# Algorithmen für Ad-hoc- und Sensornetze

## VL 05 – Lokalisierung und virtuelle Koordinaten

Dr. rer. nat. Bastian Katz

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

20. Mai 2009
(Version 2 vom 31. Mai 2009)



## Warum Lokalisierung?

**Warum wir Knotenpositionen brauchen**

- Georouting
- Topologiekontrolle
- Zuordnung der Daten (was tun, wenn's brennt?)
- ..und es kommt noch mehr..

**Was tun wir, wenn (alle/einige/viele) Knoten ihre Positionen gar nicht kennen, weil**

- GPS/Galileo nicht zur Verfügung steht
  - kostet Energie (das kostet jede andere Lokalisierung auch)
  - ist schwer, groß, teuer (verglichen mit SmartDust)
  - funktioniert nicht überall (drinnen, im Wald)
  - ist nicht besonders genau
- jedem Knoten die Position einprogrammieren wohl nicht geht?





## Heute

- Lokalisierung mit Ankerknoten
  - eine algorithmische Notiz zu einem „fremden" Terrain
- Lokalisierung ohne Ankerknoten: virtuelle Koordinaten
  - Schwere Probleme und große Lücken
- Gierige Einbettungen (wenn noch Zeit ist)
  - so *richtig* virtuelle Koordinaten





## Was man alles messen kann

- Entfernungen
  - RSSI: Received Signal Strength Indicator
    - Sendestärke bekannter $\Rightarrow$ *theoretisch* Signalstärkeverlust $d^{-\alpha}$
    - *praktisch* ist der Informationsgehalt fragwürdig
  - ToA: Time of Arrival
    - Bestimmung von Signallaufzeit (roundtrip)
  - TDoA: Time Difference of Arrival
    - Laufzeitvergleiche zweier Signale (z. B. Radio/ultrasound)
- Richtungen
  - AoA: Angle of Arrival
    - Phasenverschiebung in Antennenarrays
    - gerichtete Antennen
  - Zusatzhardware (Laser etc.)

## Lokalisierung mit Ankerknoten

Wenn GPS/Einprogrammieren so teuer ist, reicht es vielleicht, wenn *einige Ankerknoten* ihre Position kennen?

Wenn jeder Knoten genug Anker „sieht“:

- *Trilateration* bei bekannten Enfernungen
  - Drei Anker für eindeutige Lokalisierung
- *Triangulation* bei bekannten Winkeln
  - Zwei Anker für eindeutige Lokalisierung
- Bei gestörten Werten und/oder mehr Ankern
  - Least Squares etc → Schätztheorie

Was, wenn wir nur wenige Anker haben? Einige Knoten haben dann vielleicht keine Anker in ihrer Nachbarschaft!





## Lokalisierung mit wenigen Ankerknoten

### Typisches Vorgehen

Knoten schätzen Abstände/Richtungen zu Ankern über mehrere Hops und wenden *dann* Triangulation/Trilateration an.

- Furchtbar viele Verfahren und Heuristiken
- unklare Problemstellung
  - Dichte, Verteilung von Ankerknoten
- Was kann man aus algorithmischer Sicht beitragen?



## Algorithmus HOP

### Idee

In Unit-Disk-Graphen könnte die Länge eines kürzesten Weges stark genug mit der Entfernung korrellieren!

1. Berechne Hop-Distance zu Ankerknoten
   - Broadcasts von Ankerknoten genügen
2. Wähle Position so, dass Abstände der Hop-Distance bestmöglich entsprechen
   - ...was immer das heißt...



## Kompetitivität

### Problem UDG-Lokalisierung mit Ankerknoten

**Gegeben** Unit-Disk-Graph $G = (V, E)$ mit entsprechender Einbettung $\mathbf{p} : V \to \mathbb{R}^2$, Ankerknoten $V_A \subset V$

**Gesucht** Knotenpositionen $\tilde{\mathbf{p}} : V \to \mathbb{R}^2$, mit geringen Fehlern

$$\mathrm{Err}(v) := \|\mathbf{p}(v) - \tilde{\mathbf{p}}(v)\| \ .$$

In die Berechnung dürfen keine $\mathbf{p}(v)$ für ein $v \notin V_A$ einfließen.

## Optimaler Algorithmus, Kompetitivität

**Definition: Optimaler Algorithmus**

Ein *optimaler Algorithmus* wählt zu Graphen $G$ und Ankerpositionen die Knotenpositionen, die den maximalen Fehler über alle möglichen Einbettungen minimiert.

- Das muss nicht leicht sein, ist aber ein guter Vergleich:
- Kein Algorithmus ist im worst-case besser!

**Definition: Kompetitivität**

Ein Algorithmus ALG heißt *c-kompetitiv*, wenn immer gilt:

$$\mathrm{Err}_{\mathtt{ALG}}(v) \leq c \cdot \mathrm{Err}_{\mathtt{OPT}}(v) + k$$

für alle $v \in V$ und ein $k \in \mathbb{R}$.

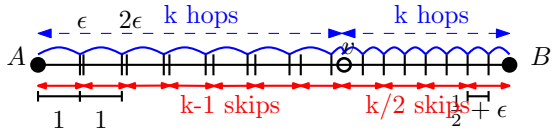




## Kompetitivität von HOP

**Satz**

HOP ist nicht kompetitiv in einer Dimension.




- für bel. $k$ setze Anker $A = 0$, $B = (k-1)(1+\epsilon) + k(\frac{1}{2}+\epsilon)$
- jeder Algo, der nur hops zählt, setzt $v$ bestenfalls in die Mitte!
  - macht ein Algo es besser, drehen wir die Instanz um!
  - Fehler in $\Theta(k)$!
- vielleicht ist das ja schon das Optimum?
- *skips* belegen Mindestabstände!
  - In diesem Fall grenzen sie die Position bis auf $O(k\epsilon)$ ein!



## Skips (formaler)

**Definition**

Ein Pfad $A = v_0, u_1, v_1, \ldots, u_s, v_s = v$ ist ein Skip-Pfad der Länge $s$ zwischen $A$ und $v$, wenn wenn

- $d(A, v_i) < d(A, v_{i+1})$
- $d(v_i, v_{i+1}) > 1$ (also $(v_i, v_{i+1}) \notin E$)

Die Länge eines längsten solchen Pfades ist die Skip-Entfernung von $v$ zu $A$.

- *hops* sind obere Schranke an Entfernung zum Anker
- *skips* sind untere Schranke

**Knobelaufgabe**

Wie berechnet man Skip-Entfernungen (in einer Dimension)?



## Algorithmus HS (Hop & Skip)

1. Berechne Graphdistanz zu allen Ankerknoten
2. Berechne Skip-Distanz zu allen Ankerknoten
3. Schneide Intervalle aus Skip- und Hop-Distanz für alle Anker
4. Wähle Punkt, der über das verbleibende Intervall den möglichen Fehler minimiert

**Satz (ohne Beweis)**

HS ist 1-kompetitiv in einer Dimension.

- Man kann zeigen, dass wirklich jede Position im Schnitt der Intervalle angenommen werden kann

In zwei Dimensionen: Fehlende Kompetitivität von HOP bleibt, aber von HS bleibt nur die Idee, auch untere Schranken für die Länge von Pfaden zu verwenden.

## Ankerfreie Lokalisierung

Viele Anwendungen funktionieren mit *plausiblen* Koordinaten so gut wie mit echten (Georouting, ...).

- Ohne Ankerknoten kann es keine echte Lokalisierung geben
- Freiheitsgrade je nach Eingabe
  - Verschiebungen (immer)
  - Drehungen (fehlende absolute Richtungsinformation)
  - Spiegelungen (fehlende Richtungsinformation)
  - Skalierungen (fehlende Entfernungsinformation)
    - schon das Wissen, es mit UDG/qUDG zu tun zu haben, trägt Entfernungsinformation!
- Vorsicht: *plausible Koordinaten können auch völlig von der Realität abweichen!*
- Trotzdem: Oft sind plausible Koordinaten besser als keine!





## UDG-Einbettung

Wenn wir davon ausgehen, dass ein Graph ein Unit-Disk-Graph ist, wie schwer ist es dann, eine Einbettung zu finden, die das belegt?

**Mindestens so schwer, wie zu entscheiden, ob ein Graph ein Unit-Disk-Graph ist!**

- Angenommen wir haben einen Algorithmus, um einen UDG einzubetten
  - wende Algo auf irgendeinen Graphen an und teste, ob Ausgabe den Graphen als UDG einbettet
  - (das sind höchstens zusätzliche $O(n^2)$)
  - (Terminierungsfragen lassen wir mal aus)



### Problem: UDG-Erkennung

Entscheide, ob ein gegebener Graph $G = (V, E)$ ein Unit-Disk-Graph ist.

### Satz

Unit-Disk-Graph-Erkennung ist NP-schwer.



## Erinnerung: Reduktion

### Beweis der NP-Schwere von $\mathcal{A}$ durch Reduktion

1. Nimm bekanntes schweres Problem $\mathcal{B}$.
2. Zeige, dass es Abbildung von Instanzen von $\mathcal{B}$ auf Instanzen von $\mathcal{A}$ gibt, die
   - in Polynomialzeit berechenbar ist,
   - Ja/Nein-Instanzen auf Ja/Nein-Instanzen abbildet

- Dann muss unser Problem auch NP-schwer sein
  - Mit einem Algo für unser Problem plus Polynomialzeit kann man ein NP-schweres Problem lösen
  - $\Rightarrow$ Dann kann man jedes NP-schwere Problem damit lösen

## Erinnerung: 3-Sat

**3Satisfiability**

Es ist NP-schwer, zu entscheiden, ob eine Formel in Konjunktiver Normalform erfüllbar ist, auch unter der Einschränkung, dass

- jede Klausel nur drei Literale enthält
- jedes Literal in maximal drei Klauseln erscheint

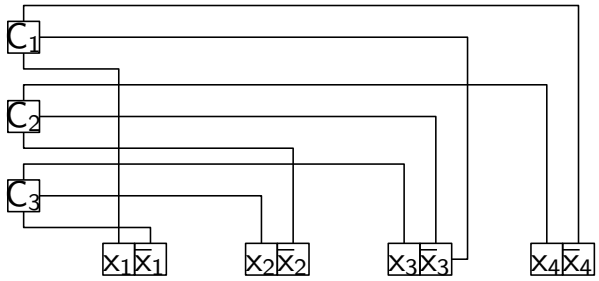
- Bsp: $(\overline{x}_1 \vee x_2 \vee x_3) \wedge (\overline{x}_2 \vee \overline{x}_3 \vee x_4) \wedge (x_1 \vee \overline{x}_3 \vee \overline{x}_4)$

Also los: wie bilden wir eine solche Formel auf einen Graphen ab, der genau dann ein UDG ist, wenn die Formel erfüllbar ist?



## Schritt 1: Orientierbarer Graph

$(\overline{x}_1 \vee x_2 \vee x_3) \wedge (\overline{x}_2 \vee \overline{x}_3 \vee x_4) \wedge (x_1 \vee \overline{x}_3 \vee \overline{x}_4)$

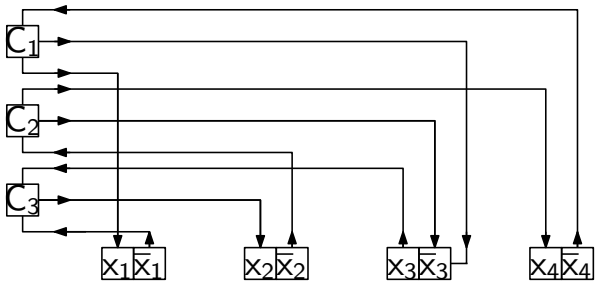



- Formel ist erfüllbar $\Leftrightarrow$ es gibt Kantenrichtungen mit
  - für jede Variable $x$ gilt: $\text{indeg}(x) = 0$ oder $\text{indeg}(\overline{x}) = 0$
  - für jede Klausel $C$ gilt $\text{outdeg}(x) > 0$



## Schritt 1: Orientierbarer Graph

$(\overline{x}_1 \vee x_2 \vee x_3) \wedge (\overline{x}_2 \vee \overline{x}_3 \vee x_4) \wedge (x_1 \vee \overline{x}_3 \vee \overline{x}_4)$

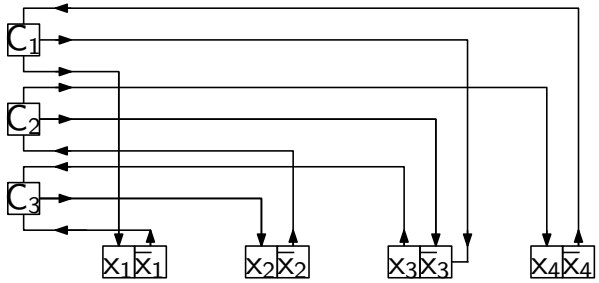



- Formel ist erfüllbar $\Leftrightarrow$ es gibt Kantenrichtungen mit
  - für jede Variable $x$ gilt: $\text{indeg}(x) = 0$ oder $\text{indeg}(\overline{x}) = 0$
  - für jede Klausel $C$ gilt $\text{outdeg}(x) > 0$



## Schritt 2: Gadgets

$(\overline{x}_1 \vee x_2 \vee x_3) \wedge (\overline{x}_2 \vee \overline{x}_3 \vee x_4) \wedge (x_1 \vee \overline{x}_3 \vee \overline{x}_4)$

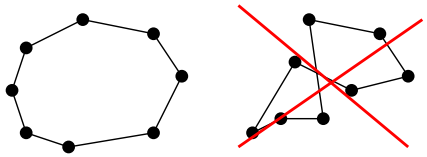



- Klassischer Gadgetbeweis: Baue Struktur nach aus
  - Variablen, Drähten, Klauseln und Kreuzungen
  - So, dass gültige Einbettungen genau gültigen Orientierungen entsprechen

## Einbettungen von Kreisen

**Lemma**

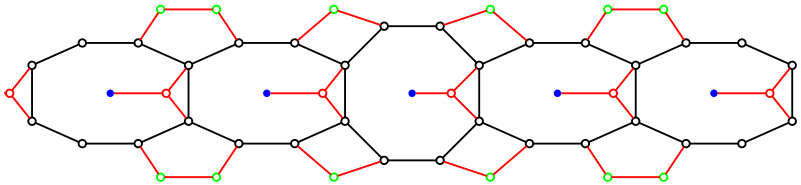
Enthält ein Graph einen knoteninduzierten Kreis, muss der in jeder Einbettung als Unit-Disk-Graph planar eingebettet werden.



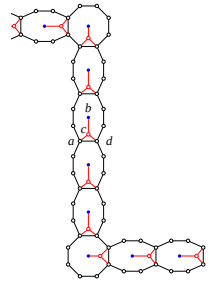
- knoteninduzierter Kreis: Menge von Knoten, die als Kreis verbunden sind (keine weiteren Kanten)
- sollte uns bekannt vorkommen!



## Schritt 2.1: Drähte

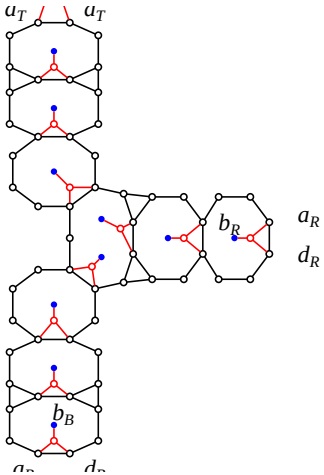


- Kreise als Käfige und Stabilistoren
  - in Käfigen nur sehr begrenzt Platz!
  - Stabilisatoren auf andern Gadgets weggelassen
- jeder Käfig kann nur einen blauen Knoten aufnehmen
  - zeigt Richtung des Drahtes an





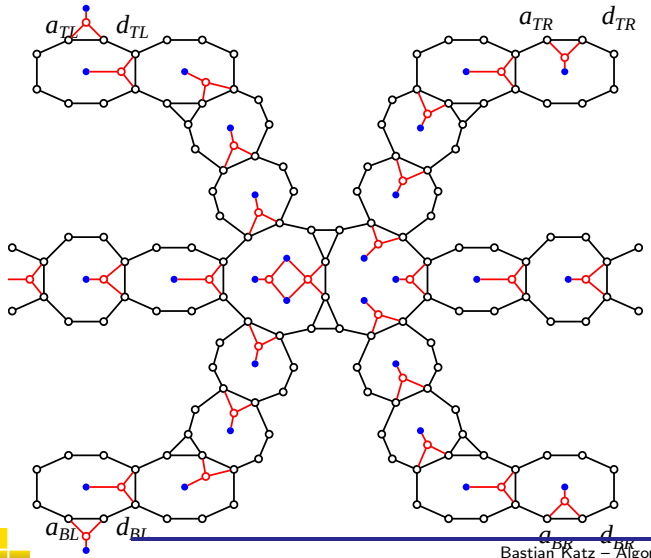

## Schritt 2.2: Klauseln



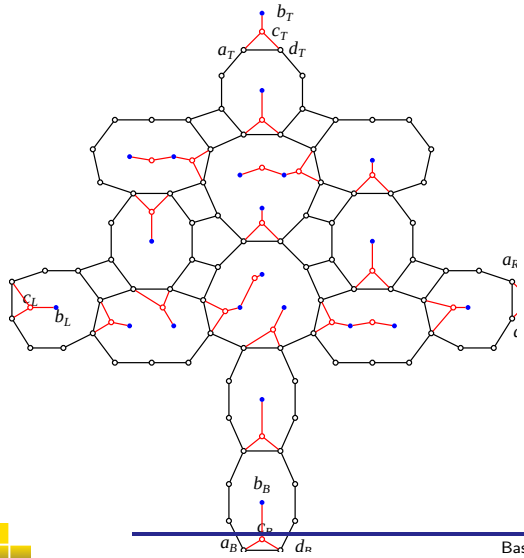

- Im wesentlichen ein Kreis der zwei blaue Knoten aufnehmen kann



## Schritt 2.3: Variablen



## Schritt 2.4: Kreuzungen






## Schwere UDG-Erkennung

**Satz**

Unit-Disk-Graph-Erkennung ist NP-schwer.

- Zu 3Sat-Formel konstruiere Graphen $G$ wie beschrieben
- Eine erfüllende Belegung der Variablen verrät uns eine eine Einbettung als UDG
  - Knotenpositionen übernehmen wir dann aus unseren Zeichnungen der Gadgets
  - nur die Einbettung der roten Elemente passen wir an
- Eine Einbettung als UDG verrät uns eine erfüllende Belegung der Formel
  - Die exakten Knotenpositionen sind dann nicht wichtig
  - kombinatorische Einbettung der roten Elemente reicht!



## Unit-Disk-Graph-Approximation

Wenn es schwer ist, einen Graphen als UDG einzubetten, kommt man dann wenigstens „dicht“ heran? Definiere *Qualität* einer Einbettung $\mathbf{p}$ als

$$q(\mathbf{p}) := \frac{\max_{\{u,v\}\in E} \|\mathbf{p}(u) - \mathbf{p}(v)\|}{\min_{\{u,v\}\notin E} \|\mathbf{p}(u) - \mathbf{p}(v)\|}$$

- $q(\mathbf{p}) < 1$: Einbettung belegt UDG, sonst nur $1/q(\mathbf{p})$-Quasi-Unit-Disk-Graph!

Man kann den Beweis noch so „aufbohren“, dass er folgende Reduktion liefert:

- Formel erfüllbar $\Rightarrow$ Graph einbettbar mit $q < 1$
- Formel nicht erfüllbar $\Rightarrow$ Graph nicht einbettbar mit $q < \sqrt{2}$.



## Was, genau, heißt das jetzt?

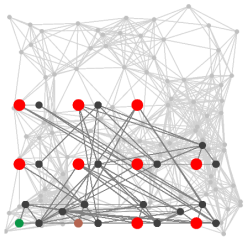
- Es ist NP-schwer, einen Unit-Disk-Graphen zu erkennen
  - also gibt es keinen Polynomialzeitalgo, der UDG entsprechend einbettet.
- Es ist auch schwer, zu erkennen, ob man einen Graphen mit Qualität $< \sqrt{2}$ einbetten kann
  - Also sind auch $1/\sqrt{2}$-Quasi-Unit-Disk-Graphen schwer zu erkennen
- Zwei Seiten derselben Medaille: UDG-Approximation und QUDG-Erkennung

## Beste bekannte Approximation

**Satz (ohne Beweis)**

Es gibt einen Algorithmus mit polynomieller Laufzeit, der mit hoher Wahrscheinlichkeit eine Einbettung mit Qualität in $O(\log^{2.5} n\sqrt{\log\log n})$ berechnet.

- Mit hoher W'keit: $p > 1 - 1/n$
- ganz grobe Skizze:
  1. LP-Lösung liefert echte Metrik auf Knoten
  2. Knoten werden geschickt in $\mathbb{R}^n$ eingebettet
  3. Einbettung wird zufällig auf den $\mathbb{R}^2$ projiziert
  4. Ergebnis wird nach festen Regeln verfeinert
- Gleich mehrere schwerste Geschütze!



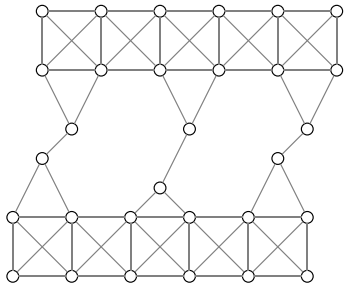
Dafür gibt's noch Schokolade: Bessere Approximation in Polynomialzeit oder bessere Schranke liefern.



## Noch ein schweres Problem

**Satz (ohne Beweis)**

Es ist NP-schwer, zu erkennen, ob man einen Graphen mit vorgegebenen Kantenlängen einbetten kann. Das gilt auch noch, wenn man sich auf UDGs einschränkt.





## Und ein ganzer Zoo von verwirrenden Ergebnissen

- Wenn man Richtungen und Entfernungen kennt, ist Lokalisierung leicht (klar)
  - aber schon beliebig kleine Fehler machen das Problem schwer
  - es macht keinen Unterschied, ob man sich auf UDGs einschränkt
- Wenn man nur Richtungen kennt, kann man in Polynomialzeit eine entsprechende Einbettung finden
  - wenn man aber nach einer entsprechenden UDG-Einbettung fragt, wird's schwer!

Das Finden plausibler Koordinaten ist schon in den meisten idealisierten Fällen schwer!

- *Heuristiken* für dichte Graphen gehen davon aus, dass plausible Lösungen „ziemlich" eindeutig sind!

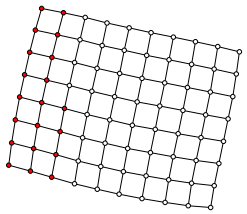


## Eine Heuristik: AFL

**Grundidee**

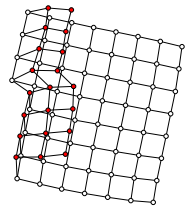
Kräftebasierte Verfahren minimieren lokalen Stress iterativ

$$\sum_{\{u,v\in E\}} (\|u,v\| - \ell_{uv})^2$$



**Problem**

Problem: Lokale Optima können Überlappungen aufweisen!



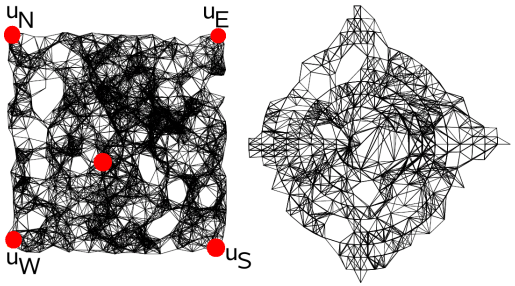
**Lösung?**

Finde überlappungsfreie Initiallösung!

## Eine Heuristik: AFL (Initiallösung)

1. Finde vier extremale und einen zentralen Knoten
   - $U_N, U_W, U_S, U_E, U_C$
2. Bestimme Hop-Entfernungen zu diesen Knoten
3. Bette Knoten an Polarkoordinaten ein

$$\rho_v = d_G(v, u_C) \quad \theta_v = \arctan \frac{d_G(v, U_N) - d_G(v, U_S)}{d_G(v, U_W) - d_G(v, U_E)}$$

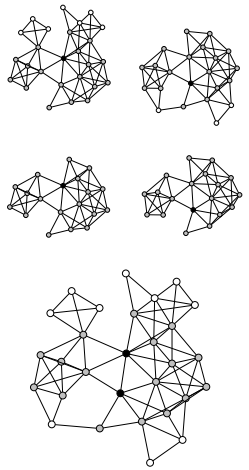



Funktioniert nicht so gut, wenn das Gebiet komplexer wird!



## Weitere heuristische Ideen

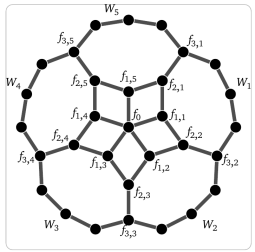
- Dichte Netze erlauben gute lokale Lösungen
  - viele lokale Lösungen zu berechnen skaliert gut!
  - setze Lösungen iterativ oder hierarchisch zusammen
- Erkennen von inneren Knoten und Randknoten
  - Ausnutzen von lokalen Strukturen





## Weitere heuristische Ideen

- Dichte Netze erlauben gute lokale Lösungen
  - viele lokale Lösungen zu berechnen skaliert gut!
  - setze Lösungen iterativ oder hierarchisch zusammen
- Erkennen von inneren Knoten und Randknoten
  - Ausnutzen von lokalen Strukturen






## Zusammenfassung

- Ankerbasierte Lokalisierung
  - fast ausschließlich Schätztheorie oder Heuristiken
  - kaum algorithmische Analyse (wegen unklarer Parameter?)
  - Ausnahme: Hop/HS zu eindimensionaler Lokalisierung
- Ankerfreie Lokalisierung
  - klarere algorithmische Probleme
  - vor allem Negativergebnisse
  - Heuristiken für dichte Graphen, aber ohne Garantien

# Literatur


1. R. O'Dell, R. Wattenhofer: *Theoretical aspects of connectivity-based multi-hop positioning*. In: Theoretical Computer Science 344:1, pp. 47-68, 2005
2. H. Breu, D.G. Kirkpatrick: *Unit Disk Graph Recognition is NP-Hard*. In: Computational Geometry. Theory and Applications 9, 1993
3. F. Kuhn, T. Moscibroda, R. Wattenhofer: *Unit Disk Graph Approximation*. In: ACM Joint Workshop on Foundations of Mobile Computing (DIALM-POMC), 2004