Algorithmen für Sensornetze — Übung 0

http://i11www.ira.uka.de/teaching/SS_05/algosens

Steffen Mecke (mecke@ira.uka.de)

SS 2005

Übersicht

Graphen

- $G = (V, E), E \subset \{\{u, v\} : u, v \in V\}$ ungerichteter Graph
- $n := |V|, m := |E|$
- inzident, adjazent
- $N(v)$, $d(v)$, $\delta(G)$, $\Delta(G)$
- Adjazenzmatrix
- vollständig, Clique, unabhängig, bipartit
- Weg, Pfad, Zykel, Kreis, zusammenhängend (einfach, stark, schwach)
- Baum, Wald, aufspannender Baum, planar, geometrisch, $G[V']$

Färbungsprobleme

Problem COLOR

Gegeben: Ein Graph $G = (V, E)$ und ein Parameter $K \in \mathbb{N}$.

Frage: Gibt es eine Knotenfärbung von G mit höchstens K Farben, so dass je zwei adjazente Knoten verschiedene Farben besitzen?

Aufgabe: Finde die minimale Färbungszahl von G.

Vollständige Subgraphen

Problem CLIQUE

Gegeben: Ein Graph $G = (V, E)$ und ein Parameter $K \in \mathbb{N}$.

Frage: Gibt es eine Clique der Größe K in G?

Aufgabe: Finde eine maximale Clique in G.

Unabhängige Mengen

Problem INDEPENDENT SET

Gegeben: Ein Graph $G = (V, E)$ und ein Parameter $K \in \mathbb{N}$.

Frage: Gibt es eine Menge I von mindestens K Knoten, so dass keine zwei Knoten in I adjazent sind?

Aufgabe: Finde eine maximale unabhängige Menge in G.

Dominierende Mengen

Problem DOMINATING SET

Gegeben: Ein Graph $G = (V, E)$ und ein Parameter $K \in \mathbb{N}$.

Frage: Gibt es eine Menge D von höchstens K Knoten, so dass jeder Knoten zu einem Knoten aus D inzident oder selbst in D ist?

Aufgabe: Finde eine minimale dominierende Menge in G.

Knotenüberdeckungen

Problem VERTEX COVER

Gegeben: Ein Graph $G = (V, E)$ und ein Parameter $K \in \mathbb{N}$.

Frage: Gibt es eine Menge C von höchstens K Knoten, so dass jede Kante zu einem Knoten aus D inzident ist?

Aufgabe: Finde eine minimale Knotenüberdeckung in G.

Kürzeste Wege

Problem Shortest Path

Gegeben: Ein Graph $G = (V, E)$, Kantenlängen $l\colon E \to \mathbb{R}$, zwei Knoten s und t.

Aufgabe: Finde einen kürzesten Weg von s nach t.

Aufgabe: Finde je einen kürzesten Weg von s zu allen Knoten in G.

Aufgabe: Finde kürzeste Wege für alle Paare (s, t).

Algorithmen, Laufzeit, Komplexität

- $O(n), \Omega(n), \Theta(n), o(n)$
- $\mathrm{poly}(n), \mathrm{polylog}(n)$
- worst case, average case, expected
- $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-vollständig, $\mathcal{NP}$-schwer

Absolute Approximationsalgorithmen

Definition

Sei Π ein Optimierungsproblem. Ein polynomialer Algorithmus $\mathcal{A}$, der für jedes $I \in D_\Pi$ einen Wert $\mathcal{A}(I)$ liefert, mit

$$|\,\mathrm{OPT}(I) - \mathcal{A}(I)| \leq K$$

und $K \in \mathbb{N}_0$ konstant, heißt **Approximationsalgorithmus mit Differenzengarantie** *oder* **absoluter Approximationsalgorithmus**.

Relative Approximationsalgorithmen

Definition

Sei Π *ein Optimierungsproblem. Ein polynomialer Algorithmus* $\mathcal{A}$*, der für jedes* $I \in D_\Pi$ *einen Wert* $\mathcal{A}(I)$ *liefert mit* $R_\mathcal{A}(I) \leq K$*, wobei* $K \geq 1$ *eine Konstante, und*

$$\mathcal{R}_\mathcal{A}(I) := \begin{cases} \frac{\mathcal{A}(I)}{\mathrm{OPT}(I)} & \text{falls } \Pi \text{ Minimierungsproblem} \\ \frac{\mathrm{OPT}(I)}{\mathcal{A}(I)} & \text{falls } \Pi \text{ Maximierungsproblem} \end{cases}$$

heißt **Approximationsalgorithmus mit relativer Gütegarantie**. $\mathcal{A}$ *heißt* ε**–approximativ**, *falls* $\mathcal{R}_\mathcal{A}(I) \leq 1 + \varepsilon$ *für alle* $I \in D_\Pi$.

Approximationsschemata

Definition

Ein (polynomiales) **Approximationsschema** *(***PAS***) für ein Optimierungsproblem* Π *ist eine Familie von Algorithmen* $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$*, so dass* $\mathcal{A}_\varepsilon$ *ein* ε*–approximierender Algorithmus ist (d.h.* $\mathcal{R}_{\mathcal{A}_\varepsilon} \leq 1 + \varepsilon$*) für alle* $\varepsilon > 0$*. Dabei bedeutet polynomial wie üblich polynomial in der Größe des Inputs* I*.*

Ein Approximationsschema $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ *heißt* **vollpolynomial** *(***FPAS***) falls seine Laufzeit zudem polynomial in* $\frac{1}{\varepsilon}$ *ist.*

Online-Algorithmen

Merkmale:

- Die Eingabe kommt „Stück für Stück“
- Der Algorithmus muss zu jedem Zeitpunkt Entscheidungen treffen, ohne die gesamte Eingabe zu kennen.
- Die relative Gütegarantie heißt „competitive ratio“
- Verglichen wird mit der besten möglichen Lösung *bei Kenntnis der gesamten Ausgabe*

Randomisierte Algorithmen

- Nichtdeterminismus: mit Wahrscheinlichkeiten gewichtete Auswahlmöglichkeiten für den nächsten Elementarschritt eines Algorithmus
- Mehrere Ausführungen des selben Algos können für die selbe Eingabe verschieden sein
- Zufallsvariablen für Laufzeit, Speicherplatz und Ergebnis
- Laufzeit oder Ergebnis „im Erwartungswert“ oder „mit hoher Wahrscheinlichkeit“
- Vorteile: oft leichter zu formulieren und zu implementieren, schneller oder „besser“
- Nachteile: oft schwerer zu analysieren, (keine Garantie für Korrektheit)

Lineare Probleme

Problem LINEAR PROGRAMMING

Gegeben: $a_{ij} \in \mathbb{R}$, $b_i, c_j \in \mathbb{R}$, $1 \leq i \leq m$, $1 \leq j \leq n$, $B \in \mathbb{R}$.

Frage: Existieren $x_1, \ldots, x_n \in \mathbb{R}$, so dass

$$\sum_{j=1}^{n} c_j \cdot x_j = B \text{ und}$$

$$\underbrace{\sum_{j=1}^{n} a_{ij} \cdot x_j \leq b_i}_{A \cdot \bar{x} \leq \bar{b}} \text{ für } 1 \leq i \leq m \text{ ?}$$

Lineare Ganzzahlige Probleme

Problem INTEGER LINEAR PROGRAMMING

Gegeben: $a_{ij} \in \mathbb{N}_0,\ b_i, c_j \in \mathbb{N}_0,\ 1 \leq i \leq m,\ 1 \leq j \leq n,\ B \in \mathbb{N}_0.$

Frage: Existieren $x_1, \ldots, x_n \in \mathbb{N}_0$, so dass

$$\sum_{j=1}^{n} c_j \cdot x_j = B \text{ und}$$

$$\underbrace{\sum_{j=1}^{n} a_{ij} \cdot x_j \leq b_i}_{A \cdot \bar{x} \leq \bar{b}} \text{ für } 1 \leq i \leq m \text{ ?}$$

Anwendungen von ILPs

- Sehr viele wichtige Probleme lassen sich als Lineare Probleme formulieren.
- Lineare Probleme lassen sich durch effiziente Algorithmen lösen ($\rightsquigarrow$ Simplex-Algorithmus, CPLEX...).
- *Ganzzahlige* Lineare Probleme lassen sich häufig *nicht* effizient lösen (ILP ist $\mathcal{NP}$-vollständig).
- Es gibt eine Riesenpalette algorithmischer Methoden, um ILPs zu lösen.

Minimale Aufspannende Bäume

Problem MINIMUM SPANNING TREE (MST)

Gegeben: Zusammenhängender Graph $G = (V, E)$, Gewichtsfunktion $c : E \longrightarrow \mathbb{R}$.

Aufgabe: Finde einen zusammenhängenden Teilgraphen $B = (V, E')$ von G, mit $E' \subseteq E$, der bezüglich c minimales Gewicht hat. Das heißt so, dass

$$c(B) = \sum_{\{u,v\} \in E'} c(\{u, v\})$$

minimal über alle solchen Teilgraphen von G ist.

Die Färbungsmethode von Tarjan

Definition

Ein **Schnitt** *in einem Graphen* $G = (V, E)$ *ist eine Partition* $(S, V \setminus S)$ *der Knotenmenge* V*. Eine Kante* $\{u, v\}$ **kreuzt** *den Schnitt* $(S, V \setminus S)$*, falls* $u \in S$ *und* $v \in V \setminus S$ *ist. Oft wird auch die Menge der Kanten, die den Schnitt* $(S, V \setminus S)$ *kreuzt, mit diesem Schnitt identifiziert.*

Regeln

Definition (Grüne Regel)

Wähle einen Schnitt in G, der von keiner grünen Kante gekreuzt wird. Unter allen ungefärbten Kanten, die diesen Schnitt kreuzen, wähle eine Kante minimalen Gewichts und färbe sie grün.

Definition (Rote Regel)

Wähle einen Kreis, *der keine rote Kante enthält. Unter allen ungefärbten Kanten, die auf diesem Kreis liegen, wähle eine Kante* maximalen Gewichts *und färbe sie rot.*

Definition (Tarjans Färbungsmethode)

1. **Solange** *noch eine der beiden Regeln anwendbar ist*
2. *wende die grüne oder die rote Regel an.*

Lemma (Färbungsinvariante)

Es gibt einen aufspannenden Baum minimalen Gewichts, der alle grünen *Kanten und* keine rote *Kante enthält.*

Satz

Tarjans Färbungsmethode erzeugt einen minimalen aufspannenden Baum (ohne Beweis).

Der Algorithmus von Prim

1. Wähle einen beliebigen Startknoten und betrachte diesen als einen "grünen Baum".
2. Wiederhole den folgenden Färbungsschritt $(|V| - 1)$–mal:
   3. Wähle eine ungefärbte Kante $e_i = \{u, v\}$ minimalen Gewichts, die genau einen Endknoten in dem grünen Baum hat, der den Startknoten enthält, und färbe sie grün.

Prims Algorithmus ist eine wiederholte Anwendung der „grünen Regel".

Satz

Der Algorithmus von Prim lässt sich mit Laufzeit $O(m \log_{2+m/n} n)$ *implementieren (*$\rightsquigarrow$ *HEAPs, ohne Beweis).*

Beweis der Korrektheit von Prims Algorithmus

- Sei dazu B_i der grüne Baum vor dem i-ten Schleifendurchlauf.
- Zeigen per Induktion: Zu jedem Zeitpunkt gibt es einen MST T_i, der alle Kanten des grünen Baums enthält.
- Für B_0 ist die Behauptung klar.
- Noch zu zeigen: $B_i \cup e_i$ ist Teil eines MST ($i > 0$).
- In T_i gibt es einen Pfad von u nach v. Dieser enthält genau eine Kante $\{x, y\}$, mit $x \in B_i$ und $y \notin B_i$.
- $T_{i+1} := T_i - \{x, y\} + \{u, v\}$ ist ein Baum mit Gewicht $c(T_{i+1}) = c(T_i) - c(\{x, y\}) + c(\{u, v\})$
- Nach Wahl von e_i gilt $c(\{x, y\}) \geq c(\{u, v\})$, also $c(T_{i+1}) \leq c(T_i)$
- Da T_i aber ein MST war, gilt Gleichheit, also ist T_{i+1} auch ein MST.

Der Algorithmus von Kruskal

1. Sortiere die Kanten nach ihrem Gewicht in nicht–absteigender Reihenfolge.
2. Durchlaufe die sortierten Kanten der Reihe nach, und wende folgenden Färbungsschritt an:
   - 3.1 **Wenn** beide Endknoten der Kante in demselben grünen Baum liegen, so färbe sie rot;
   - 3.2 **ansonsten** färbe sie grün.

Satz

Der Algorithmus von Kruskal lässt sich mit Laufzeit $O(m \log n)$ implementieren. Sind die Kanten bereits sortiert, geht es sogar noch etwas schneller ($\rightsquigarrow$ UNION-FIND).

Termine

- Nächste Vorlesung: Dienstag, 19. April, 11.30 Uhr
- Erstes Übungsblatt: Dienstag, 19. April, 11.30 Uhr
- Abgabe: Dienstag, 26. April
- Nächste Übung: Mittwoch, 27. April, 11.30 Uhr