

Cluster Probleme

Clustering: Bei großen Datenmengen die ähnlichen Daten
– d.h. mit paarweise kleinen Distanzwerten – zu gruppieren.

Das k -CENTER Problem

Eingabe: eine Zahl $k \in \mathbb{N}$, und eine Menge K von Punkten mit einer Metrik (Distanzwerten) $d : K \times K \rightarrow \mathbb{R}_{\geq 0}$

Ausgabe: eine Menge $C \subseteq K$ von k Zentren, so dass die *maximale Distanz* von C (vom *nächstliegenden* Zentrum) über alle Punkte in K minimiert wird

$$\max_{v \in K} d(v, C) \text{ minimal}$$

$$[d(v, C) = \min_{c \in C} d(v, c)]$$

Ein Greedy Algorithmus für k -CENTER

sei $v_1 \in K$ ein beliebiger Punkt, und $C := \{v_1\}$

FOR $i = 2$ TO k DO

sei $v_i \in K$ der Punkt mit der größten Distanz
zum *nächstliegenden* Zentrum in C

(der Punkt der momentan den Radius bestimmt)

setze $C := C \cup \{v_i\}$

Theorem: Dieser Greedy Algorithmus für k -CENTER ist
2-approximativ.

Theorem: k -CENTER ist nicht α -approximierbar für $\alpha < 2$

Das NP-vollständige DOMINATING SET Problem könnte sonst effizient gelöst werden.

DOMINATING SET

Eingabe: ein Graph $G(V, E)$

Ausgabe: eine kleinste Teilmenge der Knoten $D \subseteq V$ s.d. jeder Knoten mindestens einen Nachbarn in D hat.

Das k -MEDIAN Problem

Eingabe: eine Menge K von Punkten
und eine Metrik (Distanzwerte) $d : K \times K \rightarrow \mathbb{R}_{\geq 0}$

Ausgabe: eine Menge $M \subseteq K$ von k Punkten (Zentren), so dass die *Summe der Distanzen* vom nächstliegenden Zentrum über alle Punkte minimiert wird.

$$\sum_{v \in K} d(v, M) \text{ minimal}$$

Ein Algorithmus mit *Lokaler Suche*

- $i := 0$;
- sei eine Menge $M^{(0)}$ von *beliebigen* k Punkten als Zentren
- WHILE es geeignete $w \in M^{(i)}$ und $u \notin M^{(i)}$ gibt
 ersetze $w \in M^{(i)}$ durch einen Punkt u , falls dies zu einer kleineren
 Summe der Distanzen führt;

 $i := i + 1$;

Theorem: Dieser Algorithmus ist 5-approximativ.

(ohne Beweis)

Lokale Suche für min-VERTEX COVER

sei $G(V, E)$ ein Eingabegraph

- $i := 0$;
- sei $C^{(0)} = V$ die Knotenüberdeckung am Anfang
- WHILE es $v \in C^{(i)}$ gibt so dass $C^{(i)} \setminus \{v\}$ Knotenüberdeckung ist
 - $C^{(i+1)} := C^{(i)} \setminus \{v\}$;
(eine 'benachbarte' Lösung mit besserem Zielwert)
 - $i := i + 1$;

Beispiel für Nachbarschaft: Die Knotenüberdeckungen C und C' sind **benachbarte Lösungen**, wenn C' durch das Hinzufügen oder das Entfernen eines Elements aus C entsteht. Die **Nachbarschaft** $\mathcal{N}(C)$ besteht aus allen mit C benachbarten Lösungen.

Definition: Lokale Suche (*local search*)

Sei ein Minimierungsproblem $P = (\min, f, L)$ zu lösen, wobei zu jeder Lösung y eine Nachbarschaft $\mathcal{N}(y)$ (Menge benachbarter Lösungen) definiert ist.

Strikte Lokale Suche

- sei $y^{(0)}$ eine Lösung für Instanz x ; $i := 0$;
- WHILE es in $\mathcal{N}(y^{(i)})$ eine bessere Lösung gibt DO
 - bestimme einen $y \in \mathcal{N}(y^{(i)})$ so dass $f(y) < f(y^{(i)})$
 - setze $y^{(i+1)} = y$, und $i := i + 1$;
- gib die *lokal optimale Lösung* $y^{(i)}$ aus

Definition: Eine Lösung y ist *lokal optimal* falls es in $\mathcal{N}(y)$ keine Lösung mit besserem Zielwert gibt.

Unterschiede zu Greedy Algorithmen

Lokale Suchverfahren sind zu Greedy Algorithmen ähnlich, da in beiden wird eine (Teil-)Lösung in jedem Schritt *lokal* verbessert.

- Im Greedy Verfahren wird eine Lösung, über *Teillösungen* Schritt für Schritt *aufgebaut*. Es wird immer eine *beste* 'Nachbarlösung' gewählt.

(schnelle Algorithmen)

- In der lokalen Suche haben wir in jedem Schritt eine *vollständige Lösung*, die immer weiter *modifiziert* wird. Die Suche wird nicht mit dem *besten*, sondern mit einem *besseren* Nachbarn fortgesetzt.

(Laufzeit ???)

- In der lokalen Suche $y^{(0)}$ (und $y^{(i+1)}$) werden oft durch Heuristiken, oder zufällig ausgewählt.
- Ob ein lokales Optimum *schnell* gefunden wird, ist nicht klar!

Beispiele für Nachbarschaften

- Im k-MEDIAN waren zwei Mengen von Zentren $M \subseteq K$ und $M' \subseteq K$ benachbart, wenn M' durch das Ersetzen eines Zentrums aus M entstand.
- Im VERTEX COVER $C' \in \mathcal{N}(C)$ wenn $C' = C \cup \{v\}$, oder $C' = C \setminus \{v\}$ für irgendein $v \in V$.

Definition: Wenn nur Elemente aus $\{0, 1\}^n$ als Lösungen in Frage kommen, dann besteht die k-Flip Nachbarschaft $\mathcal{N}_k(y)$ einer Lösung y aus allen y' die in maximum k Positionen von y unterschiedlich sind.

($\{0, 1\}^n$ besteht aus den n -dimensionalen Vektoren über $\{0, 1\}$.)

Die obigen Beispiele sind 1-Flip, bzw. 2-Flip Nachbarschaften.

max-leaf-SPANNING-TREE

Eingabe: ein ungerichteter Graph $G(V, E)$ (ohne Gewichtung)

Ausgabe: ein Spannbaum mit größtmöglicher Blattzahl

Nachbarschaften: Seien zwei Spannbäume von G
 $T'(V, E')$ und $T''(V; E'')$ *benachbarte Lösungen*,
wenn $E'' = (E' \setminus \{e\}) \cup \{f\}$ für zwei Kanten $e, f \in E$
(2-Flip Nachbarschaft)

(falls $f \notin E'$, dann hat $E' \cup \{f\}$ einen Kreis;
entferne eine andere Kante e vom Kreis!
 $\Rightarrow T'$ hat $\leq |E|^2$ Nachbarn)

Lokale Suche für max-leaf-SPANNING-TREE

sei T ein beliebiger Spannbaum für G

WHILE es einen $T' \in \mathcal{N}(T)$ mit mehr Blättern gibt

setze $T := T'$

gib T aus

(Die WHILE Schleife wird max. n -mal durchlaufen, und es gibt stets $\leq m^2$ Nachbarn zu prüfen. \rightarrow effizient)

Theorem: Dieser Algorithmus ist 5-approximativ:

T_{OPT} hat also höchstens 5-mal so viele Blätter wie T_{ALG}

Hier zeigen wir Approximationsfaktor 10

Behauptung 1. Sei k_d die Anzahl der Knoten vom Grad d in einem Baum T . Dann gilt

$$\sum_{d \geq 3} k_d < k_1.$$

Behauptung 2. T_{OPT} hat höchstens $8k_1$ Blätter, die in T_{ALG} Knoten vom Grad 2 sind.

Korollar: Also hat T_{OPT} höchstens $k_1 + 8k_1 + k_1 = 10k_1$ Blätter, (wobei k_1 die Anzahl der Blätter in T_{ALG} ist)
 T_{ALG} ist also 10-approximativ.

Beweis der Behauptung 2.

Definition: Ein *maximaler 2-Weg* durchläuft nur Knoten von Grad 2 als innere Knoten, und endet in einem Blatt, oder in einem Knoten von höherem Grad.

(Es gibt $\leq 2k_1 - 1$ maximale 2-Wege in T_{ALG} .)

Teilbehauptung: Jeder maximale 2-Weg enthält höchstens 4 innere Knoten die in T_{OPT} Blätter sind.

(sonst wäre T_{ALG} nicht lokal optimal)

Korollar: T_{ALG} hat $< 2k_1 \cdot 4$ Knoten vom Grad 2 die in T_{OPT} Blätter sind. (das ist Behauptung 2.)

FACILITY LOCATION

Eingabe:

- eine Menge K von Kunden
- eine Menge S von möglichen Service Stationen
- für jede $s \in S$ die Betriebskosten f_s
- eine Metrik auf $K \cup S$ (Distanzwerte $d(k, s)$)

Ausgabe: Eine Menge $X \subseteq S$ von Service Stationen so dass die Summe aller Anschluss- und Betriebskosten

$$C(X) = \sum_{k \in K} \text{distanz}(k, X) + \sum_{s \in X} f_s$$

minimiert wird.

Definition: Für eine Teilmenge $X \subseteq S$ sind die *Anschlusskosten* von $k \in K$ an X die minimale Distanz:
 $\text{distanz}(k, X) = \min_{s \in X} d(k, s)$

Lokale Suche für FACILITY LOCATION

- sei $X \subseteq S$ eine beliebige Menge von Service Stationen
- WHILE das Entfernen, Hinzufügen, oder Ersetzung einer Service Station zu einer Verbesserung führt
 führe eine beliebige solche Operation aus
- gib die lokal optimale Lösung $X \subset S$ aus

Theorem: (ohne Beweis) Sei X^* ein globales Minimum und X ein lokales Minimum, dann gilt

$$C(X) \leq 3 \cdot C(X^*).$$

(Die lokale Suche ist 3-approximativ.)

Problem: Die lokale Suche für FACILITY LOCATION ist **nicht effizient!**

Theorem: (ohne Beweis) Wenn wir nur Verbesserungen akzeptieren die den aktuellen Wert mindestens um Faktor $\Delta = 1 - \varepsilon/|S|$ reduzieren, erhalten wir eine $(3 + \varepsilon)$ -approximative Lösung.

Bemerkung: Die Ausgabe ist ein sog. $(\varepsilon/|S|)$ -approximatives lokales Optimum.

δ -approximative lokale Optima

Definition: Sei (\min, f, L, A, B) ein polynomielles Suchproblem, und sei y eine Lösung für eine Instanz I . Dann ist y ein *δ -approximatives lokales Optimum*, wenn

$$(1 - \delta) \cdot f(y) \leq f(y')$$

für alle *benachbarten* Lösungen $y' \in \mathcal{N}(y)$ gilt.

Komplexität der lokalen Suche: Kann man ein lokales Optimum (irgendwie) in polynomieller Zeit berechnen?

es hängt vom (Such)problem ab...

Definition: polynomielle Suchprobleme

Ein *polynomielles Suchproblem* (opt, f, L, A, B) ist ein NP-Optimierungsproblem (mit definierten Nachbarschaften), so dass

- es einen effizienten Algorithmus A gibt, der für jede Instanz I eine Anfangslösung y_0 berechnet.
- es gibt einen effizienten Algorithmus B , der für jede Instanz I und Lösung y entscheidet ob y ein lokales Optimum ist; falls nicht, dann bestimmt B eine Nachbarlösung $B(I, y) = y' \in \mathcal{N}(y)$ mit besserem Zielwert.

$\mathcal{PLS} \subset \mathcal{NPO}$ ist die Klasse aller polynomiellen Suchprobleme.

[Standardalgorithmus (lokale Suche) für \mathcal{PLS} -Probleme

- berechne $y = A(I)$
- WHILE y kein lokales Optimum
 setze $y = B(I, y)$]

Definition: PLS-vollständigkeit

Seien $P_1, P_2 \in \mathcal{PLS}$. Das Problem P_1 ist *PLS-reduzierbar* auf P_2 ($P_1 \leq_{\mathcal{PLS}} P_2$), wenn es effiziente Transformationen Φ und Ψ gibt, s. d.

- für jede Instanz I von P_1 ist $\Phi(I)$ eine Instanz von P_2
- für jedes lokale Optimum y für P_2 und Instanz $\Phi(I)$ ist $\Psi(y, I)$ ein lokales Optimum für P_1 und Instanz I

Definition: Ein polynomielles Suchproblem ist *PLS-vollständig*, wenn $Q \leq_{\mathcal{PLS}} P$ für jedes polynomielle Suchproblem Q gilt.

(Es ist *sehr unwahrscheinlich* dass ein PLS-vollständiges problem in Polynomialzeit lokal optimierbar ist: dann wären alle Probleme in \mathcal{PLS} so; es hätte auch schwere Folgen für die Komplexitätstheorie.)

PLS-vollständige Probleme

1. MINIMUM BALANCED CUT

Eingabe: ein Graph $G(V, E)$ mit Kantengewichten $w : E \rightarrow \mathbb{R}_{\geq 0}$

Ausgabe: eine Knotenmenge $W \subset V$ mit $|W| = |V|/2$ s.d.
das Gewicht *kreuzender* Kanten minimal

kreuzende Kante: verbindet W mit $V \setminus W$

mit 2-Flip Nachbarschaft PLS-vollständig

2. **TSP:** die besten Algorithmen für TSP in der Praxis, sind Algorithmen mit lokaler Suche (Kanten werden aus der Rundreise entfernt und durch andere ersetzt).

mit der 2k-Flip Nachbarschaft für große k PLS-vollständig

Verschlechterungen während der Suche

Lokale Suche in variabler Tiefe

Kernighan-Lin Algorithmus für MINIMUM BALANCED CUT

(mit Einfrieren)

sei W eine Anfangslösung mit $|W| = |V|/2$.

REPEAT

$W_1 := W$

FOR $i = 1$ TO $n/2$ DO

- ersetze $w \in W_i$ durch $w' \notin W_i$ s.d. der Gewinn maximal

$$\text{Gewinn}(w, w') = f(W_i) - f((W_i \setminus \{w\}) \cup \{w'\})$$

diese beste Nachbar sei W_{i+1} (kann schlechter als W_i sein!)

- friere w und w' ein

(werden während FOR nicht mehr geändert)

sei W die Lösung in $\{W_1, W_2, \dots, W_{n/2}\}$ mit minimaler $f(W_i)$

UNTIL $W = W_1$ (bis W_1 bleibt minimal)

Der Metropolis Algorithmus

für jedes Minimierungsproblem wo eine Nachbarschaft $\mathcal{N}(y)$ für jede Lösung definiert ist

sei y eine Anfangslösung

WIEDERHOLE hinreichend oft

- wähle zufällig einen Nachbarn $y' \in \mathcal{N}(y)$
- IF $f(y') \leq f(y)$ then $y := y'$
- ELSE $y := y'$ mit Wahrscheinlichkeit

$$p = e^{-\frac{f(y')-f(y)}{T}}$$

(T ist ein 'Temperatur' Parameter)

Simulated Annealing

1. sei y eine Anfangslösung, und T die Anfangstemperatur
2. WIEDERHOLE hinreichend oft
 - wähle zufällig einen Nachbarn $y' \in \mathcal{N}(y)$
 - IF $f(y') \leq f(y)$ then $y := y'$
 - ELSE $y := y'$ mit Wahrscheinlichkeit

$$p = e^{-\frac{f(y')-f(y)}{T}}$$

3. wenn T noch nicht tief genug, reduziere T und GOTO 2.