

Inhalt I

Verteilte Algorithmen

- Überblick
- Formales Modell
- Algorithmus
- Komplexitätsmaße
- Leader Election in Ringen

Minimum Spanning Trees

- Grundlagen
- Algorithmus
- Beispiel
- Korrektheit
- Komplexität



Inhalt II

Dominating Set

- Grundlagen
- Unit Disk Graph
- PTAS in Unit Disk Graphen
- Lineare Programme
- Verteiltes Runden: LP \rightarrow ILP
- Verteilte Berechnung des LP



Organisatorisches

► Webseite zur Vorlesung:

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

- Folien zur Vorlesung und Übung (soweit verfügbar)
- Übungsblätter
- Referenzen auf weiterführende Literatur
- Ankündigungen zur Vorlesung



Verteilte Algorithmen

Literatur

- H. Attiya, J. Welch: *Distributed Computing*. Wiley, 2004.



Verteilte Systeme

Definition 1

Ein **verteilt System** ist eine Menge von selbständigen *Recheneinheiten*, die miteinander *kommunizieren* können.

Theorie des Verteilten Rechnens

- ▶ Ziele sind ähnlich wie bei sequentiellem Rechnen, z.B. Algorithmik, Komplexitätstheorie, usw.
- ▶ Problem: Kein einheitliches Modell für verteilte Systeme
 - ▶ Unterschiedliche Kommunikationsparadigmen
 - ▶ Sind Recheneinheiten unterscheidbar?
 - ▶ ...
- ▶ Komplexitätsmaße
- ▶ Basisalgorithmen
- ▶ Schwierigkeit von Problemen: oft Negativresultate

Verteilte Systeme

- ▶ Beispiele
 - ▶ Multiprozessorrechner
 - ▶ Cluster von Workstations
 - ▶ Internet, P2P Netze, Seti@home
 - ▶ **Sensor- und Ad Hoc-Netze**
- ▶ Grundlegende Schwierigkeiten
 - ▶ Asynchrone Kommunikation
 - ▶ Begrenztes lokales Wissen
 - ▶ Ausfälle

Kommunikationsparadigmen

- ▶ Synchron ↔ Asynchron
- ▶ Nachrichtenbasiert ↔ Shared Memory
 - ▶ Synchrone Kommunikation über Shared Memory entspricht dem PRAM Modell
 - ▶ Wir betrachten im Folgenden nur Modelle für **nachrichtenbasierte** („message passing“) verteilte Systeme

Beispiel: Broadcast

- ▶ Gegeben
 - ▶ Kommunikationsgraph $G = (V, E)$
 - ▶ Aufspannender Baum mit Wurzel $r \in V$
 - ▶ Jeder Knoten v kennt den Nachbarn, der im Baum zur Wurzel zeigt ($\text{parent}(v)$)
 - ▶ Jeder Knoten v kennt die Nachbarn, die im Baum von der Wurzel weg zeigen ($\text{children}(v)$)
 - ▶ Eine Nachricht M an einem Knoten r
- ▶ Problem
 - ▶ Die Nachricht M soll an alle Knoten des Netzes übermittelt werden

Formales Modell eines verteilten Systems (II)

- ▶ Jeder Prozessor p_i hat Zustandsmenge Q_i , darunter ausgezeichnete Anfangs- und Endzustände
- ▶ Kommunikationspuffer $\text{outbuf}_i[l]$ und $\text{inbuf}_i[l]$ ($1 \leq l \leq d_i$), jeder dieser Puffer besteht aus einer Folge von Nachrichten
- ▶ Am Anfang sind die $\text{inbuf}_i[l]$ leer
- ▶ Kommunikationspuffer können auch in die Zustände Q_i modelliert werden

Formales Modell eines verteilten Systems (I)

- ▶ n Prozessoren („Recheneinheiten“) p_1, \dots, p_n
- ▶ Zwei Prozessoren können durch einen bidirektionalen Kommunikationskanal verbunden sein
- ▶ Kommunikationsgraph („Topologie“, Netz)
 - ▶ Knoten: Prozessoren p_1, \dots, p_n
 - ▶ Kanten: Kommunikationskanäle $e_{ij} = \{p_i, p_j\}$ ($i \neq j$)
 - ▶ Der Grad (Anzahl inzidenter Kanten) von p_i sei d_i
 - ▶ Der Graph ist ungerichtet

Beispiel: Broadcast (Zustände)

- ▶ Die Zustände eines Prozessors p_i beinhalten
 - ▶ $\text{parent}(p_i) = \text{parent}_i$, ein Prozessorindex oder nil
 - ▶ $\text{children}(p_i) = \text{children}_i$, eine Menge von Prozessorindizes
 - ▶ terminated_i , eine boolean Variable
- ▶ Anfangszustände: parent_i und children_i gemäß aufspannendem Baum initialisiert, $\text{terminated}_i = \text{false}$
- ▶ $\text{outbuf}_r[j]$ enthält M für alle $j \in \text{children}_r$
- ▶ Alle anderen outbuf_i sind leer

Verteilter Algorithmus

- ▶ Berechnungsschritt $\text{comp}(i)$ bei p_i
 - ▶ p_i befindet sich in Zustand $q_i \in Q_i$
 - ▶ Abhängig von q_i und inbuf_i Zustandsübergang
 - ▶ Nach Berechnungsschritt ist inbuf_i leer
 - ▶ Für jede inzidente Kante kann eine Nachricht erstellt und zu $\text{outbuf}_i[l]$ hinzugefügt werden
- ▶ Kommunikationsschritt $\text{del}(i, j, M)$: Sende M über e_{ij}
 - ▶ Eine Nachricht M wird von $\text{outbuf}_i[l_i]$ nach $\text{inbuf}_j[l_j]$ übertragen

Beispiel: Broadcast (Algorithmus)

- ▶ Berechnungsschritt bei p_i :
 - ▶ Falls $M \in \text{inbuf}_i[k]$ für ein k
 - ▶ füge M zu $\text{outbuf}_i[j]$ hinzu, für alle $j \in \text{children}_i$
 - ▶ setze $\text{terminated}_i = \text{true}$
 - ▶ Falls $i = r$ und $\text{terminated}_r = \text{false}$, so setze $\text{terminated}_r = \text{true}$
 - ▶ Ansonsten wird nichts getan

Verteilter Algorithmus

- ▶ Konfiguration $C = (c_1, \dots, c_n)$
 - ▶ c_i enthält Zustand q_i des Prozessors p_i sowie inbuf_i und outbuf_i
 - ▶ Beschreibt vollständig den Zustand des ganzen Systems zu einem bestimmten Zeitpunkt
- ▶ Ausführung eines verteilten Algorithmus
 - ▶ Unendliche Folge von Konfigurationen und Schritten:
 $C_1, \phi_1, C_2, \phi_2, \dots$
 - ▶ Eine gültige Ausführung muss gewisse Bedingungen erfüllen

Asynchrone Kommunikation

- ▶ Keine Garantien bzgl. der Dauer einer Nachrichtenübermittlung
- ▶ Algorithmus ohne zeitliche Koordinierung („asynchron“)
- ▶ Sei eine Ausführung $C_1, \phi_1, C_2, \phi_2, \dots$
- ▶ Gültigkeitsbedingungen:
 - ▶ C_{k+1} muss durch Schritt ϕ_k aus C_k hervorgehen
 - ▶ Jeder Prozessor hat unendlich viele Berechnungsschritte
 - ▶ Jede Nachricht wird übermittelt (Achtung: „eventually“)
- ▶ C_1 besteht aus Initialzuständen mit leeren inbuf_i

Synchrone Kommunikation

- ▶ Rundenbasiert
 1. Jeder Prozessor kann eine Nachricht zu jedem Nachbarn senden
 2. Diese Nachrichten werden sofort zugestellt
 3. Jeder Prozessor führt einen Berechnungsschritt aus
- ▶ Spezialfall einer asynchronen Kommunikation:
Eine Runde besteht aus
 1. Kommunikationsschritten bis alle outbuf_i leer sind
 2. Einem Berechnungsschritt pro Prozessor
- ▶ Die Ausführung ist durch die Anfangskonfiguration C_1 festgelegt

Beispiel: Broadcast (Korrektheit)

Lemma 2

Der Broadcast Algorithmus ist korrekt, d.h. jede Ausführung ist nach endlich vielen Schritten beendet und jeder Prozessor hat die Nachricht M erhalten.

Korrektheit eines verteilten Algorithmus

- ▶ Jede gültige Ausführung ist unendlich
- ▶ Jeder Prozessor hat eine Teilmenge von Endzuständen
- ▶ Aus Endzuständen können nur Endzustände erreicht werden
- ▶ Eine gültige Ausführung ist **beendet**, wenn alle Prozessoren in einem Endzustand sind
- ▶ Ein verteilter Algorithmus ist **korrekt**, wenn alle gültigen Ausführungen nach endlich vielen Schritten enden und danach stets eine korrekte Lösung des Problems vorliegt

Beispiel: Broadcast (Korrektheit)

Beweis.

- ▶ Ein Prozessor ist im Endzustand g.d.w. er M erhalten hat
- ▶ Annahme: Es gibt einen Prozessor, der nicht terminiert
 - ▶ Sei p_i solch ein Prozessor mit minimaler Höhe im Baum
 - ▶ $p_i \neq r$, denn im ersten Berechnungsschritt von r terminiert r
 - ▶ Der Vorgänger von p_i im Baum ist terminiert
 - ▶ Dann hat der Vorgänger aber M an p_i geschickt
 - ▶ Da alle Nachrichten übermittelt werden und jeder Prozessor unendlich viele Berechnungsschritte hat, wird auch p_i nach endlich vielen Schritten M erhalten und terminieren
 - ▶ Widerspruch zur Annahme, also terminiert jeder Prozessor und erhält somit auch M

Nachrichten-Komplexität

- ▶ Maximale Anzahl an gesendeten Nachrichten über alle gültigen Ausführungen des Algorithmus
- ▶ Kann bei asynchroner und synchroner Kommunikation angewandt werden

Zeit-Komplexität

- ▶ Synchroner Kommunikation
 - ▶ Laufzeit: Maximale Anzahl der Runden
- ▶ Asynchrone Kommunikation
 - ▶ Maximale Übermittlungsdauer ist 1
 - ▶ Zeitliche Ausführung
 - ▶ Jedem Schritt wird ein nichtnegativer Zeitpunkt zugeordnet
 - ▶ Zeit startet bei 0, darf nicht abnehmen und nicht beschränkt sein
 - ▶ Pro Prozessor ist die Zeit strikt ansteigend
 - ▶ Dauer einer Nachrichtenübermittlung: Zeit zwischen Einfügen in $\text{outbuf}_i[l_i]$ und Lesen aus $\text{inbuf}_j[l_j]$
 - ▶ Normierung der maximalen Dauer auf 1
 - ▶ Laufzeit: Maximale Zeit bis zur Beendigung des Algorithmus

Beispiel: Broadcast (Nachrichten-Komplexität)

Lemma 3

Der Broadcast Algorithmus hat Nachrichtenkomplexität $n - 1$.

Beweis.

- ▶ Die Nachricht M wird genau einmal pro Kante im aufspannenden Baum versandt
- ▶ Jeder Baum mit n Knoten hat $n - 1$ Kanten

□

Beispiel: Broadcast (synchroner Zeitkomplexität)

Lemma 4

Bei jeder gültigen Ausführung im synchronen Modell empfängt jeder Prozessor mit Entfernung t von r im aufspannenden Baum die Nachricht M in Runde t .

Beispiel: Broadcast (synchrone Zeitkomplexität)

Beweis.

Induktion nach t .

- ▶ Induktionsanfang: $t = 1$. Jedes Kind von r empfängt M in der ersten Runde.
- ▶ Induktionsannahme: Jeder Prozessor mit Entfernung $t - 1$ hat M in Runde $t - 1$ empfangen.
- ▶ Induktionsschluss:
 - ▶ Sei p_i ein Prozessor mit Entfernung t im Baum
 - ▶ Der Vorgänger $\text{parent}_i = p_j$ von p_i hat also Entfernung $t - 1$, und nach Induktionsannahme hat p_i die Nachricht M in Runde $t - 1$ empfangen
 - ▶ Nach dem Algorithmus sendet p_j die Nachricht M in der darauffolgenden Runde an p_i .

□

Problemstellung

- ▶ Voraussetzungen
 - ▶ Der Kommunikationsgraph ist ein Ring aus n Prozessoren
 - ▶ Jeder Prozessor kennt seinen *linken* und *rechten* Nachbarn
- ▶ Problem
 - ▶ Genau ein Prozessor soll als *Leader* ausgezeichnet werden

Beispiel: Broadcast (synchrone Zeitkomplexität)

Lemma 5

Der Broadcast Algorithmus hat im synchronen Modell eine Zeit-Komplexität von h , wobei h die Höhe des aufspannenden Baumes ist.

Beweis.

- ▶ Die maximale Entfernung von der Wurzel im Baum ist die Höhe h des aufspannenden Baumes
- ▶ Der Beweis folgt somit aus Lemma 4

□

Anonymität, Uniformität

Definition 6

Ein verteilter Algorithmus heißt **uniform**, wenn die Anzahl n der Prozessoren nicht verwendet wird.

Definition 7

Ein verteilter Algorithmus heißt **anonym**, wenn die Prozessoren keine unterschiedlichen IDs besitzen. (D.h., die Prozessoren führen identische Programme aus.)

Satz 8

Es gibt keinen anonymen Algorithmus für das Leader Election Problem in Ringen.

„Symmetry Breaking“

- ▶ Jeder Prozessor p_i hat eindeutige ID_i
- ▶ Einfacher uniformer Algorithmus:
 - ▶ Am Anfang: jeder Prozessor p_i sendet ID_i zum linken Nachbarn
 - ▶ Prozessor p_i empfängt eine ID vom rechten Nachbarn
 - ▶ Falls $ID > ID_i$: sende ID zum linken Nachbarn
 - ▶ Falls $ID = ID_i$: sende Nachricht „terminiere“ zum linken Nachbarn und terminiere selbst als „Leader“
 - ▶ Ansonsten sende ID nicht weiter
 - ▶ Prozessor p_i empfängt Nachricht „terminiere“
 - ▶ terminiere als „Non-Leader“
- ▶ Nachrichten-Komplexität: $\mathcal{O}(n^2)$
- ▶ Bessere Algorithmen: $\Theta(n \log n)$
- ▶ Andere Möglichkeit: Randomisierung

Beweis.

- ▶ Annahme: Es gibt einen solchen Algorithmus
- ▶ *O.B.d.A.* sei der Algorithmus synchron
- ▶ Die Ausführung R des Algorithmus ist durch die Anfangskonfiguration festgelegt
- ▶ Da der Algorithmus anonym ist, sind die Zustandsübergangsfunktionen für alle Prozessoren identisch
- ▶ In jeder Runde k der Ausführung R sind alle Prozessoren am Ende der Runde im selben Zustand
 - ▶ Induktion über k
- ▶ Sobald ein Prozessor am Ende einer Runde ausgewählt wird, sind alle Prozessoren ausgewählt, also kann der Algorithmus nicht korrekt sein

□

Literatur

- ▶ R. G. Gallager, P. A. Humblet, and P. M. Spira: A Distributed Algorithm for Minimum-Weight Spanning Trees. *ACM Transactions on Programming Languages and Systems* 5:1, 66-77, 1983.

Problemstellung

- ▶ Voraussetzungen
 - ▶ Ungerichteter, zusammenhängender Kommunikationsgraph $G = (V, E)$ ($|V| = n$, $|E| = m$)
 - ▶ Kantengewichte $W : E \rightarrow \mathbb{R}$, paarweise verschieden
 - ▶ Asynchrone Kommunikation
 - ▶ Jeder Knoten kennt Kantengewichte inzidenter Kanten
- ▶ Problem
 - ▶ Bestimme einen aufspannenden Baum minimalen Gewichts (Minimum Spanning Tree, MST)

Definition 9

Sei T ein Teilbaum („Fragment“) eines Graphen $G = (V, E)$, der durch eine Teilmenge der Kanten $E_T \subset E$ induziert wird. Dann heißt eine Kante e **ausgehend** aus T wenn genau einer der beiden Endknoten von e zum Teilbaum T gehört.

Lemma 10

Sei ein Teilbaum T eines MST induziert durch eine Kantenmenge E_T und e eine aus T ausgehende Kante minimalen Gewichts. Dann ist der durch $E_T \cup e$ induzierte Baum ebenfalls ein Teilbaum eines MST.

Motivation

- ▶ Fundamentales Problem in verteilten Systemen
- ▶ Aufspannende Bäume bilden Grundlage für weitere Algorithmen wie
 - ▶ Broadcast
 - ▶ Leader Election
 - ▶ Zählen der Knoten
 - ▶ Berechnen von Funktionen wie Maximum, Summe, ...
- ▶ Energieaufwand in Sensor- und Ad Hoc-Netzen
 - ▶ Kantengewichte entsprechen Energieaufwand für Kommunikation
 - ▶ Minimiere Energieaufwand

Sequentieller Algorithmus von Prim (Wdh.)

1. Wähle einen beliebigen Startknoten und betrachte diesen als „grünen Baum“
2. Wiederhole den folgenden Färbungsschritt $n - 1$ mal
3. Wähle eine ungefärbte aus dem grünen Baum ausgehende Kante minimalen Gewichts und färbe sie grün.

- ▶ Laufzeit $\mathcal{O}(m \log_{2+m/n} n)$
- ▶ Zu sequentiellen MST-Algorithmen siehe auch Kapitel 4 im Skript zur Vorlesung „Entwurf und Analyse von Algorithmen“:

<http://i11www.ira.uka.de/algo/teaching/scripts/sources/ad.pdf>

Lemma 11

Wenn alle Kantengewichte eines Graphen paarweise verschieden sind, so ist der MST eindeutig bestimmt.

Lemma 12

Bei paarweise verschiedenen Kantengewichten ist jeder MST-Algorithmus, der Schritt für Schritt zu (mehreren) bestehenden Teilen des MST die ausgehende Kante minimalen Gewichts hinzufügt, korrekt.

Idee des Algorithmus

- ▶ Anfangszustand
 - ▶ Alle Knoten sind im Zustand Sleeping
 - ▶ Jeder Knoten bildet ein einzelnes Fragment F des MST
 - ▶ Das Level L_F jedes Fragments ist am Anfang 0
- ▶ Jedes Fragment F sucht die ausgehende Kante e_F minimalen Gewichts und versucht mit dem benachbarten Fragment F' zu verschmelzen
 - ▶ Falls $L = L'$ und $e_F = e_{F'}$, so verschmelzen F und F' zu einem Fragment mit Level $L + 1$ und core e_F
 - ▶ Falls $L < L'$, so wird F von F' absorbiert
 - ▶ Ansonsten wartet F bis F' ein höheres Level erhält

Idee des Algorithmus

- ▶ Fragment F hat zwei ausgezeichnete Kanten
 - ▶ \min_F : Die minimale ausgehende Kante
 - ▶ core_F : Kante der letzten Verschmelzung oder gleich \min_F
- ▶ Knoten v speichern
 - ▶ Level L
 - ▶ Fragment-ID (Gew. der core-Kante der letzten Verschmelzung)
 - ▶ Zu inzidenten Kanten ob sie zum MST gehören oder nicht
 - ▶ Kante in Richtung \min_F
 - ▶ Kante in Richtung core_F
- ▶ Knotenzustände
 - ▶ Sleeping: Anfangszustand
 - ▶ Find: Suche nach minimaler ausgehender Kante
 - ▶ Found: minimale ausgehende Kante ist bekannt

Aufwachen: Erster Schritt im Zustand Sleeping

- ▶ Knoten im Zustand Sleeping
 - ▶ Markiere minimale inzidente Kante als branch
 - ▶ Sende $\langle \text{connect} \rangle(0)$ über diese Kante
 - ▶ Gehe in Zustand Found
 - ▶ Bei Empfang von $\langle \text{connect} \rangle$ oder $\langle \text{test} \rangle$ weiter wie später beschrieben

Verschmelzen zweier Fragmente bei e

- ▶ Für F_1 und F_2 ist e minimale ausgehende Kante min und core
- ▶ Beide Endknoten haben $\langle \text{connect} \rangle(L)$ zum anderen Endknoten geschickt
- ▶ Broadcast von $\langle \text{init} \rangle(L + 1, W_e, \text{Find})$ an beide Fragmente über branch Kanten
 - ▶ Die Knoten updaten den Level und die Fragment-ID (W_e), senden die $\langle \text{init} \rangle$ Nachricht weiter und suchen dann ihre minimal ausgehende Kante
- ▶ Von Blättern aus Antwort von $\langle \text{report} \rangle(W)$: Gewicht der neuen minimalen ausgehenden Kante („Convergecast“)
- ▶ Neue core-Kante des Fragments wird minimal ausgehende Kante



Knoten sucht minimalen ausgehenden Kante II

- ▶ Antwort auf $\langle \text{test} \rangle(F', L')$
 - ▶ Falls $F' = F$, so sende $\langle \text{reject} \rangle()$ zurück
 - ▶ Ansonsten
 - ▶ Falls $L \geq L'$, so sende $\langle \text{accept} \rangle()$ zurück
 - ▶ Falls $L < L'$, so verzögere Antwort bis $L \geq L'$



Knoten sucht minimalen ausgehenden Kante I

- ▶ Bei Erhalt einer Nachricht $\langle \text{init} \rangle(L, F, \text{Find})$
- ▶ Welche inzidenten Kanten sind ausgehend?
- ▶ Klassifiziere Kanten als
 - ▶ branch: gehört zum MST
 - ▶ rejected: gehört nicht zum MST
 - ▶ basic: steht noch nicht fest
- ▶ Sende $\langle \text{test} \rangle(F, L)$ an minimale Kante mit Status basic
- ▶ Antwort $\langle \text{reject} \rangle()$: Nachbar gehört selbst zu F ; markiere Kante als rejected, sende $\langle \text{test} \rangle(F, L)$ an nächste Kante
- ▶ Antwort $\langle \text{accept} \rangle()$: Kante gefunden!

Antwort auf $\langle \text{init} \rangle$: Convergecast des min. Kantengew.

- ▶ Blatt eines Fragments
 - ▶ W_e sei Gewicht der minimalen ausgehenden Kante e (oder ∞)
 - ▶ Sende $\langle \text{report} \rangle(W_e)$ in Richtung core
 - ▶ Gehe in Zustand Found
- ▶ Innere Knoten
 - ▶ Nach Erhalt der $\langle \text{report} \rangle(W_{e_i})$ Nachrichten von allen ausgehenden branch-Kanten e_i
 - ▶ Nach Bestimmung der eigenen ausgehenden Kante minimalen Gewichts W_e
 - ▶ Bilde das Minimum W über die Gewichte W_{e_i} und W_e
 - ▶ Merke Richtung zur minimalen ausgehenden Kante
 - ▶ Sende $\langle \text{report} \rangle(W)$ in Richtung core
 - ▶ Gehe in Zustand Found

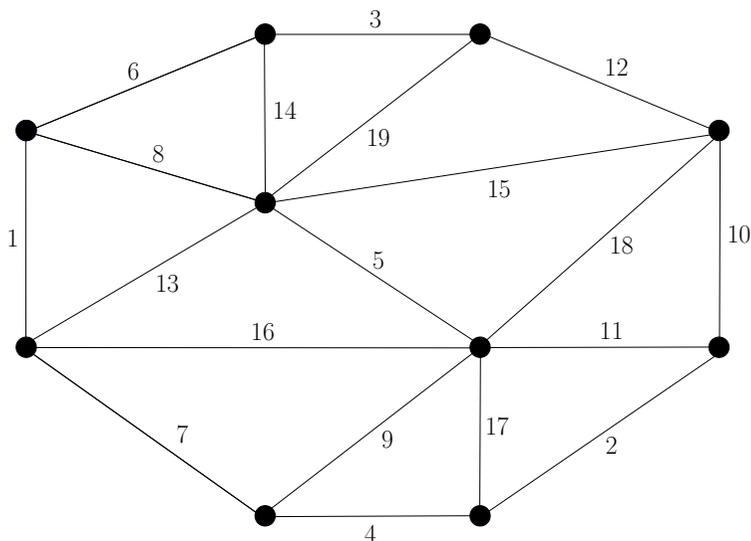


Wechsel der core-Kante und neuer $\langle \text{connect} \rangle$

- ▶ Ankunft von $\langle \text{report} \rangle(W)$ am core
 - ▶ Falls $W = \infty$ terminiere (Broadcast $\langle \text{terminate} \rangle$)
 - ▶ Sende $\langle \text{change-core} \rangle()$ entlang des Weges zur minimalen ausgehenden Kante
 - ▶ Update der Kante in Richtung core
 - ▶ Gespeicherte Kanten in Richtung neuer core-Kante bilden jetzt gerichteten Baum
 - ▶ Wurzel ist Knoten, der zur core-Kante inzident ist
 - ▶ Markiere die Kante minimalen Gewichts als branch
 - ▶ Über die neue core-Kante minimalen Gewichts wird $\langle \text{connect} \rangle(L)$ gesendet



Minimum Spanning Trees Beispiel



Eingliederung Fragmente niedrigeren Levels

- ▶ v empfängt Nachricht $\langle \text{connect} \rangle(L')$ mit $L' < L$ von u
 - ▶ Eingliederung des benachbarten Fragments
 - ▶ Markiere Kante $\{u, v\}$ als branch
 - ▶ Sende $\langle \text{init} \rangle(L, W_{\text{core}}, f)$ an u
 - ▶ Falls v in Zustand Find, so ist $f = \text{Find}$
 - ▶ Die Knoten des benachbarten Fragments nehmen an Suche nach minimaler ausgehender Kante teil
 - ▶ Falls v in Zustand Found, so ist $f = \text{Found}$
 - ▶ v hat eine ausgehende Kante kleineren Gewichts als die Kante $\{u, v\}$
 - ▶ Die Knoten des benachbarten Fragments gehen nach Weiterleitung der $\langle \text{init} \rangle(\dots)$ Nachricht in den Zustand Found



Minimum Spanning Trees Korrektheit

Korrektheit I

Lemma 13

Der verteilte MST-Algorithmus ist korrekt, d.h. der Algorithmus terminiert und die als branch klassifizierten Kanten bilden einen MST.

Beweis.

- ▶ Die branch-Kanten bilden einen MST
 - ▶ Folgt aus Lemma 12, denn es werden nur minimal ausgehende Kanten aus Fragmenten des MST als branch markiert
- ▶ Wenn der Algorithmus terminiert:
 - ▶ Für ein Fragment wird keine ausgehende Kante gefunden
 - ▶ Dieses Fragment spannt den ganzen Graph auf



Korrektheit II

- ▶ Es gibt keine „Deadlocks“
 - ▶ Betrachte beliebigen Zeitpunkt vor Terminierung
 - ▶ Nichtleere Menge von Fragmenten, jeweils mit einer ausgehende Kante minimalen Gewichts
 - ▶ Sei F ein Fragment im niedrigsten Level mit kleinster ausgehenden Kante in diesem Level
 - ▶ Jede $\langle \text{test} \rangle$ Nachricht von F wird beantwortet oder weckt einen neuen Knoten auf
 - ▶ Jede $\langle \text{connect} \rangle$ Nachricht weckt einen neuen Knoten auf oder
 - ▶ Geht an Fragment höheren Levels $\Rightarrow F$ wird eingegliedert
 - ▶ Geht an Fragment F' gleichen Levels mit gleicher minimaler ausgehenden Kante $\Rightarrow F$ und F' verschmelzen

□

Nachrichtenkomplexität II

- ▶ In jedem Level ausser 0 und dem höchsten Level kann ein Knoten
 - ▶ je einmal $\langle \text{accept} \rangle$ und $\langle \text{init} \rangle$ erhalten,
 - ▶ je einmal $\langle \text{test} \rangle$ (erfolgreich) und $\langle \text{report} \rangle$ senden, und
 - ▶ einmal $\langle \text{change-core} \rangle$ oder $\langle \text{connect} \rangle$ senden.
- $5n(-1 + \log n)$ Nachrichten
- ▶ In Level 0 kann jeder Knoten einmal $\langle \text{init} \rangle$ empfangen und einmale $\langle \text{connect} \rangle$ senden, und im höchsten Level einmal $\langle \text{report} \rangle$ senden: $5n$ Nachrichten
 - ▶ Insgesamt also höchstens $2m + 5n \log n$ Nachrichten

□

Nachrichtenkomplexität I

Lemma 13

Die Anzahl der Nachrichten im verteilten MST-Algorithmus ist $\mathcal{O}(m + n \log n)$.

Beweis.

- ▶ Ein Fragment in Level $L + 1$ enthält immer zwei Fragmente in Level L
 - ▶ Level- L Fragmente enthalten mindestens 2^L Knoten
 - ▶ Der maximale Level ist durch $\log n$ beschränkt
- ▶ Eine Kante wird höchstens einmal als rejected markiert, dabei wird eine $\langle \text{test} \rangle$ und eine $\langle \text{reject} \rangle$ Nachricht verschickt: $2m$ Nachrichten

Untere Schranke Nachrichtenkomplexität I

Lemma 13

Die Anzahl der Nachrichten für einen verteilten, uniformen und asynchronen MST-Algorithmus ist $\Omega(m + n \log n)$. D.h. unser MST-Algorithmus ist optimal bzgl. Nachrichtenkomplexität.

Beweis.

- ▶ Über jede Kante muss eine Nachricht geschickt werden, denn sonst könnte sich „in der Mitte“ dieser Kante ein Knoten befinden, der nicht gefunden würde
 - ▶ $\Omega(m)$ Nachrichten

Untere Schranke Nachrichtenkomplexität II

- ▶ Leader Election in Ringen benötigt $\Omega(n \log n)$ Nachrichten im asynchronen Fall (ohne Beweis; s. Kapitel „Verteilte Algorithmen“)
 - ▶ Algorithmus für MST \Rightarrow Algorithmus für Leader Election mit gleicher Laufzeit
 - ▶ $\Omega(n \log n)$ Nachrichten

□

Zeitkomplexität I

Lemma 13

Der verbesserte MST-Algorithmus („mit Aufwecken“) terminiert nach spätestens $\mathcal{O}(n \log n)$ Zeiteinheiten.

Beweis.

- ▶ Zeigen per Induktion: Nach $5ln - 3n$ Zeiteinheiten sind alle Knoten in Level l
- ▶ Induktionsanfang $l = 1$:
 - ▶ Nach n Zeiteinheiten sind alle Prozessoren aufgewacht
 - ▶ Jeder Prozessor hat dann eine $\langle \text{connect} \rangle$ Nachricht verschickt
 - ▶ Nach $2n$ Zeiteinheiten ist jeder Prozessor in Level 1 (Empfang oder Auslösen von $\langle \text{init} \rangle$)

Schlechte Zeitkomplexität

- ▶ Es gibt Beispiele, für die $\Omega(n^2)$ Zeiteinheiten notwendig sind
- ▶ Problem dabei
 - ▶ Nur ein Knoten wacht spontan auf
 - ▶ $\Omega(n^2)$ sequentielle Nachrichten
- ▶ Verbesserung des Algorithmus
 - ▶ Zu Beginn des Algorithmus „aufwecken“ aller Knoten
 - ▶ Schicke $\langle \text{wake-up} \rangle$ über alle Kanten, auf denen noch keine solche Nachricht empfangen wurde
 - ▶ $\mathcal{O}(n)$ Zeiteinheiten, $\mathcal{O}(m)$ Nachrichten

Zeitkomplexität II

- ▶ Induktionsschritt
 - ▶ In Level l sendet jeder Knoten max. n $\langle \text{test} \rangle$ Nachrichten, die spätestens nach $2n$ Zeiteinheiten beantwortet sind
 - ▶ Weiter $3n$ Zeiteinheiten für
 - ▶ $\langle \text{report} \rangle$
 - ▶ $\langle \text{change-root} \rangle$ und $\langle \text{connect} \rangle$
 - ▶ $\langle \text{init} \rangle$
 - ▶ Danach, zum Zeitpunkt $5ln - 3n + 5n = 5(l+1)n - 3n$, sind alle Prozessoren in Level $l+1$

□

Bessere Zeitkomplexität

- ▶ Es gibt Beispiele, für die der verbesserte Algorithmus $\Omega(n \log n)$ Zeiteinheiten benötigt werden
- ▶ $\Omega(n)$ ist triviale untere Schranke
- ▶ Der beschriebene MST-Algorithmus kann weiter verbessert werden auf optimale Zeitkomplexität von $\mathcal{O}(n)$

Dominating Set

PTAS für Dominating Sets
Literatur

- ▶ M. V. Marathe, H. Breu, H.B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz: Simple Heuristics for Unit Disk Graphs. *Networks* 25, 59–68, 1995.
<http://arxiv.org/pdf/math/9409226>
- ▶ T. Nieberg and J. L. Hurink: A PTAS for the minimum dominating set problem in unit disk graphs. Memorandum No. 1732, University of Twente. 2004.
<http://www.math.utwente.nl/publicaties/2004/1732.pdf>

Aktuelle Forschung

- ▶ Bessere Algorithmen für spezielle Graphklassen

- ▶ Durchmesser

$$D(G) = \max_{u \in V} \max_{v \in V} d(u, v)$$

- ▶ Radius

$$R(G) = \min_{u \in V} \max_{v \in V} d(u, v)$$

- ▶ MST-Radius $\mu(G, W)$

- ▶ Zeitkomplexität in Abhängigkeit von D bzw. μ

- ▶ $\mathcal{O}(D(G) + \sqrt{n} \log^* n)$

- ▶ $\mathcal{O}(\mu(G, W) + \sqrt{n})$

Dominating Set Grundlagen

Dominating Set (DS)

Definition 13

Sei $G = (V, E)$ ein ungerichteter Graph und $W \subseteq V$. Eine Teilmenge $D \subseteq V$ heisst **W -Dominating Set** g.d.w. für jeden Knoten $u \in W$ gilt, dass $u \in D$ oder es ein Kante $\{u, v\} \in E$ gibt mit $v \in D$. Im Fall $W = V$ heisst D **Dominating Set von G** .

Problem Minimum Dominating Set (MDS)

Problem MDS

Gegeben: Graph $G = (V, E)$

Gesucht: Ein Dominating Set D von G minimaler Kardinalität $|D|$

Entscheidungsproblem MDS

Gegeben: Graph $G = (V, E)$, Konstante K

Gesucht: Gibt es ein Dominating Set D von G mit Kardinalität $|D| \leq K$?

Komplexitätsstatus von MDS

- ▶ MDS ist \mathcal{NP} -vollständig
- ▶ Spezialfall des Problems Set Cover
- ▶ Bestmögliche Approximation:
 - ▶ Faktor $\mathcal{O}(\log \Delta)$ schlechter als optimale Lösung (Δ ist maximaler Knotengrad)
 - ▶ Einfacher Greedy-Algorithmus

Dominating Sets in Sensornetzen

- ▶ Routing
 - ▶ Connected Dominating Set (s. Übung)
- ▶ Energiesparen
- ▶ Clustering
- ▶ Initialisierungsphase

Approximation von MDS

1. $D := \emptyset$
2. Solange es nicht dominierte Knoten gibt
3. Sei v ein nicht dominierter Knoten, der die meisten nicht dominierten Knoten abdeckt
4. $D := D \cup \{v\}$

Independent Set (IS)

Definition 14

Sei $G = (V, E)$ ein ungerichteter Graph. Eine Teilmenge $I \subseteq V$ heisst **Independent Set** g.d.w. für jede Kante $\{u, v\} \in E$ gilt, dass nicht beide Knoten u und v in I sind.

Maximum Independent Set

Problem Maximum Independent Set

Gegeben: Graph $G = (V, E)$

Gesucht: Ein **kardinalitätsmaximal** Independent Set I von G ,
d.h. für alle Independent Sets I' gilt $|I| \geq |I'|$

- Maximum Independent Set ist \mathcal{NP} -vollständig
(I ist Independent Set $\Leftrightarrow V \setminus I$ ist Vertex Cover)

Maximal Independent Set

Problem Maximal Independent Set

Gegeben: Graph $G = (V, E)$

Gesucht: Ein **inklusionsmaximales** Independent Set I von G ,
d.h. es gibt keinen Knoten $v \in V \setminus I$ so dass
 $I \cup \{v\}$ ein Independent Set ist

- Maximal Independent Set ist in Linearzeit lösbar
 - Solange eine Knoten $v \in V \setminus I$ existiert, dessen Nachbarn alle nicht in I sind, füge v zu I hinzu

Zusammenhang Independent und Dominating Set

- Jedes kardinalitätsmaximale Independent Set ist ein inklusionsmaximales Independent Set
- Jedes inklusionsmaximale Independent Set ist ein Dominating Set
 - Sei V' ein inklusionsmaximales Independent Set in $G = (V, E)$
 - Annahme: V' ist kein Dominating Set. Dann existiert $u \in V \setminus V'$, so dass für alle Kanten $\{u, v\} \in E$ gilt $v \notin V'$
 - $V' \cup \{u\}$ ist ein Independent Set, das V' echt enthält
 - Widerspruch zur Voraussetzung „ V' inklusionsmaximal“
- Problem Minimum Independent Dominating Set

Unit Disk Graph (Wdh.)

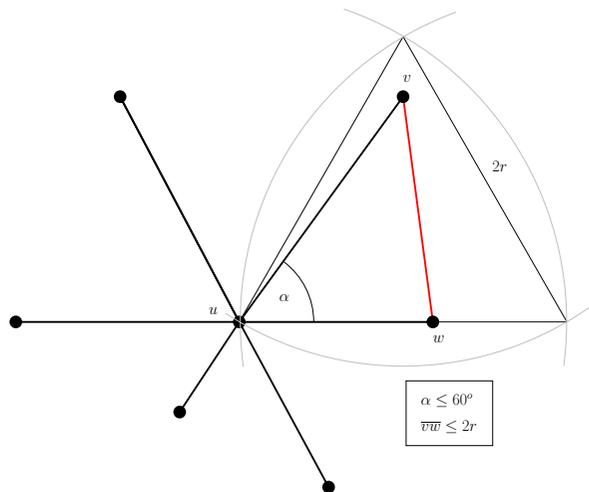
Definition 15

Ein Graph $G = (V, E)$ heisst **Unit Disk Graph (UDG)** g.d.w. es eine Einbettung der Knoten in die Ebene gibt, so dass $\{u, v\} \in E$ g.d.w. sich die Kreise um u und v mit Radius 1 schneiden.

Bemerkung

Ein Graph $G = (V, E)$ ist ein Unit Disk Graph g.d.w. es eine Einbettung der Knoten in die Ebene gibt, so dass $\{u, v\} \in E$ g.d.w. der euklidische Abstand zwischen u und v höchstens 1 ist.

Beweisskizze Sternlemma



Sternlemma

Lemma 16

Sei $G = (V, E)$ ein Unit Disk Graph. Dann enthält G keinen $K_{1,6}$ als knoteninduzierten Teilgraph.

Faktor-5 Approximation I

Satz 17

Sei $G = (V, E)$ ein Unit Disk Graph, D_{OPT} ein kardinalitätsminimales Dominating Set und I ein inklusionsmaximales Independent Set. Dann ist I ein Dominating Set mit $|I| \leq 5|D_{OPT}|$.

Bemerkung

Der Linearzeit-Algorithmus für Maximal Independent Set liefert also eine Faktor-5 Approximation für Dominating Set in Unit Disk Graphen in linearer Zeit.

Faktor-5 Approximation II

Beweis.

- ▶ Ein Knoten v in D_{OPT} dominiert höchstens 5 Knoten in I
 - ▶ Falls $v \in I$, so dominiert v keinen weiteren Knoten in I
 - ▶ Ansonsten angenommen v dominiert $v_1, \dots, v_6 \in I$.
Dann würde durch v und die v_i ein $K_{1,6}$ induziert (Knoten in I sind nicht durch Kanten verbunden), im Widerspruch zum obigen Lemma 16.
- ▶ Da D_{OPT} ganz V und somit auch ganz I dominiert, kann I höchstens 5-mal mehr Knoten enthalten als D_{OPT} .

□

Eigenschaften des PTAS

- ▶ Keine geometrische Einbettung als Unit Disk Graph gegeben
 - ▶ Es ist \mathcal{NP} -vollständig solch eine Einbettung zu konstruieren
 - ▶ Viele andere Algorithmen basieren darauf
- ▶ Algorithmus ist **robust**, d.h. in polynomialer Zeit wird entweder eine Lösung berechnet oder ein Nachweis, dass G kein Unit Disk Graph ist

Polynomiales Approximationschema (PAS) (Polynomial Time Approximation Scheme, PTAS)

- ▶ Eingabe: Unit Disk Graph $G = (V, E)$ und eine Konstante ε
- ▶ Gesucht: Dominating Set $D \subseteq V$ mit $|D| \leq (1 + \varepsilon)|D_{OPT}|$
- ▶ Laufzeit des Algorithmus polynomial in $|G|$

Nachbarschaft I

Definition 18

Sei $G = (V, E)$ ein ungerichteter Graph, $v \in V$ ein Knoten und $W \subseteq V$ eine Teilmenge aller Knoten.

- ▶ $N(v) := \{u \in V \mid \{u, v\} \in E\} \cup \{v\}$ ist die **abgeschlossene Nachbarschaft** von v
- ▶ $N(W) := \bigcup_{u \in W} N(u)$ bezeichnet die **abgeschlossene Nachbarschaft** von W
- ▶ $N^r(v) := N(N^{r-1}(v))$ definiert rekursiv die **r -Nachbarschaft** von v , wobei $N^1(v) := N(v)$

Nachbarschaft II

Definition 19

Der **graphentheoretische Abstand** $d(v, w)$ zweier Knoten $v, w \in V$ ist die minimale Anzahl der Kanten auf einem v - w -Weg.

Bemerkung

- ▶ $D \subseteq V$ ist ein Dominating Set g.d.w. $N(D) = V$.
- ▶ $N^r(v)$ enthält alle Knoten w mit $d(v, w) \leq r$.

2-Separation II

Lemma 21

Sei $\mathcal{S} = \{S_1, \dots, S_k\}$ eine 2-Separation, D_{OPT} ein Dominating Set minimaler Kardinalität von G , und $D(S_i)$ seien S_i -Dominating Sets minimaler Kardinalität in G . Dann gilt

$$\sum_{i=1}^k |D(S_i)| \leq |D_{OPT}|.$$

2-Separation I

Definition 20

Sei $G = (V, E)$ ein ungerichteter Graph. Eine Menge $\mathcal{S} := \{S_1, \dots, S_k\}$ mit $S_i \subseteq V$ ($1 \leq i \leq k$) heisst **2-Separation** g.d.w. für je zwei Knoten $s \in S_i$ und $\bar{s} \in S_j$ mit $i \neq j$ gilt, dass $d(s, \bar{s}) > 2$.

2-Separation III

Beweis.

- ▶ Da \mathcal{S} 2-Separation sind die $N(S_i)$ paarweise disjunkt
- ▶ $D_{OPT} \cap N(S_i)$ ist S_i -Dominating Set
- ▶ $D(S_i)$ ist **kardinalitätsminimales** S_i -Dominating Set, also $|D(S_i)| \leq |D_{OPT} \cap N(S_i)|$
- ▶ Somit folgt

$$\sum_{i=1}^k |D(S_i)| \leq \sum_{i=1}^k |D_{OPT} \cap N(S_i)| \leq |D_{OPT}|$$

□

Algorithmus

1. $X := V, D := \emptyset$
2. Solange X nicht leer sei $v \in X$
3. $r := 0$
4. Solange $|D(N^{r+2}(v))| > (1 + \varepsilon)|D(N^r(v))|$
5. $r := r + 1$
6. $X := X \setminus N^{r+2}(v), D := D \cup D(N^{r+2}(v))$

Bemerkungen

- ▶ $D(W)$ bezeichnet ein kardinalitätsminimales W -Dominating Set in G (Enumeration aller möglichen Lösungen)
- ▶ $N^r(v)$ bezeichnet Nachbarschaft im von X induzierten Graph

Bezeichnungen

- ▶ Betrachte Schritt i der Hauptschleife (2)
- ▶ v_i und X_i sind das v bzw. X aus Schritt i
- ▶ r_i bezeichne das maximale r in Schritt i
- ▶ $N_i^{r_i+2} := N_{X_i}^{r_i+2}(v_i)$ die Knoten, die in Schritt i entfernt werden
- ▶ $N_i := N_{X_i}^{r_i}(v_i)$
- ▶ k sei die Anzahl der Schritte in der Hauptschleife

Korrektheit I

Lemma 22

Nach Ausführung des Algorithmus gilt

$$\bigcup_{i=1}^k N_i^{r_i+2} = V$$

und $\mathcal{N} := \{N_1, \dots, N_k\}$ ist eine 2-Separation in G .

Korrektheit II

Satz 23

Die Knotenmenge $D := \bigcup_{i=1}^k D(N_i^{r_i+2})$ ist ein Dominating Set von G , wobei $|D| \leq (1 + \varepsilon)|D_{OPT}|$.

Bemerkung

Dieser Satz gilt für beliebige ungerichtete Graphen.

Korrektheit III

Beweis.

D ist Dominating Set von G weil mit obigem Lemma 22 jeder Knoten von G in einem N_i^{r+2} enthalten ist und somit von einem Knoten in D dominiert wird. Weiter gilt

$$\begin{aligned}
 |D| &= \left| \bigcup_{i=1}^k D(N_i^{r+2}) \right| \leq \sum_{i=1}^k |D(N_i^{r+2})| \\
 &\leq (1 + \varepsilon) \sum_{i=1}^k |D(N_i)| \quad \text{[Def. Algorithmus]} \\
 &\leq (1 + \varepsilon) D_{OPT} \quad \text{[Lemma 21 und 22]}
 \end{aligned}$$



Laufzeit

Satz 25

Sei G ein Unit Disk Graph und $\varepsilon > 0$ eine Konstante. Dann ist die Laufzeit des Algorithmus polynomial in der Anzahl der Knoten n .



Intermezzo: Independent Set in UDG

Lemma 24

Sei $G = (V, E)$ ein Unit Disk Graph, $v \in V$, und $I^r \subseteq N^r(v)$ ein Independent Set. Dann gilt

$$|I^r| \leq (2r + 1)^2 \in \mathcal{O}(r^2).$$

Beweis.

- ▶ Betrachte korrekte Einbettung des UDG
- ▶ Die Knoten in I^r sind nicht durch Kanten verbunden
- ▶ Einheitskreise mit Fläche π um die Knoten in I^r sind disjunkt
- ▶ All diese Kreise sind in Kreis um v mit Radius $2r + 1$ enthalten
- ▶ $|I^r| \leq \pi(2r + 1)^2 / \pi = (2r + 1)^2$



Beweis Laufzeit I

Beweis.

- ▶ Betrachte den Schleifendurchlauf mit größten Wert von r
 - ▶ Gesamtlaufzeit wird dadurch dominiert
- ▶ Es gilt $|D(N^r(v))| \leq (2r + 1)^2$
 - ▶ Nach Lemma 24 hat ein beliebiges kardinalitätsmaximales Independent Set I^r höchstens $(2r + 1)^2$ Knoten
 - ▶ I^r ist auch ein Dominating Set
 - ▶ Ein kardinalitätsminimales Dominating Set hat also höchstens $(2r + 1)^2$ Knoten
- ▶ Berechnung von $D(N^r(v))$ durch Enumeration
 - ▶ Es gibt $\mathcal{O}(n^\alpha)$ viele Kandidaten, wobei $\alpha \in \mathcal{O}(r^2)$



Beweis Laufzeit II

- ▶ r ist beschränkt durch eine Konstante c abhängig nur von ε
 - ▶ O.B.d.A. sei r gerade (sonst endet folgende Ungl. bei $N^1(v)$)
 - ▶ Es gilt nach Konstruktion des Algorithmus:

$$\begin{aligned} (2r + 1)^2 &\geq |D(N^r(v))| > (1 + \varepsilon)|D(N^{r-2}(v))| > \dots \\ &> (1 + \varepsilon)^{r/2}|D(N^0(v))| \\ &= (\sqrt{1 + \varepsilon})^r \end{aligned}$$

- ▶ $(\sqrt{1 + \varepsilon})^r$ wächst asymptotisch schneller als $(2r + 1)^2$
- ▶ Obige Ungleichung kann also nur für $r \leq c$ gelten, wobei c nur von ε abhängt
- ▶ Gesamtlaufzeit ist polynomial in n
 $(\mathcal{O}(n^{c^2})$ mit $c \in \mathcal{O}(\frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon}))$



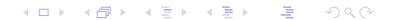
Verteilter Algorithmus für Dominating Set
Literatur

- ▶ F. Kuhn und R. Wattenhofer: Constant-Time Distributed Dominating Set Approximation. *Proceedings 22nd ACM Symposium on Principles of Distributed Computing (PODC)*, Boston, Massachusetts, USA, Juli 2003.
http://dgc.ethz.ch/members/pascal/refs/ds_2003_kuhn.pdf



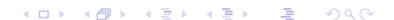
Robustheit

- ▶ Algorithmus bricht ab falls kein $N^r(v)$ -Dominating Set der Größe $(2r + 1)^2$ gefunden wird
- ▶ Kontruierere dann ein maximales Independent Set I^r von $N^r(v)$
 - ▶ $|I^r| > (2r + 1)^2$
 - ▶ Nachweis, dass der Graph kein Unit Disk Graph ist
- ▶ Falls der Graph kein Unit Disk Graph ist wird
 - ▶ entweder ein Dominating Set D mit $|D| \leq (1 + \varepsilon)D_{OPT}$ geliefert
 - ▶ oder ein Nachweis geliefert, dass der kein Unit Disk Graph ist,
 - ▶ in polynomialer Laufzeit in der Größe des Graphen.



Notation

- ▶ Graph $G = (V, E)$, Knotenanzahl $n = |V|$
- ▶ Knoten $V = \{1, \dots, n\}$
- ▶ Matrizen: $A, N, I \in \mathbb{R}^{n \times n}$
- ▶ Vektoren: $x, y, b, c \in \mathbb{R}^n$
- ▶ Knotengrad: δ_i
- ▶ Maximaler Knotengrad in r -Nachbarschaft:
 $\delta_i^{(r)} := \max\{\delta_j | j \in N^r(i)\}$



Mathematische Modellierung

- ▶ Modelliere Teilmenge $D \subseteq V$
 - ▶ Pro Knoten $i \in V$ eine Variable $x_i \in \{0, 1\}$
 - ▶ $x_i = 1$ g.d.w. $i \in D$
- ▶ Sei A die Adjazenzmatrix von $G = (V, E)$
 - ▶ A ist $n \times n$ Matrix
 - ▶ $A_{ij} = 1$ g.d.w. $\{i, j\} \in E$
- ▶ Nachbarschaftsmatrix $N := A + I$
 - ▶ I ist die $n \times n$ Einheitsmatrix
 - ▶ $N_{ij} = 1$ g.d.w. $j \in N(i)$
- ▶ $D \subseteq V$ ist Dominating Set g.d.w. $N \cdot x \geq 1$

ILP für Dominating Set (Integer Linear Program)

$$\begin{aligned} \min \quad & \sum_{i=1}^n x_i \\ N \cdot x \quad & \geq 1^n \\ x \in \quad & \{0, 1\}^n \end{aligned}$$

Relaxierung: LP (Linear Program)

$$\begin{aligned} \min \quad & \sum_{i=1}^n x_i \\ N \cdot x \quad & \geq 1^n \\ x \quad & \geq 0^n \end{aligned}$$

Duales LP

$$\begin{aligned} \max \quad & \sum_{i=1}^n y_i \\ N \cdot y \quad & \leq 1^n \\ y \quad & \geq 0^n \end{aligned}$$

Dualitätssatz

Satz 26

Sei $\min \sum_{i=0}^n x_i$ die Zielfunktion eines LP und $\max \sum_{i=0}^n y_i$ die Zielfunktion des dualen LP. Dann gilt für zulässige Lösungen x' und y'

$$\sum_{i=0}^n y'_i \leq \sum_{i=0}^n x'_i.$$

(ohne Beweis)

Untere Schranke II

Beweis.

- ▶ Ein Dominating Set D liefert eine zulässige LP-Lösung x
- ▶ Setze $y_i := 1/(\delta_i^{(1)} + 1)$
- ▶ y ist zulässige Lösung für das duale LP

$$\begin{aligned} (N \cdot y)_i &= \sum_{j \in N(i)} y_j \leq \sum_{j \in N(i)} \frac{1}{\delta_j + 1} \\ &= (\delta_i + 1) \frac{1}{\delta_i + 1} = 1 \end{aligned}$$

- ▶ Mit dem Dualitätssatz gilt

$$\sum_{i=1}^n y_i \leq \sum_{i=1}^n x_i = |D|$$

Untere Schranke I

Lemma 27

Sei $\delta_i^{(1)}$ das Maximum aller Knotengrade in $N(i)$. Für ein Dominating Set D gilt

$$\sum_{i=1}^n \frac{1}{\delta_i^{(1)} + 1} \leq |D|.$$

Algorithmus Runden

Gegeben: α -Approximation $x^{(\alpha)}$ für Relaxierung

Gesucht: Dominating Set D über Variablen x_D

1. berechne $\delta_i^{(2)}$
2. $p_i := \min \left\{ 1, x_i^{(\alpha)} \ln(\delta_i^{(2)} + 1) \right\}$
3. $x_{D,i} := \begin{cases} 1 & \text{mit Wahrscheinlichkeit } p_i \\ 0 & \text{sonst} \end{cases}$
4. sende $x_{D,i}$ an alle Nachbarn
5. falls $x_{D,j} = 0$ für alle $j \in N(i)$
6. $x_{D,i} := 1$

Erwartungswert Runden

Satz 28

Sei Δ der Maximalgrad in G , und $x^{(\alpha)}$ eine α -Approximation für das LP. Dann ist x_D eine zulässige ILP-Lösung und der Erwartungswert der Größe des Dominating Set D ist

$$E[|D|] \leq (1 + \alpha \ln(\Delta + 1)) |D_{OPT}|.$$

Beweis Erwartungswert II

- ▶ Sei q_i die Wahrscheinlichkeit, dass kein Knoten in $N(i)$ in Schritt 3 ausgewählt wurde: $q_i = \prod_{j \in N(i)} (1 - p_j)$
- ▶ Falls ein $j \in N(i)$ existiert mit $p_j = 1$ so ist $q_i = 0$.
- ▶ Im Folgenden betrachte nur i mit $p_j < 1$ für alle $j \in N(i)$.
- ▶ Für $j \in N(i)$ gilt $\delta_i^{(1)} \leq \delta_j^{(2)}$
- ▶ Also gilt für alle $j \in N(i)$:

$$0 < 1 - p_j = 1 - x_j^{(\alpha)} \ln(\delta_j^{(2)} + 1) \leq 1 - x_j^{(\alpha)} \ln(\delta_i^{(1)} + 1)$$

Beweis Erwartungswert I

Beweis

- ▶ Für eine optimale Lösung x^* des LP gilt $\sum_{i=1}^n x_i^* \leq |D_{OPT}|$.
- ▶ Für die Anzahl X der in Schritt 3 ausgewählten Knoten gilt:

$$\begin{aligned} E[X] &= \sum_{i=1}^n p_i \leq \sum_{i=1}^n x_i^{(\alpha)} \ln(\delta_i^{(2)} + 1) \\ &\leq \ln(\Delta + 1) \sum_{i=1}^n x_i^{(\alpha)} \leq \alpha \ln(\Delta + 1) \sum_{i=1}^n x_i^* \\ &\leq \alpha \ln(\Delta + 1) |D_{OPT}|. \end{aligned}$$

Zwei Ungleichungen

Für eine endliche Menge von positiven reellen Zahlen \mathcal{A} gilt

$$\prod_{x \in \mathcal{A}} x \leq \left(\frac{\sum_{x \in \mathcal{A}} x}{|\mathcal{A}|} \right)^{|\mathcal{A}|}$$

und für $1 \leq x \leq n$ gilt

$$\left(1 - \frac{x}{n} \right)^n \leq e^{-x}.$$

Beweis Erwartungswert III

$$\begin{aligned}
 q_i &= \prod_{j \in N(i)} (1 - p_j) \leq \prod_{j \in N(i)} \left(1 - x_j^{(\alpha)} \ln(\delta_i^{(1)} + 1)\right) \\
 &\leq \left(1 - \frac{\sum_{j \in N(i)} x_j^{(\alpha)} \ln(\delta_i^{(1)} + 1)}{\delta_i + 1}\right)^{\delta_i + 1} \\
 &\leq \left(1 - \frac{\ln(\delta_i^{(1)} + 1)}{\delta_i^{(1)} + 1}\right)^{\delta_i^{(1)} + 1} \leq e^{-\ln(\delta_i^{(1)} + 1)} \\
 &= \frac{1}{\delta_i^{(1)} + 1}
 \end{aligned}$$

Algorithmus LP

Gegeben: Δ, k Gesucht: zulässige LP-Lösung x

1. $x_i := 0$, farbe_{*i*} := weiss
2. für $l := k - 1$ bis 0
3. für $m := k - 1$ bis 0
4. sende farbe_{*i*} an alle Nachbarn
5. $\tilde{\delta}(i) := |\{j \in N(i) \mid \text{farbe}_j = \text{weiss}\}|$
6. falls $\tilde{\delta}(i) \geq (\Delta + 1)^{l/k}$
7. $x_i := \max \left\{ x_i, \frac{1}{(\Delta + 1)^{m/k}} \right\}$
8. sende x_i an alle Nachbarn
9. falls $\sum_{j \in N(i)} x_j \geq 1$ setze farbe_{*i*} := grau

Beweis Erwartungswert IV

► Für die Anzahl Y der in Schritt 6 ausgewählten Knoten gilt mit Lemma 27:

$$E[Y] = \sum_{i=1}^n q_i \leq \sum_{i=1}^n \frac{1}{\delta_i^{(1)} + 1} \leq |D_{OPT}|$$

► Insgesamt ergibt sich

$$E[|D|] = E[X] + E[Y] \leq (1 + \alpha \ln(\Delta + 1)) |D_{OPT}|$$

► x_D ist zulässig: in Schritt 6 wird für alle noch nicht dominierten Knoten $x_{D,i} := 1$ gesetzt □

Invariante 1

Lemma 29

Am Anfang jeden Durchlaufs der äußeren Schleife gilt

$$\tilde{\delta}(i) \leq (\Delta + 1)^{(l+1)/k}.$$

Beweis.

- Für $l = k - 1$ folgt $\tilde{\delta}(i) \leq \Delta + 1$ nach Definition von Δ
- Ansonsten: Sei i Knoten mit $\tilde{\delta}(i) > (\Delta + 1)^{(l+1)/k}$
 - Am Ende der vorigen Iteration $l + 1$ wurde $x_i := 1$ gesetzt
 - Alle Knoten in $N(i)$ wurden grau markiert
 - Also $\tilde{\delta}(i) = 0$ □

Aktive Knoten

Definition 30

Knoten mit $\tilde{\delta}(i) \geq (\Delta + 1)^{l/k}$ heissen **aktiv**.

Für weiße Knoten wird die Anzahl aktiver Knoten in $N(i)$ mit $a(i)$ bezeichnet. Für graue Knoten ist $a(i) = 0$.

Erhöhung in innerer Schleife

- ▶ Variable z_i pro Knoten
- ▶ $z_i := 0$ vor Beginn der inneren Schleife
- ▶ Bei Erhöhung von x_i um d
 - ▶ Verteile d auf die z_j für alle *weißen* $j \in N(i)$
- ▶ Es gilt also

$$\sum_{i \in V} z_i = \sum_{i \in V} x_i$$

Invariante 2

Lemma 31

Am Anfang jeden Durchlaufs der inneren Schleife gilt

$$a(i) \leq (\Delta + 1)^{(m+1)/k}.$$

Beweis.

- ▶ Für $m = k - 1$ folgt $a(i) \leq \Delta + 1$ nach Definition von Δ .
- ▶ Ansonsten: Sei i Knoten mit $a(i) > (\Delta + 1)^{(m+1)/k}$
 - ▶ Am Ende der vorigen Iteration $m + 1$ gilt für aktive Knoten $x_j \geq 1/(\Delta + 1)^{(m+1)/k}$
 - ▶ Die Summe der x -Werte in $N(i)$ ist größer als 1
 - ▶ i wurde demnach grau markiert, und $a(i) = 0$ □

Invariante 3

Lemma 32

Am Ende jeden Durchlaufs der äußeren Schleife gilt

$$z_i \leq \frac{1}{(\Delta + 1)^{\frac{l-1}{k}}}.$$

Invariante 3 - Beweis I

Beweis.

- ▶ Für Knoten i kann z_i nur erhöht werden solange i weiss ist
- ▶ Phase I:
 - ▶ Alle Iterationen bei denen i weiss bleibt
 - ▶ Es gilt stets $\sum_{j \in N(i)} x_j < 1$
 - ▶ Erhöhungen der x -Variablen werden auf $(\Delta + 1)^{l/k}$ z -Variablen aufgeteilt, also

$$z_i \leq \frac{\sum_{j \in N(i)} x_j}{(\Delta + 1)^{l/k}} < \frac{1}{(\Delta + 1)^{l/k}}$$

nach Phase I

Invariante 3 - Beweis III

- ▶ z_i wird nur für weisse Knoten erhöht
 - ▶ nach Phase II ändert sich z_i nicht mehr
- ▶ Phase I und Phase II zusammengezählt ergibt

$$z_i \leq \frac{(\Delta + 1)^{\frac{1}{k}} - 1}{(\Delta + 1)^{\frac{l}{k}}} + \frac{1}{(\Delta + 1)^{\frac{l}{k}}} = \frac{1}{(\Delta + 1)^{\frac{l-1}{k}}}$$

□

Invariante 3 - Beweis II

- ▶ Phase II: Die Iteration bei der i grau wird
 - ▶ Aktive Knoten $j \in N(i)$ waren schon in voriger Iteration aktiv, und $x_j \geq 1/(\Delta + 1)^{(m+1)/k}$
 - ▶ x_j wird nun auf $1/(\Delta + 1)^{m/k}$ erhöht
 - ▶ Die Differenz wird auf mindestens $(\Delta + 1)^{l/k}$ z -Variablen verteilt
 - ▶ Die Anzahl aktiver Knoten in $N(i)$ ist $a(i)$, somit erhöht sich z_i um höchstens

$$\frac{\frac{1}{(\Delta+1)^{\frac{m}{k}}} - \frac{1}{(\Delta+1)^{\frac{m+1}{k}}}}{(\Delta+1)^{\frac{l}{k}}} a(i) \leq \frac{(\Delta+1)^{\frac{1}{k}} - 1}{(\Delta+1)^{\frac{l}{k}}}$$

Ergebnis

Satz 33

Der Algorithmus LP berechnet eine zulässige Lösung x für das Lineare Programm, wobei x eine $k(\Delta + 1)^{2/k}$ -Approximation der optimalen Lösung darstellt. Der Algorithmus terminiert nach $2k^2$ Runden.

Beweis Satz I

Beweis

- ▶ In jeder inneren Schleife werden 2 Nachrichten gesendet
 - ▶ Insgesamt $2k^2$ Runden
- ▶ Die ausgegebene Lösung ist zulässig
 - ▶ Beim letzten Schleifendurchlauf wird $x_i := 1$ gesetzt für jeden noch weissen Knoten i
 - ▶ Für graue Knoten i ist $\sum_{j \in N(i)} x_j \geq 1$
 - ▶ Also $N \cdot x \geq 1$

Beweis Satz III

- ▶ Betrachte $y_i := z_i / (\Delta + 1)^{2/k}$
 - ▶ Es gilt $\sum_{j \in N(i)} y_j \leq 1$ für alle i
 - ▶ Also ist y zulässige Lösung des dualen LP
 - ▶ Sei x^* optimale Lösung des LP
 - ▶ Dualitätssatz: $\sum_{i=1}^n y_i \leq \sum_{i=1}^n x_i^*$
- ▶ Damit gilt $\sum_{i=1}^n z_i \leq (\Delta + 1)^{2/k} \sum_{i=1}^n x_i^*$
- ▶ $\sum_{i=1}^n z_i$ ist gleich der Erhöhung der x_i in einem Schritt der äußeren Schleife, also

$$\sum_{i=1}^n x_i \leq k(\Delta + 1)^{2/k} \sum_{i=1}^n x_i^*.$$

□

Beweis Satz II

- ▶ Betrachte Durchlauf der äußeren Schleife:

- ▶ Invariante 1:
Es gibt höchstens $(\Delta + 1)^{(l+1)/k}$ positive z_j für $j \in N(i)$
- ▶ Invariante 3: $z_j \leq (\Delta + 1)^{-(l+1)/k}$
- ▶ Somit

$$\sum_{j \in N(i)} z_j \leq \frac{(\Delta + 1)^{\frac{l+1}{k}}}{(\Delta + 1)^{\frac{-(l+1)}{k}}} = (\Delta + 1)^{\frac{2}{k}}$$

