



Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

Approximationsalgorithmen

Klausuranmeldung

- Bis Freitag, 30.7.

Approximationsalgorithmen

Last Balanzierung

- m identische Maschinen $\{1, \dots, m\}$
- n Aufgabe $\{1, \dots, n\}$
- Job j hat Länge $t(j)$
- Aufgabe: Platziere die Aufgabe auf den Maschinen, so dass diese möglichst „balanziert“ sind

- Sei $A(i)$ die Menge der Aufgaben auf Maschine i
- Sei $T(i) = \sum_{j \in A(i)} t(j)$

- Makespan: $\max T(i)$
- Präzise Aufgabe: Minimiere Makespan

Approximationsalgorithmen

GreedyLoadBalancing

1. Setze $T(i)=0$ und $A(i)=\emptyset$ für alle Maschinen $i \in \{1, \dots, m\}$
2. **for** $j=1$ **to** n **do**
3. Sei $M(i)$ eine Maschine mit $T(i) = \min_{k \in \{1, \dots, m\}} T(k)$
4. Weise Aufgabe j Maschine i zu
5. $A(i) \leftarrow A(i) \cup \{j\}$
6. $T(i) \leftarrow T(i) + t(j)$

Approximationsalgorithmen

Satz 84

- Algorithmus GreedyLoadBalancing hat ein Approximationsverhältnis von mindestens $2 - 1/m$.

Beweis

- GreedyLoadBalancing verteilt zunächst die kurzen Aufgaben gleichmäßig
- Danach wird die lange Aufgabe zugewiesen
- Makespan: $2m-1$

Maschine 1: ■■

Maschine 2: ■■

Maschine 3: ■■ ■■■■

Approximationsalgorithmen

Satz 84

- Algorithmus GreedyLoadBalancing hat ein Approximationsverhältnis von mindestens $2 - 1/m$.

Beweis

- Damit ist das Approximationsverhältnis mindestens $(2m-1)/m = 2 - 1/m$.

Approximationsalgorithmen

Beobachtung

- Für jede Problem Instanz ist der optimale Makespan mindestens

$$T^* = \frac{1}{m} \sum_{j=1}^n t(j)$$

- Begründung: Bestenfalls können wir die Aufgaben genau auf die m Maschinen aufteilen und jede Maschine hat Last Gesamtlast/Anzahl Maschinen

Approximationsalgorithmen

Satz 85

- Algorithmus GreedyLoadBalancing ist ein 2-Approximationsalgorithmus für das Lastbalanzierungsproblem.

Beweis

- Sei i^* die Maschine, die maximale Last in der vom Algorithmus berechneten Zuteilung erhält
- Sei j^* die Aufgabe, die Maschine i^* als letzte zugewiesen wurde
- Es gilt: $T(k) \geq T(i^*) - t(j^*)$ für alle Maschinen k , da zum Zeitpunkt der Zuweisung von j^* , $T(i^*)$ Minimum der $T(k)$ war
- Somit folgt für die Kosten Opt einer optimalen Zuweisung:

$$Opt \geq \frac{1}{m} \sum_{k=1}^m T(k) \geq T(i^*) - t(j^*)$$

Approximationsalgorithmen

Satz 85

- Algorithmus GreedyLoadBalancing ist ein 2-Approximationsalgorithmus für das Lastbalanzierungsproblem.

Beweis

- Außerdem gilt sicher $\text{Opt} \geq t(j^*)$
- Es folgt

$$T(i^*) = (T(i^*) - t(j^*)) + t(j^*) \leq 2\text{Opt}$$

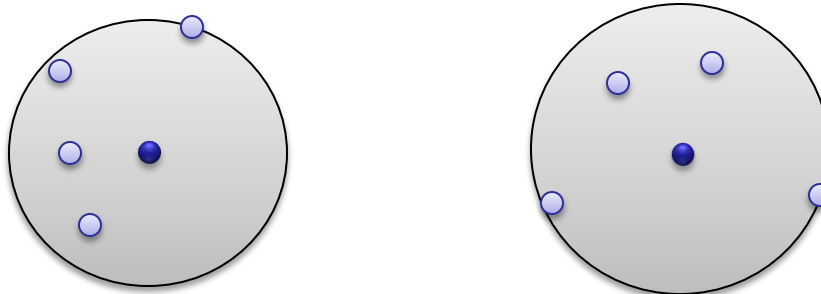
Approximationsalgorithmen

Das (diskrete) k-Center Clustering Problem

- Gegeben: Menge P von n Punkten in der Ebene
- Gesucht: Menge $C \subseteq P$ von k Zentren, so dass die maximale Distanz der Punkte zum nächstgelegenen Zentrum, d.h.
$$\text{cost}(P,C) = \max_{p \in P} d(p,C)$$
 minimiert wird, wobei
- $\text{dist}(p,C) = \min_{q \in C} \text{dist}(p,q)$ und $\text{dist}(p,q)$ bezeichnet den Abstand von p und q (Euklidische Distanz)

Approximationsalgorithmen

Beispiel



Alternative Formulierung

Finde k Scheiben mit Zentrum aus P , die alle Punkte abdecken und deren Maximaler Radius minimiert wird.

Approximationsalgorithmen

Typische Anwendung

- Punkte symbolisieren Städte
- Will Mobilfunkmasten mit möglichst geringer Leistung aufstellen, so dass alle Städte versorgt sind

Allgemeiner

- Punkte (typischerweise in höheren Dimensionen) sind „Beschreibungen von Objekten“
- Will Objekte in Gruppen von ähnlichen Objekten unterteilen

Approximationsalgorithmen

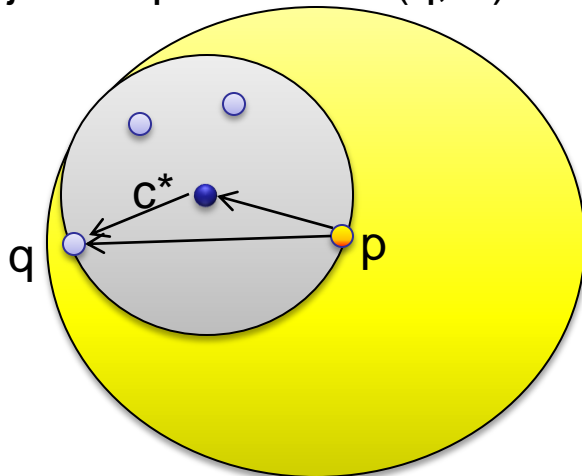
Ein Gedankenexperiment

- Nehmen wir an, wir kennen die Kosten r einer optimalen Lösung, d.h. wir wissen, dass man mit Scheiben mit Radius r und Zentrum aus P die Punkte abdecken kann
- Wir werden zeigen, dass wir dann einen einfachen 2-Approximationsalgorithmus finden können

Approximationsalgorithmen

Idee

- Wir nutzen Existenz von Lösung C^* mit Radius (Kosten) r
- Betrachte Punkt $p \in P$
- Dann gibt es Zentrum $c^* \in C^*$ mit $\text{dist}(p, c^*) \leq r$
- Nehmen wir nun p als Zentrum anstelle von c^* und verdoppeln wir den Radius, so decken wir jeden Punkt q ab, der von c^* mit Radius r abgedeckt wurde, d.h.
- für jedes $q \in P$ mit $\text{dist}(q, c^*) \leq r$ gilt $\text{dist}(p, q) \leq \text{dist}(p, c^*) + \text{dist}(c^*, q) \leq 2r$



Approximationsalgorithmen

k-Center1(P,k)

1. $C = \emptyset$; $P' \leftarrow P$
2. **while** $P' \neq \emptyset$ **do**
3. Wähle beliebigen Punkt $p \in P'$
4. $C = C \cup \{p\}$
5. Lösche alle Punkte aus P' mit Distanz höchstens $2r$ von p
6. **if** $|C| \leq k$ **then return** C
7. **else** return „Es gibt keine Menge von k Zentren mit Radius r “

Offensichtlich gilt

Jede Menge von k Zentren, die der Algorithmus zurückgibt hat Kosten höchstens $2r$.

Approximationsalgorithmen

Lemma 86

- Wenn Algorithmus k-Center1 mehr als k Zentren auswählt, dann gilt für jede Menge $C^* \subseteq P$ von k Zentren, dass $\text{cost}(P, C^*) > r$ ist.

Beweis (durch Widerspruch)

- Annahme: Es gibt C^* mit $\text{cost}(P, C^*) \leq r$ und $|C^*| \leq k$ und $|C| > k$.
- Sei C die Menge der Zentren, die k-Center1 auswählt
- Da $C \subseteq P$ gibt es für jedes $p \in C$ (mindestens) ein $c^* \in C^*$ mit $\text{dist}(p, c^*) \leq r$
- Wir nennen c^* nah zu p
- Behauptung: Kein c^* kann nah zu zwei $p \in C$ sein (Beweis später)
- Damit folgt: $|C^*| \geq |C|$ und somit Widerspruch zu $|C^*| \leq k$ und $|C| > k$.

Approximationsalgorithmen

Lemma 86

- Wenn Algorithmus k-Center1 mehr als k Zentren auswählt, dann gilt für jede Menge $C^* \subseteq P$ von k Zentren, dass $\text{cost}(P, C^*) > r$ ist.

Beweis (durch Widerspruch)

- Behauptung: Kein c^* kann nah zu zwei $p \in C$ sein
- Beweis der Behauptung:
- Alle Paare von Zentren p, q aus C haben Abstand $> 2r$
- Wäre nun für ein Zentrum c^* $\text{dist}(p, c^*) \leq r$ und $\text{dist}(q, c^*) \leq r$, so würde $\text{dist}(p, q) \leq \text{dist}(p, c^*) + \text{dist}(c^*, q) = \text{dist}(p, c^*) + \text{dist}(c^*, q) \leq 2r$ gelten. Widerspruch!

Approximationsalgorithmen

Was, wenn wir r nicht kennen?

- Wir wissen nicht, welche Punkte Distanz größer als $2r$ von den bisher ausgewählten Zentren haben
- Idee: Wähle immer den am weitesten entfernten Punkt, d.h. der $\text{dist}(p,C)$ maximiert
- Gibt es einen Punkt mit $\text{dist}(p,C) > 2r$, dann ist es dieser

Approximationsalgorithmen

k-Center2(P,k)

1. Wähle beliebigen Punkt p aus P und setze $C=\{p\}$
3. **while** $|C|<k$ **do**
3. Wähle Punkt $p\in P$, der $\text{dist}(p,C)$ maximiert
4. $C=C\cup\{p\}$
5. **return** C

Approximationsalgorithmen

Satz 87

- Algorithmus k-Center2 ist ein 2-Approximationsalgorithmus für das diskrete k-Center Clustering Problem. Algorithmus k-Center2 kann mit Laufzeit $O(nk)$ implementiert werden.

Beweis

- Zunächst zur Laufzeit:
- Um den Algorithmus in $O(nk)$ Zeit zu implementieren, müssen wir jeden Schleifendurchlauf in $O(n)$ Zeit erledigen können. Dazu speichern wir uns für jeden Punkt p den Wert $\text{dist}(p,C)$.
- Wird nun ein neues Zentrum c in C eingefügt, so müssen wir nur für jeden Punkt überprüfen, ob $\text{dist}(p,c) < \text{dist}(p,C)$ ist und ggf. $\text{dist}(p,C)$ aktualisieren. Dies geht insgesamt in $O(n)$ Zeit.

Approximationsalgorithmen

Satz 87

- Algorithmus k-Center2 ist ein 2-Approximationsalgorithmus für das diskrete k-Center Clustering Problem. Algorithmus k-Center2 kann mit Laufzeit $O(nk)$ implementiert werden.

Beweis

- Wir bezeichnen nun mit r die Kosten einer optimalen Lösung C^* .
Annahme: Algorithmus liefert Menge C mit Kosten $>2r$.
- Dann gibt es einen Punkt p mit $\text{dist}(p,C) > 2r$
- Da der Algorithmus immer den Punkt auswählt, der maximalen Abstand zu den bisher ausgewählten Zentren hat, haben alle ausgewählten Zentren Abstand $>2r$ zur den bisher ausgewählten Zentren
- Somit würde Algorithmus k-Center1 auf dieser Eingabe mehr als k Zentren zurückgeben. Damit gilt nach Lemma 86 $\text{cost}(P,C^*) > r$. Widerspruch!

Approximationsalgorithmen

Zusammenfassung & Kommentare

- Man kann viele Probleme approximativ schneller lösen als exakt (für die drei Beispiele sind keine Algorithmen mit Laufzeit $O(n^c)$ für eine Konstante c bekannt)
- Gierige Algorithmen sind häufig Approximationsalgorithmen