Algorithmen für Ad-hoc- und Sensornetze

VL 01 – Einführung und erste Schritte

Dr. rer. nat. Bastian Katz

Lehrstuhl für Algorithmik I
Institut für theoretische Informatik
Universität Karlsruhe (TH)
Karlsruher Institut für Technologie

22. April 2009
(Version 3 vom 24. April 2009)

Organisatorisches

- Sprechstunde
  - jederzeit nach Vereinbarung, `katz@kit.edu`
  - Raum 318, Gebäude 50.34
- Webseite zur Vorlesung:
  `http://i11www.iti.uni-karlsruhe.de/teaching/sommer2009/sensornetze/index`
  - Folien zur kommenden Vorlesung (immer montags?)
  - korrigierte Foliensätze, Druckversionen
  - Referenzen und weiterführende Literatur
  - Ankündigungen zur Vorlesung

Prüfungen

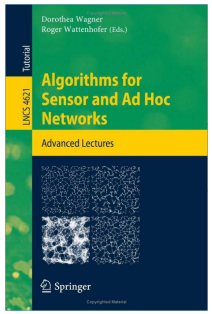
- Prüfungsgebiete
  - Hauptdiplom/Master Info: 2 SWS, 3 LP
    - 01 Theoretische Grundlagen
    - 02 Algorithmentechnik
  - Diplom/Master InWi: 2 SWS, 4 LP
    - Modul „Advanced Algorithms“
  - Graduiertenkolleg GRK1194
- Prüfungsstoff
  - Fragestellungen, Motivationen
  - Algorithmische Ideen und Beweisskizzen
- Parallele Vorlesungen am ITI Wagner:
  - Algorithmen für planare Graphen
  - Algorithmen zur Visualisierung von Graphen
  - Algorithmen für Routenplanung

Voraussetzungen (nice to have)

- Grundbegriffe der Graphentheorie
  - Graph, Knoten, Kante, Grad, inzident, adjazent, Nachbarschaft, Baum, Wald, Kreis, planar, bipartit, Clique
- Algorithmen und Algorithmenanalyse
  - Asymptotische Laufzeit, $O(n)$, $o(n)$, $\Omega(n)$, $\Theta(n)$
  - **P**, **NP**, **NP**-schwer, **NP**-vollständig
  - Lineare Probleme, Approximationsalgorithmen, Randomisierung
- Nachfragen!

Literatur

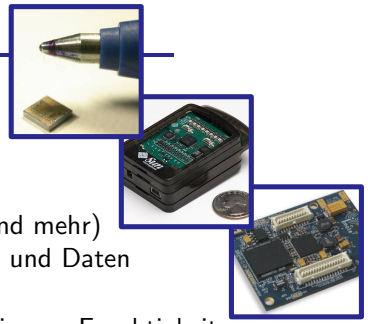
- Kurzskripte *Grundlagen*, `http://i11www.ira.uka.de/information/scripts`
  - *Begriffe der Graphentheorie*
  - *Algorithmen und ihre Laufzeit*
  - *Die Komplexitätsklassen* **P**, **NP** *und* **NPC**
- D. Wagner, R. Wattenhofer (Hrsg.): *Algorithms for Sensor and Ad Hoc Networks*
- Literaturangaben zu einzelnen Themen in der Vorlesung (Webseite, Folien)



Heute

- Organisatorisches
- Einige Worte zu Sensornetzen
  - Sensorknoten, Anwendungen, drahtlose Kommunikation
- Vorlesungsübersicht
- Erste Schritte
  - Broadcast
  - Leader Election
  - Routing: Link Reversal

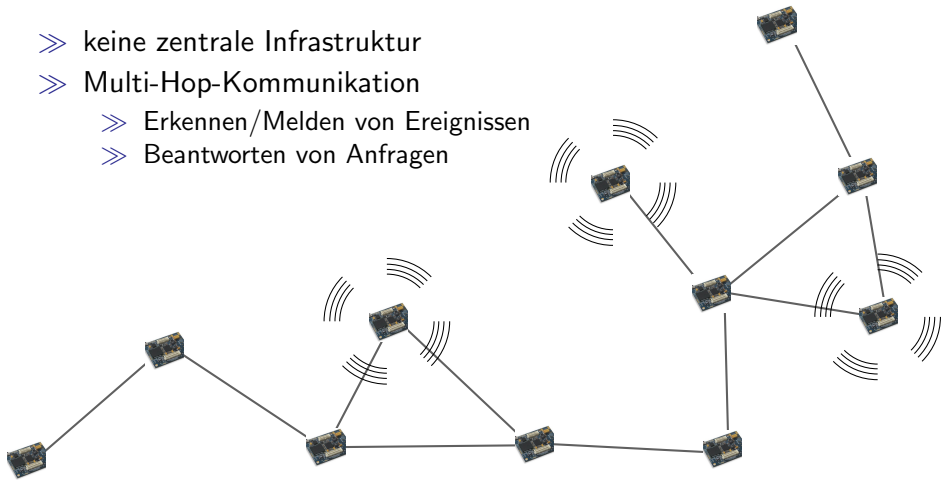
Sensorknoten (aka motes)



- Mikroprozessor
  - wenige MHz bis einige hundert MHz
  - energiesparende Zustände
- Speicher
  - ab 3kB Arbeitsspeicher (bis 512kB und mehr)
  - tw. zusätzlich Flashspeicher für Code und Daten
- Sensorik
  - Temperatur, Licht, Druck, Beschleunigung, Feuchtigkeit uvm.
- Drahtlose Kommunikation
  - Reichweiten von wenigen Metern bis einige hundert Meter
  - Langsame Entwicklung von Standards (z. B. auf Basis von IEEE 802.15.4 wie ZigBee)
- Energieversorgung
  - heute: Batterien
  - in Zukunft: Solarzellen und anderes „Energy Harvesting“

Ausbringen

- keine zentrale Infrastruktur
- Multi-Hop-Kommunikation
  - Erkennen/Melden von Ereignissen
  - Beantworten von Anfragen

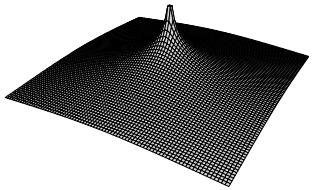


Anwendungen

- Environmental Monitoring
  - Buschfeuer, Gletscherbewegungen
  - Bewässerung von Agrarflächen
- Animal Monitoring
  - Überwachung von Brutstätten, Bewegungen
  - „Intelligente Zäune“
- Structural Monitoring
  - Strukturelle Integrität von Gebäuden
- Überwachung und Tracking
  - Grenz/Gebietsüberwachung
  - Lokalisierungsaufgaben
- ...

Drahtlose Kommunikation – Signalausbreitung

Was selbst Algorithmiker mal gehört haben sollten

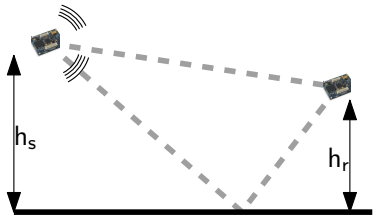


- gemeinsames Medium
- Signalabfall
  - „free space“

$$P_r = \frac{P_s G_s G_r \lambda^2}{(4\pi)^2 L d(r,s)^2}$$

  - „two ray ground“

$$P_r \sim \frac{P_s G_s G_r h_s^2 h_r^2}{d(r,s)^4}$$



- plus alle möglichen Effekte
  - Abschattung, Reflexionen, Diffusion, Antennenform,...

Drahtlose Kommunikation – Empfang

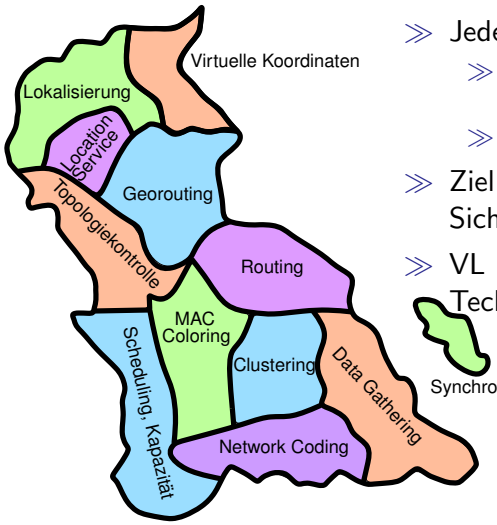
Was selbst Algorithmiker mal gehört haben sollten

- Signalstärke am Empfänger kann ausreichen
  - um die Nachricht zu verstehen
  - um zu verstehen, dass eine Nachricht gesendet wurde
  - um andere Nachrichten zu stören
  - Verbindungen („Links“)
    - Knoten können prinzipiell Nachrichten austauschen
    - (oft nur: geringe Fehlerwahrscheinlichkeit)
    - Verbindungsmodelle ab nächster Woche
  - Interferenz
    - Kommunikation kann nicht beliebig parallel stattfinden
    - einfache Interferenzmodelle ab übernächster Woche
    - komplexere Modelle in der zweiten Hälfte

Herausforderungen

- Geringes Energiebudget
  - geringe Leistung ↔ jahrelange Bereitschaft
  - Balancierung, Aufteilung von Aufgaben
- Geringer Speicher, geringe Rechenleistung
  - verteilte Lösungen ↔ komplexe Selbstorganisation
- Geometrie
  - enger Zusammenhang zwischen Positionen und Daten/Aufgaben
  - gemeinsame Nutzung und Störung der Funkkanäle
- Redundanz, Unzuverlässigkeit und Dynamik
  - Knoten können ausfallen, sich bewegen
  - viele Knoten können Aufgaben redundant ausführen

Vorlesungsübersicht



- Jede VL *ein* Problem in WSN
  - viele Themen algorithmisch sehr isoliert
  - sehr unterschiedliche Sichtweisen
- Ziel: Zurechtfinden auf der Landkarte, Sichtweisen kennenlernen
- VL über neue Ideen, nicht über Technologien!

Hardwaredesign, Telematik, Robotik

Praxis

Realität | Testbed

Simulation

Modelle | Beweise

Theorie

Graphentheorie, Kombinatorik, Geometrie

klassische Sicht

Verbindungen
beliebige Graphen ??? ???

Mobilität, Dynamik
statisch hochdynamisch

Kommunikationsmodell
Message passing ??? beliebige Interferenzen

Knotenpositionen
keine Positionen beliebige Positionen ???

Sensornetze

Verteilte Systeme

Definition

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

- Kommunikationsparadigmen
  - synchron ↔ asynchron
  - nachrichtenbasiert ↔ shared memory
- Ad-hoc Netze sind von Natur aus nachrichtenbasiert!
- Synchrone Kommunikation deutlich einfacher zu analysieren (aber nur durch Synchronisation möglich: VL13)

Verteilte Systeme (etwas formaler)

- n Prozessoren $p_1, \ldots, p_n$ („Knoten“)
- Kommunikationsgraph („Topologie“, „Netz“)
  - Knoten: Prozessoren
  - Kanten: Mögliche Kommunikationspartner
  - nicht immer ungerichtet/symmetrisch
  - oft werden asymmetrische Links einfach ignoriert

Verteilter Algorithmus

- Jeder Prozessor führt *dasselbe* Programm aus
  - Programme starten gleichzeitig (meistens), ggf. in verschiedenen Zuständen (Eingabe)
  - Runde: Berechnungsschritt und Kommunikationsschritt
  - beliebige Berechnungen im Berechnungsschritt
  - *Verfassen* von Nachrichten an Nachbarn in Berechnungsschritt
  - *Zustellen* von Nachrichten in Kommunikationsschritt
- nach jeder Runde hat jeder Prozessor p_i einen *Zustand* C_i
  - enthält zugestellte Nachrichten aus der letzten Runde
  - eine Teilmenge der Zustände sind *Endzustände*
  - aus Endzuständen können nur Endzustände erreicht werden
  - eine Ausführung ist beendet, wenn alle Prozessoren in Endzuständen sind
- Oft: Prozessoren in Endzuständen sind auch inaktiv

Korrektheit eines verteilten Algorithmus

Definition

Ein verteilter Algorithmus ist *korrekt*, wenn seine Ausführung immer nach endlich vielen Schritten endet und danach eine korrekte Lösung des Problems vorliegt.

- Eingaben liegen (wenn überhaupt) verteilt vor
- Ausgabe/Lösung besteht aus Zustand der Knoten bei Terminierung
- Bis auf weiteres: Nachrichten sind kurz, z. B. $O(\log n)$ Bits

Beispiel: Broadcast, Flooding

Gegeben Symmetrischer, zusammenhängender Kommunikationsgraph $G = (V, E)$, Knoten $r \in V$.

Problem Sende eine Nachricht M von r an alle Knoten.

Flooding in Knoten p_i

wenn $p_i = r$ **dann**
  sende M an alle Nachbarn, nimm Endzustand an
sonst
  warte, bis Nachricht M empfangen wird
  sende M an alle Nachbarn, nimm Endzustand ein

Korrektheit Flooding

Lemma

Der Flooding-Algorithmus ist korrekt, d. h. nach endlich vielen Schritten ist die Ausführung beendet und alle Knoten haben M erhalten.

Beweis:

- Ein Prozessor ist in Endzustand $\Leftrightarrow$ er hat M erhalten
- Annahme: Es gibt einen Prozessor p_i der nicht terminiert.
  - keiner seiner Nachbarn terminiert
  - kein Knoten, von dem es einen Weg zu p_i gibt, terminiert
  - Widerspruch: r terminiert sicher und G ist zusammenhängend!

Komplexitätsmaße

- Laufzeit (Zeitkomplexität)
  - Maximale Anzahl der Runden bis alle Knoten Endzustand erreicht haben
- Nachrichtenkomplexität
  - Maximale Anzahl der Nachrichten, die insgesamt verschickt werden
  - Maximale Anzahl der Nachrichten, die ein einzelner Knoten verschicken muss
  - Manchmal in SN: Dieselbe Nachricht an alle Nachbarn zählt nur einfach (One-Hop-Broadcast)

Komplexität Flooding

Lemma

Der Flooding-Algorithmus hat Nachrichtenkomplexität $2|E|$ und Zeitkomplexität D.

- $D := \max_{u,v \in V} d_G(u, v)$: Durchmesser des Graphen
- Beweis:
  - Jeder Knoten sendet genau einmal eine Nachricht zu allen Nachbarn: $\sum_{p_i \in V} \deg(p_i) = 2|E|$
  - Induktion: Nach k Runden sind alle Knoten in Abstand $\leq k-1$ zu r im Endzustand.
    - IA r nimmt Endzustand in Runde 1 ein.
    - IS Jeder Knoten u mit Abstand k hat einen Nachbarn v mit Abstand $k-1$
      $\stackrel{\text{IV}}{\Rightarrow}$ v nimmt Endzustand spätestens in Runde k ein (und sendet M an u)
      $\Rightarrow$ u terminiert spätestens in Runde $k+1$

Anonymität, Uniformität

Definition (Uniformität)

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

Definition (Anonymität)

Ein verteilter Algorithmus heißt *anonym*, wenn die Prozessoren keine unterschiedlichen IDs besitzen.

Beispiel Leader Election

Gegeben Symmetrischer, zusammenhängender Kommunikationsgraph G

Problem Genau ein Prozessor soll als *Leader* ausgezeichnet werden

Satz

Es gibt noch nichtmal einen anonymen Algorithmus für das Leader-Election-Problem in Ringen und damit auch keinen für beliebige Graphen.

Satz

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

Beweis:

- ≫ Vereinfachung: Jeder unterscheidet „linken“ und „rechten“ Nachbarn gleich
- ≫ Induktion: Knoten beginnen jede Runde im selben Zustand
  - IA Runde 0: Keine IDs, keine Eingabe, Knoten beginnen im selben Zustand
  - IS Runde $k \curvearrowright k+1$:
    - ≫ Alle Knoten beginnen Runde k im selben Zustand
    - ⇒ Alle Knoten führen dieselbe Berechnung aus
    - ⇒ Alle Knoten enden im selben Zustand
    - ⇒ Alle Knoten senden dieselbe Nachricht nach links (rechts)
    - ⇒ Alle Knoten beginnen Runde $k+1$ im selben Zustand
- ≫ Kein Knoten kann alleine in Leader-Zustand enden.

Satz

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

Beweis:

- ≫ Vereinfachung: Jeder unterscheidet „linken“ und „rechten“ Nachbarn gleich
- ≫ Induktion: Knoten beginnen jede Runde im selben Zustand
  - IA Runde 0: Keine IDs, keine Eingabe, Knoten beginnen im selben Zustand
  - IS Runde $k \curvearrowright k+1$:
    - ≫ Alle Knoten beginnen Runde k im selben Zustand
    - ⇒ Alle Knoten führen dieselbe Berechnung aus
    - ⇒ Alle Knoten enden im selben Zustand
    - ⇒ Alle Knoten senden dieselbe Nachricht nach links (rechts)
    - ⇒ Alle Knoten beginnen Runde $k+1$ im selben Zustand
- ≫ Kein Knoten kann alleine in Leader-Zustand enden.

Leader Election in Ringen mit IDs

Leader Election, Knoten p_i kennt $\texttt{ID}_i$

sende $\texttt{ID}_i$ an linken Nachbarn
wenn *eine ID empfangen wird* **dann**
  wenn $ID > ID_i$ **dann**
    sende ID an linken Nachbarn
  wenn $ID = ID_i$ **dann**
    sende Nachricht „terminiere“ an linken Nachbarn
    terminiere als Leader
wenn *Nachricht „terminiere“ empfangen wird* **dann**
  sende Nachricht „terminiere“ an linken Nachbarn
  terminiere als Non-Leader

- ≫ Nachrichtenkomplexität: $O(n^2)$
- ≫ bessere Algorithmen: $O(n \log n)$

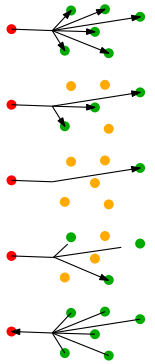
Knobelaufgabe

Wie könnte ein Leader-Election-Algorithmus für allgemeine Graphen aussehen?

Routing

Routing is the act of moving information across a network from a source to a destination

- Broadcast
  - ein Knoten sendet (gleiche) Nachricht an alle
- Multicast
  - ein Knoten sendet (gleiche) Nachricht an viele andere
- Unicast
  - ein Knoten sendet Nachricht an einen anderen Knoten
- Anycast
  - ein Knoten sendet Nachricht an irgendeinen Knoten aus einer Zielmenge
- Convergecast
  - Alle (viele) Knoten senden Nachrichten an eine Senke

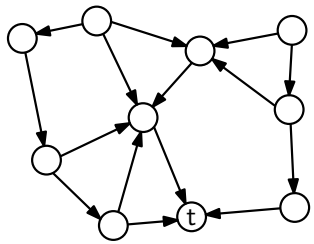


Klassische Routingprotokolle

- proaktiv ↔ reaktiv
  - proaktiv: Routinginformationen werden im Voraus verteilt
    - \+ bei wenig Mobilität einmaliger Aufwand
    - \- in der Regel höherer Speicherbedarf
  - reaktiv: Routinginformationen werden bei Bedarf ermittelt
    - \+ bei viel Mobilität kein permanenter Overhead
    - \- Routen müssen bei Bedarf erst gefunden werden
- Beispiel Reaktiv: Flooding
  - meistens ungeeignet: alle bekommen Nachricht (mehrfach)
  - manchmal die einzige Lösung (hohe Mobilität)
- Beispiel Proaktiv: Link-State-Routing
  - Knoten tauschen regelmäßig gesamten Graphen aus
  - Pakete werden immer auf kürzeste Wege geschickt

Link Reversal Routing

- Einfaches proaktives Protokoll für Convergecasts (eine Senke)
- Richte Kanten so, dass *jeder* gerichtete Pfad zur Senke führt
  - ⇒ Schicke Pakete einfach über eine beliebige ausgehende Kante!

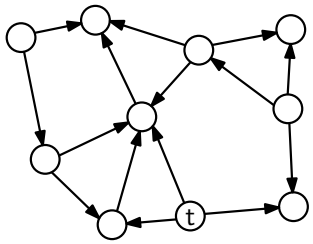


- Wie orientiert man Kanten so?

Link Reversal Routing: DAGs

Definition

Ein gerichteter Graph ohne Zykel heißt *DAG* (directed acyclic graph). Er heißt t-zielorientiert, wenn t der einzige Knoten ohne ausgehende Kante ist.



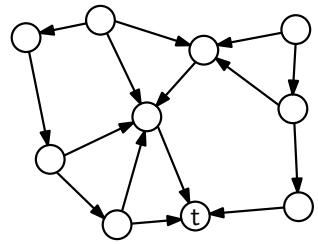
- DAG: Pakete können nicht im Kreis wandern
  - landen irgendwann bei Knoten ohne ausg. Kante
- t-orientiert: Pakete landen bei t!
- Das wollen wir (für gegebenes t)!

Organisatorisches Einführung Vorlesungsübersicht Erste Schritte

Link Reversal Routing: DAGs

Definition

Ein gerichteter Graph ohne Zykel heißt *DAG* (directed acyclic graph). Er heißt t-zielorientiert, wenn t der einzige Knoten ohne ausgehende Kante ist.



- DAG: Pakete können nicht im Kreis wandern
  - landen irgendwann bei Knoten ohne ausg. Kante
- t-orientiert: Pakete landen bei t!
- Das wollen wir (für gegebenes t)!

Problemstellung

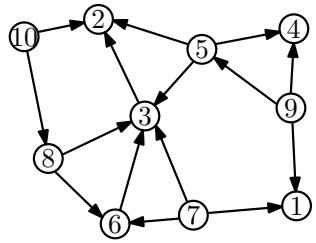
Gegeben Ungerichteter Kommunikationsgraph $G = (V, E)$, *Senke* $t \in V$

Gesucht Orientierung der Kanten, so dass der entstehende gerichtete Graph G' ein t-zielorientierter DAG ist

Initialer DAG

Richte alle Kanten von höherer ID zur niedrigeren

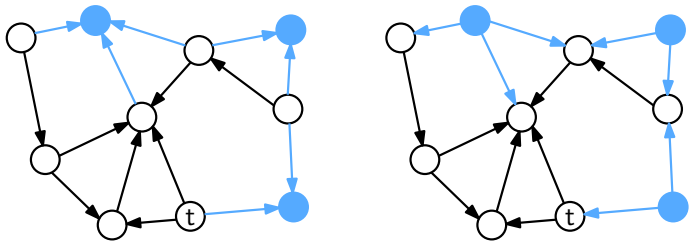
- jeder Knoten sendet seine ID an alle Nachbarn
- jeder Knoten merkt sich Richtungen inzidenter Kanten
- das ist ein DAG: Auf jedem Pfad nehmen IDs ab!



Full Link Reversal

Falls Knoten $u \neq t$ keine ausgehende Kante hat, drehe alle inzidenten Kanten um!

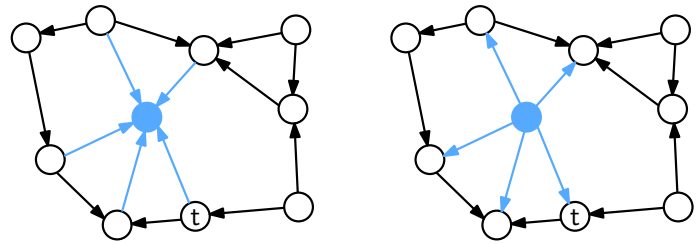
- Knoten ohne ausgehende Kanten schicken „reverse“-Nachricht an alle Nachbarn
- das bleibt ein DAG: Gedrehte Kanten liegen auf keinem Zykel!
- Irgendwann ist t die einzige Senke!? Terminiert das immer?



Full Link Reversal

Falls Knoten $u \neq t$ keine ausgehende Kante hat, drehe alle inzidenten Kanten um!

- Knoten ohne ausgehende Kanten schicken „reverse“-Nachricht an alle Nachbarn
- das bleibt ein DAG: Gedrehte Kanten liegen auf keinem Zykel!
- Irgendwann ist t die einzige Senke!? Terminiert das immer?

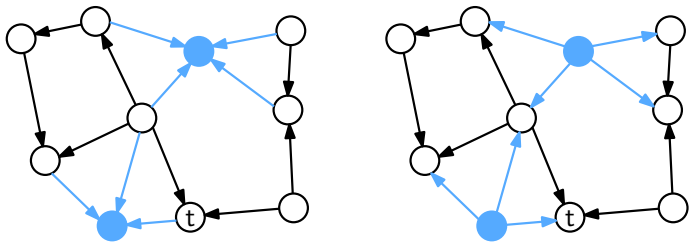


Bastian Katz – Algorithmen für Ad-hoc- und Sensornetze
Lehrstuhl für Algorithmik I
Institut für theoretische Informatik
Universität Karlsruhe (TH)
Karlsruher Institut für Technologie
35/ 44
Organisatorisches Einführung Vorlesungsübersicht Erste Schritte

Full Link Reversal

Falls Knoten $u \neq t$ keine ausgehende Kante hat, drehe alle inzidenten Kanten um!

- Knoten ohne ausgehende Kanten schicken „reverse“-Nachricht an alle Nachbarn
- das bleibt ein DAG: Gedrehte Kanten liegen auf keinem Zykel!
- Irgendwann ist t die einzige Senke!? Terminiert das immer?

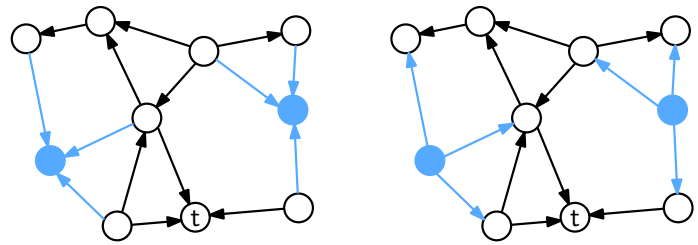


Bastian Katz – Algorithmen für Ad-hoc- und Sensornetze
Lehrstuhl für Algorithmik I
Institut für theoretische Informatik
Universität Karlsruhe (TH)
Karlsruher Institut für Technologie
35/ 44

Full Link Reversal

Falls Knoten $u \neq t$ keine ausgehende Kante hat, drehe alle inzidenten Kanten um!

- Knoten ohne ausgehende Kanten schicken „reverse“-Nachricht an alle Nachbarn
- das bleibt ein DAG: Gedrehte Kanten liegen auf keinem Zykel!
- Irgendwann ist t die einzige Senke!? Terminiert das immer?

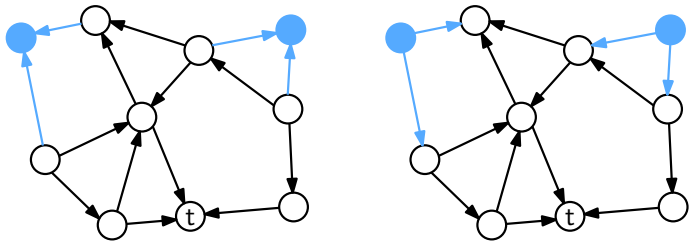


Bastian Katz – Algorithmen für Ad-hoc- und Sensornetze
Lehrstuhl für Algorithmik I
Institut für theoretische Informatik
Universität Karlsruhe (TH)
Karlsruher Institut für Technologie
35/ 44

Full Link Reversal

Falls Knoten $u \neq t$ keine ausgehende Kante hat, drehe alle inzidenten Kanten um!

- ≫ Knoten ohne ausgehende Kanten schicken „reverse“-Nachricht an alle Nachbarn
- ≫ das bleibt ein DAG: Gedrehte Kanten liegen auf keinem Zykel!
- ≫ Irgendwann ist t die einzige Senke!? Terminiert das immer?

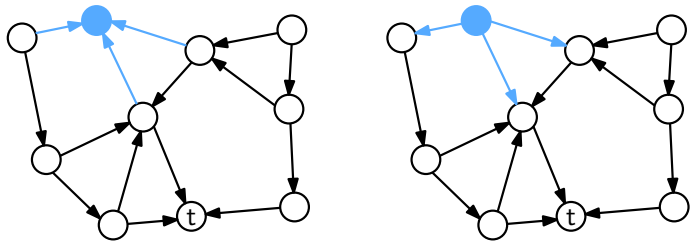


Organisatorisches Einführung Vorlesungsübersicht Erste Schritte

Full Link Reversal

Falls Knoten $u \neq t$ keine ausgehende Kante hat, drehe alle inzidenten Kanten um!

- ≫ Knoten ohne ausgehende Kanten schicken „reverse“-Nachricht an alle Nachbarn
- ≫ das bleibt ein DAG: Gedrehte Kanten liegen auf keinem Zykel!
- ≫ Irgendwann ist t die einzige Senke!? Terminiert das immer?



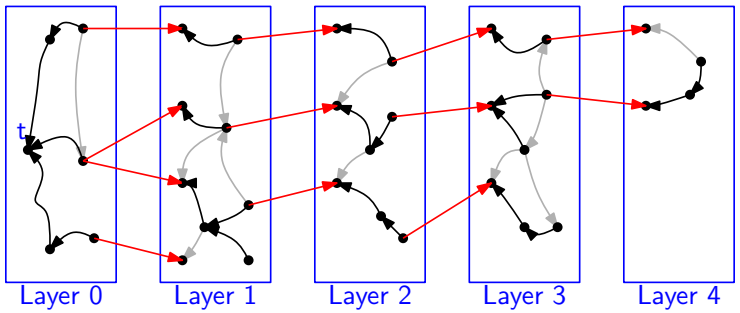
Bastian Katz – Algorithmen für Ad-hoc- und Sensornetze
Lehrstuhl für Algorithmik I
Institut für theoretische Informatik
Universität Karlsruhe (TH)
Karlsruher Institut für Technologie
35/ 44
Organisatorisches Einführung Vorlesungsübersicht Erste Schritte

Korrektheit: Layer

Initial- und Zwischenlösungen induzieren *Layer*:

Definition

Ein Knoten u liegt im kleinsten Layer k, so dass es einen (ungerichteten) Pfad von u zu t gibt, der nur k Kanten entgegen der Orientierung nutzt.



Bastian Katz – Algorithmen für Ad-hoc- und Sensornetze
Lehrstuhl für Algorithmik I
Institut für theoretische Informatik
Universität Karlsruhe (TH)
Karlsruher Institut für Technologie
36/ 44

Korrektheit Full Link Reversal 1

Betrachte parallele Änderungen in beliebiger Reihenfolge:

Lemma

Wenn ein Knoten u seine Kanten umorientiert, verringert sich u's Layer um 1, alle anderen Knoten bleiben in ihrem Layer.

Beweis (Teil 1)

- ≫ Sei u aus Layer k.
  - ⇒ es gibt einen Pfad von u nach t, der k Kanten entgegen der (alten) Orientierung nutzt. Darunter ist genau eine zu u inzidente Kante.
  - ⇒ dieser Pfad nutzt $k-1$ Kanten entgegen der neuen Orientierung
  - ⇒ nach der Umorientierung ist u in Layer $k-1$

Bastian Katz – Algorithmen für Ad-hoc- und Sensornetze
Lehrstuhl für Algorithmik I
Institut für theoretische Informatik
Universität Karlsruhe (TH)
Karlsruher Institut für Technologie
37/ 44

Korrektheit Full Link Reversal 2

Betrachte parallele Änderungen in beliebiger Reihenfolge:

Lemma

Wenn ein Knoten u seine Kanten umorientiert, verringert sich u's Layer um 1, alle anderen Knoten bleiben in ihrem Layer.

Beweis (Teil 2)

- Sei $v \neq u$,
  - Sei P ein ungerichteter Pfad von v zu t
  - Anzahl der Kanten, die P entgegen der Orientierung nutzt, ändert sich durch das Umorientieren nicht:
    - Fall 1: P enthält u nicht (trivial)
    - Fall 2: P enthält u, dann ändern sich zwei Kanten unterschiedlicher Richtung auf P
- $\Rightarrow$ v's Layer ändert sich nicht.

Korrektheit Full Link Reversal 2

Betrachte parallele Änderungen in beliebiger Reihenfolge:

Lemma

Wenn ein Knoten u seine Kanten umorientiert, verringert sich u's Layer um 1, alle anderen Knoten bleiben in ihrem Layer.

Beweis (Teil 2)

- Sei $v \neq u$,
  - Sei P ein ungerichteter Pfad von v zu t
  - Anzahl der Kanten, die P entgegen der Orientierung nutzt, ändert sich durch das Umorientieren nicht:
    - Fall 1: P enthält u nicht (trivial)
    - Fall 2: P enthält u, dann ändern sich zwei Kanten unterschiedlicher Richtung auf P
- $\Rightarrow$ v's Layer ändert sich nicht.

Korrektheit Full Link Reversal 3

Satz

Full Link Reversal führt $O(n^2)$ Umorientierungen durch und terminiert nach $O(n^2)$ Schritten in einem t-zielorientierten DAG.

Beweis:

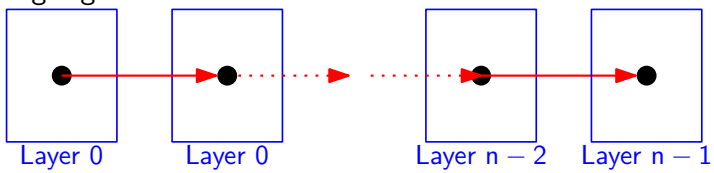
- in jedem Schritt ist der Graph ein DAG
- kein Knoten kann in einem Layer $> n$ starten
- in jedem Schritt verringert mindestens ein Knoten sein Layer
- nach spätestens n^2 Schritten sind alle Knoten in Layer 0
- wenn alle Knoten in Layer 0 sind, ist der DAG t-zielorientiert

Worst-Case-Netzwerke 1

Satz

Es gibt Initiallösungen auf Graphen bei denen der Full-Reversal-Algorithmus insgesamt $\Theta(n^2)$ Umorientierungen ausführt.

- sogar ganz einfache: falsch orientierte Pfade



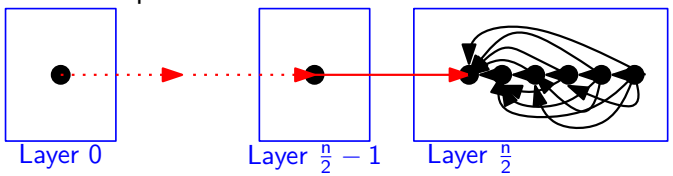
- $\sum_{i=1}^{n}(i-1) \in \Theta(n^2)$
- $\Rightarrow$ Schranke für die Zahl der Umorientierungen scharf!
- Und die Laufzeitschranke?
  - In diesem Beispiel nur $\Theta(n)$ Schritte

Worst-Case-Netzwerke 2

Satz

Es gibt Initiallösungen auf Graphen bei denen der Full-Reversal-Algorithmus insgesamt $\Theta(n^2)$ Schritte benötigt.

- etwas komplexere:



- vollständiger DAG auf $n/2$ Knoten in Layer $n/2$
- jeder davon braucht $n/2$ Umorientierungen
- immer nur einer kann zur Zeit Senke sein
- ⇒ Laufzeit $\Omega(n^2)$, Laufzeitschranke $O(n^2)$ damit auch scharf!

Full Link Reversal in dynamischen Netzen

- Hervorragend geeignet, um auf Änderungen in der Topologie zu reagieren
- Neue Knoten, neue Kanten einfach!? Siehe Update!
- Ausfälle, die Senken erzeugen werden mit FLR gelöst
  - Im schlimmsten Fall sehr aufwendig!

Was mitnehmen?

- Vorlesungsübersicht
  - Jeden Tag ein Thema, algorithmische Perlen
- Sensor- und Ad-hoc-Netze
  - Knoten, Anwendungen, drahtlose Kommunikation
- Verteilte Systeme
  - Algorithmus, Endzustände, Anonymität/IDs
  - Broadcast, Leader Election in Ringen
- Erster Sensornetzwerkalgorithmus: Full Link Reversal
  - Einfaches Idee, nichttriviale Analyse
  - Leicht umzusetzen, aber im worst-case *sehr* schlecht

Literatur

1. H. Attiya, J. Welch: *Distributed Computing*. Wiley, 2004.
2. C. Busch, S. Surapaneni, S. Tirthapura: *Analysis of link reversal routing algorithms for mobile ad hoc networks*. In SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures, 2003, ACM