar sein, d.h. sie müssen dem ANSI C11/C++11-Standard genügen. Empfohlen wird C++!!!

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) sowie dem Clang-Analyzer (http://clang-analyzer.llvm.org/) auf Fehler überprüft, welche zu Abzügen führen.

Eine mathematische Einführung in C++ findet sich insbesondere in dem Buch "Algorithmische Mathematik" von Hougardy und Vygen [7].

Einführungsaufgabe

Zum Einstieg implementieren Sie einen optimalen Algorithmus für kleine Instanzen durch Enumeration aller Partitionen von $\mathcal{D}$ in $\left\lceil \frac{|\mathcal{D}|}{u} \right\rceil$ bis$|\mathcal{D}|$ Partitionsmengen. Jede Partition definiert eine Lösung, indem für jede Partitionsmenge ein Standort geöffnet wird, der im Schwerpunkt seiner Partitionsmenge platziert wird.

Spätester Abgabetermin für die Einführungsaufgabe ist der 17.04.2016, 24 Uhr.

Wer die Einführungsaufgabe nicht meistert, ist durchgefallen!

4 Aufgaben

4.1 Un-/Schwach-Beschränkte Standortprobleme

Bei den Aufgaben in diesem Abschnitt soll zunächst ein (schwach-beschränktes) Standortproblem mit einer endlichen Menge von möglichen Standorten approximiert werden. Hierbei bietet es sich zum Beispiel an, alle Kundenlokationen als möglichen Standort auszuwählen.

Ist die Zuordnung mittels eines Facility-Location-Algorithmus gefunden, werden die Standorte abschließend bei fester Zuordnung optimiert, d.h. in den Schwerpunkt ihrer jeweiligen Kunden gelegt.

Aufgabe 1: Implementieren und vergleichen sie die Algorithmen nach Jain-Vazirani und Dual Fitting um ein Soft-Capacitated Facility-Location-Problem zu lösen, wie es in Korte und Vygen [3] in Satz 22.22 sowie Abschnitt 22.3 beschrieben ist.

Aufgabe 2: Implementieren und vergleichen Sie Dual Fitting mit Dual Fitting und Skalierung & Greedy Augmentierung, ein Soft-Capacitated Facility-Location-Problem zu lösen, wie es in Korte und Vygen [3] in Satz 22.22 sowie den Abschnitten 22.3 und 22.4 beschrieben ist.

Aufgabe 3: Wenden Sie für $k = \left\lceil \frac{|\mathcal{D}|}{u} \right\rceil, \ldots \left(\left\lceil \frac{|\mathcal{D}|}{u} \right\rceil + \log |\mathcal{D}| \right)$ den k-bounded Jain-Vazirani-Algorithmus an und geben Sie die beste Lösung aus, wie er in Abschnitt 22.5 auf Korte und Vygen [3] beschrieben ist.

Aufgabe 4: Verwenden Sie die lokale Suche nach Arya, um ein Soft-Capacitated Facility-Location-Problem zu lösen, wie es in Korte und Vygen [3] in Satz 22.22 sowie Abschnitt 22.6 beschrieben ist.

Aufgabe 5: Verwenden Sie den Algorithmus von Vygen für das Universal-Facility-Location-Problem, um das Soft-Capacitated Facility-Location-Problem zu lösen. Dieser ist in Korte und Vygen [3] Abschnitt 22.8 beschrieben.

4.2 k-Means & Lloyds Algorithm

Beim k-Means-Problem ist die Anzahl der Standorte fest vorgegeben und es wird versucht die Standorte so zu platzieren und Kunden zuzuordnen, dass die Kosten minimiert werden.

Basis aller Implementierung in diesem Abschnitt ist Lloyd's Algorithmus für k-Mean. Dieser führt unter der Annahme $u = \infty$ iterativ die folgenden Schritte auf einer beliebigen initialen Lösung durch:

1. Optimiere alle Standorde bei fester Zuordnung,
2. Optimiere die Zuordnung bei festen Standorten (unter der Annahme $u = \infty$,

bis nur noch marginale Verbesserungen erzielt werden.

Die folgenden Implementierungen sollten für $k = \left\lceil \frac{|\mathcal{D}|}{u} \right\rceil, \ldots \left\lceil \frac{|\mathcal{D}|}{u} \right\rceil + \log |\mathcal{D}|$

1. eine Lösung des k-Means-Problems bestimmen,
2. solange Standorte mit Kapazitätsverletzungen aufspalten, bis (1) erfüllt ist,
3. alle Standorte bei fester Zuordnung optimal platzieren

und am Ende die beste Lösung ausgeben.

Aufgabe 6: Implementieren und vergleichen Sie die Beschleunigungen nach Elkan [5] mit Heap-Ordered k-Means nach Hamerly & Drake [6]. (Abschnitte 2.4.3 und 2.5 aus [6]).

Aufgabe 7: Implementieren und vergleichen Sie die Beschleunigungen nach Drake & Hamerly [4] mit dem Annular Center Sorting by Norm nach Hamerly & Drake [6]. (Abschnitte 2.4.5 und 2.4.6 aus [6]).

Aufgabe 8: Implementieren Sie den k-means++ Algorithmus nach Arthur and Vassivitskii [2].

Aufgabe 9: Bei dieser Aufgabe geht es darum Lloyd's Algorithmus durch eine bessere Startlösung zu verbessern, wie es von Ostrovsky, Rabani, Schulman und Swamy vorgeschlagen

wird [11]. Lloyd's Algorithmus darf elementar implementiert werden.

Aufgabe 10: In dieser verwenden Sie das Approximations-Schema nach Arora, Raghavan und Rao [1] für das k-Median-Problem ($c_{ij} = \|i - j\|_1$ anstelle der quadrierten euklidischer Kosten), um die Zuordnung zu bestimmen, deren Standorte Sie anschließend, wie bei den anderen Aufgaben, bei fester Zuordnung bzgl. der quadratischen Abstände optimieren.

4.3 Iteratives Minimum-Cost-Flow & Mean-Placement

Ein Problem an LLoyds Algorithmus ist, dass die Kapazitäten zunächst nicht eingehalten werden. Dieses Problem kann behoben werden, indem die Zuordnung bei festen Standorten unter Einhaltung der Kapazitätsschranken mit einem Minimum-Cost-Flow-Algorithmus gelöst wird, anstatt die Kunden einfach am nächsten Standort anzuschließen.

Aufgabe 11: Verwenden Sie den Successive-Shortest-Path-Algorithmus [3], Abschnitt 9.4

Aufgabe 12: Verwenden Sie den Netzwerk-Simplex-Algorithmus [3], Abschnitt 9.6.

4.4 Baumkosten

Aufgabe 13: In dieser Aufgabe weicht die Problemstellung etwas ab. Die Standort-Kosten bestehen weiterhin aus $k|I|$. Für die Service-Kosten wird für jeden Standort und seine Kunden ein l_1-minimaler Spannbaum bestimmt. Die Service-Kosten bestehen aus der Summe der Spannbaum-Längen.

Als Basis dient Algorithmus A aus "Approximation Algorithms for Network Design and Facility Location with Service Capacities" von Maßberg and Vygen [10]. Der Spannbaum-Algorithmus sollte auf n Senken in der in der Zeit $\mathcal{O}(n \log n)$ laufen. Dazu kann z.B. eine Delaunay-Triangulierung verwendet werden (siehe [8]). Anders als in [10] gilt die einfache Kapazitätsschranke (1) und Steinerbäume dürfen durch Spannbäume approximiert werden.

5 Abgabe und Abschlussvortrag

Spätester Abgabetermin für die Einführungsaufgabe ist der 17.04.2016, 24 Uhr.
Spätester Abgabetermin für das Programmierpraktikum ist der 17.07.2016, 24 Uhr.
Am Ende des Semesters muss jeder Teilnehmer seine Implementierung im Rahmen eines Block-Seminars vorgestellen. Hierbei sollen in 15 Minuten der Algorithmus, die interessantesten Code-Fragmente, sowie experimentelle Ergebnisse vorgestellt werden. Voraussichtlicher Termin für das Blockseminar ist der 22.07.2016.

Literatur

[1] Sanjeev Arora, Prabhakar Raghavan, and Satish Rao, *Approximation schemes for euclidean k-medians and related problems*, Proceedings of the thirtieth annual ACM symposium on Theory of computing, ACM, 1998, pp. 106–113.

[2] David Arthur and Sergei Vassilvitskii, *k-means++: The advantages of careful seeding*, Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[3] B. Korte and J. Vygen, *Combinatorial Optimization*, 5th edition, Springer, 2012.

[4] Jonathan Drake and Greg Hamerly, *Accelerated k-means with adaptive distance bounds*, 5th NIPS workshop on optimization for machine learning, 2012.

[5] Charles Elkan, *Using the triangle inequality to accelerate k-means*, ICML, vol. 3, 2003, pp. 147–153.

[6] Greg Hamerly and Jonathan Drake, *Accelerating lloyd's algorithm for k-means clustering*, Partitional Clustering Algorithms, Springer, 2015, pp. 41–78.

[7] S. Hougardy and J. Vygen, *Algorithmische Mathematik*, Springer, 2015.

[8] Michael Jünger, Volker Kaibel, and Stefan Thienel, *Computing delaunay triangulations in manhattan and maximum metric*, Institut für Informatik, Universität zu Köln, 1995.

[9] B. Korte and J. Vygen, *Combinatorial optimization: Theory and algorithms*, 5th ed., Springer, 2012.

[10] Jens Maßberg and Jens Vygen, *Approximation algorithms for a facility location problem with service capacities*, ACM Transactions on Algorithms (TALG) **4** (2008), no. 4, 50.

[11] Rafail Ostrovsky, Yuval Rabani, Leonard J Schulman, and Chaitanya Swamy, *The effectiveness of lloyd-type methods for the k-means problem*, Journal of the ACM (JACM) **59** (2012), no. 6, 28.