Programmierpraktikum Diskrete Optimierung

Maximale unabhängige/stabile Mengen
Teil I: Exakte und heuristische Verfahren auf allgemeinen Graphen
Teil II: Approximationsschemata auf Unit Disk Graphen

SoSe 2012

Allgemeine Bemerkungen:

In diesem Programmierpraktikum sollen maximale unabhängige/stabile Mengen in Graphen bestimmt werden. Für einen Graphen $G = (V, E)$ ist eine Teilmenge $V' \subset V$ unabhängig (stabil), wenn $G[V']$ keine Kanten enthält. Es ist für $\epsilon > 0$ im allgemeinen Fall NP-schwer, das Problem mit einer Güte von $n^{1-\epsilon}$ zu approximieren. Das bedeutet, dass auf allgemeinen Graphen keine nicht-trivialen Approximationsgüten erreicht werden können, falls $P \neq NP$. Allerdings gibt es verschiedene exakte und heuristische Algorithmen, von denen einige implementiert werden sollen. Neben den Standard-Benchmarks für das Stabile-Mengen-Problem, sollen die allgemeinen Algorithmen auch auf Unit Disk Graph Instanzen angewandt und mit den spezialisierten Algorithmen hierfür verglichen werden. Im Falle der Heuristiken sollen dabei auch obere Schranken bestimmt werden, um die Güte der Lösungen a-posteriori abzuschätzen.

Spezialfall: Unit Disk Graphen

Neben exakten Lösungsverfahren auf ungerichteten Graphen soll das Problem auch für eingeschränkte Instanzen bearbeitet werden. *Unit Disk Graphen* stellen eine insbesondere in der Anwendung wichtige Graphenklasse, sie sind ein einfaches graphische Modell vieler geometrischer Fragestellungen. Ein Graph $G = (V, E)$ ist ein Unit Disk Graph (UDG), wenn es eine Abbildung $f : V \to \mathbb{R}^2$ gibt (geometrische Einbettung), die

$$(u, v) \in E \iff \|f(u) - f(v)\| \leq 1$$

erfüllt. In diesem Praktikum gehen wir davon aus, dass der Graph durch diese Abbildung, also durch die Mittelpunkte der Kreisscheiben, gegeben ist[1].

Das maximale unabhängige Mengen Problem auf UDGs ist NP-schwer, so dass man hier oft unter Ausnutzung der geometrischen Eigenschaften auf approximative Verfahren zurückgreift, um eine Lösung zu erhalten. Wir sagen, dass ein (polynomieller) Algorithmus $\mathcal{A}$ eine c-Approximation ist, wenn er für jede Eingabeinstanz G eine unabhängige Menge $\mathcal{A}(G)$ liefert, deren Größe mindestens c^{-1} mal der Größe einer optimalen Lösung ist.

In unserem Fall von UDGs liefert jeder (!) *Greedy*-Ansatz bereits eine 5-Approximation. Hierbei wird iterativ immer ein Knoten des (Rest-)Graphen zur Lösungsmenge hinzugefügt und anschließend dieser mitsamt seiner Nachbarschaft aus dem Graphen entfernt. Eine bessere Güte liefert der sog. *LeftMostGreedy*-Ansatz, bei dem immer ein Knoten mit minimaler x-Koordinate gewählt wird und der eine 3-Approximation ergibt. Hat man keine geometrische Einbettung, so kann man diese Güte trotzdem erreichen, indem ein Knoten v gesucht wird, der in seiner Nachbarschaft $\Gamma(v)$ maximal 3 unabhängige Knoten zulässt.

Beschränken wir nun die Eingabeinstanzen auf UDGs, so existieren für das Problem sogar sog. *polynomielle Approximationsschemata* (PTAS), d.h. zu jedem gegebenem $\varepsilon > 0$ gibt es eine $(1 + \varepsilon)$-Approximation, die in polynomieller Zeit läuft. Die Laufzeit hängt in diesem Fall jedoch auch vom gewählten Parameter ε ab, ist aber für fest gewähltes ε polynomiell in der Instanzgröße.

[1] Es sei angemerkt, dass die Bestimmung einer geometrischen Einbettung eines UDG ein nichttriviales und NP-schweres Problem ist.

Aufgaben:

Für jede der Aufgaben ist zunächst eine Einarbeitungsaufgabe zu bearbeiten, für die es eine Zwischenbesprechung geben wird. Hier sollen auch Fragen zur Bearbeitung und Probleme besprochen werden. Sie können sich natürlich auch jederzeit mit Fragen und Problemen melden. Die Einarbeitungsaufgabe soll bis zum 30. April 2012 abgegeben werden. Ein Termin für die endgültige Abgabe und eine entsprechende Präsentation wird dann auch festgelegt.

Teil I: Algorithmen für Allgemeine Graphen

Das maximale unabhängige Mengen Problem soll auf allgemeinen Graphen exakt oder heuristisch in Kombination mit einer a-posteriori Güte-Abschätzung gelöst werden. Es gibt vier exakte und ein heuristisches Verfahren. Die Graphen werden ausschließlich als Adjazenzliste über das DIMACS-Format eingelesen, da geometrischen Informationen nicht verwendet werden.

Die einfachste Heuristik ist der *Minimum-Degree-Greedy*-Algorithmus. Hierbei wird iterativ immer der Knoten des (Rest-)Graphen mit kleinstem Grad zur Lösungsmenge hinzugefügt und anschließend mitsamt seiner Nachbarschaft aus dem Graphen entfernt.

Gruppe 1: Measure & Conquer Approach von Fomin, Grandoni und Kratsch

Implementieren sie den *Measure & Conquer Approach* von Fomin, Grandoni und Kratsch, welcher mit einer Laufzeit von $O^\star(2^{0.287n})$ einer der theoretisch schnellsten bekannten exakten Algorithmen für das Problem ist.

Einstiegsaufgabe

Implementieren sie eine Einleseroutine für Graphen im DIMACS-Format und den *Minimum-Degree-Greedy*-Algorithmus.

Gruppe 2: Branch-&-Bound-Algorithmus von Balas und Yu

Implementieren sie den *Branch-&-Bound-Algorithmus* von Balas und Yu, indem Sie den dortigen Cliquen-Algorithmus auf das Komplement des Eingabegraphen anwenden.

Einstiegsaufgabe

Implementieren sie eine Einleseroutine für Graphen im DIMACS-Format und den *Minimum-Degree-Greedy*-Algorithmus.

Gruppe 3: Branch-&-Bound-Algorithmus von Sewell

Implementieren sie den *Branch-&-Bound-Algorithmus* von Sewell, welcher einer der schnellsten praktischen exakten Algorithmen für das Problem ist.

Einstiegsaufgabe

Implementieren sie eine Einleseroutine für Graphen im DIMACS-Format und den *Minimum-Degree-Greedy*-Algorithmus.

Gruppe 4: Heuristiken von Werneck et al. sowie Balas und Niehaus

Implementieren sie folgende Heurstik: Generieren Sie zunächst einige Greedy-Lösungen und wenden sie darauf die schnelle Lokale Suche nach Werneck et al. an. Anschließend sollen aus dieser Grundmenge durch Crossover-Schritte nach Balas und Niehaus mit Hilfe von bipartiten Matchings bessere Lösungen generiert werden. Um die Güte der Lösung abzuschätzen, soll als obere Schranke eine Cliquen-Überdeckung berechnet werden, indem mit

einem Greedy-Algrithmus iterativ eine unabhängige Menge auf dem Komplement-Graphen (also eine Clique im Originalgraphen) bestimmt wird und diese aus dem Graphen entfernt wird, bis alle Knoten gelöscht sind. Die Anzahl der Iterationen/Cliquen ist dann eine obere Schranke für die maximale Größe einer unabhängigen Menge.

Einstiegsaufgabe

Implementieren sie eine Einleseroutine für Graphen im DIMACS-Format und den *Minimum-Degree-Greedy*-Algorithmus.

Gruppe (5) (Backup): Branch-&-Cut nach Rebenack et al.

Implementieren sie den Branch-&-Cut Algorithmus nach Rebenack et al. unter Verwendung eines LP-Solvers.

Einstiegsaufgabe

Implementieren sie eine Einleseroutine für Graphen im DIMACS-Format und den *Minimum-Degree-Greedy*-Algorithmus.

Teil II: Algorithmen für Unit Disk Graphen

Ganz allgemein soll das maximale unabhängige Mengen Problem auf Unit Disk Graphen approximativ gelöst werden. Neben der Instanz wird ein Parameter $\varepsilon > 0$ erwartet, der die Approximationsgüte der zu erstellenden Lösung festlegt.
Hierfür gibt es 3 unterschiedliche Verfahren, die von je einer Gruppe gelöst werden sollen. Die Implementierungen sollen natürlich auch die in den Arbeiten dazu angegebenen asymptotischen Laufzeiten haben und insbesondere für festes ε eine polynomielle Laufzeit erreichen. Da alle Verfahren für kleine Teilgraphen eine optimale Lösung berechnen, kann aus praktischer Sicht vielleicht auch ein anderes optimales Lösungsverfahren (vgl. andere Gruppen) eine schnellere Ausführung ermöglichen. Es steht jeder Gruppe frei, dies auch empirisch zu untersuchen!

Gruppe 6: Shifting Strategie von Hochbaum und Maass

Implementieren Sie die *Shifting Strategie* von Hochbaum und Maass, welche das Gebiet des Graphen in $k \times k$ große Quadrate unterteilt.

Einstiegsaufgabe

Implementieren sie eine Einleseroutine für Unit Disk Graphen und den *LeftMostGreedy*-Algorithmus.

Gruppe 7: DP-Strategie von Matsui

Implementieren Sie die *erweiterte Shifting Strategie* von Matsui, welche das Gebiet des Graphen in Streifen unterteilt und mit Hilfe eines Dynamischen Programms eine Lösung für jeden Streifen erstellt.

Einstiegsaufgabe

Implementieren sie eine Einleseroutine für Unit Disk Graphen und den *LeftMostGreedy*-Algorithmus.

Gruppe 8: Expanding Neighborhood Ansatz

Implementieren Sie die *lokale Strategie*, welche lokale Nachbarschaften im Graphen betrachtet. Da dieses Verfahren nicht auf eine geometrische Darstellung angewiesen ist, benutzen Sie bitte das DIMACS Eingabeformat für allgemeine, ungerichtete Graphen.

Einstiegsaufgabe

Implementieren sie eine Einleseroutine für Graphen im DIMACS-Format und den *LeftMostGreedy*-Algorithmus.

Datenformat und Testinstanzen:

Die Programme in diesem Praktikum sind in ISO-C++ zu schreiben!

Testinstanzen finden Sie auf der Homepage der Veranstaltung. Für UDGs sind sowohl geometrische, als auch korrespondierende adjazenz-basierte Instanzen im DIMACS-Format vorhanden.

Geometrisches Format

Die Eingabe des Graphen (durch Angabe der Kreismittelpunkte) wird als Textdatei gegeben. Alle Werte sind ganzzahlig und ihre Darstellung ist in 32-Bit möglich. Um entsprechend Rundungsfehler in der Bestimmung der Adjazenz zu vermeiden, werden skalierte Werte genommen, so dass alle Kreise einen konstanten, gegebenen Radius haben, der ebenfalls in der Instanzdatei kodiert ist.

Hierbei wird folgendes Format verwendet:

1. Zeile	`n`	`n` Anzahl Knoten
2. Zeile	`r`	`r` Durchmesser der Kreisscheiben
3. Zeile	`id x y`	`id` Knotennr., `x` `y` Koordinaten
	...	...
		Im Anschluss kann Kommentar stehen:

Mit einem `'c'` beginnende Zeilen (Kommentare) oder leere Zeilen werden ignoriert.

Eine Kante $(u, v) \in E$ existiert dann im Graphen $G = (V, E)$, wenn $(x_u - x_v)^2 + (y_u - y_v)^2 \leq r^2$ gilt.

DIMACS-Format

Das DIMACS-Format definiert einen Graphen über eine Adjazenzliste, wie folgt:

1. Mit einem `'c'` beginnende Zeilen (Kommentare) oder leere Zeilen werden ignoriert.
2. Die erste nicht-ignorierte Zeile beginnt mit einem 'p' (Problem) und hat das folgende Format:
   `'p' <Problem-Typ> n m`,
   wobei n = Zahl der Knoten und m = Zahl der Kanten zwei natürliche Zahlen sind.
3. Zeilen, die mit 'e' beginnen, definieren eine Kante:
   `'e' i j`,
   wobei die Kante die Knoten i und j verbindet. Die Anzahl der mit 'e' beginnenden Zeilen ist 'm' (gemäß der Problem-Zeile).
4. Zeilen, die mit einem `'n'` beginnen, spezifizieren ein Knotengewicht und werden in diesem Praktikum ignoriert.

Programmaufruf und Ausgabe

Das Programm soll so aufgerufen werden, dass man den Namen der Instanzdatei mit übergibt. Wenn also `program` ein Programm zur Lösung einer (Teil-)Aufgabe ist, und `datei` die Instanz kodiert, so ist der Aufruf:
`program datei`
Im Falle eines PTAS soll eine entsprechende Approximationsgüte `eps` > 0 mit angegeben werden, also:
`program datei eps`
Die Ausgabe soll in eine separate Datei `datei.out` erfolgen, in der ersten Zeile die Kardinaliät der Lösung, gefolgt von den Nummern der Lösungsknoten darunter.