Analyse und experimentelle Auswertung verschiedener Routing-Protokolle mit virtuellen Koordinaten auf Sensornetzen

Inaugural-Dissertation

zur Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen Fakultät
der Heinrich-Heine-Universität Düsseldorf

vorgelegt von

Daniel Gaußmann

aus Düsseldorf

Düsseldorf, April 2019

aus dem Institut für Informatik
der Heinrich-Heine-Universität Düsseldorf

Gedruckt mit der Genehmigung der
Mathematisch-Naturwissenschaftlichen Fakultät der
Heinrich-Heine-Universität Düsseldorf

Berichterstatter:

1. Prof. Dr. Egon Wanke
2. Jun.-Prof. Dr. Kalman Graffi

Tag der mündlichen Prüfung: 14. Juni 2019

Erklärung

Ich versichere an Eides Statt, dass die Dissertation von mir selbständig und ohne unzulässige fremde Hilfe unter Beachtung der „Grundsätze zur Sicherung guter wissenschaftlicher Praxis an der Heinrich-Heine-Universität Düsseldorf“ erstellt worden ist.

Des Weiteren erkläre ich, dass ich eine Dissertation in der vorliegenden oder in ähnlicher Form noch bei keiner anderen Institution eingereicht habe.

Düsseldorf, April 2019 Daniel Gaußmann

Zusammenfassung

In den letzten Jahren erhielten sogenannte *Sensornetze* eine erhöhte Aufmerksamkeit, auch und vor allem wegen der Steigerung der Leistungsfähigkeit von kleinen, kostengünstigen Prozessoren und Sensormodulen, wodurch mehr und mehr Einsatzzwecke realisierbar und finanzierbar werden.

Zwei große Problemstellungen in diesem Gebiet sind das Routen durch die Netzwerke und das Erkennen von Löchern und Rändern im Netzwerk. Zu beiden Fragestellungen wird in dieser Arbeit ein Beitrag geleistet.

Für das Routen bestimmen wir zunächst mittels *Multidimensionaler Skalierung (MDS)* aus der Struktur der Netzwerke heraus virtuelle Koordinaten. Diese benutzen wir, um die Zuverlässigkeit von Greedy-Routing deutlich zu erhöhen. Dabei verzichten wir ausdrücklich auf strenge formale Bedingungen an die Netzwerke (z.B. die Unit-Disk-Eigenschaft). In der experimentellen Auswertung ermitteln wir, wie sich dieses Verfahren und einige Modifikationen davon unter verschiedenen Bedingungen verhalten.

Auch im Kapitel zur Erkennung von Löchern wird auf starke formale Anforderungen an die Netzwerke verzichtet. Dem hier vorgestellten Verfahren liegt eine einfache, aber sehr effektive Idee zugrunde, die in vielen Fällen auch dann sehr gute Ergebnisse liefert, wenn andere Verfahren an ihre Grenzen kommen - insbesondere bei wenig dichten Netzwerken. Die Ergebnisse werden dabei besser, wenn die Netzwerke eher die üblichen formalen Anforderungen erfüllen und dadurch die Resultate sich besser mit der intuitiven Anschauung in Einklang bringen lassen.

Im letzten Kapitel schließlich wird ein neues Verfahren vorgestellt, das die Zustellung einer Nachricht garantiert, was mit Greedy-Methoden trotz aller Modifikationen in der Ebene nicht möglich ist. Das Routen über *Hierarchische Bipartition (HBR)* kann dabei als alleiniges Routing-Protokoll verwendet werden, aber auch als Ausweichprotokoll, wenn Greedy-Methoden fehlschlagen. Das Verfahren benötigt nur wenig Speicher für die Routingtabellen und liefert im Mittel Wege, die nur wenig länger sind als die optimalen Wege.

Als roter Faden durch die drei Kapitel zieht sich die Erkenntnis, dass die Erkennung der Eckpunkte eines Netzwerkes eine entscheidende Rolle spielt. Vor allem bei MDS können damit die Ergebnisse stark verbessert werden - auch unter erschwerten Bedingungen wie einer sehr lückenhaften Kenntnis über die Struktur des Netzwerkes.

Abstract

In recent years, so-called sensor networks have received increased attention, particularly because of the increase in the performance of small, low-cost processors and sensor modules, which makes more and more applications feasible and affordable.

Two major problems in this field are routing through the networks and the detection of holes and boundaries in the network. In this thesis, a contribution is made to both of these questions.

For routing, we first determine virtual coordinates from the structure of the networks using multidimensional scaling (MDS). We use these to significantly increase the reliability of greedy routing. We explicitly renounce strict formal conditions of the networks (e.g. the unit disk property). In the experimental evaluation we determine how this approach and some of its modifications behave under different conditions.

The chapter on hole detection also avoids strong formal requirements for the networks. The method presented here is based on a simple but very effective idea, which in many cases provides very good results even when other methods reach their limits - especially on less dense networks. The results will improve if the networks fulfil the usual formal requirements and thereby the results can be better reconciled with the intuitive perspective.

Finally, in the last chapter, a new procedure is presented that guarantees the delivery of a message, which is not possible with greedy methods in the plane despite all modifications. Routing via *Hierarchical Bipartition (HBR)* can be used as a stand-alone routing protocol, but also as a fallback protocol if greedy methods fail. The procedure requires only little memory for the routing tables and provides on the average routing paths that are only slightly longer than the optimal paths.

The common theme throughout the three chapters is the realization that the recognition of the corner points of a network plays a decisive role. Especially with MDS, the results can be greatly improved - even under difficult conditions such as a very incomplete knowledge of the structure of the network.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Als Einstieg in die Themen dieser Arbeit wollen wir kurz das Umfeld betrachten und einen einen kleinen Einblick in den Bereich „Sensornetze“ geben. Was ist das überhaupt, wozu kann man sie benutzen, und wo liegen einige Herausforderungen in diesem Gebiet? Im Anschluss daran folgen die obligatorischen Definitionen, die im Verlauf der gesamten Arbeit gebraucht werden. Neben den absolut grundlegenden Dingen zur Klärung der Notation (*Graphen*, *Wege*) auch einige etwas weitergehende wie *eingebetteter Graph* und *Unit-Disk-Graph*. Das Kapitel schließt dann mit der Erläuterung einiger einfacher Routing-Protokolle und deren Einschränkungen, von denen schließlich einige in dieser Arbeit teilweise behoben werden.

1.1 Sensornetze

Ein *Sensornetz* oder *Sensornetzwerk* besteht aus vielen einzelnen *Sensoren*, auch *Sensorknoten* genannt, die untereinander über eine meist drahtlose Verbindung kommunizieren können. Ein einzelner Sensor besteht dabei in der Regel aus einer Sensorkomponente, einer Datenverarbeitungseinheit, einer Energieversorgung und einer Sende- und Empfangseinheit. Die Sensorkomponente kann Daten über die Umgebung messen (z.B. Temperatur, Luftdruck, Luftfeuchtigkeit, Helligkeit, Bewegung). In der Datenverarbeitungseinheit können die gemessenen Daten verarbeitet und gespeichert werden und schließlich über die Sendeeinheit zu anderen Sensorknoten und/oder auf diesem Weg zu einer zentralen Sammelstelle geschickt werden. Die dafür notwendige Energie wird über eine Batterie bereitgestellt oder durch Module zur Energiegewinnung (z.B. Solarzellen) selbst erzeugt.

Die einzelnen Sensorknoten sind oft sehr klein, besitzen nur eine beschränkte Rechen- und Speicherkapazität und eine knappe Energieversorgung. Dabei ist „sehr klein“ durchaus eine relative Größenangabe und meint oft die Größe einer Münze oder einer Streichholzschachtel. Im Prinzip können auch Smartphones als solche Sensorknoten betrachtet werden - Smartphones können miteinander kommunizieren, sie besitzen vielfältige Sensoren (z.B. Kamera, Mikrophon, Beschleunigungssensoren, Kompass, GPS), und sie können Daten verarbeiten.

In der englischen Literatur findet man häufig die Bezeichnungen WSN (*Wireless Sensor Network*) und MANET (*Mobile Ad Hoc Network*). In einigen Arbeiten werden diese Begriffe mehr oder weniger synonym verwendet, andere legen Wert auf eine Unterscheidung [GMG07]. Dabei werden MANETs oft als drahtlose, lokale Netzwerke betrachtet, die „mit Menschen zu tun haben“ und bezeichnen eher die Netzwerke aus Smartphones, PDAs, Laptops und anderen mobilen Geräten, die wir heutzutage mit uns herum tragen. Die Anzahl der Geräte in solchen Netzwerken (z.B. innerhalb eines Warenlagers oder eines Krankenhauses) ist oft recht übersichtlich, die Leistungsfähigkeit der einzelnen Geräte relativ hoch, und die Kommunikation untereinander läuft oft direkt über eine feste Basistation. Auch das Thema Energieversorgung spielt eine untergeordnete Rolle, wenn die Geräte jeden Abend an der Steckdose aufgeladen werden können.

Die Bezeichnung WSN kommt eher bei Netzwerken zum Einsatz, deren Fokus auf der Beobachtung der Umgebung liegt. Diese Netzwerke sind unabhängiger vom Menschen und die Anzahl der beteiligten Geräte wird als deutlich höher angenommen. Dafür ist die Leistungsfähigkeit der einzelnen Geräte geringer. Die Kommunikation läuft wegen einer eingeschränkten Sendereichweite nicht direkt mit einer (möglicherweise auch gar nicht vorhandenen oder nicht benötigten) Basisstation, sondern wird von Knoten zu Knoten durch das Netzwerk weitergeleitet. Auch die Energieversorgung wird zu einem wichtigerem Gesichtspunkt, wenn die einzelnen Sensorknoten im Gelände verteilt werden und dann über Tage, Wochen oder Monate hinweg arbeiten sollen.

Im Kontext dieser Arbeit betrachten wir eher WSNs, die wir hier als Sensornetze oder Sensornetzwerke bezeichnen. Das Konzept der drahtlosen Verbindung wird immer implizit angenommen.

Die Sendereichweite der einzelnen Sensoren ist also stark eingeschränkt, so dass ein Sensorknoten nicht direkt mit allen anderen Knoten im Netzwerk kommunizieren kann. Stattdessen wird eine Nachricht von einem Startknoten s zu einem Zielknoten t über mehrere Zwischenknoten durch das Netzwerk

geleitet. Dafür müssen Routing-Protokolle implementiert werden, über die entschieden wird, zu welchem Nachbarknoten ein Sensorknoten eine gerade empfangene Nachricht weiterleiten soll, damit sie am eigentlichen Ziel ankommt.

Die Größe des Sensornetzwerkes - also die Anzahl der beteiligten Sensoren - liegt je nach Anwendung bei einigen Dutzend, ein paar Hundert, einigen Tausend oder gar weitaus mehr Sensoren. Die Netzwerke, die wir in dieser Arbeit betrachten, bestehen aus einigen Hundert bis einigen Tausend Sensorknoten.

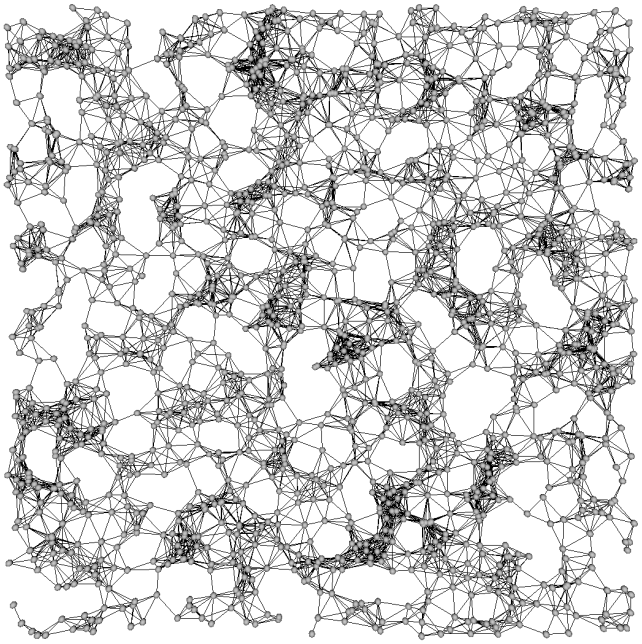

Abbildung 1.1: Eine grafische Darstellung eines Sensornetzwerks mit 1500 zufällig platzierten Sensoren in einem quadratischen Gebiet. Die Sensoren werden als Punkte bzw. kleine Kreise dargestellt, die Kommunikationsverbindungen zwischen zwei Sensoren als Linien.

1.1.1 Anwendungsgebiete für Sensornetzwerke

Auch wenn wir in dieser Arbeit von einem theoretischen Blickwinkel aus an dieses Thema herangehen, darf ein kurzer Blick auf die Praxis nicht fehlen. Die „Überwachung der Umgebung“ mit Hilfe von Sensornetzwerken ist erstmal ein sehr weit gefasster Begriff.

Wie in fast jedem Forschungsbereich gibt es natürlich auch hier eine ganze Reihe von militärischen Anwendungen - sowohl offensive als auch defensive. Die wollen wir hier aber weitestgehend ausklammern. Auch dann, wenn sich bei Sensornetzen in einer feindlichen Umgebung weitere interessante Forschungsaspekte ergeben: Wie lässt sich ein Netzwerk vor Angriffen schützen? Wie steht es um die Integrität der gewonnenen Daten, auch wenn der „Feind" Zugriff auf die Hardware hat?

Eine Attacke, die auch im Kontext dieser Arbeit eine gewisse Relevanz hat, ist die sogenannte *Wurmloch-Attacke* [HPJ02]. Wir werden eine Methode vorstellen, wie man aus der Graphstruktur eines Netzwerkes (also den vorhandenen Kanten zwischen den Knoten) auf die Position der Knoten im Gelände schließen kann. Dabei machen wir uns zu Nutzen, dass Knoten, die mit einer Kante verbunden sind, auch tatsächlich nahe beieinander liegen. Bei der Wurmloch-Attacke werden über zusätzlich aufgestellte Sende- und Empfangseinheiten zwischen einzelnen Knoten zusätzliche Verbindungen aufgebaut, die dadurch die Positionsermittlung stören und letztlich zu falschen Ergebnissen führen.

Aber glücklicherweise gibt es auch zivile Einsatzmöglichkeiten für diese recht neue Technologie.

Ein Bereich ist die Überwachung von Tierpopulationen, um mehr über das Verhalten der Tiere zu erfahren. So lassen sich zum Beispiel Bewegungsprofile einzelner Tiere durch Anbringen von GPS-Halsbändern erstellen, wodurch sich auch Rückschlüsse über das Sozialverhalten der Tiere ziehen lassen [HSBH$^+$09]. Ein anderer Ansatz sind sogenannte Kamerafallen, die ein bestimmtes Gebiet auf Tierbewegungen hin überwachen [KKJ$^+$09]. Relevant werden solche Beobachtungen zum Beispiel bei der Bekämpfung von Tierseuchen, bei Konflikten zwischen Mensch und Tier in einem Siedlungsgebiet, oder bei der Invasion einer vom Menschen eingeschleppten Tierart, die die vorhandene Fauna zu verdrängen droht.

In der modernen Landwirtschaft können Sensornetzwerke zur Steigerung der Erträge eingesetzt werden. Durch Beobachtung des Mikroklimas innerhalb eines Feldes, also beispielsweise unterschiedlich hohe Luft- und Bodenfeuchtigkeit, kann der Ausbreitung einiger Parasiten und Krankheiten entgegengewirkt werden [Bag05]. In [PE08] wird unter anderem vorgestellt, wie ein Sensornetzwerk dafür benutzt wird, um über strategisch sinnvoll aufgestellte Windmaschinen an notorischen Kältepunkten auf einer Farm dafür zu sorgen, dass die Temperaturen nicht unter den Gefrierpunkt fallen, um dadurch Frostschäden an den Pflanzen zu vermeiden.

Wenn man es etwas spektakulärer haben möchte, dann gibt es Projekte zur Unterstützung im Katastrophenfall. Sowohl bei der Unterstützung der Ersthelfer vor Ort, aber auch zur Vorhersage von Überflutungen [BRR08] oder dem rechtzeitigen Erkennen von Waldbränden [SHK06], um mit dem Löschen beginnen zu können, bevor die Brände außer Kontrolle geraten.

1.1.2 Problemstellungen bei Sensornetzwerken

Bei diesen und vielen weiteren Einsatzmöglichkeiten gibt es unterschiedliche Hürden zu bewältigen. Einige lassen sich mit bekannten Standardmitteln lösen, für andere müssen neue Wege erdacht werden. Dabei sind unterschiedliche Bereiche der Informatik an der Findung von Lösungen beteiligt - von der Entwicklung der Hardware, die den Anforderungen im Einsatz genügt; über die Implementierung von passenden Algorithmen; hin zu theoretischen Aussagen, ob beispielsweise einige der auftauchenden Probleme NP-vollständig sind, und man die exakte Lösung eher beiseite schieben sollte, um sich auf approximative Lösungen zu konzentrieren.

- Die Entwicklung der Sensorknoten selbst. Sensorknoten sollen oft klein, leicht, robust, langlebig und kostengünstig sein. Eine große Herausforderung bei der Miniaturisierung ist vor allem die Kommunikation mit anderen Sensorknoten. Bei Verwendung von Radiowellen erzwingen kleinere Antennen höhere Frequenzen, was nicht gut mit geringem Energieverbrauch vereinbar ist [KKP99]. Bei optischen Verbindungen ist eine Sichtverbindung erforderlich, was in vielen Anwendungsfällen nicht gewährleistet werden kann. Das wird unter Umständen zusätzlich erschwert durch ungünstige Umweltbedingungen, z.B. beim Einsatz von Sensoren in einem Waldgebiet (Bäume als Hindernisse für den Funkverkehr) oder unter Wasser [APM05].

  Bei Sensornetzen mit besonders vielen Sensorknoten spielt auch die Kostenfrage eine wichtige Rolle. Ziel ist zum Beispiel ein Preis von weniger als 1$ pro Sensor [Rea00], der oft schon vom Preis für die Sende- und Empfangseinheit überschritten wird.

- Entwicklung und Auswahl geeigneter Kommunikationsprotokolle. Das fließt bei der Entwicklung der Sensorknoten selbst auch schon ein, da natürlich auch geeignete Hardware vorhanden sein muss. Das ist aber auch abhängig vom geplanten Einsatzzweck und der erwarteten Datenrate. Beispiele wären WLAN, Bluetooth, ZigBee oder Ultrabreitband [sLwScS07]. So sind für eine längere Batterielebensdauer beispielsweise WLAN oder Ultrabreitband für hohe Datenraten besser geeignet,

und ZigBee und Bluetooth besser für kleine Datenraten geeignet.

Neben der physikalischen Ebene treten auch Herausforderungen an der Transportebene auf. Die verwendeten Protokolle sollten energiesparend sein, um die Lebensdauer der einzelnen Sensorknoten nicht unnötig zu reduzieren, und sie sollten auch fehlertolerant sein, um gegebenenfalls Ausfälle einiger Knoten oder fehlerhafte Übertragungen kompensieren zu können.

- Topologiekontrolle, also die Auswahl bestimmter Kommunikationsverbindungen, um die Kommunikation effizienter zu gestalten. Der Relative Nachbarschaftsgraph [Tou80], Morelia-Graph [BCG+04], XTC [WZ03], oder minimale Spannbäume [GHS83] sind Beispiele dafür. Dabei wird der Kommunikationsgraph ausgedünnt, und die Kommunikation läuft nur über bestimmte Kanten bzw. Wege, die für die Erfüllung der jeweiligen Aufgabe besonders effizient sind. So können beispielsweise in einem minimalen Spannbaum mit einer Wurzel effizient Informationen aus dem Sensornetzwerk in einem Knoten gesammelt werden.

- Datensammlung und Datenverteilung im Netzwerk. Dabei sollen sämtliche Messdaten aus dem Sensornetzwerk zu einer sogenannten *Senke* geleitet werden. Oder andersherum eine Information von einer *Quelle* aus im gesamten Netzwerk verteilt werden. Das triviale *Fluten* des gesamten Netzwerkes, bei dem jeder Knoten eine erhaltene Nachricht an alle anderen Sensorknoten in Kommunikationsreichweite sendet, ist zwar sehr einfach zu implementieren, aber offensichtlich nicht besonders effizient. Besonders dann, wenn das Netzwerk sehr dicht ist (d.h. jeder Sensorknoten ist mit vielen anderen Knoten verbunden) werden dadurch sehr, sehr viele unnötige Nachrichten verschickt, weil ein Empfänger eine Nachricht längst von einem anderen Knoten erhalten hat.

  Einen Einstieg in dieses *Broadcasting* (ein Knoten verteilt eine Information an alle Knoten) und *Gossiping* (alle Knoten verteilen ihre Information an alle Knoten im Netzwerk) liefert zum Beispiel [HHL88].

  Bei der Datensammlung interessanter Informationen an einer Senke kommt es auch vor, dass mehrere Knoten in einem Gebiet dasselbe Phänomen beobachten (z.B. einen starken Anstieg der Temperatur, was auf einen Brand hinweisen kann) und diese Information an die Basisstation weiterleiten wollen. Dann ist es sinnvoll, wenn die Daten unterwegs zusammengefasst werden, um dadurch die gesamte Nachrichtenmenge zu reduzieren [HKB99].

- Problemstellungen bei der Selbstorganisation, bei der das Sensornetzwerk selbst einige Knoten für zusätzliche Aufgaben bestimmt (z.B. *Leader Election* [MWV00]), oder Veränderungen und Fehler im Netzwerk erkennt und darauf reagiert [PH07].

  Dort hinein spielt auch die Erkennung von Rändern oder Löchern im Netzwerk. Wenn ein Sensornetzwerk ein bestimmtes Gebiet überwachen soll, dann ist es sicherlich sinnvoll, dass das gesamte Gebiet lückenlos von den verteilten Sensoren überdeckt wird. Wenn einige Knoten ausfallen, und dadurch einige Bereiche nicht mehr erfasst werden, dann ist das einerseits offensichtlich schlecht für die Erfüllung der Aufgabe, kann aber andererseits auch bereits ein Indiz dafür sein, dass das, was man über das Netzwerk erkennen möchte, in diesem Bereich gerade stattfindet - z.B. ein Waldbrand, der einige Knoten zerstört hat.

- Positionserkennung. Wenn von einem Sensornetzwerk ein bestimmtes Gebiet überwacht werden soll, dann ist es offensichtlich sinnvoll, wenn die Positionen der einzelnen Knoten bekannt sind. Jeden Sensor mit einem GPS-Modul auszustatten ist dabei eine Möglichkeit, die aber nicht immer sinnvoll realisierbar ist. Denn dieses zusätzliche Modul erhöht die Produktionskosten der Sensoren und auch den Energiebedarf. Wenn die relative Position der Knoten untereinander aber aus der Graphstruktur ermittelt werden kann, dann würden einige wenige Knoten mit GPS-Modul ausreichen, um aus diesen relativen Koordinaten absolute Koordinaten zu bestimmen. Eine Methode dafür werden wir uns in Kapitel 3 näher anschauen, auch wenn wir dort den Fokus nicht auf eine möglichst akkurate Positionierung legen, sondern eher darauf, dass wir mit den ermittelten „virtuellen“ Koordinaten möglichst gut in dem Netzwerk routen können.

Einen erweiterten Überblick in verschiedene Aspekte rund um das Thema Sensornetze bietet z.B. die Arbeit *Wireless sensor networks: A survey* [ASSC02].

1.2 Modellierung

Im Folgenden geben wir kurz die allgemein bekannten Begriffe und Definitionen an, um die Bezeichnungen zu klären, die wir im Rahmen dieser Arbeit verwenden. Wir betrachten ein Sensornetzwerk als *Graph*. Die Sensorknoten entsprechen den *Knoten* des Graphen, die Kommunikationsverbindungen zwischen den Sensorknoten entsprechen den *Kanten*.

Definition 1 (Graph, Einbettung) Ein *Graph* ist ein Paar $G = (V, E)$, wobei V eine Menge von *Knoten* und $E \subseteq \big\{\{u, v\}\big|u, v \in V\big\}$ eine Menge von *Kanten* ist.

Ein *kantengewichteter Graph* ist ein Graph $G = (V, E, \omega)$ mit einer Kantengewichtsfunktion $\omega : E \to \mathbb{R}$.

Ein *d-dimensional eingebetteter Graph* $G = (V, E, p)$ ist ein Graph mit einer *Einbettung* $p : V \to \mathbb{R}^d$, dessen Knoten im $\mathbb{R}^d$ eingebettet sind.

In dieser Arbeit betrachten wir nur kantengewichtete Graphen und keine knotengewichtete Graphen, die ganz analog definiert werden. Wir verwenden daher im Folgenden die kürzere Schreibweise *gewichtete Graphen*.

Im Kontext von Sensornetzwerken, bei denen die Einbettung nicht nur ein theoretisches Konstrukt ist, sondern den Positionen der Sensoren im Gelände entspricht, ist in der Regel $d = 2$. In einigen Fällen sind auch dreidimensionale Einbettungen ($d = 3$) interessant. Wenn die Dimension der Einbettung aus dem Kontext heraus klar ist (oder an der Stelle irrelevant ist), verwenden wir die einfachere Formulierung *eingebetteter Graph*.

Da höhere Dimensionen in unserem Universum nicht darstellbar sind, verzichten wir in dieser Arbeit weitgehend darauf, auch wenn die Einbettung von Graphen in höherdimensionale Räume nicht uninteressant ist. Eine Problemstellung dabei ist zum Beispiel die Frage nach einer Greedy-Einbettung [FPW09]. Dabei wird für einen Graphen $G = (V, E)$ eine Einbettung $p : V \to \mathbb{R}^d$ gesucht, so dass mit diesen Koordinaten Greedy-Routing möglich wird. In der zitierten Arbeit ist die Dimension von der Knotenzahl abhängig und liegt in $\mathcal{O}(\log^2 |V|)$.

In der Einbettung zeichnen wir Knoten wie üblich als Punkte oder kleine Kreise. Kanten zwischen zwei Knoten werden als gerade Linien zwischen den entsprechenden Punkten gezeichnet, manchmal auch als gebogene Kurven, wenn es der Übersichtlichkeit dient.

Definition 2 (planarer Graph) Ein *Graph* $G = (V, E)$ ist ein *planarer Graph*, wenn es eine Einbettung $p : V \to \mathbb{R}^2$ gibt, so dass sich keine zwei Kanten in der Darstellung kreuzen. Eine solche Einbettung heißt *planare Einbettung*. Durch diese planare Einbettung wird die Ebene in zusammenhängende Bereiche aufgeteilt, die *Fenster* heißen.

Nach dem Satz von Satz von Wagner und Fáry können alle planaren Graphen so eingebettet werden, dass alle Kanten als gerade Linien gezeichnet werden können, ohne dass sie sich kreuzen. Gelegentlich ist es aber sinnvoller, einzelne

Kanten als Kurven zu zeichnen.

Definition 3 (Teilgraph) Sei $G = (V, E)$ ein Graph. Ein Graph $G' = (V', E')$ ist ein *Teilgraph* von G, wenn

1. $V' \subseteq V$ und
2. $E' \subseteq E \cap \big\{\{u, v\} \,|\, u, v \in V'\big\}$.

G' ist ein *induzierter Teilgraph*, wenn $E' = E \cap \big\{\{u, v\} \,|\, u, v \in V'\big\}$.

Den von einer Knotenmenge $V' \subseteq V$ induzierten Teilgraphen bezeichnen wir auch mit $G' = G[V']$.

Definition 4 (Nachbar, Knotengrad) Sei $G = (V, E)$ ein Graph und $u \in V$ ein Knoten. Ein Knoten $v \in V$ ist ein *Nachbar* von u, wenn $\{u, v\} \in E$. Der *Knotengrad* $\deg(u)$ von u ist die Anzahl der Nachbarn von u.

$$\deg(u) \;:=\; \big|\{v \in V \mid \{u, v\} \in E\}\big|$$

Der durchschnittliche Knotengrad $\deg(G)$ von G ist

$$\deg(G) \;:=\; \frac{1}{|V|} \sum_{u \in V} \deg(u).$$

Definition 5 (Nachbarschaft) Sei $G = (V, E)$ ein Graph und $u \in V$ ein Knoten. Die *Nachbarschaft* $N(u)$ von u ist die Menge aller Knoten, die über eine Kante mit u verbunden sind, einschließlich des Knotens selbst.

$$N(u) \;:=\; u \;\cup\; \big\{v \in V \;\big|\; \{u, v\} \in E\big\}$$

Für ein $k \in \mathbb{N}$ ist die *k-Hop-Nachbarschaft* $N_k(u)$ von u die Menge aller Knoten v, für die es einen Weg von u nach v der Länge $\leq k$ gibt.

$$\begin{aligned} N_0(u) &:= \{u\} \\ N_k(u) &:= N_{k-1}(u) \;\cup\; \big\{v \in V \;\big|\; \exists u' \in N_{k-1}(u) \text{ mit } \{u', v\} \in E\big\} \\ &\quad \text{für } k \geq 1 \end{aligned}$$

Definition 6 (Weg, Kreis) Sei $G = (V, E)$ ein Graph und $s \neq t \in V$ zwei Knoten. Ein *Weg* von s nach t ist eine Folge von Knoten $p = (u_0, \ldots, u_n)$ mit

1. $u_0 = s$ und $u_n = t$
2. $\{u_i, u_{i+1}\} \in E$ für alle i.

Ein Weg ist ein *einfacher Weg*, wenn jeder Knoten nur einmal besucht wird, also $u_i \neq u_j$ für $i \neq j$.

Ein *Kreis* ist ein Weg mit $s = t$.

Ein *einfacher Kreis* ist ein Kreis, bei dem alle Knoten mit Ausnahme des Start- bzw. Zielknotens nur einmal besucht werden, also $u_i \neq u_j$ für $i \neq j$ mit $\{i, j\} \neq \{0, n\}$.

Definition 7 (Weglänge, Distanz) Sei $G = (V, E)$ ein Graph und $p = (u_0, \ldots, u_n)$ ein Weg. Die *Länge* des Weges p ist die Anzahl der Kanten auf dem Weg

$$\delta_G(p) \ := \ n$$

Ein Weg p ist ein *kürzester Weg* von s nach t, wenn es keinen Weg von s nach t mit kleinerer Länge gibt, also $\delta_G(p) \leq \delta_G(p')$ für alle Wege p' von s nach t.

Die *Distanz* zwischen zwei Knoten u und v in G ist die Länge eines kürzesten Weges von u nach v.

$$\text{dist}_G(u, v) \ := \ \min\{\delta_G(p) \mid p \text{ ist ein Weg von } u \text{ nach } v\}$$

Für einen gewichteten Graphen $G = (V, E)$ mit einer Kantengewichtsfunktion $\omega : E \to \mathbb{R}$ ist die Länge eines Weges $p = (u_0, \ldots, u_n)$ $(n \geq 1)$ die Summe der Kantengewichte entlang des Weges

$$\delta^{\omega}_G(p) \ := \ \sum_{i=0}^{n-1} \omega(\{u_i, u_{i+1}\})$$

Kürzester Weg und *Distanz* sind für gewichtete Graphen dann analog zum ungewichteten Fall definiert. Wenn der Graph G oder die Kantengewichtsfunktion ω aus dem Kontext heraus klar ist, verwenden wir auch eine vereinfachte Schreibweise $\delta(p)$ bzw. $\text{dist}(u, v)$ ohne Index.

Bemerkung 8 *Auch wenn wir Graphen und gewichtete Graphen hier so definiert haben, wie es allgemein üblich ist, betrachten wir im Folgenden nur Graphen mit den folgenden Einschränkungen, sofern nicht anders vermerkt:*

- *Wir betrachten nur* zusammenhängende *Graphen, d.h. zu jedem Knotenpaar u, v existiert auch ein Weg von u nach v.*
- *Wir nehmen bei gewichteten Graphen an, dass alle Kantengewichte positiv sind. Dadurch werden insbesondere Betrachtungen zu kürzesten*

Wegen etwas übersichtlicher. Wir betrachten also nur Kantengewichtsfunktionen

$$\omega : E \to \mathbb{R}_{>0}.$$

Im Bereich der Sensornetzwerke haben wir es nicht mit beliebigen Graphen zu tun. Die gerade angesprochenen Einschränkungen sind da nur ein Anfang. Natürlich sollte *ein* Netzwerk zusammenhängend sein - ansonsten würde man eher von *zwei* Netzwerken sprechen. Ebenso ist es sinnvoll anzunehmen, dass die Übertragung einer Nachricht von einem Knoten zu einem Nachbarn etwas kostet, und dadurch kein Zeitgewinn oder gar eine Energieersparnis erreicht werden kann. Daher kommt die Einschränkung auf positive Kantengewichte.

Wenn wir das Netzwerk als eingebetteten Graphen betrachten, dann ist ein Knoten in einem Sensornetzwerk nicht mit beliebigen anderen Knoten in der Ebene verbunden, sondern wegen der eingeschränkten Sendeleistung der Hardware nur mit den Knoten in seiner Nähe. Ein gängiges Modell dafür sind Unit-Disk-Graphen.

Definition 9 Ein Graph $G = (V, E)$ ist ein *Unit-Disk-Graph* (kurz: UDG), wenn es eine Einbettung $p : V \to \mathbb{R}^2$ gibt mit

$$\{u, v\} \in E \quad \Leftrightarrow \quad \|p(u) - p(v)\| \leq 1 \text{ für } u \neq v$$

Eine solche Einbettung heißt *Unit-Disk-Einbettung*. Dabei bezeichnet $\|.\|$ den üblichen euklidischen Abstand.

Oft verwendet man auch die äquivalente Bedingung $\{u, v\} \in E \quad \Leftrightarrow \quad \|p(u) - p(v)\| \leq r$, mit $r \in \mathbb{R}^+$, wenn man den Senderadius der Knoten nicht auf 1 beschränken will.

Eine Erweiterung dazu sind die q-Quasi-Unit-Disk-Graphen. Hier versucht man zu modellieren, dass eine Verbindung zu Knoten am äußeren Limit des Senderadius existieren *kann*, aber nicht *muss*. Nur innerhalb eines kleineren Radius q muss dann eine Verbindung bestehen.

Definition 10 Ein Graph $G = (V, E)$ ist ein *q-Quasi-Unit-Disk-Graph* (kurz q-QUDG) mit $0 \leq q \leq 1$, wenn es eine Einbettung $p : V \to \mathbb{R}^2$ gibt mit

$$\begin{aligned} \{u, v\} \in E \quad &\Rightarrow \quad \|p(u) - p(v)\| \leq 1 \\ \|p(u) - p(v)\| \leq q \quad &\Rightarrow \quad \{u, v\} \in E \text{ für } u \neq v \end{aligned}$$

Verwendet man beliebige Radien r, dann lautet hier die zweite Bedingung entsprechend $\|p(u) - p(v)\| \leq rq \Rightarrow \{u, v\} \in E$ für $u \neq v$.

Bemerkung 11 *Ein 1-QUDG ist ein Unit-Disk-Graph, und ein 0-QUDG ist ein beliebiger Graph.*

Ein in theoretischen Arbeiten oft genutzter Wert für den Parameter q ist $q = 1/\sqrt{2}$. Der Grund dafür ist, dass viele Aussagen über Unit-Disk-Graphen auch für q-Quasi-Unit-Disk-Graphen mit $q \geq 1/\sqrt{2}$ gelten, z.B. die Folgende.

Lemma 12 *Sei $G = (V, E)$ ein Unit-Disk-Graph und p eine Unit-Disk-Einbettung für G. Gegeben seien weiter vier Knoten $u_1, u_2, v_1, v_2 \in V$ und zwei Kanten $\{u_1, v_1\}, \{u_2, v_2\} \in E$, die sich in der Darstellung der Einbettung schneiden. Dann existiert mindestens eine weitere Kante zwischen diesen vier Knoten.*

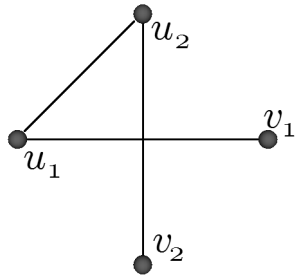

Abbildung 1.2: Zur Veranschaulichung des Lemmas

Für den Beweis überlegt man sich, wie man die vier Knoten positionieren kann, so dass der minimale Abstand der Knoten untereinander möglichst groß wird. Das ist dann der Fall, wenn die beiden Kanten die Länge 1 haben und sich mittig im rechten Winkel schneiden. Über den Satz des Pythagoras erkennt man dann, dass o.B.d.A. der Abstand zwischen u_1 und u_2 in der Einbettung kleiner oder gleich $1/\sqrt{2}$ sein muss (vgl. Abbildung 1.2). Daher existiert diese Kante in einem Unit-Disk-Graphen und auch in einem q-Quasi-Unit-Disk-Graphen mit $q \geq 1/\sqrt{2}$.

Leider ist die Eigenschaft „G ist ein Unit-Disk-Graph“ nicht einfach zu testen. Es gilt nämlich:

Satz 13 *Es ist NP-schwer zu entscheiden, ob ein gegebener Graph ein Unit-Disk-Graph ist. [BK98]*

Satz 14 *Es ist NP-schwer zu entscheiden, ob ein gegebener Graph ein $1/\sqrt{2}$-Quasi-Unit-Disk-Graph ist [KMW04].*

Damit ist es auch nicht effizient möglich, zu einem gegebenen Unit-Disk-Graphen G eine passende Einbettung zu finden.[1]

Für Sensornetzwerke bedeutet das: Wenn die Sensorknoten „irgendwo liegen“, dann kann aus der Graphstruktur nicht ohne weiteres ermittelt werden, wo genau „irgendwo“ ist. Eine absolute Positionierung (z.B. in GPS-Koordinaten) ist sowieso ohne weitere Informationen unmöglich, aber auch eine relative Positionierung der Knoten untereinander kann im Allgemeinen nicht exakt aus der Graphstruktur ermittelt werden.

Es ist jedoch durchaus denkbar, dass eine brauchbare Einbettung gefunden werden kann, die für einige Einsatzzwecke ausreichend ist. Es kann nur nicht sicher gestellt werden, dass die gefundene Einbettung eine Unit-Disk-Einbettung ist, selbst wenn die tatsächlich vorhandene Einbettung den Anforderungen einer UDG-Einbettung genügt. Der umgekehrte Fall ist ebenso denkbar: Die tatsächliche Einbettung des Netzwerkes in der Ebene ist keine UDG-Einbettung, obwohl der zugrundeliegende Graph ein Unit-Disk-Graph ist.

1.3 Das Modell in der Praxis

Die Modellierung von Sensornetzwerken über eingebettete Unit-Disk- und q-Quasi-Unit-Disk-Graphen ist allgemein Standard. Bei der Betrachtung von Kantengewichten ist die euklidische Distanz zwischen den Sensoren eine weit verbreitete Wahl. In der Praxis ergeben sich dabei jedoch einige Probleme. Diese werden im Rahmen dieser Arbeit weder gelöst noch im Detail betrachtet. Aber es gibt sie, und sie sollten bei der Auswertung von theoretischen oder simulierten Erkenntnissen nicht völlig außer Acht gelassen werden.

1.3.1 Existenz von Kanten

Theoretisch betrachtet sind UDG und q-QUDG gute Modelle für Sensornetzwerke. In der Realität sieht die Sache oft ein wenig anders aus. Besonders bei kleinen Sensorknoten mit kleinen Antennen sorgen diverse physikalische Effekte dafür, dass der zugrundeliegende Graph weder ein Unit-Disk-Graph noch ein Quasi-Unit-Disk-Graph mit einem nicht zu kleinen Parameter q ist. Oder zumindest ist die tatsächliche Einbettung dieses Graphen keine UDG-

[1] Natürlich nur unter der Annahme, dass P $\neq$ NP gilt, wovon ich in dieser Arbeit aber ausgehe.

oder q-QUDG-Einbettung.

Zwar ist die Existenz von Verbindungen zwischen sehr weit entfernten Knoten eher unwahrscheinlich, jedoch können einzelne Verbindungen zwischen sehr nahe beieinander liegenden Knoten durch Hindernisse abgeschirmt werden, oder durch Reflexion und Interferenzen so stark gestört werden, dass eine Kommunikation mit den gegebenen Mitteln nicht mehr möglich ist.

Wir werden daher in dieser Arbeit nicht nur das (Q)UDG-Modell benutzen, sondern auch Graphen betrachten, bei der die Existenz bzw. Nicht-Existenz von Kanten weniger restriktiv definiert wird.

1.3.2 Kantengewichte und Distanzmessung

Für eine möglichst gute Ermittlung der Positionen ist es sinnvoll, nicht nur die Existenz einer Verbindung zu kennen, sondern auch die Länge dieser Verbindung, also das Gewicht der Kanten. Es gibt einige Ansätze, diese Länge zu bestimmen.

Eine weit verbreitete Methode ist *RSSI* (*Radio Signal Strength Indicator*). Dabei wird die Distanz aus der ankommenden Signalstärke berechnet. Für die empfangene Signalstärke P_r gilt vereinfacht

$$P_r = \frac{P_t}{d^\alpha}$$

Dabei ist P_t die Stärke des Senders, d die Distanz zwischen Sender und Empfänger und α ein Parameter dafür, wie die sehr die Stärke des Signals mit steigender Distanz abnimmt. Im einfachsten Modell ist $\alpha = 2$, realistischer ist meistens $\alpha = 4$, allgemein akzeptiert wird ein Wert $2 \leq \alpha \leq 6$ [San05]. Es ist wenig verwunderlich, dass eine exakte Distanzmessung damit schwierig wird, wenn der gewünschte Wert in der vierten oder gar sechsten Potenz von den Messwerten abhängt. Tatsächlich zeigt sich durchgehend in praktischen Tests, dass die damit gewonnenen Distanzwerte eher unbefriedigend sind [BMPC08], [WX07], [ZZZZ08].

Andere Methoden wie *Time Difference of Arrival* haben auch ihre Tücken, um es vorsichtig zu formulieren. Sie erfordern eine stärkere Hardware oder eine exakte Synchronisierung der Knoten untereinander [GG03], und sind damit im Bereich Sensornetzwerke fast gänzlich ungeeignet.

Es bleibt also im Hinterkopf zu behalten, dass eine exakte Distanzmessung im Kontext von Sensornetzwerken schwierig ist und ungenaue Ergebnisse liefert. Wir werden daher bei der Positionierung der Knoten auch mit diskreten Kan-

tengewichten arbeiten, die die Entfernung der Knoten zueinander in einigen wenigen diskreten Schritten abschätzen. Das geschieht in der Hoffnung, dass zumindest die Messung „nah“ und „fern“ einigermaßen akkurat ist.

1.4 Einfache Routingverfahren

Das Problem, das in dieser Arbeit hauptsächlich behandelt wird, ist im Grunde ein sehr einfaches, welches auch schon in den Grundvorlesungen zu Graphentheorie behandelt wird:

Kürzester Weg *(Shortest Path)*	
Gegeben:	Ein Graph $G = (V, E)$ und zwei Knoten $s, t \in V$
Gesucht:	Ein kürzester Weg von s nach t

Das Problem, das hier in der einfachsten Variante definiert ist, lässt sich erweitern zu dem Problem *Single-Source-Shortest-Path*, bei dem ein kürzester Weg von einem Startknoten $s \in V$ zu allen anderen Knoten $u \in V$ gesucht wird; und zu *All-Pairs-Shortest-Path*, bei dem zwischen jedem Knotenpaar $(u, v) \in V \times V$ ein kürzester Weg gefunden werden soll.

Um dieses Problem zu lösen, gibt es einige einfache und effiziente Algorithmen. Bei ungewichteten Graphen reicht eine einfache Breitensuche aus. Bei gewichteten Graphen gibt es dann beispielsweise den Algorithmus von Dijkstra [Dij59].

Im Bereich der Sensornetzwerke sind diese Methoden oft nicht sinnvoll anwendbar. Hier wird in der Regel angenommen, dass man keine globale Sicht auf das gesamte Netzwerk hat. Die Verfahren laufen oft nicht auf Netzwerkebene, sondern lokal an einzelnen Knoten. Ein einzelner Knoten kennt in der Regel nur seine Nachbarschaft und vielleicht einige weitere „wichtige“ Knoten im Netzwerk, aber nicht die gesamte Netzwerkstruktur. Die Entscheidung, zu welchem Nachbarknoten eine Nachricht geschickt wird, damit sie am Ziel ankommt, muss dann anders getroffen werden.

Auf Grund dieser Einschränkungen begnügen wir uns in diesem Kontext oft mit einem *kurzen* Weg, oder im Extremfall sogar mit *überhaupt* einem Weg.

1.4.1 Fluten

Das einfachste Verfahren dafür ist das sogenannte *Fluten*. Dabei wird eine empfangene Nachricht einfach an alle Nachbarknoten weitergeleitet, falls dies noch nicht geschehen ist. Von einer globalen Sichtweise aus entspricht das Fluten in etwa einem Durchlauf des Graphen mit Breitensuche.

Der Vorteil liegt auf der Hand: Wenn das Netzwerk zusammenhängend ist, und somit der Zielknoten t vom Startknoten s aus erreicht werden kann, dann kommt die Nachricht mit Sicherheit an.

Der Nachteil ist aber ebenso offensichtlich - ein ungeheuer großer Kommunikationsaufwand. Selbst wenn der kürzeste Weg von s nach t nur wenige Knoten enthält, so wird die Nachricht dennoch im gesamten Netzwerk verteilt. Außerdem kommt es - besonders in dichten Netzwerken - zu dem Effekt, dass ein Knoten eine Nachricht von mehreren seiner Nachbarknoten erhält.

1.4.2 Greedy-Routing

Auch wenn *greedy* bei Algorithmen ein allgemeines Prinzip bezeichnet, so ist bei Routingverfahren in der Regel die in diesem Abschnitt vorgestellte Variante diejenige, die mit *Greedy-Routing* gemeint ist. Die Verfahren in den kommenden Abschnitten arbeiten zwar auch *greedy*, werden aber anders bezeichnet.

Beim Greedy-Routing wird eine Einbettung des Graphen benötigt. Dabei wird eine Nachricht immer zu dem Knoten in der Nachbarschaft weitergeleitet, dessen Position in der Einbettung der Position des Zielknoten am nächsten liegt.

Kürzester Weg *(Shortest Path)* (2)	
Gegeben:	Ein eingebetteter Graph $G = (V, E, p)$ und zwei Knoten $s, t \in V$, sowie deren Einbettungen $p(s)$ und $p(t)$.
Gesucht:	Ein kürzester Weg von s nach t

Pseudo-Code für das Greedy-Routing ist in Algorithmus 1 zusammengefasst. Ausgehend vom aktuellen Knoten u (initialisiert mit dem Startknoten $u := s$) wird der beste Fortschritt bei dem Knoten u' erreicht, für den

$$\|p(u') - p(t)\|$$

minimal ist. Um mögliche Endlosschleifen zu verhindern, muss zusätzlich der neue Knoten u' näher am Ziel liegen als u. Kann kein solcher Knoten gefunden werden, weil man auf diesem Wege in eine Sackgasse hineingeraten ist, dann schlägt das Greedy-Routing fehl. Ansonsten wird bei u' der nächste Knoten auf dem Weg zu t gesucht.

Algorithmus 1 : Einfaches Greedy-Routing

Eingabe : Ein eingebetteter Graph $G = (V, E, p)$ ein Startknoten $s \in V$ und ein Zielknoten $t \in V$

Ausgabe : Bei Erfolg ein Weg von s nach t, ansonsten eine Fehlermeldung

```
1  Initialisiere den Weg mit Startknoten s
2  Setze u := s
3  while u ≠ t do
4  |  Bestimme u' ∈ N(u) mit ‖p(u') − p(t)‖ ist minimal
5  |  if ‖p(u') − p(t)‖ < ‖p(u) − p(t)‖ then
6  |  |  Füge u' dem Weg hinzu
7  |  |  Setze u := u'
8  |  else
9  |  |  Abbruch und Ausgabe einer Fehlermeldung
10 |  end
11 end
12 if u = t then
13 |  Ausgabe des Weges
14 end
```

Der Code beschreibt das Verfahren aus globaler Sicht auf das Netzwerk. Er lässt sich aber problemlos auf ein verteiltes Verfahren umformulieren, indem der Code innerhalb der while-Schleife an Knoten u durchgeführt wird, wenn u eine Nachricht für t empfängt, und dann die Weiterleitung der Nachricht an Knoten u' durchführt.

1.4.3 Kompass-Routing

Eine andere Variante ist das *Kompass-Routing* [KSU99]. Dieses Verfahren ist eine intuitive Wahl für den Fall, dass man in einer Stadt zu einem bestimmten Ort laufen will (und keinen Stadtplan oder andere Hilfen wie Wegweiser zur Verfügung hat). Im Modell entsprechen die Straßenkreuzungen den Knoten in einem Graphen, und die Straßen zwischen den Kreuzungen den Kanten.

Möchte man von der Kreuzung, an der man sich gerade befindet, zu einem bestimmten Punkt in der Stadt laufen, dann würde man sicherlich intuitiv die Straße wählen, die möglichst gut in die Richtung des Ziels führt. Etwas formaler:

Wähle auf dem Weg zu t an einem Knoten u als nächstes einen Knoten u', mit

$$\|p(u') - p(t)\| < \|p(u) - p(t)\|,$$

für den der Winkel zwischen dem Liniensegment der Kante $\{u, u'\}$ und der Verbindungslinie von $p(u)$ nach $p(t)$

$$\angle p(u')p(u)p(t)$$

minimal wird.

1.4.4 Cost-Over-Progress

Eine weitere Variante bezieht die Kantengewichte mit in die Betrachtung ein. Im Bereich der Sensornetzwerke ist es oft sinnvoll, sparsam mit den Energiereserven der Sensoren umzugehen. Ein wichtiger Punkt dabei ist es, den Energiebedarf für das Senden von Nachrichten möglichst klein zu halten. Der Energiebedarf, um eine Nachricht über eine Distanz d zu verschicken, steigt aber nicht linear mit der Entfernung, sondern (mindestens) quadratisch. Es ist daher mit Blick auf den Energieverbrauch nicht unbedingt sinnvoll, mit jedem Schritt möglichst nahe an das Ziel zu kommen.

Bei Cost-Over-Progress [KNS06] wird die Nachricht an einen Nachbarknoten u' weitergeleitet, für den

$$\frac{\omega(u, u')}{\|p(u) - p(t)\| - \|p(u') - p(t)\|}$$

minimal wird. Die gängige Funktion zur Modellierung des Energieverbrauchs ist

$$\omega(u, v) \;:=\; a + b \cdot \|p(u) - p(v)\|^c$$

mit Parametern $a, b > 0$, $c \geq 2$. Ein optimales Routen unter diesen Voraussetzung leitet die Nachricht in äquidistanten Schritten in Richtung des Ziels weiter. Der beste Fortschritt (auch *charakteristische Distanz* genannt, vgl. [SL01]) ist

$$d^* = \sqrt[c]{\frac{a}{b \cdot (c-1)}}.$$

Je nach Wahl der Parameter lassen sich damit auch die beiden zuvor vorgestellten Verfahren beschreiben. Wählt man $\omega(u, v) = a$ für alle u, v, dann erhält man das Greedy-Routing. Bei $\omega(u, v) = \|p(u) - p(v)\|$ bekommt man ein Verfahren wie Kompass-Routing.

1.4.5 Ausweichprotokolle

Schlagen die verschiedenen Varianten des Greedy-Routing fehl, müssen Alternativen gesucht werden, um dennoch die Nachricht zum Zielknoten weiterzuleiten. Die einfachste Methode ist natürlich der Fallback zum Fluten - aber es ist klar, dass das nicht das optimale Verfahren sein kann.

Ein Ansatz liefert das sogenannte *Facetten-Routing* aus [BMSU01], bei dem ein planarer Teilgraph des Netzwerkes verwendet wird. Das bedeutet in diesem Fall, dass die Kantenmenge so reduziert wird, dass die vorhandene Einbettung der Knoten eine planare Einbettung mit geraden Kantenlinien ist. Die Nachricht wird dabei an den Knoten entlang des Fensters weitergeleitet, in dem die gedachte Linie von Startknoten zum Zielknoten liegt. Kreuzt eine Kante diese gedachte Linie, wird in das nächste Fenster gewechselt, bis schließlich das Ziel erreicht wird.

1.4.6 Virtuelle Koordinaten

Für das Routen können nicht nur die tatsächlichen (oft ja unbekannten) Koordinaten verwendet werden, sondern auch virtuelle Koordinaten, die z.B. aus der Graphstruktur heraus ermittelt werden.

Ein Ansatz dafür ist die Verwendung von besonderen Knoten, die als Anker, Landmarke oder Leuchtturm dienen (in der englischen Literatur oft *landmarks*, *beacons* oder *anchors*). Varianten sind zum Beispiel VCap [CCDU05] oder das Beacon-Vector-Routing [FRZ+05]. Hier bekommt jeder Knoten eine virtuelle Koordinate, die über die Abstände des Knotens zu einer vorher bestimmten Menge von *beacon nodes* definiert wird. Die Eindeutigkeit der virtuellen Koordinaten ist dabei nicht unbedingt gegeben, und es müssen für eine sichere Zustellung einer Nachricht ggf. weitere Vorkehrungen getroffen werden.

Die Frage nach der Eindeutigkeit der so erzeugten virtuellen Koordinaten mit einer minimalen Anzahl von Ankerknoten führt zu dem Problem der *Metrischen Dimension*, das für allgemeine Graphen NP-vollständig ist [GJ90], aber auch für viele Graphklassen. Auch für solche, die im Bereich Sensornetzwer-

ke eine große Rolle spielen, z.B. planare Graphen [DPSJvL11] oder Gabriel-Unit-Disk-Graphen [HW12]. Für weitere Varianten und einen Überblick über aktuelle Ergebnisse im Bereich der metrischen Dimension sei auf [VHW19] verwiesen.

Eine weitere Methode basiert auf einem Baum-basierten Koordinatensystem wie RPT (oft auch LTP) [CMT07] oder HECTOR [MRSRS08]. Die zugrundeliegende Idee dabei ist es, entlang eines vorher aufgebauten Baumes durch den Graphen zu routen.

1.4.7 Ausblick

Greedy-Verfahren sind eine gute Sache wenn sie funktionieren, da sie leicht zu verstehen und ohne großen Aufwand zu implementieren sind. Leider ist das Finden von kürzesten Wegen - oder überhaupt von Wegen von einem Startknoten zu einem Zielknoten - kein Problem, das sich im Allgemeinen mit Greedy-Methoden lösen lässt.

In dieser Arbeit werden wir im ersten Teil versuchen, die Erfolgsquote für Greedy-Routing zu verbessern. Dazu verwenden wir die sogenannte Multidimensionale Skalierung, um aus der Graphstruktur heraus eine Einbettung des Graphen in die Ebene zu konstruieren, die einerseits recht gut an die tatsächliche Einbettung heranreicht; die aber andererseits weniger Sackgassen für Greedy-Routing aufweist und somit häufiger eine Nachricht erfolgreich zum Ziel durchstellen kann. In experimentellen Auswertungen werden wir betrachten, wie sehr die Verwendung dieser rekonstruierten Koordinaten die Erfolgsquote unter verschiedenen Bedingungen verbessern kann.

Im zweiten Teil beschäftigen wir uns mit der Erkennung von Rändern und insbesondere Löchern in Sensornetzwerken, die die Positionsbestimmung und das Greedy-Routing auf den (tatsächlichen und virtuellen) Koordinaten oft erschweren. Der hier vorgestellte Algorithmus zur Locherkennung funktioniert dabei auch sehr gut auf sehr dünnen Netzwerken mit einem geringen durchschnittlichen Knotengrad.

Im dritten Teil schließlich widmen wir uns einem neuen Verfahren, mit deren Hilfe man aus den Sackgassen, in die Greedy-Routing gelegentlich hineinläuft, wieder entkommt und dadurch das Finden eines Weges zum Ziel garantieren kann. Dieses Routing-Verfahren kann dabei sowohl als Ausweichprotokoll verwendet werden, als auch als alleiniges Protokoll. In der experimentellen Auswertung wird deutlich, für welche Fälle das Verfahren alleine bzw. ein hybrides Verfahren sinnvoller ist.

Kapitel 2

Software zur experimentellen Auswertung

Ein großer Bestandteil dieser Arbeit ist die experimentelle Auswertung der vorgestellten Algorithmen. Dabei wurde vor allem ein Augenmerk darauf gelegt, wie sich einige Graphparameter auf die Güte der Ergebnisse auswirken. Zu diesem Zweck wurde für diese Arbeit eine Software entwickelt, in der die hier vorgestellten Algorithmen implementiert wurden. Über eine grafische Benutzerschnittstelle können die unterstützten Graphparameter eingestellt werden. Die Auswirkungen dieser Parameter können dann an einzelnen Graphen genauer untersucht werden, oder zur statistischen Auswertung an vielen Graphen nacheinander simuliert werden.

Die meisten Abbildungen mit Graphen in dieser Arbeit sind ebenfalls mit dieser Software erstellt worden.

2.1 Implementierung

Als Programmiersprache und -Umgebung wurde *Delphi* von *Embarcadero Technologies* gewählt. Wegen des starken Anteils von Operationen auf Matrizen der Algorithmen in Kapitel 3 wurde zunächst auch mit den in der Mathematik üblichen Sprachen wie Matlab (bzw. die freie Alternative GNU Octave) experimentiert, die dort ihre volle Stärke ausspielen können. Dabei gelangt man jedoch bei der dort verwendeten Skriptsprache recht schnell an andere Grenzen, so dass sich insgesamt die Verwendung einer universell einsetzbaren Entwicklungsumgebung als deutlich sinnvoller herausgestellt hat.

Für die Matrizen-Operationen wurden die entsprechenden Pakete des *ALGLIB*-Projektes verwendet [Boc]. Dort werden sowohl einfache Operationen wie die Multiplikation zweier Matrizen als auch komplexere wie die Bestimmung der inversen Matrix oder der Singulärwertzerlegung implementiert.

Die dreidimensionale Darstellung der Netzwerke wurde mit *GLScene* realisiert, einer auf OpenGL basierenden Bibliothek für Delphi und C++Builder, die visuelle Komponenten zum Rendern von 3D-Szenen anbietet. Der Fokus liegt dabei auf einer einfachen Anwendbarkeit für den Entwickler, ohne dabei zu starke Einbußen an die Performance machen zu müssen [GL].

Die Realisierung von Graphen geschieht über eine selbst entwickelte Klasse `TSensorNetwork`[1]. Kern dieser Klasse ist eine Liste von Knoten vom Typ `TSensorNode`. Die Speicherung der Kanten ist über Adjazenzlisten an den Knoten-Objekten realisiert, für die Verwaltung der graphischen Darstellung wird zusätzlich eine globale Kantenliste verwendet.

Die Knotenklasse besitzt neben der Liste der adjazenten Knoten eine ganze Reihe weiterer Properties und Methoden, über die prinzipiell auch die Simulation verteilter Berechnungen möglich wäre. Darauf liegt in dieser Arbeit aber nicht der Schwerpunkt. Die implementierten Algorithmen sind oft realisiert als eine Mischform aus globalen Algorithmen, die die Kenntnis des gesamten Netzwerkes erfordern, und lokalen (verteilten) Algorithmen, wie sie in einem real existierenden Sensornetzwerk in den Programmen der einzelnen Knoten ablaufen könnten. Letzteres wird auch teilweise schon durch den objektorientierten Ansatz induziert - so ist beispielsweise das `visited`-Flag bei einer Breitensuche in einem Graphen mit n Knoten kein globales Array der Größe n, sondern eine boolsche Variable innerhalb der Knoten-Klasse.

Die Implementierung der Algorithmen läuft dann in von der Basis-Netzwerk-Klasse abgeleiteten Klassen, die um entsprechende Methoden und Variablen erweitert wurden.

2.1.1 Grafische Benutzerschnittstelle

Die Software bietet eine grafische Benutzerschnittstelle, über die Netzwerke erstellt und bearbeitet werden können. Einzelne Knoten können erstellt, verschoben, dupliziert und gelöscht werden. Beim Verschieben der Knoten werden optional automatisch passende Kanten für das gewählte Graphmo-

[1] Namenskonvention in Delphi bzw. Object Pascal generell: Klassenbezeichner beginnen immer mit einem großen `T`.

dell eingefügt bzw. wieder entfernt, wenn der Knoten in die Nähe anderer Knoten verschoben wird.

Ebenso können einzelne Kanten hinzugefügt und entfernt werden, oder aber es können zwischen den vorhandenen Knoten automatisch Kanten entsprechend ihrer Entfernung zueinander erstellt (und gelöscht) werden. Das Ergebnis hängt dann von der gewählten maximalen Kantenlänge und dem gewünschten Kantenmodell ab.

Neben der völlig freien Erzeugung von Graphen gibt es die Möglichkeit, einige bestimmte Graphen mit variabler Knotenzahl zu erzeugen, z.B. Wege, Sterne (mit 3, 4 oder 5 Strahlen), Kreise und Gitter. Diese Graphen sind bei der Betrachtung einiger Spezialfälle interessant. Wenn bei der Erstellung eines Netzwerkes die Topologie jedoch derart bekannt ist, sind die in dieser Arbeit behandelten Verfahren nicht unbedingt optimal - in solchen Fällen würde man eher zu hochspezialisierten Methoden greifen, die genau auf diese Netzwerktopologie zugeschnitten sind.

Bei dreidimensional eingebetteten Graphen ist außerdem die Möglichkeit zur Rotation über alle drei Achsen gegeben, um die dreidimensionale Gestalt des Netzwerkes besser erfassen zu können.

In der Anzeige des Netzwerkes können einige Aspekte der implementierten Algorithmen hervorgehoben werden, wie zum Beispiel Ankerknoten, Routingwege zwischen zwei ausgewählten Knoten, Anzeige der Knoten an den tatsächlichen Positionen oder den berechneten virtuellen Koordinaten (soweit das sinnvoll ist), oder eine farbliche Markierung bestimmter Teilbereiche des Netzwerkes.

Wichtiger für die Testzwecke ist jedoch die Generierung zufälliger Graphen anhand verschiedener Parameter. Dabei sollen nacheinander viele Graphen erstellt werden, die Algorithmen darauf angewendet werden, um dann die Ergebnisse auswerten zu können.

2.2 Erzeugung zufälliger Graphen mit verschiedenen Parametern

Eine einfache Methode zur Erzeugung eines zufälligen Graphen $G = (V, E)$ mit n Knoten ist es, für jedes Knotenpaar $\{u, v\}$ mit $u, v \in V$ unabhängig zu entscheiden, ob die Kante $\{u, v\}$ in die Kantenmenge E aufgenommen werden soll oder nicht. In diesem einfachen Modell gibt es natürlich je nach Einsatzzweck einige Probleme mit den erzeugten zufälligen Graphen. So wird

zwar bei einer geordneten Knotenmenge und einer Wahrscheinlichkeit von $P(\{u, v\} \in E) = \frac{1}{2}$ für alle $\{u, v\}$ jeder Graph mit derselben Wahrscheinlichkeit erstellt. Betrachtet man jedoch ungeordnete Knotenmengen (wie es bei Graphen allgemein üblich ist), dann muss man isomorphe Graphen berücksichtigen. So gibt es nur einen Graphen mit $|V| = 3$ Knoten und $|E| = 0$ Kanten, aber drei Graphen mit $|V| = 3$ Knoten und $|E| = 2$ Kanten (vgl. Abbildung 2.1), die aber alle zueinander isomorph sind. Möchte man solche Dinge berücksichtigen und benötigt man eine Gleichverteilung der erzeugten Graphen unter Berücksichtigung von Isomorphien, dann wird die Erzeugung zufälliger Graphen natürlich deutlich komplizierter.

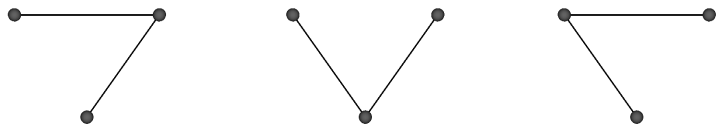

Abbildung 2.1: Darstellung dreier paarweise isomorpher Graphen mit $|V| = 3$ und $|E| = 2$

Für Sensornetze bzw. für unsere geplanten Zwecke spielen diese Betrachtungen jedoch keine oder nur eine untergeordnete Rolle. Ob eine Kante existiert oder nicht, hängt in erster Linie ab von der Position der Knoten in der Einbettung und ihrer Entfernung zueinander.

2.2.1 Positionierung der Knoten

Zunächst betrachten wir die Verteilung der Knoten auf einem festgelegten Gebiet. Die offensichtliche und einfachste Methode zur Verteilung einer bestimmten Anzahl von Knoten ist eine gleichmäßige Verteilung. Die Knoten werden dabei mittels eines einfachen Pseudozufallszahlengenerators positioniert, indem die x- und y-Koordinaten einzeln ausgewürfelt werden.

In einer zweiten Variante wird eine ungleichmäßigere Verteilung angestrebt. Hier werden an zufällig ausgewählten Positionen (*Zentren*) direkt mehrere Knoten ausgesetzt, die dann nach einer Normalverteilung zirkulär um dieses Zentrum herum verteilt werden. Dazu wird ein Winkel gewählt (gleichverteilt zwischen 0 und 2π), und eine normalverteilte Distanz zum Zentrum. Dadurch werden mehr Knoten in der Nähe des Zentrums angeordnet. Die gewünschte Gesamtanzahl von Knoten wird gleichmäßig auf die Zentren verteilt. Die normalverteilten Zufallswerte werden aus den üblichen gleichverteilten über die Polarmethode aus [MB64] berechnet.

Algorithmus 2 : Verteilung der Knoten

Eingabe : Gewünschte Knotenzahl n
rechteckiges Verteilungsgebiet
$[x_{min}, x_{max}] \times [y_{min}, y_{max}] \in \mathbb{R}^2$
optional: Anzahl der Zentren $s \leq n$
optional: Streuparameter σ

```
1  switch Knotenmodus do
2      case Gleichmäßig do
3          for i = 1, ..., n do
4              Wähle zufälliges x mit x_min ≤ x ≤ x_max;
5              Wähle zufälliges y mit y_min ≤ y ≤ y_max;
6              Füge V einen neuen Knoten u hinzu;
7              Setze p(u) := (x, y);
8          end
9      case Zentren do
10         for i = 1, ..., s do
11             Wähle zufälliges x mit x_min ≤ x ≤ x_max;
12             Wähle zufälliges y mit y_min ≤ y ≤ y_max;
13             for j = 1, ..., n/s do
14                 Setze d := σ · RandomNormal();
15                 Wähle zufälligen Winkel 0 ≤ α < 2π;
16                 Füge V einen neuen Knoten u hinzu;
17                 Setze p(u) := (x + d cos α, y + d sin α);
18             end
19         end
20 end
```

Hindernisse

Als weitere Option können wir in dem rechteckigen Verteilungsgebiet Bereiche markieren, die für eine Positionierung gesperrt sein sollen. Wird ein Knoten über die Zufallswahl in diesem Bereich positioniert, dann wird dieser Knoten verworfen und wird nicht neu positioniert. In der Implementierung wird dies realisiert durch ein passend großes Schwarz-Weiß-Bitmap.

Dadurch soll ein Verteilen der Sensoren in einer Gegend simuliert werden, in dem durch topographische Besonderheiten (z.B. Gebäude oder Gewässer) einige Bereiche nicht mit Sensoren bestückt werden können.

Bei der Festlegung der Kanten kann in Zusammenhang mit solchen knoten-

freien Gebieten die zusätzliche Bedingung gewählt werden, dass eine Kante $\{u, v\}$ nur dann eingefügt werden kann, wenn zwischen den beiden Knoten eine Sichtverbindung besteht, d.h. die gerade Verbindung von $p(u)$ nach $p(v)$ berührt kein gesperrtes Gebiet. Dies wird pixelweise entlang der Linie von $p(u)$ nach $p(v)$ getestet. Für die Implementierung besteht hier eine kleine Stolperfalle, da durch die dabei auftretenden Treppenartefakte ein Test von $p(u)$ nach $p(v)$ unter Umständen ein anderes Ergebnis liefert als ein Test von $p(v)$ nach $p(u)$.

Dadurch sollen eingeschränkte Verbindungen simuliert werden, wie sie z.B. zwischen Knoten auf einem innerstädtischen Straßennetz mit Gebäuden auftreten können, die die Verbindungen blockieren. Für Abbildung 2.2 (unten) dient ein Straßenzug in Berlin Schöneberg (52°30'N, 13°21'E) als Vorlage, welchen wir auch später in den experimentellen Auswertungen verwenden.

Dreidimensionale Netzwerke

In einigen Anwendungsfällen können auch dreidimensional ausgelegte Netzwerke interessant sein. Damit sind nicht nur Netzwerke in einem hügeligen Gelände gemeint, sondern frei schwebende Sensorknoten für atmosphärische Beobachtungen oder Unterwasser-Szenarien. Auch solche Netzwerke können erzeugt, angezeigt und getestet werden. In Abildung 2.2 ist der Übersichtlichkeit wegen nur ein kleines 3D-Netzwerk mit wenigen Knoten zu sehen. Die in den Simulationen verwendeten Netze sind deutlich größer.

In diesem Fall wird in Algorithmus 2 zusätzlich zu den Koordinaten $x, y \in \mathbb{R}$ eine dritte Koordinate $z \in \mathbb{R}$ gewählt. Bei einer ungleichmäßigen Verteilung über Zentren werden für jeden Knoten zwei Winkel vom Zentrum gewählt und die Position des Knotens entsprechend darüber bestimmt.

Hindernisse können hier ebenfalls über ein Bitmap definiert werden. Wegen des höheren Speicherbedarfs allerdings mit einer deutlich gröberen Auflösung. Für die Software ist ein fester Skalierungsfaktor von 25 gewählt, d.h. ein Pixel in dem Hindernis-Bitmap entspricht einem Block der Größe $25 \times 25 \times 25$ in der Testumgebung, in der das Netzwerk ausgelegt werden soll. Die Bitmap-Datei enthält dann die einzelnen Ebenen der Höhe 25 untereinander angeordnet. Für ein Gebiet der Größe $1000 \times 1000 \times 1000$ wird also ein Bitmap mit den Ausmaßen 40×1600 benötigt.

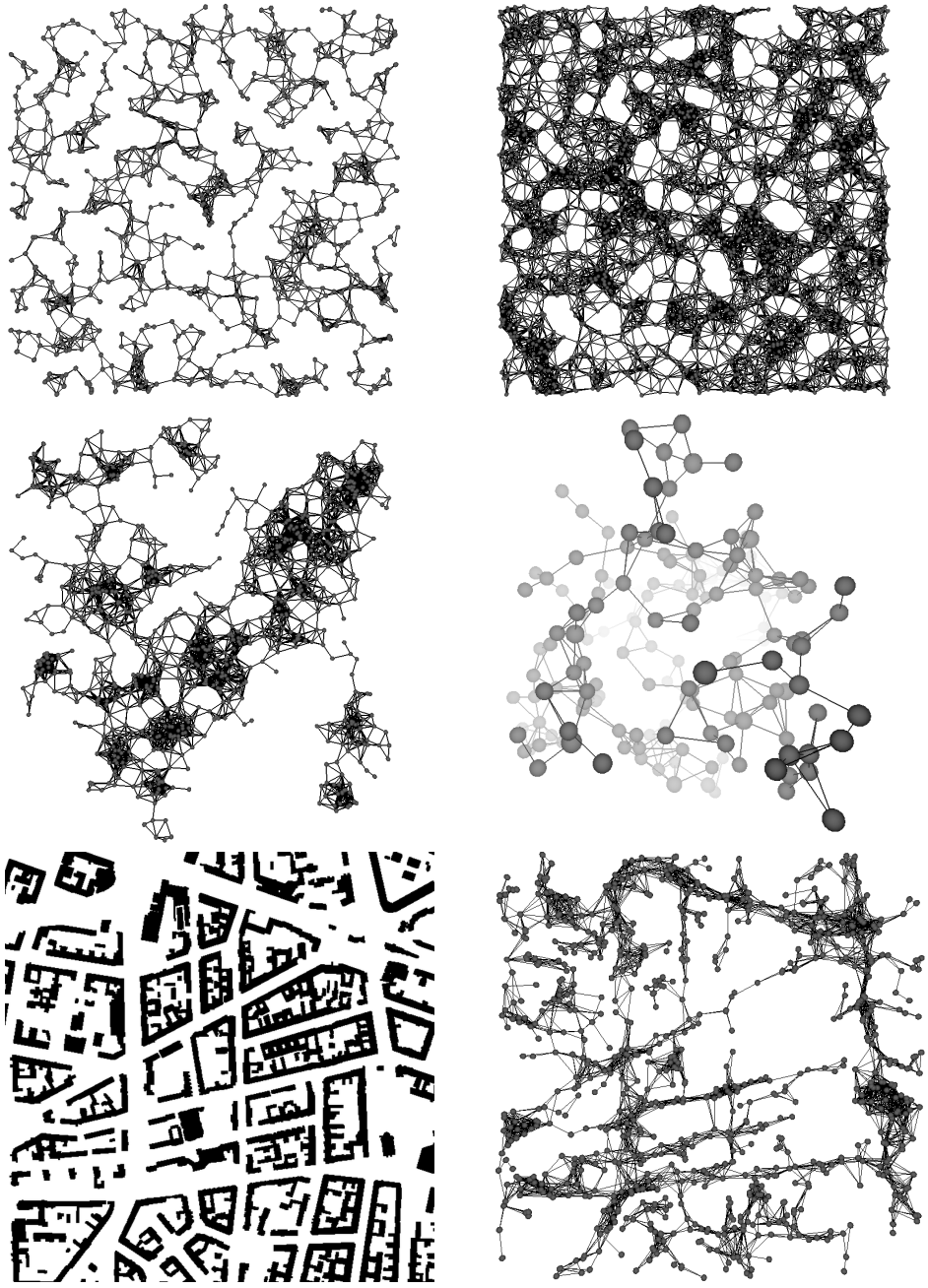

Abbildung 2.2: Einige Beispiele für generierte Netzwerke. Oben zwei Unit-Disk-Graphen mit 800 bzw. 2000 Knoten bei gleichem Senderadius. In der Mitte ein Graph mit einer Knotenverteilung über Zentren (links) und 3D-Auslegung (rechts). Unten ein Hindernis-Bitmap und ein daraus resultierender Graph bei Notwendigkeit einer Sichtverbindung zwischen zwei Knoten.

2.2.2 Festlegung der Kanten

Für die Erzeugung der Kanten wählen wir drei Varianten.

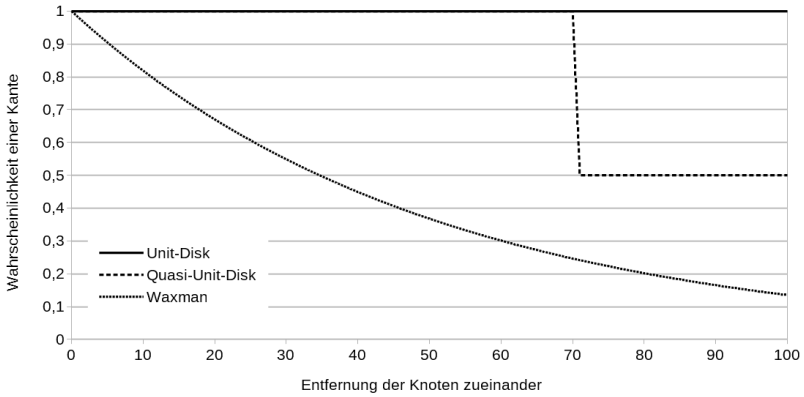

Abbildung 2.3: Wahrscheinlichkeiten für die Erzeugung einer Kante in Abhängigkeit der Entfernung zweier Knoten bei Unit-Disk-Graphen, Quasi-Unit-Disk-Graphen und Waxman-Modell mit $r = 100$.

Zuerst die im Bereich Sensornetzwerke offensichtliche Wahl für Unit-Disk-Graphen (kurz *UDG*): Verbinde zwei Knoten $u, v \in V$ genau dann durch eine Kante, wenn $\|p(u) - p(v)\| \leq r$, wobei der Senderadius r ein weiterer Eingabeparameter ist. Für die Wahrscheinlichkeit, dass zwei Knoten u und v mit einer Kante verbunden sind gilt also:

$$P\big(\{u,v\} \in E\big) \;:=\; \begin{cases} 1 & \text{falls } \|p(u) - p(v)\| \leq r \\ 0 & \text{sonst} \end{cases}$$

Als zweites wählen wir $\frac{1}{\sqrt{2}}$-Quasi-Unit-Disk-Graphen ($QUDG$). Dabei setzen wir für die Kanten im äußeren Radius eine feste Wahrscheinlichkeit von $p = 0.5$, also

$$P\big(\{u,v\} \in E\big) \;:=\; \begin{cases} 1 & \text{falls } \|p(u) - p(v)\| \leq \frac{r}{\sqrt{2}} \\ 0.5 & \text{falls } \frac{r}{\sqrt{2}} < \|p(u) - p(v)\| \leq r \\ 0 & \text{sonst} \end{cases}$$

Zuletzt wählen wir ein Modell, das auch in der Literatur sehr oft verwendet wird, und auf Waxman [Wax88] zurückgeht.

$$P\big(\{u,v\} \in E\big) \;:=\; \begin{cases} \beta \exp\big(-\frac{\|p(u)-p(v)\|}{\alpha r}\big) & \text{falls } \|p(u) - p(v)\| \leq r \\ 0 & \text{sonst} \end{cases}$$

Die Parameter α und β liegen beide im halboffenen Intervall $(0, 1]$. Größere Werte für β erzeugen generell dichtere Graphen. Über den Parameter α lässt sich der Anteil an kurzen Kanten steuern - bei einem kleinen Wert für α fällt die Wahrscheinlichkeit sehr schnell mit zunehmender Distanz ab, bei einem großen langsamer.

Im Modell von Waxman fehlt eigentlich die Nebenbedingung „falls $\|p(u) - p(v)\| \leq r$“, da dort nicht explizit auf Funknetzwerke mit einer beschränkten maximalen Kantenlänge eingegangen wird. Beliebig lange Kanten, und seien sie noch so unwahrscheinlich, ergeben im Bereich der Sensornetze aber keinen Sinn, wenn man mal von Angriffen wie z.B. die Wurmlochattacke auf die Topologie des Netzes absieht (z.B. [HPJ02]).

In den in dieser Arbeit durchgeführten Testläufen wählen wir $\beta = 1$ und $\alpha = 0.5$.

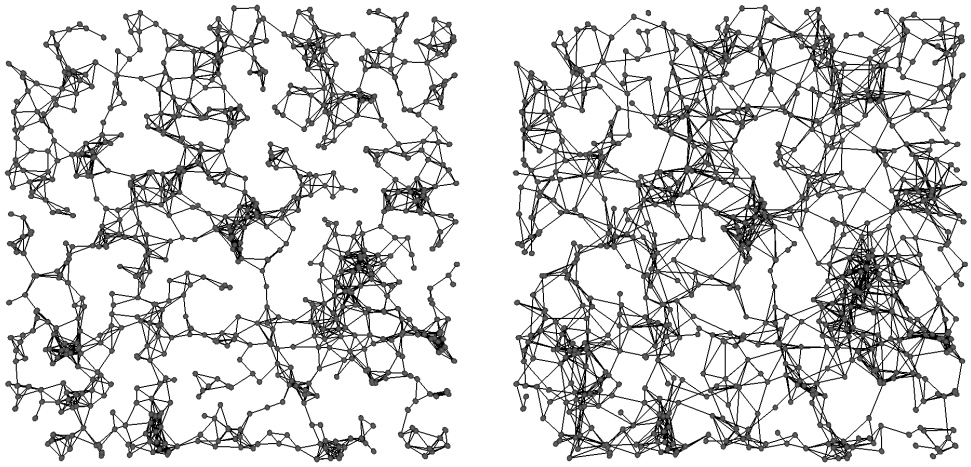

Abbildung 2.4: Die eingebettete Knotenmenge des UDG mit 800 Knoten aus Abbildung 2.2 mit $1/\sqrt{2}$-QUDG- und Waxman-Kanten. Die maximale Sendereichweite ist erhöht, um einen vergleichbaren durchschnittlichen Knotengrad zu erreichen.

In der experimentellen Auswertung wählen wir bei den unterschiedlichen Kantenmodellen unterschiedliche maximale Senderadien r. Die Werte sind dabei so gewählt, dass in den entstehenden Graphen der durchschnittliche Knotengrad (in etwa) übereinstimmt, um eine gewisse Vergleichbarkeit der Ergebnisse zu ermöglichen. Daher wählen wir in der Regel bei einer zweidimensionalen Auslegung der Knoten für UDG, q-QUDG und Waxman-Kanten die Senderadien von $r = 50$, $r = 58$ bzw. $r = 93$.

2.2.3 Zusammenhängende Graphen

Bei den in dieser Arbeit behandelten Algorithmen ist es sinnvoll, nur zusammenhängende Graphen zu betrachten. Wenn es keinen Weg zwischen zwei Knoten s und t gibt, dann ist auch ein Vergleich von Routingalgorithmen an dieser Stelle wenig sinnvoll. Oder anders formuliert: Wenn ein Sensornetzwerk aus zwei (oder mehr) Zusammenhangskomponenten besteht, dann handelt es sich dabei nicht um *ein* Sensornetzwerk, sondern um zwei (oder mehrere) Sensornetzwerke.

Bei der zufälligen Erstellung der Graphen ist es jedoch recht wahrscheinlich, dass die dabei entstehenden Graphen nicht zusammenhängend sind, und wenn es nur ein paar einzelne Knoten sind, die nicht mit dem Rest des Netzwerkes verbunden sind. Für die automatischen Testreihen sind solche kleinen Zusammenhangskomponenten störend.

Daher wird vor der Anwendung der Positionierungs- und Routingalgorithmen die größte Zusammenhangskomponente des erstellten Graphen bestimmt. Die Knoten der kleineren Komponenten werden aus dem Graphen entfernt. Bleiben dabei weniger als zwei Drittel der Knoten des ursprünglich erstellten Graphen übrig, dann wird dieser Graph verworfen und nicht weiter getestet. Wenn aufgrund dessen zu viele Graphen verworfen werden müssen, dann wird der gesamte Test mit diesen Parametern abgebrochen. Das wird als Zechen dafür gewertet, dass die Erzeugung von Graphen mit diesen Parametern zu keinen (oder nur wenig) sinnvollen Ergebnissen führt. Als Grenze dafür wählen wir in den Auswertungen zehn in Folge verworfene Graphen. In den Testreihen in den folgenden Kapiteln sind die Parameter so gewählt, dass dieser Fall nicht auftritt.

Kapitel 3

Bestimmung virtueller Koordinaten mittels Multidimensionaler Skalierung

In diesem Teil der Arbeit betrachten wir einen Lösungsansatz, wie aus der Graphstruktur eine Einbettung der Knoten in die Ebene (oder in den dreidimensionalen Raum) ermittelt werden kann, die erstens der tatsächlichen Einbettung mehr oder weniger gut entspricht, und/oder zweitens die Erfolgsquote für Greedy-Routing gegenüber der tatsächlichen Einbettung verbessert. Diese beiden Ziele können in vielen Fällen nicht gleichzeitig erreicht werden. Eine Einbettung, die eher der tatsächlichen Einbettung entspricht, sorgt oft für eine schlechtere Erfolgsquote beim Greedy-Routing und umgekehrt.

Dazu verwenden wir die sogenannte *Multidimensionale Skalierung*. In den folgenden Abschnitten wird zunächst das Prinzip der Multidimensionalen Skalierung vorgestellt und die mathematischen Hintergründe für einen iterativen Algorithmus erläutert. Daran anschließend folgen einige eigene Beobachtungen und Modifikationen des Grundprinzips. Zum einen stellen wir eine einfache aber sehr effektive Methode vor, wie schon nach wenigen Iterationen eine sehr gute Einbettung gefunden werden kann. Zum anderen eine Modifikation, die einen Nachteil des Verfahrens behebt und in vielen Fällen die gefundene Einbettung in der Hinsicht verbessert, dass diese näher an die tatsächliche Einbettung heran kommt.

3.1 Einführung

Unter dem Begriff *Multidimensionale Skalierung* (kurz: MDS) werden Verfahren zusammengefasst, die folgendes Problem lösen:

Eingabe: n Punkte $y_1, \ldots, y_n \in \mathbb{R}^{m_1}$

Ausgabe: n Punkte $x_1, \ldots, x_n \in \mathbb{R}^{m_2}$ mit $m_2 < m_1$, so dass die Abstände zwischen den Punkten möglichst gut erhalten bleiben, d.h.

$$\|y_i - y_j\|_{m_1} \sim \|x_i - x_j\|_{m_2} \; \forall i, j$$

Was genau „möglichst gut“ bedeutet, wird durch die *loss function* festgelegt. Weit verbreitet ist die euklidische Norm und die Minimierung der Differenzquadrate

$$\sum_{i \neq j} \left(\|y_i - y_j\|_{m_1} - \|x_i - x_j\|_{m_2} \right)^2 .$$

Da zur Berechnung der Punkte im Raum mit geringerer Dimension nicht die eigentlichen Koordinaten der Punkte verwendet werden, sondern nur die Distanzen zwischen ihnen, wird das Problem, das mit Multidimensionaler Skalierung gelöst wird, oft auch so spezifiziert:

MULTIDIMENSIONALE SKALIERUNG *(MDS)*

Gegeben: Eine Matrix $\Delta = (\delta_{ij}) \in \mathbb{R}^{n \times n}$ mit

1. Δ ist symmetrisch, also $\delta_{ij} = \delta_{ji}$ für $1 \leq i, j \leq n$
2. $\delta_{ii} = 0$ für $1 \leq i \leq n$

Gesucht: n Punkte $x_1, \ldots, x_n \in \mathbb{R}^m$, so dass die Abstände zwischen den Punkten möglichst gut den δ_{ij} entsprechen, z.B. durch Minimierung von

$$\sum_{i \neq j} (\delta_{ij} - \|x_i - x_j\|)^2.$$

Die Eingabematrix Δ kann als Distanzmatrix von Punkten in einem höher dimensionierten Raum betrachtet werden. Die Ausgabe ist dann eine Einbettung dieser Punkte in einen Raum mit geringerer Dimension. Wenn die Eingabematrix einer Distanzmatrix zu einer Menge von Punkten im $\mathbb{R}^m$ ist, dann sollte MDS die Koordinaten dieser Punkte liefern - ggf. bis auf Rotation, Verschiebung und Spiegelung.

In der Literatur wird in Arbeiten zum Thema Positionierung der Knoten in Sensornetzwerken sehr häufig das Buch [BG97] zitiert. Dieses Werk stammt nicht unbedingt aus dem mathematisch-naturwissenschaftlichen Bereich, sondern richtet sich vor allem auch an Anwender jenseits der Mathematik.

Ein Anwendungsbeispiel daraus sind die Korrelationen einiger Verbrechensraten in den USA, wie sie in Tabelle 3.1 aufgeführt sind.

Verbrechen	Nr.	1	2	3	4	5	6	7
Mord	1	1.00	0.52	0.34	0.81	0.28	0.06	0.11
Vergewaltigung	2	0.52	1.00	0.55	0.70	0.68	0.60	0.44
Raub	3	0.34	0.55	1.00	0.56	0.62	0.44	0.62
Körperverletzung	4	0.81	0.70	0.56	1.00	0.52	0.32	0.33
Einbruch	5	0.28	0.68	0.62	0.52	1.00	0.80	0.70
Diebstahl	6	0.06	0.60	0.44	0.32	0.80	1.00	0.55
Autodiebstahl	7	0.11	0.44	0.62	0.33	0.70	0.55	1.00

Tabelle 3.1: Korrelation zwischen Verbrechensraten in 50 US-Staaten (vgl. [BG97])

Diese Tabelle enthält nur einige wenige Daten, ist aber auch so schon relativ unübersichtlich. Wie die einzelnen Verbrechensraten zueinander in Beziehung stehen, ist nicht auf einen Blick ersichtlich. Wendet man auf diese Tabelle einen MDS-Algorithmus an, bekommt man eine zweidimensionale Darstellung wie in Abbildung 3.1. Diese ist nicht nur leichter für den Menschen zu überblicken, sie lässt auch weitere Interpretationen zu. So könnte man die X-Achse als einen Verlauf zwischen Angriffen auf Personen und Angriffen auf Eigentum ansehen, und die Y-Achse als eine Einteilung nach versteckten und eher öffentlichen Verbrechen.

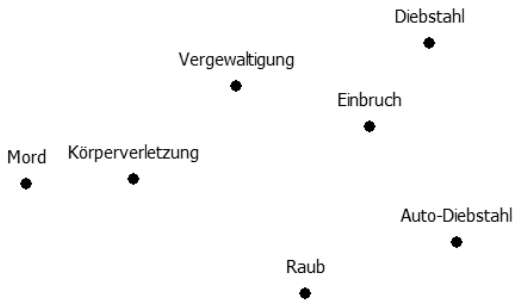

Abbildung 3.1: Eine zweidimensionale Anordnung der Verbrechensraten durch MDS

MDS wird in diesem Beispiel also eingesetzt, um Objekte, von denen eine gewisse Ähnlichkeitsbeziehung bekannt ist, sinnvoll in der Ebene anzuordnen, um in einem nächsten Schritt aus dieser Anordnung weitere Schlüsse ziehen zu können. Die Eingabematrix Δ wird dabei als *(Un-)Ähnlichkeitsmatrix* oder auch *Distanzmatrix* bezeichnet - je nachdem ob ein hoher Wert eine starke Ähnlichkeit angibt (wie in dem Beispiel mit den Verbrechensraten) oder eine hohe Distanz (wie wir es bei Sensornetzen brauchen werden). Die Werte (δ_{ij}) werden oft aus Statistiken entnommen, durch Umfragen gewonnen oder eben durch Analyse des Sensornetzwerks.

Für unsere Zwecke lassen sich diese Methoden ebenfalls einsetzen. Wir haben Sensorknoten gegeben, die eine tatsächliche Anordnung in der Ebene besitzen. Diese Anordnung ist jedoch oft unbekannt und soll ermittelt werden. Der Ansatz ist, aus den Daten, die die einzelnen Knoten ermitteln können (z.B. kürzeste Wege zu anderen Knoten), eine Distanzmatrix aufzubauen, aus der dann mit den Verfahren der MDS Positionen ermittelt werden. Die Hoffnung ist, dass die ermittelten Distanzwerte ungefähr den euklidischen Distanzen entsprechen, und dass MDS aus dieser mehr oder weniger stark fehlerbehafteten Distanzmatrix eine Anordnung berechnet, die der tatsächlichen mehr oder weniger gut entspricht.

Wie genau diese Anordnung im Einzelfall berechnet wird, ist unterschiedlich und hängt auch von weiteren Umständen ab. Beispielsweise davon, ob die Distanzmatrix vollständig ist, d.h. ob zu jedem $i \neq j$ ein δ_{ij} ermittelt werden konnte. Da Δ auch symmetrisch ist, kann in diesem Fall von einer von Δ abgeleiteten Matrix B_Δ dann eine Zerlegung der Form

$$B_\Delta = Q\Lambda Q'$$

berechnet werden (*Spektral-* oder *Eigenwertzerlegung*), bei der Λ eine Diagonalmatrix ist. Die Koordinatenmatrix der Punkte wird dann auf

$$X = Q_+\Lambda_+^{1/2}$$

gesetzt, wobei Q_+ die Matrix ist, die nur aus den ersten zwei oder drei Spalten von Q besteht und Λ_+ die entsprechend verkleinerte Diagonalmatrix - je nachdem ob eine zwei- oder dreidimensionale Anordnung gewünscht wird.

Dieses Verfahren näher zu erläutern ist hier nicht zielführend. In unserem Anwendungsfall ist die Matrix Δ oft *nicht* vollständig gegeben, da wir nicht voraussetzen wollen, dass wir zu jedem Knotenpaar eine Distanzschätzung ermitteln können. Daher ist diese MDS-Variante nicht brauchbar.

Stattdessen nutzen wir die sogenannte *Stress Majorisierung*, bzw. die Minimierung der *Stressfunktion* durch den SMACOF-Algorithmus (*Scaling by majorizing a convex function*).

3.2 Der SMACOF-Algorithmus

Für die nähere Erläuterung des Verfahrens benötigen wir zunächst einige grundlegende Begriffe und Schreibweisen aus dem Bereich der linearen Algebra.

Eine $m \times n$-Matrix $A \in \mathbb{R}^{m \times n}$ mit Einträgen aus den reellen Zahlen ist eine Matrix mit m Zeilen und n Spalten. Wenn wir die einzelnen Elemente einer Matrix benötigen, benutzen wir auch die Schreibweise $A = (a_{ij})$. Ein einzelnes Element a_{ij} $(1 \leq i \leq m,\ 1 \leq j \leq n)$ steht in der Matrix A in der i-ten Zeile und der j-ten Spalte. Eine Matrix heißt *quadratisch*, wenn $n = m$ gilt. Sie heißt *symmetrisch*, wenn für alle Elemente $a_{ij} = a_{ji}$ gilt. Eine quadratische Matrix heißt *Diagonalmatrix*, wenn $a_{ij} = 0$ für alle $i \neq j$.

Die *Spur* einer Matrix A ist die Summe ihrer Diagonalelemente. Die Spur bezeichnen wir mit $tr(A)$, also

$$tr(A) \ := \ \sum_{i=1}^{n} a_{ii}$$

Die Diagonalmatrix, für die alle $a_{ii} = 1$ sind, heißt *Einheitsmatrix* und wird mit I bezeichnet. Wenn es für eine quadratische Matrix A eine Matrix A^{-1} gibt mit $AA^{-1} = A^{-1}A = I$, dann ist A *invertierbar* und A^{-1} heißt die *Inverse* zu Matrix A.

Ist $A = (a_{ij}) \in \mathbb{R}^{m \times n}$ eine Matrix, dann bezeichnen wir mit $A' = (a_{ji}) \in \mathbb{R}^{n \times m}$ die *Transponierte* der Matrix, bei der Zeilen und Spalten vertauscht sind.

Für nicht invertierbare Matrizen A wird die allgemeinere Pseudo-Inverse definiert. Wir verwenden hier die *Moore-Penrose-Inverse* A^{+}. Diese ist die ein-

deutig bestimmte Matrix, die die folgenden Bedingungen erfüllt.[1]

$$\begin{aligned} AA^+A &= A^+ \\ A^+AA^+ &= A \\ (AA^+)' &= AA^+ \\ (A^+A)' &= A^+A \end{aligned}$$

Eine $m \times 1$ Matrix heißt *Spaltenvektor*, eine $1 \times n$-Matrix heißt *Zeilenvektor*.

Spaltenvektoren werden wir in der Regel mit einem Kleinbuchstaben (z.B. a) bezeichnen, Matrizen mit mehreren Zeilen und Spalten mit Großbuchstaben (z.B. A). Zeilenvektoren schreiben wir als a'.

Eine Menge von n Spaltenvektoren im $\mathbb{R}^m$ kann zu einer $m \times n$-Matrix zusammengefasst werden. Wir sprechen dann von einer *Koordinatenmatrix*. Für Zeilenvektoren entsprechend.

Sofern nicht anders vermerkt, bezeichnen wir mit $\|a\|$ die *euklidische Norm* eines Vektors $a \in \mathbb{R}^n$, definiert als

$$\|a\| := \sqrt{\sum_{i=1}^{n} a_i^2}.$$

Anschaulich entspricht $\|a\|$ der Länge des Vektors a.

Als Matrixnorm verwenden wir die *Frobeniusnorm*, d.h. die Wurzel aus der Summe der Quadrate aller Elemente.

$$\|A\| := \sqrt{\sum_{i,j} a_{ij}^2}$$

Die Zusammenfassung und die folgende Herleitung des SMACOF-Verfahrens sind angelehnt an [BG97] und [JZ04].

Definition 15 (Stress) Für eine Distanzmatrix $\Delta = (\delta_{ij})$ und eine Koordinatenmatrix $X = (x_{ij})$ ist der *Stress* $\sigma_\Delta(X)$ definiert als

$$\sigma_\Delta(X) := \sum_{i<j} (\delta_{ij} - \|x_i - x_j\|)^2$$

[1] Wir betrachten nur reelle Matrizen und übergehen daher die Unterschiede zwischen adjungierten und tranponierten Matrizen.

Wenn nicht alle δ_{ij} bekannt sind, wird diese Definition erweitert zu

$$\sigma_{\Delta,W}(X) := \sum_{i<j} w_{ij}(\delta_{ij} - \|x_i - x_j\|)^2$$

Dabei ist W eine symmetrische Gewichtsmatrix $W = (w_{ij})$ mit

$$\begin{aligned} 0 \leq w_{ij} \leq 1 \quad & \text{falls } \delta_{ij} \text{ bekannt ist und} \\ w_{ij} = 0 \quad & \text{sonst.} \end{aligned}$$

Wenn Δ und W aus dem Zusammenhang klar sind, schreiben wir der Übersichtlichkeit wegen auch einfach $\sigma(X)$.

Die Größe der Einträge w_{ij} gibt dabei an, wie „vertrauenswürdig“ der Wert für δ_{ij} ist. In der Regel wählen wir bei bekanntem δ_{ij} einfach $w_{ij} = 1$, und bei unbekanntem δ_{ij} setzen wir $w_{ij} = 0$

Ziel ist es, diesen Stress bei gegebenen Δ und W zu minimieren. Dazu wird ein iteratives Verfahren verwendet:

1. Wähle zufällige Koordinaten Z
2. Majorisiere den Stress $\sigma(X)$ durch eine Funktion $\tau_Z(X)$ mit

$$\begin{aligned} \sigma(X) &\leq \tau_Z(X) \quad \text{für alle } X \\ \sigma(Z) &= \tau_Z(Z); \end{aligned}$$

3. Suche ein Z' mit $\tau_Z(Z') < \tau_Z(Z)$
4. Breche ab, wenn $\sigma(Z) - \sigma(Z') < \varepsilon$. Ansonsten
5. Setze $Z = Z'$ und fahre fort bei 2.

Das Verfahren wird abgebrochen, wenn die Verbesserung des Stresswertes unter eine vorher festgelegte Schranke $\varepsilon > 0$ fällt.

Im Folgenden schauen wir uns etwas genauer an, wie die Funktion $\tau_Z(X)$ gewählt wird. Dabei gehen wir nicht zu sehr auf die mathematischen Hintergründe ein, sondern beschränken uns auf einen groben Überblick, der es aber dennoch erlaubt, das Verfahren nachzuvollziehen.

Die Stressfunktion $\sigma_\Delta(X)$ lässt sich durch einfaches Ausmultiplizieren schreiben als:

$$\begin{aligned} \sigma_\Delta(X) &= \sum_{i<j} w_{ij}(\delta_{ij} - \|x_i - x_j\|)^2 \\ &= \sum_{i<j} w_{ij}\delta_{ij}^2 \; + \; \sum_{i<j} w_{ij}\|x_i - x_j\|^2 \; - 2 \sum_{i<j} w_{ij}\delta_{ij}\|x_i - x_j\| \end{aligned}$$

Der erste Summand ist unabhängig von X, er spielt für die Optimierung also keine Rolle. Betrachten wir die beiden anderen getrennt voneinander.

Der zweite Summand lässt sich umschreiben zu

$$\sum_{i<j} w_{ij}\|x_i - x_j\|^2 = \sum_{i<j} w_{ij} tr\big(X'(A_{ij})X\big) = \sum_{i<j} tr\big(X'(w_{ij}A_{ij})X\big)$$

Dabei ist A_{ij} eine Matrix mit $a_{ii} = a_{jj} = 1$, $a_{ij} = a_{ji} = -1$ und die übrigen Elemente 0, also von dieser Gestalt:

$$A_{ij} \;:=\; \begin{pmatrix} 0 & \vdots & 0 & \vdots & 0 \\ \cdots & 1 & \cdots & -1 & \cdots \\ 0 & \vdots & 0 & \vdots & 0 \\ \cdots & -1 & \cdots & 1 & \cdots \\ 0 & \vdots & 0 & \vdots & 0 \end{pmatrix}$$

Die Summe kann man in die Spur-Funktion hineinziehen, also

$$\sum_{i<j} w_{ij}\|x_i - x_j\|^2 \;=\; tr\big(X'\sum_{i<j} w_{ij}A_{ij}X\big) \;=:\; tr\big(X'VX\big)$$

Die neu eingeführte Matrix V hat dabei eine gleichmäßige Struktur, für $n = 3$ sieht sie so aus:

$$V \;:=\; \sum_{i<j} w_{ij}A_{ij} \;=\; \begin{pmatrix} w_{12}+w_{13} & -w_{12} & -w_{13} \\ -w_{12} & w_{12}+w_{23} & -w_{23} \\ -w_{13} & -w_{23} & w_{13}+w_{23} \end{pmatrix}$$

Betrachten wir nun den dritten Summanden. Wir erweitern ihn mit einer weiteren Liste von Koordinaten $z_1, \ldots, z_n \in \mathbb{R}^m$ zu

$$\begin{aligned} & -2\sum_{i<j} w_{ij}\delta_{ij}\|x_i - x_j\| \\ = \;& -2\sum_{i<j} w_{ij}\delta_{ij}\frac{\|x_i - x_j\|\|z_i - z_j\|}{\|z_i - z_j\|} \\ = \;& -2\sum_{i<j} w_{ij}\delta_{ij}\frac{\sqrt{\sum_{a=1}^m (x_{ia}-x_{ja})^2}\sqrt{\sum_{a=1}^m (z_{ia}-z_{ja})^2}}{\|z_i - z_j\|}. \end{aligned}$$

Eine Abschätzung mit der Cauchy-Schwarz'schen Ungleichung ergibt

$$\leq \;-\sum_{i<j} w_{ij}\delta_{ij}\frac{\left(\sum_{a=1}^m (x_{ia}-x_{ja})\right)\left(\sum_{a=1}^m (z_{ia}-z_{ja})\right)}{\|z_i - z_j\|}$$

Diese Ungleichung hat auch dann noch Gültigkeit, wenn für einge $i \neq j$ die Koordinaten z_i und z_j gleich sind. Wie oben lässt sich das weiter umformen zu

$$\begin{aligned} &= -\sum_{i<j} w_{ij}\delta_{ij}\frac{tr(X'A_{ij}Z)}{\|z_i - z_j\|} \\ &= -tr\Big(X'\big(\sum_{i<j}\frac{w_{ij}\delta_{ij}}{\|z_i - z_j\|}A_{ij}\big)Z\Big) \\ &=: -tr(X'B_Z Z) \end{aligned}$$

Die Matrix B_Z ist ähnlich regelmäßig aufgebaut wie V, sie hat die Einträge

$$\begin{aligned} b_{i,j} &= \begin{cases} -\frac{w_{ij}\delta_{ij}}{\|z_i - z_j\|} & \text{für } i \neq j \text{ und } \|z_i - z_j\| \neq 0 \\ 0 & \text{für } i \neq j \text{ und } \|z_i - z_j\| = 0 \end{cases} \\ b_{i,i} &= -\sum_{j=1, j\neq i}^{n} b_{ij} \end{aligned}$$

Insgesamt erhalten wir für den Stress also

$$\sigma(X) \leq \sum_{i<j} w_{ij}\delta_{ij}^2 \; + \; tr(X'VX) \; - \; 2tr(X'B_Z Z) \; =: \; \tau_Z(X)$$

Die Gleichheit gilt bei $Z = X$. Wir können eine Koordinatenmatrix mit einem besseren Stresswert bestimmen, indem wir die Funktion $\tau_Z(X)$ mit den gängigen Methoden minimieren.

Bilden wir zunächst die Ableitung die Ableitung von $\tau_Z(X)$

$$\nabla\tau_Z(X) = 2VX - 2B_Z Z$$

Setzen wir die Ableitung $\nabla\tau_Z(X) = 0$ und lösen nach X auf, erhalten wir

$$X = V^+ B_Z Z$$

Zu beachten ist dabei, dass die Matrix V keinen vollen Rang hat, also nicht invertierbar ist. Mit V^+ ist hier die Moore-Penrose-Inverse (auch Pseudo-Inverse) gemeint, die sich auf Grund der besonderen Gestalt der Matrix V auch so berechnen lässt (vgl. [BG97]):

$$V^+ \; = \; (V + \mathbb{1}\mathbb{1}')^{-1} - n^{-2}\mathbb{1}\mathbb{1}'$$

Wenn $w_{ij} = 1$ für alle i, j gilt, dann vereinfacht sich das zu

$$V^+ \;=\; \frac{1}{n} J_n$$

Damit ergibt sich aus dem allgemeinen Schema von Seite 47 eine etwas genauere Beschreibung des Verfahrens wie in Algorithmus 3 aufgeführt.

Algorithmus 3 : MDS mit dem SMACOF-Algorithmus

Eingabe : Eine Distanzmatrix $\Delta \;:=\; (\delta_{ij}) \in \mathbb{R}^{n \times n}$ und Gewichte $w_{ij} \in \mathbb{R}$, $1 \leq i, j \leq n$

Ausgabe : Eine Koordinatenmatrix $Z \in \mathbb{R}^{n \times m}$ mit m-dimensionalen Koordinaten

1 Berechne $V^+ \;:=\; \left(\sum_{i<j} w_{ij} A_{ij}\right)^+ \;=\; (V + \mathbb{1}\mathbb{1}')^{-1} - n^{-2}\mathbb{1}\mathbb{1}'$
2 Setze $k \;:=\; 0$
3 Wähle zufällige Startkoordinaten $Z \;:=\; X_0$
4 **repeat**
5 Setze $k \;:=\; k + 1$
6 Berechne $X_k \;:=\; V^+ B_Z Z$
7 Setze $Z \;:=\; X_k$
8 **until** $\sigma(X_{k-1}) - \sigma(X_k) \leq \varepsilon$;
9 Ausgabe: Z

Die Implementierung ist dann straight-forward. Wenn passende Bibliotheken für die Multiplikation und Invertierung von Matrizen vorhanden sind, ist das nur etwas Tipparbeit.

Interessant ist dann noch der Teil *vor* dem SMACOF-Algorithmus. Nämlich die Frage, wie man die Distanzmatrix für die Eingabe findet, und welche Einflüsse die Wahl auf das Ergebnis hat. Damit beschäftigen wir uns in einem der nächsten Abschnitte. Zunächst aber etwas zur Laufzeit des Verfahrens und ein paar Betrachtungen dazu, wie sich der SMACOF-Algorithmus allgemein auf Graphen verhält.

3.2.1 Laufzeit

Auch wenn der Algorithmus selbst sehr übersichtlich ist, haben es die einzelnen Schritte durchaus in sich, wenn es um die Betrachtung der Laufzeit geht. Den Aufwand schätzen wir ab in Abhängigkeit der Anzahl der anzuordnenden Objekte.

In der Initialisierungsphase müssen einmalig die Matrizen V und V^+ berechnet werden. Die Matrix $n \times n$-Matrix V kann in $\mathcal{O}(n^2)$ aufgebaut werden: Für die n Diagonalelemente werden jeweils $\mathcal{O}(n)$ Additionen benötigt, die übrigen $\mathcal{O}(n^2)$ Einträge sind jeweils in konstanter Zeit berechenbar.

Ausschlaggebend für die Initialisierung ist jedoch die Berechnung der Inversen $(V+\mathbb{1}\mathbb{1}')^{-1}$, was mit den gängigen Methoden $\mathcal{O}(n^3)$ Zeit benötigt[2].

In jedem Iterationsschritt werden zwei Matrixmultiplikationen benötigt. Da die Matrix Z keine quadratische Matrix ist, sondern nur die Dimension $n \times m$ hat, mit konstantem $m = 2$ oder $m = 3$, haben wir hier nur eine Laufzeit in $\mathcal{O}(n^2)$. Die Überprüfung der Abbruchbedingung, also die Berechnung des Stresswertes, benötigt ebenfalls $\mathcal{O}(n^2)$ Zeit.

Es gilt also:

Satz 16 *Bei einer gegebenen Distanzmatrix $\Delta \in \mathbb{R}^{n \times n}$ benötigt die Berechnung von virtuellen Koordinaten mittels des SMACOF-Verfahrens eine Laufzeit von*

$$O(n^3 + k \cdot n^2),$$

wobei k die Anzahl der benötigten Iterationen bis zum Erreichen der Abbruchbedingung bezeichnet.

Die gesamte Laufzeit hängt somit davon ab, wie gut das Verfahren konvergiert, d.h. wie viele Iterationsschritte bis zur gewünschten Genauigkeit benötigt werden. Wir werden beobachten, dass der Worst-Case hier nicht besonders schön aussieht, und dass eine Beschränkung der Iterationen auf einen vorher festgelegten Maximalwert sinnvoll ist. In üblichen Fällen ist aber die Anzahl der benötigten Iterationen durch eine Konstante abschätzbar, und eine Erhöhung der Knoten- und Kantenzahl wirkt sich eher positiv aus. Dazu mehr in Abschnitt 3.3.4.

Für die Anwendung auf Netzwerke bzw. Graphen ist außerdem die Bestimmung der Eingabematrix Δ wichtig, was ebenfalls recht aufwändig werden kann. Es wird sich aber zeigen, dass in der Regel das gesamte Verfahren mit kubischer Laufzeit auskommt.

[2]Wir beschränken uns an dieser Stelle auf die einfache Methode zur Matrixmultiplikation bzw. die grobe Abschätzung für dessen Aufwand, und betrachten nicht die asymptotisch schnelleren Methoden (z.B. aus [STR69])

3.3 MDS auf Graphen

Zur Erinnerung: Wir haben einen Graphen $G = (V, E)$ gegeben, von dem wir annehmen, dass er ein Sensornetzwerk modelliert. Im Idealfall ist G also ein Unit-Disk-Graph oder ein q-Quasi-Unit-Disk-Graph. Zumindest können wir aber annehmen, dass G eine Einbettung besitzt, bei der die Existenz einer Kante zwischen zwei Knoten u und v sehr stark mit einer räumlichen Nähe der beiden Knoten korreliert.

Damit können wir hoffen, dass wir aus der Graphstruktur eine Distanzmatrix ableiten können, die (mehr oder weniger) stark mit den euklidischen Abständen der Knoten in der realen (aber für uns unbekannten) Einbettung korrelieren.

Ein erster Ansatz, der ganz ohne zusätzlichen Aufwand auskommt, wäre, als Distanzmatrix die Adjazenzmatrix des Graphen zu verwenden, also:

Für einen Graphen $G = (V, E)$ mit einer durchnummerierten Knotenmenge $V = \{u_1, \ldots, u_n\}$ wähle als Eingabematrix $\Delta = (\delta_{ij})$ für den SMACOF-Algorithmus die δ_{ij} wie folgt

$$\begin{array}{lll} \delta_{ij} := 0 & & \text{für } i = j \\ \delta_{ij} := 1 & & \text{für } \{u_i, u_j\} \in E \\ \delta_{ij} := \text{undef.} & & \text{sonst} \end{array}$$

Wir nehmen also an, dass wir nur die direkten Nachbarknoten kennen und schätzen die Entfernung zu diesen mit 1 ab.

Um es kurz zu machen: Das funktioniert nicht besonders gut, wie man beispielhaft in Abbildung 3.2 recht deutlich erkennen kann. Die Ursache dafür müssen wir nicht lange suchen. Wie bei jedem iterativen Minimierungsverfahren besteht auch bei SMACOF die Gefahr, in ein *lokales* Minimum hineinzulaufen, das mit dem globalen Minimum (sofern dieses überhaupt existiert) nicht viel gemein hat. Zwar existieren in der Rekonstruktion viele sehr lange Kanten, die den Stress entsprechend erhöhen, aber das Verfahren findet ausgehend von dieser Iteration keine bessere Positionierung mit weniger Stress - es ist in einem lokalen Minimum gefangen.

Weiter veranschaulichen lässt sich das leider nicht. Für einen Graphen $G = (V, E)$ ist die zugehörige Stressfunktion $\sigma : \mathbb{R}^{2|V|} \to \mathbb{R}$ eine sehr hochdimensionale Funktion, die sich nur für einen Graphen mit genau einem Knoten darstellen lässt. Und in diesem Fall ist $\sigma(X) = 0$ für alle $X \in \mathbb{R}^2$, was die Suche nach Koordinaten mit minimalem Stress etwas uninteressant macht.

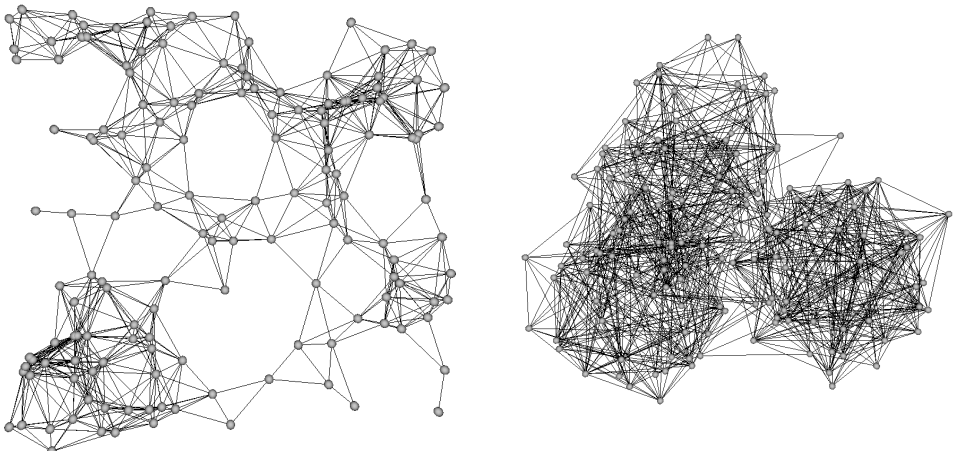

Abbildung 3.2: Ein Unit-Disk-Graph mit 150 Knoten in einer Unit-Disk-Einbettung (links) und die Rekonstruktion durch MDS unter ausschließlicher Verwendung der Nachbarschaftsbeziehungen (rechts)

Nur mit lokalen Nachbarschaftsuntersuchungen kommen wir also nicht weit. Daran ändert sich prinzipiell auch nichts, wenn wir nicht nur die direkte Nachbarschaft, sondern die k-Hop-Nachbarschaft zum Aufbau der Distanzmatrix verwenden.

Erweitert man die Distanzmatrix und ermittelt Distanzinformationen für alle Knotenpaare, dann sehen die Ergebnisse oft deutlich besser aus. In Abbildung 3.3 wurde für den Graphen aus der vorigen Abbildung eine vollständige Distanzmatrix durch Lösung von All-Pairs Shortest Path ermittelt. Die für die Berechnung verwendete Distanzmatrix sieht also so aus:

$$\begin{aligned} \delta_{ij} &:= 0 && \text{für } i = j \\ \delta_{ij} &:= \text{dist}(u_i, u_j) && \text{sonst} \end{aligned}$$

In der MDS-Rekonstruktion erkennt man leicht die Grobstruktur des Ausgangsgraphen. Die vier größeren „Löcher“ sind ebenso erkennbar wie die sehr dichte Struktur in der linken unteren Ecke. Die rekonstruierten Positionen sind natürlich nicht absolut gesehen mit den ursprünglichen fast identisch. Das ganze Netzwerk ist möglicherweise verschoben, gedreht, gespiegelt und/oder gestreckt. Die Darstellung der Abweichung der realen von den virtuellen Koordinaten (Abbildung 3.3 rechts) entstand nach einer manuellen Korrektur der berechneten Werte durch Drehung, Spiegelung und Streckung.

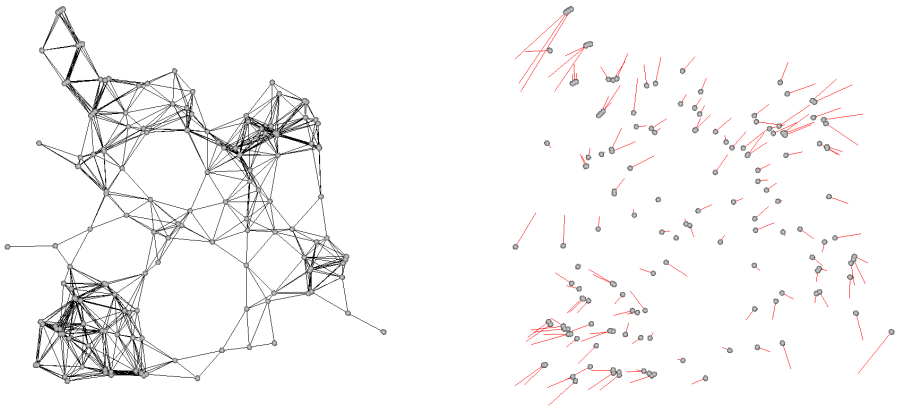

Abbildung 3.3: Die Rekonstruktion des Graphen aus Abbildung 3.2 durch MDS unter Verwendung von All-Pairs Shortest Path (links) und die Abweichung zu den realen Positionen (rechts).

3.3.1 Skalierung und Normierung

Der Stress, den wir in dem Verfahren minimieren, ist (wie schon beschrieben) definiert als

$$\sigma_\Delta(X) \;:=\; \sum_{i<j}(\delta_{ij} - \|x_i - x_j\|)^2$$

Eine offensichtliche Frage ist dann natürlich, wie sich das Verfahren verhält, wenn wir einerseits die Eingabewerte δ_{ij} skalieren, und andererseits die Einbettung des Graphen in unterschiedlich große Gebiete vornehmen wollen. Wenn wir einen Graphen auf einem Gebiet von (beispielsweise) 1000×1000 Einheiten einbetten, dann sind die Werte $\|x_i - x_j\|$ anders, als wenn wir den Graphen in ein deutlich größeres (oder kleineres) Gebiet einbetten.

Betrachten wir zunächst den zweiten Fall, der sich im Grunde selbst erledigt, da die sich das Verfahren in diesem Punkt selbst normiert.

Der wesentliche Teil eines Iterationsschrittes ist die Multiplikation

$$X_k \;:=\; V^+ B_Z Z$$

Die Matrix V und auch die Pseudoinverse V^+ ist von den δ_{ij} und auch von der bisher berechneten Näherung der Koordinaten Z unabhängig, spielt für die Betrachtung hier also keine Rolle.

Die Matrix B_Z enthält (hier nur verkürzt, genaue Definition auf Seite 49) die Einträge

$$b_{i,j} = \frac{w_{ij}\delta_{ij}}{\|z_i - z_j\|}$$

Ein Umskalieren der Koordinatenmatrix Z hat daher keinen Einfluss auf das Verfahren: Ein eventueller Skalierungsfaktor steht in der Multiplikation $B_Z Z$ einmal im Nenner, und einmal im Zähler.

Etwas interessanter ist der Fall der Eingabedistanzen δ_{ij}.

Ein Skalieren der Werte für δ_{ij} mit einem Faktor α zieht sich komplett durch die Rechnung hindurch und sorgt für eine entsprechende Skalierung der Ergebniskoordinaten. Damit skaliert auch der Stresswert $\sigma_\Delta(X)$ entsprechend. Lassen wir das Verfahren bei einem festen ε abbrechen, so werden die Ergebnisse unter Umständen schlechter, wenn die Distanzwerte vorher hochskaliert wurden. Daher ist es sinnvoll, auch die Abbruchbedingung

$$\sigma(X_{k-1}) - \sigma(X_k) \leq \varepsilon$$

entsprechend zu skalieren.

Ein weiterer wichtiger Punkt für die Abbruchbedingung ist die Tatsache, dass je nach Graph der beste erreichbare Stresswert sehr unterschiedlich ausfallen kann.

Bei zufälligen Graphen mit 100 Knoten erreichen die Stresswerte oft eine Größenordnung von 10^3, bei Graphen mit 1000 Knoten liegen die Werte oft auch im Bereich von 10^6. Aber auch bei konstanter Knotenzahl gibt es bei den Graphen sehr unterschiedliche Stresswerte bei der optimalen Einbettung. Die optimale Positionierung eines Weges mit 100 Knoten hat einen Stresswert von Null, bei einem Kreis mit 100 Knoten liegt der optimale Stresswert in der Größenordnung von 10^6. Und das gilt alleine schon für die Verwendung von Hop-Distanzen.

Bei solchen extremen Schwankungen eine feste Abbruchbedingung von $\varepsilon > 0$ zu wählen ist nicht sinnvoll. Stattdessen wird die Abbruchbedingung normiert durch

$$\frac{\sigma(X_{k-1}) - \sigma(X_k)}{\sigma(X_{k-1})} \leq \varepsilon$$

Eine Genauigkeit von $\varepsilon = 10^{-5}$ bis $\varepsilon = 10^{-7}$ hat sich für die im Rahmen dieser Arbeit durchgeführten Tests als brauchbare Schranke erwiesen.

Das für Graphen angepasste SMACOF-Verfahren ist in Algorithmus 4 zusammengefasst.

3.3.2 Typische Effekte bei MDS

Bevor wir uns näher anschauen, wie gut die MDS-Rekonstruktion unter verschiedenen Bedingungen aussieht (und wie man die Ergebnisse auch verbes-

Algorithmus 4 : Multidimensionale Skalierung auf Graphen

Eingabe : Ein Graph $G = (V, E)$ mit durchnummerierten Knoten $V = \{u_1, \ldots, u_n\}$

Ausgabe : Eine Einbettung der Knoten $p : V \to \mathbb{R}^2$, gegeben durch eine Koordinatenmatrix $Z \in \mathbb{R}^{n \times 2}$

1 Bestimme für alle Paare $\{u_i, u_j\}$ die Länge eines kürzesten Weges $\mathrm{kw}(u_i, u_j)$ von u_i nach u_j
2 Bilde die Distanzmatrix $\Delta = (\delta_{ij})$, mit $\delta_{ij} := \mathrm{kw}(u_i, u_j)$
3 Setze $w_{ij} := 1$ für alle $1 \leq i, j \leq n$
4 Berechne $V^+ := \left(\sum_{i<j} w_{ij} A_{ij}\right)^+ = (V + \mathbb{1}\mathbb{1}')^{-1} - n^{-2}\mathbb{1}\mathbb{1}'$
5 Setze $k := 0$
6 Wähle zufällige Startkoordinaten $Z := X_0$
7 **repeat**
8 Setze $k := k + 1$
9 Berechne $X_k := V^+ B_Z Z$
10 Setze $Z := X_k$
11 **until** $\left(\sigma(X_{k-1}) - \sigma(X_k)\right) / \sigma(X_{k-1}) \leq \varepsilon$;
12 Ausgabe: Z

sern kann), wollen wir zunächst die Grenzen dieses Verfahrens diskutieren und einige typische Effekte betrachten, die immer wieder auftreten.

Zuerst einmal benutzt MDS über die Eingabe-Matrix nur die Graphstruktur und hat keinerlei Einblick in die tatsächliche Einbettung des Netzwerks in die Ebene. Das führt unter anderem dazu, dass ein Weg (im graphentheoretischen Sinne) auch als „gerader Weg“ (anschaulich gesprochen) abgebildet wird (vgl. Abbildung 3.4). Denn diese Einbettung ist offensichtlich optimal:

Sind die Knoten auf dem Weg mit $u_1, \ldots, u_n$ durchnummeriert, dann gilt $\delta_{ij} = |i - j|$ für die durch Ermittlung der kürzesten Wege gewonnenen Distanzwerte. Für eine äquidistante Einbettung der Knoten auf der X-Achse

$p : V \to \mathbb{R}^2$ mit $p(u_i) := (i, 0)$ gilt dann

$$\begin{aligned}\sigma(p) &= \sum_{i \neq j} \left(\delta_{ij} - \|p(u_i) - p(u_j)\|\right)^2 \\ &= \sum_{i \neq j} \left(|i-j| - \sqrt{(0-0)^2 + (i-j)^2}\right)^2 \\ &= \sum_{i \neq j} \left(|i-j| - |i-j|\right)^2 \\ &= 0\end{aligned}$$

Eine interessante Randbeobachtung dabei ist, dass es sehr viele Iterationen dauert, bis der Weg ordentlich „glatt gebügelt“ ist. Bei den sonst üblicherweise nötigen 100 Iterationen ist noch recht deutlich eine Krümmung zu erkennen. Erst bei einigen 1000 Iterationen verschwindet diese sichtbare Krümmung und man erhält in sehr guter Näherung das zu erwartende Ergebnis. Obwohl das Ergebnis hier mehr oder weniger offensichtlich ist, konvergiert das Verfahren nur sehr, sehr langsam zu einem Optimum.

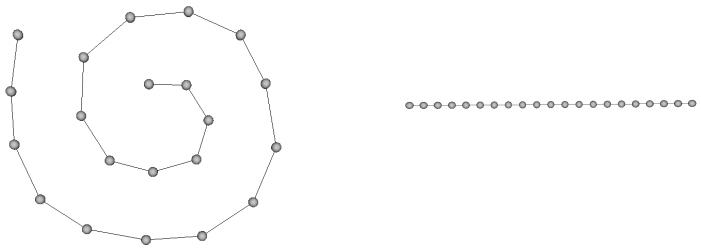

Abbildung 3.4: Ein Weg ist ein Weg. Auch wenn die Knoten tatsächlich als Spirale angeordnet sind (links) wird MDS die Knoten linear anordnen (rechts).

Eine weitere Eigenschaft ist, dass MDS bei Unit-Disk-Graphen oft keine Unit-Disk-Einbettung findet. Das Verfahren eignet sich also nicht für einen heuristischen Test, ob ein Graph ein Unit-Disk-Graph ist.

Tendenziell läuft die Rekonstruktion schon in die Richtung, dass zwei durch eine Kante verbundene Knoten nahe beieinander positioniert werden und andere weiter entfernt voneinander. Trotzdem kann die Minimierung des Stress dazu führen, dass nicht verbundene Knoten etwas zusammengerückt platziert werden, was dann die Unit-Disk-Eigenschaft verletzt. In Abbildung 3.5 ist links ein Unit-Disk-Graph in einer UDG-Einbettung gegeben. Die MDS-Positionierung daneben ist jedoch keine Unit-Disk-Einbettung: Der Winkel,

den die beiden Kanten $\{u, w\}$ und $\{v, w\}$ bilden, ist deutlich kleiner als 60°. Die euklidische Distanz der Knoten u und v in der Einbettung ist also kleiner als ihre Distanz zu Knoten w.

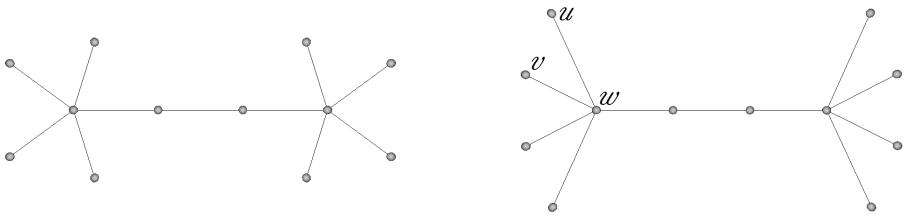

Abbildung 3.5: Für einen Unit-Disk-Graphen wird nicht unbedingt eine Unit-Disk-Einbettung gefunden.

Bei größeren, zufällig erstellten Netzwerken lassen sich auch immer wieder vergleichbare Effekte beobachten. Zum einen werden kleine Gruppen von Knoten, die sehr dicht beieinander liegen, oft etwas gleichmäßiger verteilt, wie man in Abbildung 3.6 (oben) recht gut erkennen kann. Wenn das Netzwerk einige Bereiche hat, in denen sich keine Knoten befinden („Löcher“), oder das Netzwerk keine konvexe Gestalt hat, dann verzerrt sich die Rekonstruktion: Löcher im Inneren des Netzwerkes werden eher kreisförmig abgebildet (ähnlich wie Seifenblasen auf einer Wasseroberfläche), Auswüchse am Rand des Netzwerkes strecken sich radial nach außen. Diese Effekte sind bei zufällig erstellten Graphen mit Unit-Disk-Kanten besonders auffällig. Bei weniger strengen Kantenmodellen (besonders im Waxman-Modell) sind die Effekte auch vorhanden, springen einem aber nicht so direkt ins Auge.

Bei regelmäßig in einem Gitter angeordneten Knoten werden die Positionen sehr gut ermittelt. Nur an den Rändern erkennt man eine kissenförmige Verzerrung.

Sorgt man im Gitter jedoch dafür, dass einige Verbindungen fehlen, indem man einige Knoten leicht verschiebt, dann fällt es schon schwerer, in der Rekonstruktion das ursprüngliche Netzwerk wiederzuerkennen. Durch ein paar wenige fehlende Verbindungen kommt es dazu, dass zwei nahe beieinander liegende Knoten im Graphen eine recht hohe Distanz bekommen, wodurch die Abschätzung der Distanz durch Kürzeste-Wege-Methoden stark von der euklidischen Distanz abweicht. Dadurch verstärken sich die Effekte, die man auch bei zufälligen Graphen beobachten kann.

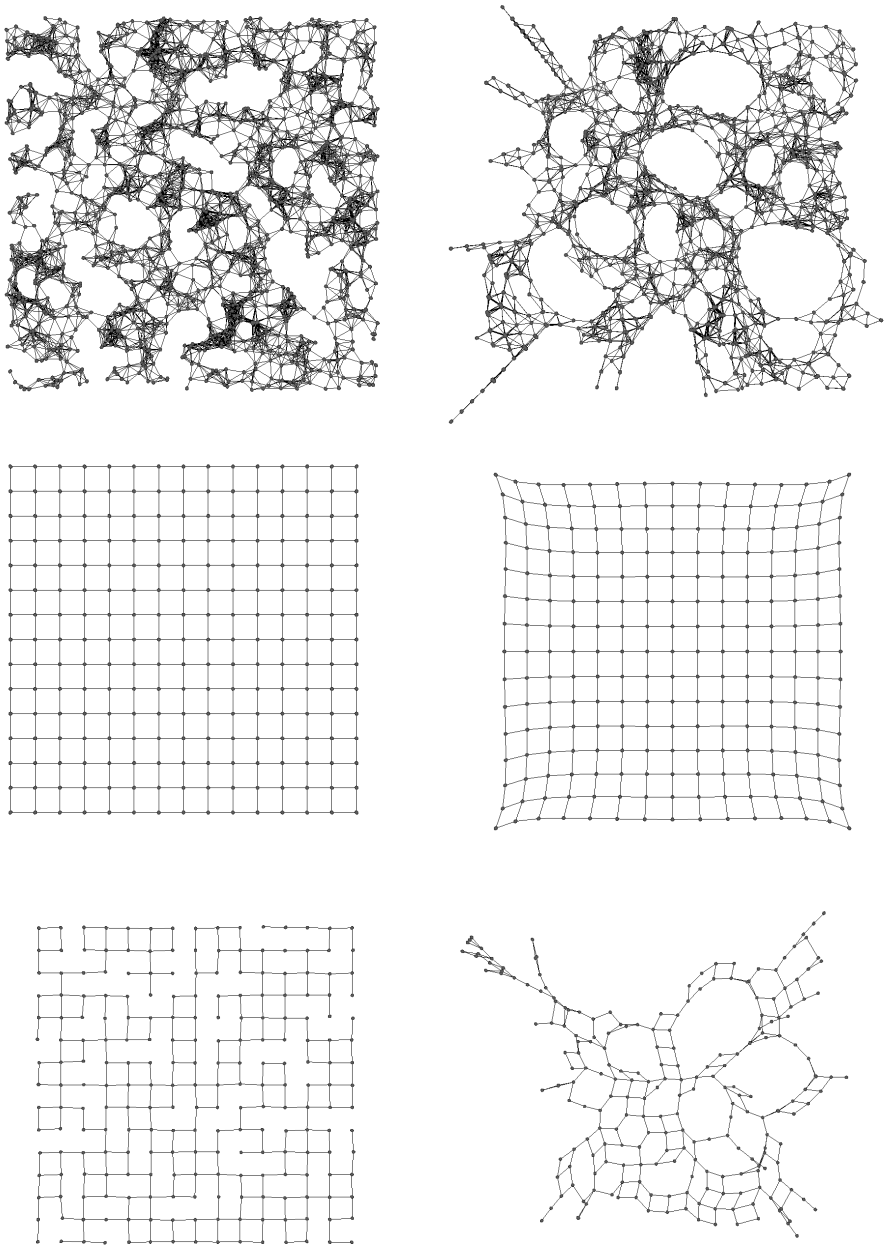

Abbildung 3.6: Weitere typische Effekte bei der Rekonstruktion der Knotenpositionen durch MDS. Links jeweils eine Unit-Disk-Einbettung, rechts das Ergebnis nach Anwendung von MDS.

3.3.3 Bessere Startkonfigurationen

Bei der Betrachtung einiger Durchläufe in einigen Graphen kann man beobachten, dass ausgehend von einer zufällig ungünstigen Positionierung der Knoten bei der Initialisierung das Verfahren gelegentlich sehr viele Iterationen benötigen kann. Recht klar lässt sich das beim Gitter beobachten, wenn die ursprüngliche Verteilung der Knoten nach einigen Iterationen zu einer „Verdrehung“ geführt hat. Aus dieser Lage gibt es zwar in der Regel ein Entkommen, so dass sich das Gitter nach einigen weiteren Iterationen korrekt entfaltet. In anderen Fällen bleibt das Verfahren jedoch in einem lokalen Minimum gefangen, das offensichtlich keine global optimale Lösung darstellt (vgl. Abbildung 3.7).

Es ist also ein naheliegender Gedanke, von einer zufälligen Initialisierung abzusehen, um dadurch solche Effekte zu vermeiden.

Eine Initialisierung aller Koordinaten mit einem festen Wert, z.B. $p(u) = (0,0)$ für alle Knoten $u \in V$ ist nicht sinnvoll. Die Matrix B_Z, die in jeder Iteration mit der aktuellen Koordinatenmatrix multipliziert wird, ist in diesem Fall die Nullmatrix, wodurch das Verfahren sofort abbricht. Auch eine gleichmäßige Initialisierung aller Knoten durch z.B. $p(u_i) := (i,0)$ funktioniert nicht. In diesem Fall hat die initiale Koordinatenmatrix X_0 den Spaltenrang 1, der durch eine Multiplikation mit einer anderen Matrix nicht erhöht werden kann. Während der Iteration entfaltet sich das Netzwerk also nicht in die Ebene, sondern bleibt eindimensional.

Die nächste Idee wäre dann diese: Wähle zwei Knoten $u_1, u_2 \in V$ und setze

$$\begin{aligned} p(u_1) &= (1,0) \\ p(u_2) &= (0,1) \\ p(v) &= (0,0) \text{ für alle } v \text{ mit } v \neq u_1 \text{ und } v \neq u_2 \end{aligned}$$

Die Wahl der beiden Knoten u_1 und u_2 kann zufällig geschehen, oder geschickter dadurch, dass zwei Knoten gewählt werden, die einen möglichst großen Abstand zueinander haben, wie in Algorithmus 5 skizziert.

Bemerkung 17 *Diese Methode liefert nicht den* Durchmesser $D(G)$ *des Graphen (also den größten Abstand zwischen zwei Knoten), sondern nur eine grobe Annäherung an den Durchmesser mit einem Fehlerfaktor von 2, d.h. der Durchmesser des Graphen kann doppelt so groß sein wie die Entfernung der beiden so ermittelten Knoten u_1 und u_2. Für den Beweis dieser Aussage*

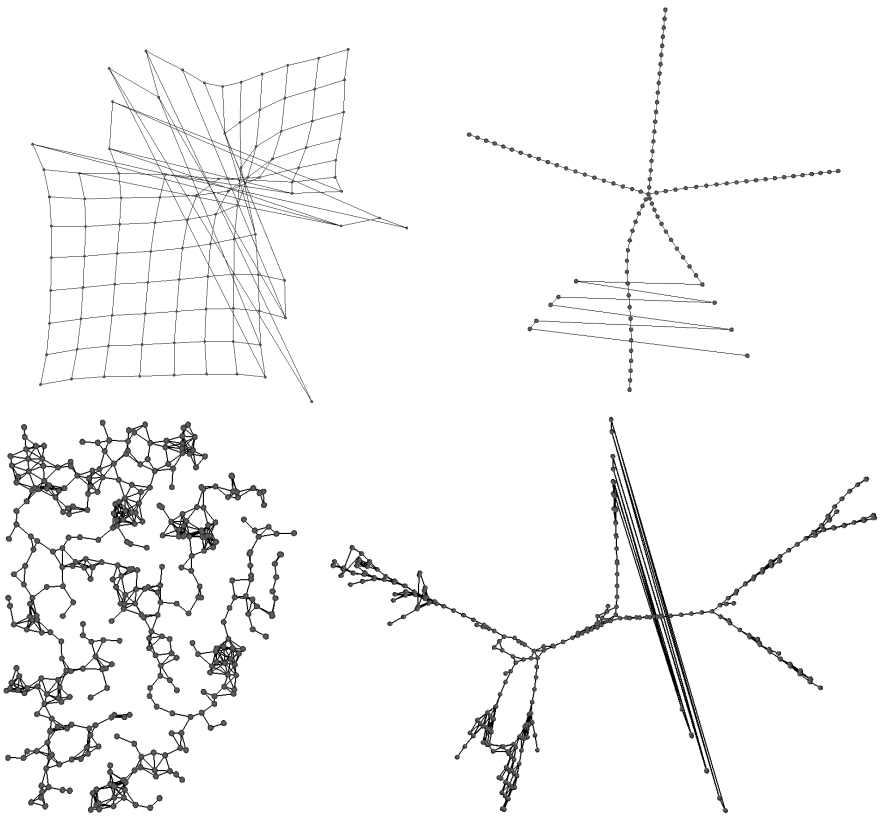

Abbildung 3.7: Einige Beispiele für Effekte nach ungünstigen Startkonfigurationen. Ein „verdrehtes“ Gitter, bei dem recht viele Iterationen bis zum erfolgreichen Abbruch des Verfahrens benötigt werden (links oben), ein Stern vom Grad 5 mit 101 Knoten, der offensichtlich nicht optimal eingebettet ist (rechts oben), und ein Unit-Disk-Graph mit knapp 400 Knoten und sehr schwachem Zusammenhang mit einer durch MDS rekonstruierten nicht optimalen Einbettung, die in einem lokalen Minimum feststeckt.

Algorithmus 5 : Bestimmung zweier Knoten mit großem Abstand.

Eingabe : Ein Graph $G = (V, E, p)$
Ausgabe : Zwei Knoten $u_1, u_2 \in V$ mit großer Distanz $\text{dist}(u_1, u_2)$

1 Wähle einen zufälligen Knoten $w \in V$
2 Wähle einen Knoten $u_1 \in V$ mit $\text{dist}(u_1, w)$ maximal
3 Wähle einen Knoten $u_2 \in V$ mit $\text{dist}(u_2, u_1)$ maximal

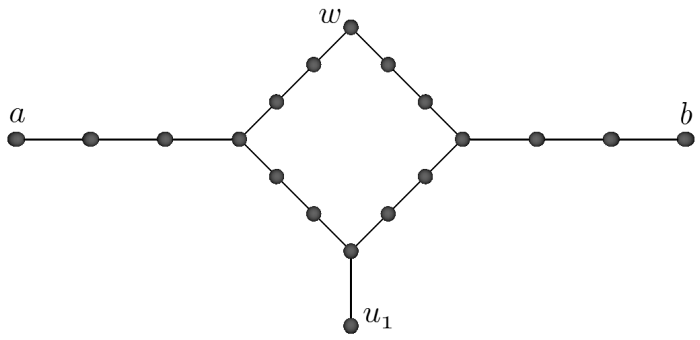

Abbildung 3.8: Mit der hier vorgestellten einfachen Methode findet man nicht immer den Durchmesser des Graphen.

betrachtet man einerseits zwei Knoten $x, y \in V$ *mit* $\text{dist}(x, y) = D(G)$*. Dann gilt:*

$$\begin{aligned} D(G) &= \text{dist}(x, y) \\ &\leq \text{dist}(x, u_1) + \text{dist}(u_1, y) \\ &\leq \text{dist}(u_2, u_1) + \text{dist}(u_1, u_2) \\ &= 2\,\text{dist}(u_1, u_2) \end{aligned}$$

Andererseits ist diese Abschätzung auch scharf, wie die folgende Abbildung 3.8 zeigt. Wird als Ausgangsknoten der Knoten w *gewählt, dann ist der davon am weitesten entfernte Knoten der Knoten* u_1 *mit einer Distanz* $\text{dist}(w, u_1) = 7$*. Als Knoten* u_2 *kommen dann die Knoten* a, b *oder* w *in Frage, alle mit einer Distanz* $\text{dist}(u_1, u_2) = 7$*. Der Durchmesser des Graphen wird aber über die beiden Knoten* a *und* b *definiert, mit* $\text{dist}(a, b) = 12$*. Das Beispiel lässt sich beliebig erweitern, indem der zentrale Kreis auf* $4k$ *Knoten, und entsprechend die beiden Wegstücke zu* a *und* b *auf je* k *Knoten vergrößert werden. Der Fehler zwischen der gefunden Distanz und dem tatsächlichen Durchmesser ist dann* $\frac{2k+1}{4k}$*.*

Etwas aufwändigere Verfahren zur Approximierung des Durchmessers ei-

nes Graphen mit besserer Fehlerabschätzung finden sich beispielsweise in [BFLO06] oder [ACM96].

Diese einfache Methode hat sich aber schon als sehr brauchbar für unsere Zwecke herausgestellt. Besonders da sie oft schon nach sehr wenigen Iterationsschritten die endgültige Auslegung gut erahnen lässt, wenn das Verfahren bei einer zufälligen Initialisierung noch dabei ist, das initiale Chaos zu entwirren. Ein Beispiel dafür ist in Abbildung 3.9 zu sehen.

Bei Graphen mit geringem Zusammenhang wie im unteren Beispiel in Abbildung 3.7 werden deutlich weniger Knoten derart falsch eingebettet, und auch die Zahl der benötigten Iterationen bis zum Erreichen der Abbruchbedingung wird im Mittel geringer (vgl. nächster Abschnitt). Bei dichteren Graphen reduziert sich die mittlere Zahl der Iterationen. Offensichtlich falsch eingebettete Knoten sind dort aber auch bei einer zufälligen Initialisierung nicht mehr (oder nur in wenigen Einzelfällen) zu beobachten.

Die benötigte Anzahl der Iterationen lässt sich in einigen Fällen noch weiter reduzieren, wenn nicht nur zwei Knoten geschickt initialisiert werden, sondern noch ein paar weitere. Bei zweidimensionalen Netzwerken in einem rechteckigen Gebiet bietet es sich beispielsweise an, vier Eckknoten auf $(1, 1)$, $(1, -1)$, $(-1, -1)$ und $(-1, 1)$ zu initialisieren, und den Rest mittig auf $(0, 0)$ zu legen. Diese Zusatzannahmen können in anderen Fällen aber auch negative Auswirkungen haben. Wie diese anschaulichen Eckknoten eines Netzwerkes gefunden werden können, werden wir im Kapitel zur Rand- und Locherkennung näher beschreiben - denn an der Stelle werden wir diese Knoten auch benötigen.

3.3.4 Konvergenzverhalten des Verfahrens

Für die Laufzeit eines iterativen Verfahrens ist nicht nur die Laufzeit eines Iterationsschrittes wichtig, sondern auch und vor allem die Anzahl der notwendigen Iterationen, um zu einem „guten“ Ergebnis zu kommen. Hierfür ist die *Konvergenzgeschwindigkeit* des Verfahrens interessant.

Allgemein wird oft unterschieden zwischen sublinearer, linearer, superlinearer Konvergenz und Konvergenz der Ordnung p.

Definition 18 Sei (x_n) eine konvergente Zahlenfolge und x^* der Grenzwert, gegen den die Folge konvergiert.

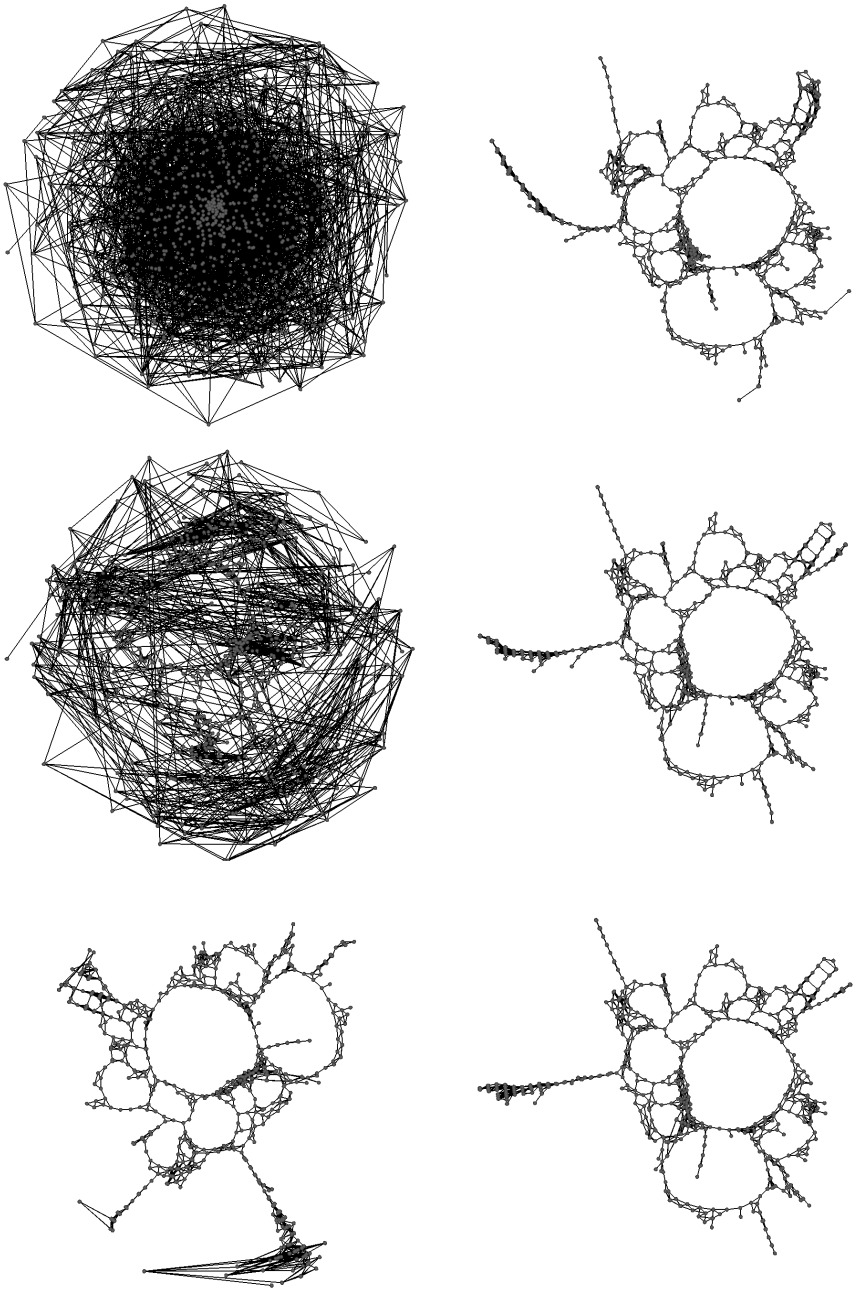

Abbildung 3.9: Unterschiedliche Entwicklung der Einbettung mit einer zufälligen Initialisierung der Knoten (links) und einer gezielten Initialisierung zweier Knoten (rechts). Gezeigt ist jeweils der Stand nach 10, 25 und 50 Iterationen.

- Die Folge (x_n) konvergiert *linear*, wenn es ein $0 < c < 1$ gibt mit

$$\lim_{n\to\infty} \frac{|x_{n+1} - x^*|}{|x_n - x^*|} = c$$

- Die Folge (x_n) konvergiert *sublinear*, wenn $c = 1$.
- Die Folge (x_n) konvergiert *superlinear*, wenn $c = 0$.
- Die Folge (x_n) konvergiert *mit Ordnung* p, wenn es ein $p > 1$ und $c > 0$ gibt mit

$$\lim_{n\to\infty} \frac{|x_{n+1} - x^*|}{|x_n - x^*|^p} = c$$

Das Problem bei dieser Definition ist für unseren Fall, dass wir den Grenzwert - also den minimalen Stress - nicht kennen. Wir müssen uns mit dem jeweils letzten errechneten Stresswert begnügen, bevor das Verfahren abgebrochen wird.

Beobachtungen

Eine Beweisführung, dass das SMACOF-Verfahren ein „schönes“ Konvergenzverhalten (linear, superlinear oder gar der Ordnung $p > 1$) hat, ist leider nicht möglich. Bei einigen Graphen zeigt sich sehr klar ein sublineares Konvergenzverhalten - beispielsweise bei einem Weg und einem vollständigen Graphen. Bei einigen anderen regelmäßig konstruierten Graphen wie Kreisen, Gittern und Sternen zeigt sich recht deutlich eine mehr oder weniger gute lineare Konvergenz, ebenso bei vielen „zufälligen“ Graphen.

In Abbildung 3.10 ist das Konvergenzverhalten bei einem Durchlauf von MDS für einige Graphen abgebildet. Der hier beispielhaft gezeigte zufällige Unit-Disk-Graph hat einen durchschnittlichen Knotengrad von ungefähr $\deg(G) \sim 10$.

Auf der Y-Achse ist der Wert

$$\frac{|\sigma(X_{k+1}) - \sigma(X^*)|}{|\sigma(X_k) - \sigma(X^*)|}$$

abgetragen, wobei $\sigma(X^*)$ der Stresswert ist, bei dem das Verfahren abgebrochen wurde, da entweder die willkürlich festgelegte Zahl von 1000 Iterationen erreicht wurde, oder weitere Berechnungen wegen einer Annäherung an die Maschinengenauigkeit keinen Sinn mehr ergaben. Die Werte aus den letzten paar Iterationen, bei denen der Quotient durch Rundungs-Ungenauigkeiten

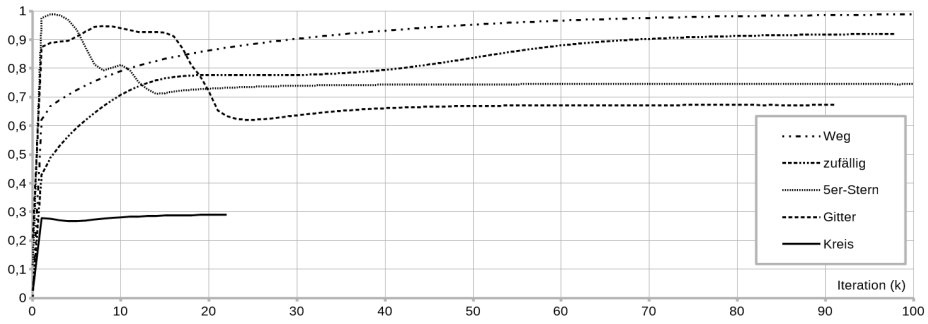

Abbildung 3.10: Konvergenzverhalten bei einigen ausgewählten Graphen mit je 100 Knoten

ein chaotisches Verhalten zeigt, sind abgeschnitten. Kleine Werte bedeuten eine hohe Verbesserung pro Iterationsschritt, kleine Werte sind also besser.

Abgesehen vom Weg (und auch dem vollständigen Graphen, nicht im Bild) ist das Verfahren nach höchstens knapp über 100 Iterationen beendet. Beim Weg ist auch nach 1000 Iterationen gerade mal eine Genauigkeit von 10^{-3} erreicht. Bis zu der bei vielen Graphen sinnvollen Genauigkeit von 10^{-7} werden mehrere 10.000 Iterationen benötigt. An der Erfolgsquote für das Greedy-Routing, die uns hier hauptsächlich interessiert, ändern diese vielen zusätzlichen Iterationen natürlich nichts mehr.

Wir sind ja nicht so sehr daran interessiert, wie sich das Verfahren im Unendlichen verhält, sondern eher daran, wie schnell das Verfahren an einen Punkt gelangt, an dem eine ausreichend hohe Genauigkeit erreicht wird. Diese ist dann erreicht, wenn sich die Positionen der Knoten nur noch um kleine Bruchteile ihrer Senderadien verändern. Eine deutliche Verbesserung der Einbettung oder eine Erhöhung der Erfolgschancen beim Greedy-Routing ist dann nicht mehr zu erwarten.

In Abbildung 3.11 ist die Zahl der benötigten Iterationen für MDS auf jeweils 1000 zufälligen Unit-Disk-Graphen mit verschiedenen Knotenzahlen n und konstantem Sendereichweite r bzw. bei konstantem n und variablem r zu sehen. Der hauptsächlich relevante Faktor scheint hier also nicht die Knotenzahl zu sein, sondern vielmehr der durchschnittliche Knotengrad bzw. die daraus bei dieser Form der Auslegung resultierende grobe Graphstruktur. In den Beispielkonfigurationen in Abbildung 3.11 liegt der durchschnittliche Knotengrad bei ungefähr $\deg(G) = 5$ bis $\deg(G) = 9$.

Für Graphen mit geringem Knotengrad werden im Schnitt etwas mehr Itera-

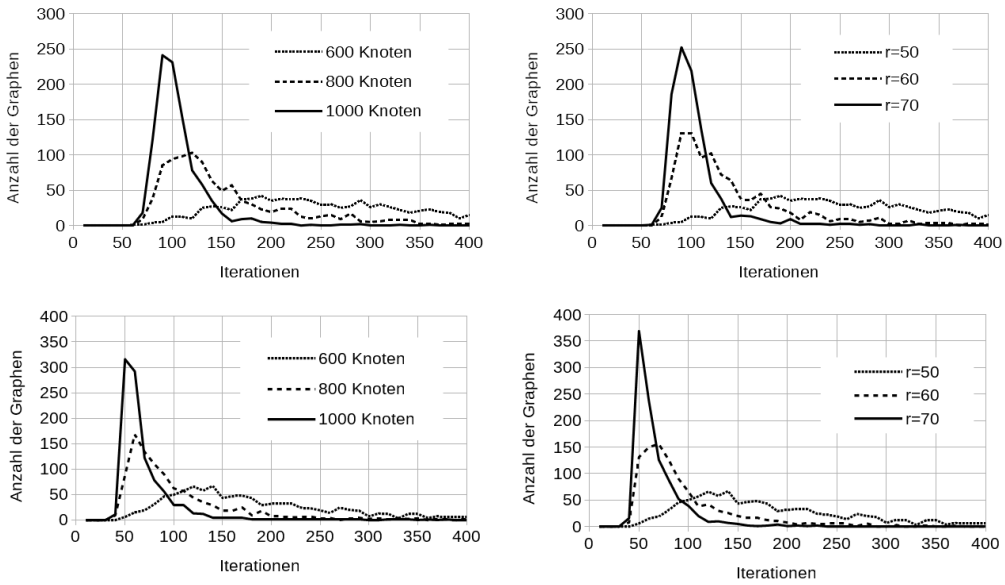

Abbildung 3.11: Verteilung der benötigten Iterationen bei je 1000 zufälligen Unit-Disk-Graphen auf einem Gebiet der Größe 1000×1000 mit variabler Knotenzahl und festem Senderadius $r = 50$ (links), und bei fester Knotenzahl $n = 600$ und variablem Senderadius. Oben mit zufälliger Initialisierung, unten mit einer gezielten Initialisierung zweier Knoten. Zum Glätten der Kurven wurden die benötigten Iterationen auf volle Zehner aufgerundet.

tionen benötigt, und die Streuweite der benötigten Schritte ist deutlich größer als bei Graphen mit höherem Knotengrad. Bei den Graphen mit vielen Knoten, die wir in der experimentellen Auswertung verwenden, ist die Spitze im Bereich von ca. 50 bzw. ca. 100 Iterationen noch ausgeprägter. Dieser Effekt setzt sich aber nicht beliebig weit fort. Bei Graphen mit $n = 600$ Knoten wird diese klar ausgeprägte Spitze ab einem durchschnittlichen Knotengrad von ungefähr $60 - 70$ wieder weniger stark ausgeprägt, die Streuweite wird größer und die im Mittel benötigten Iterationen wachsen deutlich an. Diese Entwicklung deckt sich mit der Beobachtung, dass das Konvergenzverhalten bei vollständigen Graphen ebenso schlecht ist wie bei Wegen.

3.4 Variationen des Verfahrens

Das bisher beschriebene Verfahren kann man auf verschiedene Arten modifizieren, um bessere Ergebnisse zu erzielen. Was im Einzelnen „besser“ bedeu-

tet, hängt von der Anwendung ab. Möchte man möglichst gut die Einbettung des Graphen rekonstruieren, so sind andere Dinge sinnvoll als für das Ziel, eine möglichst hohe Erfolgsquote bei Greedy-Routing zu erreichen.

3.4.1 Bewertung einer Einbettung

Wie bereits früher in Abbildung 3.6 zu erkennen, ist ein typischer Effekt der Multidimensionalen Skalierung, dass Löcher im Graphen (anschaulich gesprochen) in der Auslegung rundlich erscheinen - ganz unabhängig davon, welche Form sie in der ursprünglichen Unit-Disk-Einbettung haben. Die dadurch verursachten Verzerrungen verfälschen natürlich die Positionierungen der Knoten und sind unerwünscht, wenn man die tatsächlichen Koordinaten möglichst gut rekonstruieren will.

Um die Güte einer Einbettung praktikabel zu bewerten, benötigen wir zuerst eine Methode dafür bzw. ein Maß für die Güte einer Einbettung. Dabei liegt die Betonung auch auf *praktikabel*. Wünschenswert wäre - zumindest bei Unit-Disk-Graphen - wenn wir eine rekonstruierte Einbettung mit allen gültigen Unit-Disk-Einbettungen vergleichen könnten und darüber die Güte bzw. den Fehler der gefundenen Einbettung angeben könnten. Da wir aber nicht einmal *eine* gültige UDG-Einbettung effizient finden können, sind *alle* gültigen erst recht nicht möglich.

Für die Bewertung einer Einbettung müssen wir also auf das zurückgreifen, was wir haben. Und das ist die vorhandene Einbettung des Graphen. Im Einzelfall kann das zu unschönen Ergebnissen führen - die Betrachtungen zu Wegen haben das ja bereits gezeigt. Bei zufälligen Graphen treten solche Extremfälle aber selten auf. Die meisten sinnvollen Einbettungen haben eine gut erkennbare Ähnlichkeit.

Ein erster Ansatz für die Quantifizierung der Ähnlichkeit zweier Einbettungen wäre der folgende.

Für einen Unit-Disk-Graphen $G = (V, E)$ und zwei Einbettungen $p, p' : V \to \mathbb{R}^2$ des Graphen in die Ebene betrachten wir die Summe der Differenzquadrate

$$\tilde{\Delta}(p, p') \;:=\; \sum_{u \in V} \|p(u) - p'(u)\|^2$$

Die Schwächen dieser einfachen Definition sind offensichtlich. Nehmen wir

zwei Einbettungen $p, p' : V \to \mathbb{R}^2$ mit

$$p'(u) \;:=\; p(u) + (\delta_x, \delta_y)' \text{ für alle } u \in V,$$

dann wird die Abweichung von p bzgl. p' beliebig groß, wenn wir δ_x und δ_y groß genug wählen. Trotzdem würde man anschaulich gesprochen sicherlich nicht behaupten wollen, dass die beiden Einbettungen besonders verschieden sind - sie sind ja nur verschoben.

Daher definieren wir Ähnlichkeitstransformationen, die an der Güte einer Einbettung nichts verändern sollen

- Verschiebung um Vektor $t = (\delta_x, \delta_y)' \in \mathbb{R}^2$:

$$p'(u) \;:=\; p(u) + t$$

- zentrische Streckung mit Faktor $c \in \mathbb{R}^+$:

$$p'(u) \;:=\; c \cdot p(u)$$

- Spiegelung an einer Ursprungsgeraden mit Neigungswinkel α:

$$p'(u) \;:=\; \begin{pmatrix} \cos 2\alpha & \sin 2\alpha \\ \sin 2\alpha & -\cos 2\alpha \end{pmatrix} p(u)$$

- Rotation um Winkel α:

$$p'(u) \;:=\; \begin{pmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{pmatrix} p(u)$$

Die Matrizen in der obigen Gestalt, die für eine Spiegelung bzw. Drehung sorgen, heißen *Spiegelungs-* bzw. *Rotationsmatrizen*.

Andere Transformationen wie z.B. Scherungen oder allgemeine Streckungen (d.h. unterschiedliche Skalierung in x bzw. y-Richtung) könnten zwar in einigen Fällen durchaus sinnvoll sein, in der Allgemeinheit jedoch nicht.

Definition 19 (Ähnlichkeit zweier Einbettungen) Sei $G = (V, E)$ ein Graph und $p, p' : V \to \mathbb{R}^2$ zwei Einbettungen des Graphen in die Ebene. Die *Ähnlichkeit* der beiden Abbildungen ist definiert als

$$\Delta(p, p') \;:=\; \frac{1}{|V|} \min_{c,R,S,q,t} \sum_{u \in V} \Big\| p(u) - \big(c \cdot R \cdot S^q \cdot p'(u) + t\big) \Big\|^2$$

mit $c \in \mathbb{R}^+$, R eine Rotationsmatrix, S eine Spiegelungsmatrix, $q \in \{0, 1\}$ und $t \in \mathbb{R}^2$.

Für die Bewertung der gefundenen Einbettung bestimmen wir also zunächst eine Ähnlichkeitstransformation, die die rekonstruierte Einbettung möglichst gut auf die tatsächliche abbildet. Dazu verwenden wir eine leicht modifizierte Variante des Verfahrens aus [Ume91].

Dort wird für zwei Punktmengen $X = \{x_1, \dots x_n\}$ und $Y = \{y_1, \dots y_n\}$ eine Rotationsmatrix R, ein Verschiebungsvektor t und ein Streckfaktor c ermittelt, so dass

$$\sum_{i=1}^{n} \|y_i - (cRx_i + t)\|^2$$

minimal wird.

Hierfür werden die Punktmengen zuerst um den Koordinatenursprung zentriert und dann die Kovarianzmatrix Σ_{xy} gebildet.

$$\begin{aligned}
\mu_x &= \frac{1}{n}\sum_{i=1}^{n} x_i \\
\mu_y &= \frac{1}{n}\sum_{i=1}^{n} y_i \\
\Sigma_{xy} &= \frac{1}{n}\sum_{i=1}^{n} (y_i - \mu_y)(x_i - \mu_x)
\end{aligned}$$

Berechnet man die Singulärwertzerlegung von $\Sigma_{xy} = UDV^T$, dann ist die optimale Ähnlichkeitsabbildung gegeben durch

$$\begin{aligned}
R &= UV^T \\
t &= \mu_y - cR\mu_x \\
c &= \frac{1}{\sigma_x^2} tr(DS)
\end{aligned}$$

mit

$$\sigma_x^2 = \frac{1}{n}\sum_{i=1}^{n} \|x_i - \mu_x\|^2$$

Wenn die Determinante von Σ_{xy} negativ ist, dann ist die so berechnete Matrix R jedoch keine Rotationsmatrix, sondern eine Spiegelungsmatrix, was für die Problemstellung in [Ume91] ausdrücklich *nicht* erwünscht ist. In diesem

Fall wird dort die Berechnung von R modifiziert, um in jedem Fall nur eine Drehung, und keine Spiegelung zu erhalten.

Für unsere Zwecke wollen wir aber eine Spiegelung *und* eine Drehung erlauben. Falls also $\mathrm{Det}(\Sigma_{xy}) < 0$, dann führen wir die Berechnungen ein weiteres Mal durch mit den gespiegelten Punkten $X' = \{x'_1, \ldots, x'_n\}$, $x'_i = cRx_i + t$ und erhalten so das für unseren Fall sinnvolle Ergebnis.

Da die Matrix Σ_{xy} nur eine 2×2 Matrix ist, ist auch die an und für sich aufwändige Singulärwertzerlegung in konstanter Zeit möglich. Der Aufwand für die Bewertung einer Einbettung liegt also in $\mathcal{O}(n)$. Damit haben wir eine praktikable Möglichkeit, die Ähnlichkeit zweier verschiedener Einbettungen zu bewerten.

3.4.2 Zwei Durchläufe von MDS

Eine einfache Verbesserung der Einbettung lässt sich erreichen, wenn die Multidimensionale Skalierung zweimal durchgeführt wird.

In einem ersten Durchlauf werden wie bisher die Positionen mit Hilfe von All-Pairs Shortest Path bestimmt (vgl. Algorithmus 4), mit den bereits bekannten Effekten.

In einem zweiten Durchlauf wird eine neue Distanzmatrix aufgebaut, die nur die Nachbarschaftsbeziehungen der Knoten berücksichtigt.

Das ursprünglich bei der ausschließlichen Verwendung der Adjazenzmatrix beobachtete Durcheinander in der Rekonstruktion ist hier nicht mehr zu erwarten. Die Ursache dafür, nämlich die zufällige initiale Anordnung der Knoten, ist hier ja nicht mehr gegeben. Tatsächlich wird durch diese einfache Methode die Einbettung der Knoten bei zufälligen Graphen oft verbessert.

In Abbildung 3.12 sind die nachbehandelten Graphen zu sehen, die wir schon früher betrachtet haben. Beim zufälligen Graph ist das grobe Layout deutlich besser zu erkennen. Ebenso ist auffällig, dass die Knoten lokal gleichmäßiger positioniert werden - kaum verwunderlich, wenn alle benachbarten Knoten eine angenommene Distanz von 1 haben und keine Information darüber vorliegt, ob sie nun eine Distanz von 1 oder nur 0.1 haben.

Die Kisseneffekte beim Gitter (nicht in der Abbildung) verschwinden erwartungsgemäß nahezu vollständig. Sie wären in der Abbildung nicht mehr zu erkennen und sind daher ausgelassen.

Das gestörte Gitter verändert sich hingegen kaum sichtbar. Allerdings haben

Algorithmus 6 : Zweiter Durchlauf für MDS

Eingabe : Ein Graph $G = (V, E)$ mit durchnummerierten Knoten $V = \{u_1, \ldots, u_n\}$
Eine Einbettung der Knoten $p : V \to \mathbb{R}^2$, gegeben durch eine Koordinatenmatrix $X_0 \in \mathbb{R}^{n\times 2}$

Ausgabe : Eine neue Einbettung der Knoten $p : V \to \mathbb{R}^2$, gegeben durch eine Koordinatenmatrix $Z \in \mathbb{R}^{n\times 2}$

1 Bilde die Distanzmatrix $\Delta = (\delta_{ij})$, mit

$$\delta_{ij} := \begin{cases} 1 & \text{falls } \{u_i, u_j\} \in E \\ 0 & \text{sonst} \end{cases}$$

Setze $w_{ij} := \delta_{ij}$ für alle $1 \leq i, j \leq n$

2 Berechne $V^+ := \left(\sum_{i<j} w_{ij} A_{ij}\right)^+ = (V + \mathbb{1}\mathbb{1}')^{-1} - n^{-2}\mathbb{1}\mathbb{1}'$

3 Setze $k := 0$

4 Setze $Z := X_0$

5 **repeat**

6 Setze $k := k + 1$

7 Berechne $X_k := V^+ B_Z Z$

8 Setze $Z := X_k$

9 **until** $\sigma(X_{k-1}) - \sigma(X_k) \leq \varepsilon$;

10 Ausgabe: Z

diese sehr geringen Änderungen einen recht starken Einfluss auf die Erfolgsquote beim Greedy-Routing, wie wir im nächsten Abschnitt sehen werden. Bei einigen Einbettungen scheint das Greedy-Routing sehr empfindlich auf kleine Änderungen zu reagieren.

Der visuelle Eindruck bestätigt sich in den Zahlen. In Tabelle 3.2 sind die Werte für die drei Graphen zusammengefasst, jeweils nach dem ersten und nach dem zweiten Durchlauf. Neben dem Mittelwert, über den wir die Ähnlichkeit ja definiert haben, sind auch Median und maximaler Fehler notiert.

Die Werte in der Tabelle sind dabei auf die maximalen Kantenlängen normiert. Ein Wert von 1 bedeutet also, dass die geschätzte Position des Knotens um den Senderadius des Knotens von seiner tatsächlichen abweicht. Das ist besonders beim Vergleich der beiden zufälligen Graphen zu beachten, denn diese wurden gebildet auf der gleichen eingebetteten Knotenmenge, aber mit den auch sonst verwendeten unterschiedlichen Senderadien $r = 50$ für Unit-

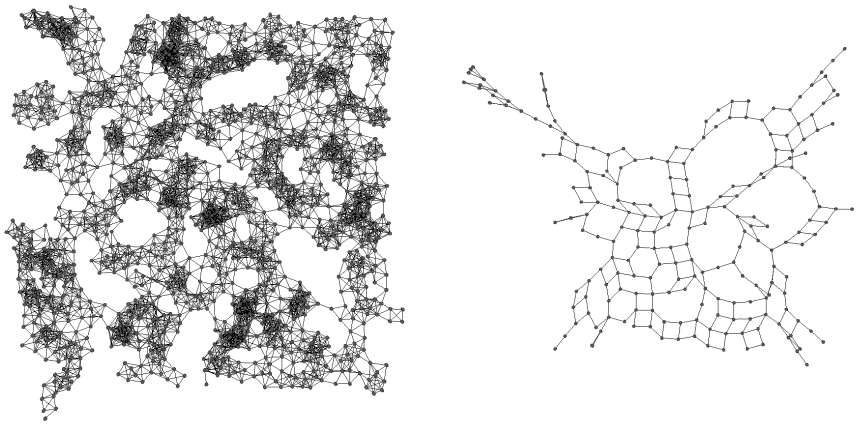

Abbildung 3.12: Einbettung des zufälligen Graphen und des gestörten Gitters aus Abbildung 3.6 nach einem weiteren Durchlauf von MDS, bei dem nur Informationen über benachbarte Knoten verwendet wurden.

Disk-Graphen und $r = 93$ im Waxman-Modell, um einen ähnlich hohen Knotengrad zu erreichen.

Der Fehler wird beim zufälligen Graphen deutlich kleiner und beim Gitter verschwindet er fast vollständig. Der Fehler beim gestörten Gitter ändert sich nur unwesentlich und wird sogar etwas größer.

3.4.3 Kantengewichte

Eine andere mögliche Methode zur Verbesserung der Einbettungen ist die Verwendung von Kantengewichten in der Distanzmatrix. Im Idealfall kennt hier ein Knoten nicht nur seine Nachbarn, sondern auch die Distanz zu ihnen. Einige theoretisch machbare, aber in der Praxis oft sehr ungenaue Methoden wurden in der Einleitung ja aufgeführt. Eine exakte Distanzmessung ist vermutlich unrealistisch, eine diskrete Abschätzung der Distanz in zwei, vier oder auch einigen weiteren Abstufungen könnte aber durchaus möglich sein. Für den zufälligen Graphen sind in Tabelle 3.2 auch die Fehler für unterschiedliche Kantengewichte eingetragen. Für exakte euklidische Distanzen und diskrete Distanzen mit zwei, vier und acht Abstufungen, jeweils auch nach einem zweiten Durchlauf nach obiger Methode.

Definition 20 Sei $G = (V, E, p)$ ein eingebetteter Graph, $u, v \in V$ zwei Knoten und $e = \{u, v\} \in E$ eine Kante.

	Mittelwert	Median	Maximum
zufälliger Graph (UDG)			
ω^H (Hop)	0.28 (0.16)	0.22 (0.13)	2.23 (1.39)
ω_2^D (zwei Abstufungen)	0.19 (0.11)	0.15 (0.096)	1.48 (1.02)
ω_4^D (vier Abstufungen)	0.19 (0.12)	0.15 (0.099)	1.65 (1.09)
ω_8^D (acht Abstufungen)	0.15 (0.099)	0.11 (0.074)	1.42 (1.09)
ω^E (euklidisch)	0.14 (0.031)	0.10 (0.015)	1.40 (1.06)
zufälliger Graph (Waxman)			
ω^H (Hop)	0.38 (0.65)	0.27 (0.62)	1.46 (1.88)
ω_2^D (zwei Abstufungen)	0.20 (0.19)	0.18 (0.18)	1.14 (1.40)
ω_4^D (vier Abstufungen)	0.15 (0.08)	0.13 (0.07)	0.97 (0.82)
ω_8^D (acht Abstufungen)	0.11 (0.04)	0.10 (0.038)	0.93 (1.23)
ω^E (euklidisch)	0.091 (0.0064)	0.071 (0.0029)	0.95 (1.20)
Gitter (Hop)	0.10 (0.00013)	0.056 (0.000087)	0.35 (0.00045)
gestörtes Gitter (Hop)	1.04 (1.11)	0.85 (0.93)	5.58 (5.83)

Tabelle 3.2: Ähnlichkeit der rekonstruierten Einbettung zur tatsächlichen Einbettung bei einigen Graphen. In Klammern jeweils der Wert nach einem zweiten Durchlauf unter alleiniger Verwendung der Nachbarschaften.

Die *Hop-Kantengewichtsfunktion* $\omega^H : E \to \mathbb{R}$ ist definiert durch

$$\omega^H(\{u, v\}) \;:=\; 1$$

Die *euklidische Kantengewichtsfunktion* $\omega^E : E \to \mathbb{R}$ ist definiert durch

$$\omega^E(\{u, v\}) \;:=\; \|p(u) - p(v)\|$$

Sei weiter $k > 1$ eine natürliche Zahl und d_{max} eine maximale Distanz. Die *diskrete Kantengewichtsfunktion* $\omega_k^D : E \to \mathbb{R}$ ist definiert durch

$$\omega_k^D(\{u, v\}) \;:=\; \left\lceil k \cdot \frac{\|p(u) - p(v)\|}{d_{max}} \right\rceil$$

Bei der Betrachtung der Tabelle fällt auf, dass die Positionierung mit Hilfe der euklidischen Distanzwerte deutlich besser ist und im Mittel (und Median) nach einem zweiten Durchlauf sehr exakt ist.

Bei den diskreten Kantengewichten ist ebenfalls eine deutliche Verbesserung gegenüber den reinen Hop-Distanzen zu beobachten, allerdings spielt die Anzahl der Abstufungen eine eher untergeordnete Rolle. Zumindest in diesem

speziellen zufälligen Graphen. Werden die Abstufungen feiner, dann werden die Ergebnisse erwartungsgemäß besser und nähern sich den Werten der Einbettung an, die mit Hilfe der euklidischen Distanzen ermittelt wurde.

Eine interessante Beobachtung ist, dass die Einbettung des Graphen mit Waxman-Kanten über Hopdistanzen einen größeren Fehler aufweist als die des UDG, jedoch kann hier noch stärker von der Verwendung der euklidischen Distanzen profitiert werden.

Beim (gestörten) Gitter sind unterschiedliche Kantengewichte uninteressant, da alle Kanten (fast) die gleiche Kantenlänge haben. Andere Kantengewichte würden die Einbettung in diesem Fall nur skalieren, was im Zuge des Vergleichs mit der originalen Einbettung wieder rückgängig gemacht wird.

3.4.4 Höhere Dimensionen

Wenn wir die rekonstruierten Koordinaten nur zum Routen verwenden wollen, und nicht in erster Linie an den tatsächlichen Positionen interessiert sind, dann ist die Frage berechtigt, warum wir es bei zwei Dimensionen belassen sollen. Die Multidimensionale Skalierung ermöglicht es, aus der Eingabematrix Δ nicht nur zweidimensionale Koordinaten zu berechnen, sondern auch höherdimensionale.

Als Motivation ein einfaches Beispiel - ein Sterngraph mit fünf Strahlen. In der zweidimensionalen Rekonstruktion gelangt man mit Greedy-Routing nicht vom Ende eines Strahls zum Ende eines benachbarten Strahls. Der Winkel, den zwei Strahlen darin bilden, ist dafür zu spitz. In der dreidimensionalen Rekonstruktion werden die Winkel groß genug, um mit Greedy-Routing eine 100%ige Erfolgsquote zu erreichen.

Auch beim Gitter bekommen wir in einer dreidimensionalen Einbettung eine 100%ige Erfolgsquote - zumindest in der hier gewählten Größe. Die kissenförmige Verzerrung ist an den Ecken weniger stark ausgeprägt (auch wenn das in der Abbildung bei der vorderen Ecke so aussieht), stattdessen bekommt das Gitter eine Sattelform.

Das Hinzufügen einer weiteren Dimension kann also die Erfolgsquote für das Routen positiv beeinflussen.

In vielen Fällen sind bei zweidimensional ausgelegten Netzen nach der Multidimensionalen Skalierung in drei Dimensionen die ursprünglichen zwei Dimensionen weiterhin erkennbar. Anschaulich gesprochen: Wenn das originale Netzwerk auf einem Blatt Papier eingebettet ist, dann sieht das Netzwerk

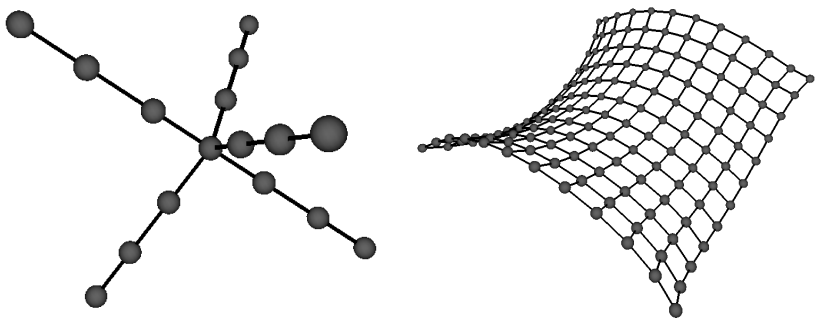

Abbildung 3.13: Optimale 3D-Einbettung eines Sterngraphen mit fünf Strahlen (links) und eines Gitters (rechts).

mit den rekonstruierten 3D-Koordinaten wie ein gewelltes oder auch wie ein mehr oder weniger stark aufgequollenes Stück Pappe aus, manchmal auch teilweise verdreht oder eingerissen und hochgeklappt - aber nicht komplett zerknüllt.

Einige Probleme, die im zweidimensionalen auftreten, verschwinden im dreidimensionalen automatisch. So kommt beispielsweise das Verheddern der Strahlen in den Sterngraphen dort nicht mehr vor. Dafür kommt es bei zufälligen Graphen gelegentlich zu vergleichbaren Effekten, besonders bei einer zufälligen Initialisierung der Knotenpositionen. Hier machen nämlich gelegentlich die Einbuchtungen in Löchern Probleme. Wenn initial einige Knoten oberhalb, und einige Knoten unterhalb der Hauptebene positioniert werden, dann kann es vorkommen, dass eine solche Positionierung in ein lokales, aber offensichtlich nicht globales Minimum darstellt. Bei einer geschickten Initialisierung der Knoten, die entsprechend erweitert werden muss, um einen dreidimensionalen Raum aufzuspannen, verschwinden diese Effekte weitgehend.

3.5 Experimentelle Auswertung

Zunächst betrachten wir weiter die drei bekannten Graphen - den zufälligen Graphen mit knapp 1500 Knoten, das Gitter mit 15×15 Knoten und das gestörte Gitter mit ebenso vielen Knoten. Wir betrachten dabei alle möglichen Wege und vergleichen die Erfolgsquoten für Greedy-Routing unter Verwendung der tatsächlichen und der rekonstruierten Positionen.

Wie erwartet bzw. erhofft ist die Erfolgsquote mit den rekonstruierten Koordinaten in der Regel höher als bei den tatsächlichen Koordinaten. Ebenso ist

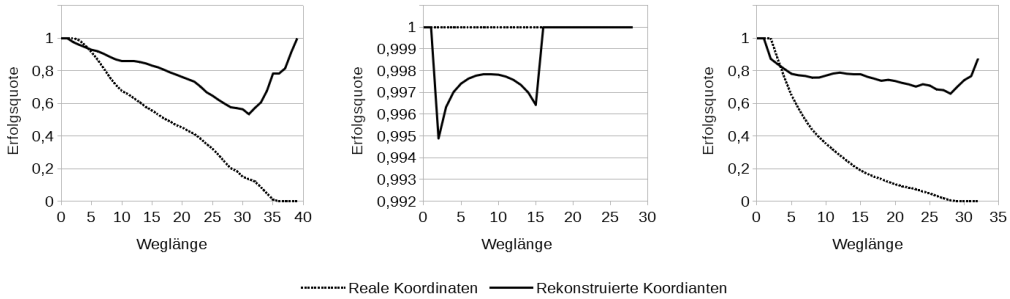

Abbildung 3.14: Erfolgsquote für Greedy-Routing in Abhängigkeit der Hop-Distanz von Start- und Zielknoten für einen zufällig generierten Graphen (links), das Gitter (Mitte) und ein gestörtes Gitter (rechts).

nur wenig verwunderlich, dass die Erfolgsquote bei längeren Wegen oft niedriger ist als bei kurzen Wegen. Je größer die Distanz zwischen zwei Knoten, desto häufiger kann es passieren, dass durch das Greedy-Routing eine falsche Entscheidung getroffen wird, die nicht zum Ziel führt.

Der Anstieg der Erfolgsquote bei dem hier gewählten zufälligen Graphen und dem gestörten Gitter bei sehr langen Wegen ist auffällig, aber im Allgemeinen nicht zu beobachten. Außerdem sollte man hier nicht zu viel Bedeutung hineininterpretieren. Der hintere ansteigende Teil der Kurve repräsentiert gerade mal 2650 Wege. Bei insgesamt über 2 Millionen getesteten Wegen ist das ein zu vernachlässigender Anteil.

Beim Gitter ist die Erfolgsquote mit realen Koordinaten wenig überraschend konstant 100%. Bei Verwendung der virtuellen Koordinaten sind es nicht ganz 100%. Die Kurve dort erscheint im ersten Moment ein wenig merkwürdig. Ein Blick in die absoluten Zahlen klärt diesen Verlauf jedoch schnell auf. Wenn nicht alle Wege gefunden werden, dann schlagen für das virtuelle Routen jeweils genau 8 Wege fehl. Das sind genau die Wege, in denen der Zielknoten ein Nachbarknoten eines Eckknoten ist, und der Startknoten auf der jeweils anderen an dieser Ecke beteiligten Kante[3] liegt. In diesem Fall kommt man mit Greedy-Routing durch die kissenförmige Verzerrung im Eckbereich nicht zum Ziel.

Von diesen Wegen gibt es für die Weglängen 2 bis 15 in einem 15×15-Gitter jeweils genau 8 Stück. Aus der unterschiedlichen Gesamtzahl der Wege mit entsprechender Länge resultiert dann die gleichmäßige Kurve und die

[3]Kante hier einmal im Sinne von „Außenkante des rechteckigen Gebildes, welches das Gitter in der Ebene einnimmt“.

Erfolgsquoten von „nur“ $99,5\%$ bei kurzen Weglängen.

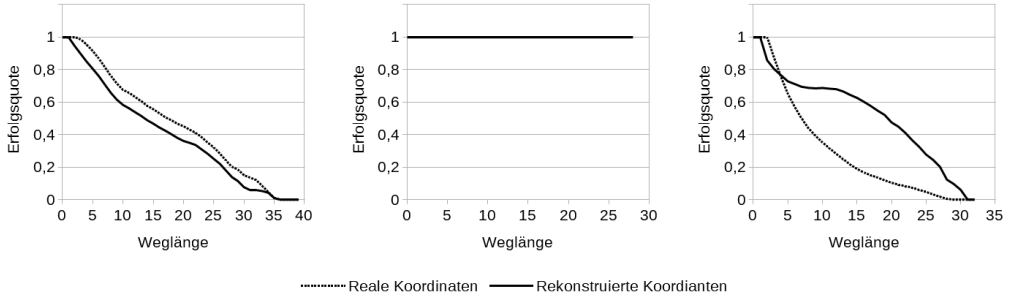

Abbildung 3.15: Erfolgsquote für Greedy-Routing mit den Koordinaten nach einem zweiten Durchlauf von MDS bei zufälligem Graphen, Gitter und gestörtem Gitter.

Zuletzt der Vergleich der Erfolgsquote mit den „verbesserten“ Positionierungen nach zwei Durchläufen aus dem vorigen Abschnitt. Hier zeigt sich sehr deutlich, dass sich die Erfolgsquote beim Routen mit den rekonstruierten Koordinaten der Quote jener mit den tatsächlichen Koordinaten angleicht. Sowohl im positiven Sinne wie beim Gitter, bei dem die Erfolgsquote nach dem zweiten Durchlauf und dem Verschwinden der Kisseneffekte auf 100% steigt, als auch und vor allem im negativen Sinne bei den anderen Graphen. Bei dem zufälligen Graphen ist das nicht verwunderlich - durch den zweiten Durchlauf von MDS werden die an den Rand gedrückten Einbuchtungen in die Löcher wieder entplättet, sodass wieder mehr Sackgassen entstehen können.

Bei dem gestörten Gitter überrascht etwas die deutliche Verschlechterung, obwohl die Einbettung sich nur unwesentlich ändert. Aber auch diese kleinen Veränderungen an einzelnen Knoten reichen offensichtlich aus, um im Sinne von Greedy-Routing zu einer falschen (d.h. nicht zum Ziel führenden) Entscheidung zu kommen. Allerdings ist der Rückgang der Erfolgsquote nicht ganz so dramatisch, wie er anhand der Kurve erscheint. Insgesamt verringert sich die Erfolgsquote nur von ca. 74% auf ca. 66%.

3.5.1 Testläufe auf vielen Graphen

Im Folgenden betrachten wir die allgemeine Erfolgsquote für Greedy-Routing bei zufälligen Graphen mit verschiedenen Parametern. Die Testwerte basieren auf der Auswertung von jeweils 100 zufällig erzeugten Graphen mit jeweils 1000 zufällig ausgewählten Wegen.

Da es offensichtlich sein sollte, dass für das Greedy-Routing generell dichtere Graphen von Vorteil sind, werden wir für einen Vergleich der unterschiedlichen Kantenmodelle (Unit-Disk-Graph, Quasi-Unit-Disk-Graph und Waxman) den Senderadius bzw. die maximale Kantenlänge so anpassen, dass bei gleicher Knotenzahl auf dem gleichen Gebiet in etwa der gleiche durchschnittliche Knotengrad erreicht wird.

Gleichmäßige Verteilung der Knoten

Zuerst betrachten wir Unit-Disk-Graphen mit einer gleichmäßigen Knotenverteilung. Die Ergebnisse für das Greedy-Routing mit den tatsächlichen Koordinaten sind wenig überraschend. Bei geringer Knotenzahl n und einem daraus resultierendem schwächeren Zusammenhang ist die Erfolgsquote für Greedy-Routing generell niedriger als bei dichteren Graphen, bei denen es weniger Sackgassen gibt.

Die Rekonstruktion der Koordinaten durch MDS sorgt schon bei bloßer Nutzung der Hop-Distanzen für eine deutliche Verbesserung der Erfolgsquote. Bei den Graphen mit sehr geringer Dichte ist dieser Effekt besonders deutlich zu beobachten.

Bemerkenswert ist jedoch, dass bei Graphen mit geringer Knotendichte eine Abschätzung der Kantenlängen einen negativen Effekt auf die Erfolgsquote hat: Ein Versuch, die Erfolgsquote zu verbessern, indem zusätzliche Informationen genutzt werden, führt also nicht zu dem gewünschten Ergebnis. Die Ursache dürfte darin begründet sein, dass bei einer Abschätzung der Kantenlängen die Einbettung näher an die tatsächliche heranreicht(vgl. Abschnitt 3.4.3), was generell kontraproduktiv für das Greedy-Routing zu sein scheint.

Nur bei dichteren Graphen sorgt eine Abschätzung der Distanzen zwischen den Knoten für eine weitere (kleine) Verbesserung der Erfolgsquote, die dort aber ohnehin auf einem sehr hohen Niveau liegt (d.h. $\geq 90\%$).

Bei einer Erweiterung der MDS-Koordinaten auf drei Dimensionen mit Verwendung von Hop-Distanzen (Kurve „3-D Hop“ in der Abbildung) ist die Entwicklung umgekehrt. Hier zeigen sich bei den weniger dichten Graphen teils deutliche Verbesserungen gegenüber der zweidimensionalen Rekonstruktion. Bei dichten Graphen ist dann jedoch eine 3D-Rekonstruktion nicht hilfreich oder sogar kontraproduktiv.

Beobachtung 21 *Bei Unit-Disk-Graphen mit gleichmäßiger Verteilung der Knoten reicht die Verwendung von Hop-Distanzen meist aus. In dünnen*

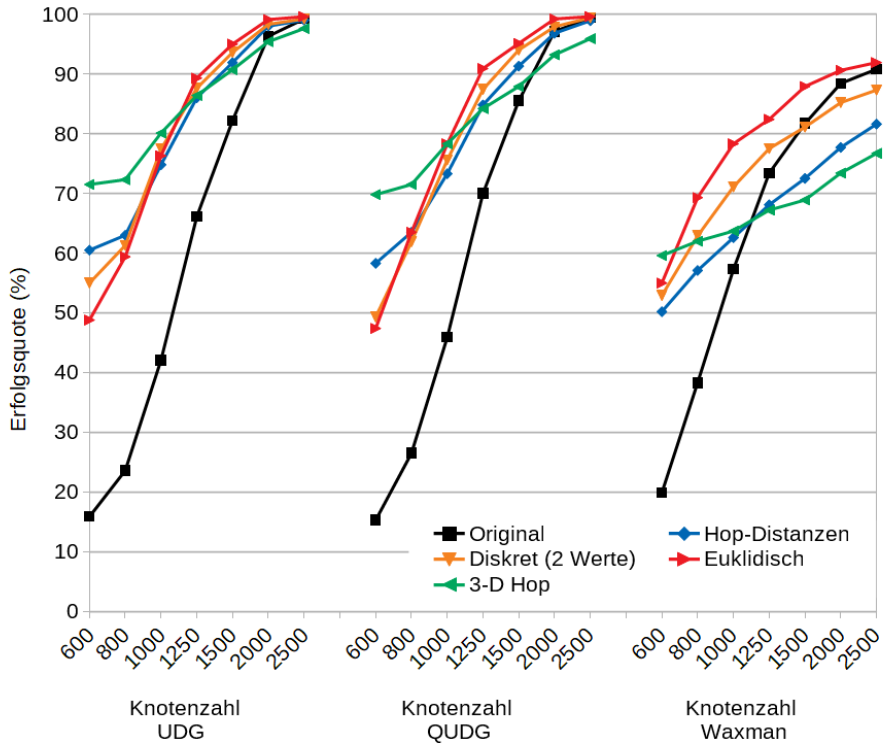

Abbildung 3.16: Erfolgsquoten in Prozent für Greedy-Routing bei gleichmäßiger Verteilung der Knoten auf einem Gebiet der Größe 1000×1000 mit UDG ($r = 50$, links), Q-UDG ($r = 58$, Mitte) und Waxman-Kanten ($r = 93$, rechts)

Netzwerken ist eine dreidimensionale Auslegung von Vorteil, in dichteren Netzwerken kann eine Abschätzung der Kantenlängen die Resultate noch leicht verbessern.

Bei $1/\sqrt{2}$-Quasi-Unit-Disk-Graphen sind die Beobachtungen sehr ähnlich. Um einen ähnlichen Knotengrad wie bei den Uni-Disk-Graphen zu erhalten, haben wir hier den Senderadius der Knoten auf $r = 58$ erhöht. Eine exakte Übereinstimmung des durchschnittlichen Knotengrades lässt sich natürlich nicht sicherstellen - jedoch werden so die Werte besser vergleichbar.

Auch hier sind Abschätzungen der Kantenlängen bei wenig dichten Graphen eher kontraproduktiv, bei höherer Dichte werden die Ergebnisse etwas besser. Bei der dreidimensionalen Auslegung wieder das umgekehrte Verhalten. Bei dünnen Netzwerken lässt sich eine Verbesserung beobachten, bei dichten Graphen eine Verschlechterung gegenüber der Basis-Variante.

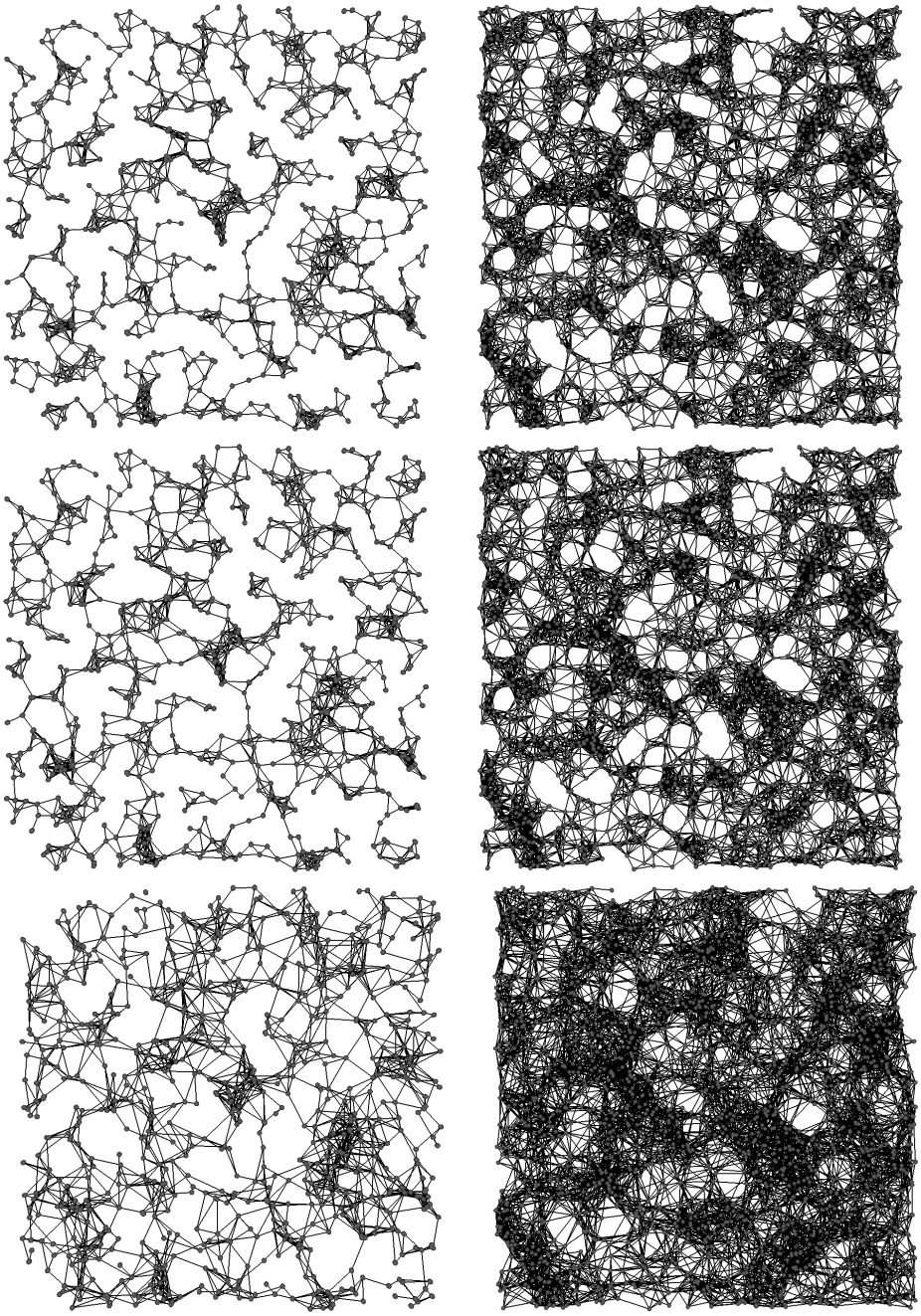

Abbildung 3.17: Einige Beispiele für die getesteten Netzwerke bei gleichmäßiger Knotenverteilung mit 800 Knoten (links) und 2000 Knoten (rechts). Oben Graphen mit Unit-Disk-Kanten ($r = 50$), in der Mitte $1/\sqrt{2}$-Quasi-Unit-Disk-Kanten ($r = 58$), unten Kanten im Waxman-Modell ($r = 93$).

Beobachtung 22 *Auf $1/\sqrt{2}$-Quasi-Unit-Disk-Graphen verhält sich die Multidimensionale Skalierung sehr ähnlich wie auf Unit-Disk-Graphen.*

Um im Waxman-Modell vergleichbare Knotengrade zu bekommen, muss die Sendereichweite der Knoten in unserem Szenario auf $r = 93$ erhöht werden. Ein Nebeneffekt davon ist, dass die durchschnittliche Weglänge der getesteten Wege stark abnimmt.

Diese reduzierte Weglänge dürfte auch der ausschlaggebende Punkt dafür sein, dass die Erfolgsquote für das Routen mit den tatsächlichen Koordinaten bei geringer Knotenzahl recht deutlich über den Werten der anderen Kantenmodelle liegt. Je weniger Kanten auf dem Weg liegen, desto seltener kann eine falsche Entscheidung getroffen werden, die in eine Sackgasse führt. Bei hohen Knotengraden verliert die geringere Weglänge an Einfluss, und die Erfolgsquoten fallen trotz geringerer Weglänge hinter die Quoten der anderen Modelle zurück.

Interessant ist, dass die Rekonstruktion der Koordinaten mit MDS nicht von diesem Effekt profitieren kann, zumindest nicht bei der Verwendung von Hop-Distanzen. Hier verschlechtern sich die Erfolgsquoten teilweise sogar - ein Effekt, der bei den anderen Modellen längst nicht in diesem Ausmaß beobachtet werden kann.

Eine weitere Abweichung von dem bisher beobachteten Verhalten ist die durchgängige Verbesserung der Erfolgsquote durch eine Abschätzung der Kantenlängen. Da hier sehr lange Kanten existieren, und sehr kurze Kanten fehlen können, ist eine Abschätzung der Länge für MDS anscheinend sehr hilfreich. Bei sehr dichten Graphen ist diese Abschätzung der Kantenlängen sogar notwendig, um wenigstens die Erfolgsquoten zu erreichen, die beim Routen mit den tatsächlichen Koordinaten beobachtet werden können.

Eine Rekonstruktion in einer weiteren Dimension ist auch hier für wenig dichte Graphen sinnvoll, für dichte Graphen nicht.

Beobachtung 23 *Bei Graphen mit Kanten im Waxman-Modell ist eine Abschätzung der Kantenlängen hilfreich und führt zu einer Verbesserung der Erfolgsquote beim Greedy-Routing gegenüber der Rekonstruktion mit reinen Hopdistanzen.*

Verteilung durch Zentren

Bei einer ungleichmäßigen Verteilung der Knoten lassen sich einige ähnliche Effekte beobachten, aber auch ein paar Abweichungen. Beim direkten Ver-

gleich der Abbildungen ist zu beachten, dass bei der Verteilung durch Zentren keine Graphen mit $n = 600$ und $n = 800$ Knoten getestet wurden. Der Grund dafür ist der häufige fehlende Zusammenhalt der Graphen in diesen Fällen. Die größte Zusammenhangskomponente hat dabei oft nur eine Größe von $\ll \frac{n}{2}$, so dass aus einem Test auf diesen Graphen keine sinnvollen Ergebnisse abgelesen werden können.

Auffallend ist zunächst, dass die Erfolgsquoten mit steigender Knotenzahl weniger stark ansteigen als bei einer rein zufälligen Verteilung der Knoten. Bei der Art und Weise, wie die Graphen hier aufgebaut werden, ist das auch wenig verwunderlich: Ein guter Teil der zusätzlichen Knoten erhöht nur den Knotengrad in der Nähe der Zentren, nicht aber unbedingt auch den Grad des Zusammenhangs des gesamten Graphen in seiner Makrostruktur. Besonders an den Randgebieten ist auch bei Graphen mit vielen Knoten eine Zerfaserung zu erkennen, die auch mit der Multidimensionalen Skalierung nicht vollständig aufgelöst werden kann, und die dann zu Fehlschlägen beim Greedy-Routing führt.

Eine Abschätzung der Kantenlängen ist auch hier im Waxman-Modell sinnvoll. Bei Unit-Disk- und Quasi-Unit-Disk-Graphen hat eine Abschätzung der Kantenlängen bei dieser Form der Knotenverteilung so gut wie keine Auswirkungen auf die Erfolgsquote beim Greedy-Routing.

Im Gegensatz zur gleichmäßigen Knotenverteilung kann hier im Waxman-Modell durchgehend eine höhere Erfolgsquote beim Routen mit den rekonstruierten Koordinaten beobachtet werden als mit den originalen. Dieser positive Effekt wird mit zunehmender Knotenzahl aber auch hier geringer.

Die Verwendung von 3D-Rekonstruktionen ist bei Unit-Disk- und Quasi-Unit-Disk-Graphen durchgehend eine gute Option. Bei der gleichmäßigen Knotenverteilung trat dieser Effekt nur auf bei Netzwerken mit weniger Knoten und einem dadurch verursachten geringen durchschnittlichen Knotengrad. Im Waxman-Modell bleibt auch bei vielen Knoten die dreidimensionale Rekonstruktion besser als die mit Hop-Distanzen, fällt jedoch hinter der Rekonstruktion mit einer Abschätzung der Kantenlängen zurück.

Beobachtung 24 *Bei einer stark ungleichmäßigen Verteilung der Knoten ist eine dreidimensionale Auslegung der Knoten sinnvoll und führt zu besseren Ergebnissen.*

Eine Abschätzung der Kantenlängen ist hier nur im Waxman-Modell sinnvoll, und hat in den anderen Modellen praktisch keine Auswirkungen.

Auf den ersten Blick erscheint es reizvoll, diese Unterschiede zur gleichmäßi-

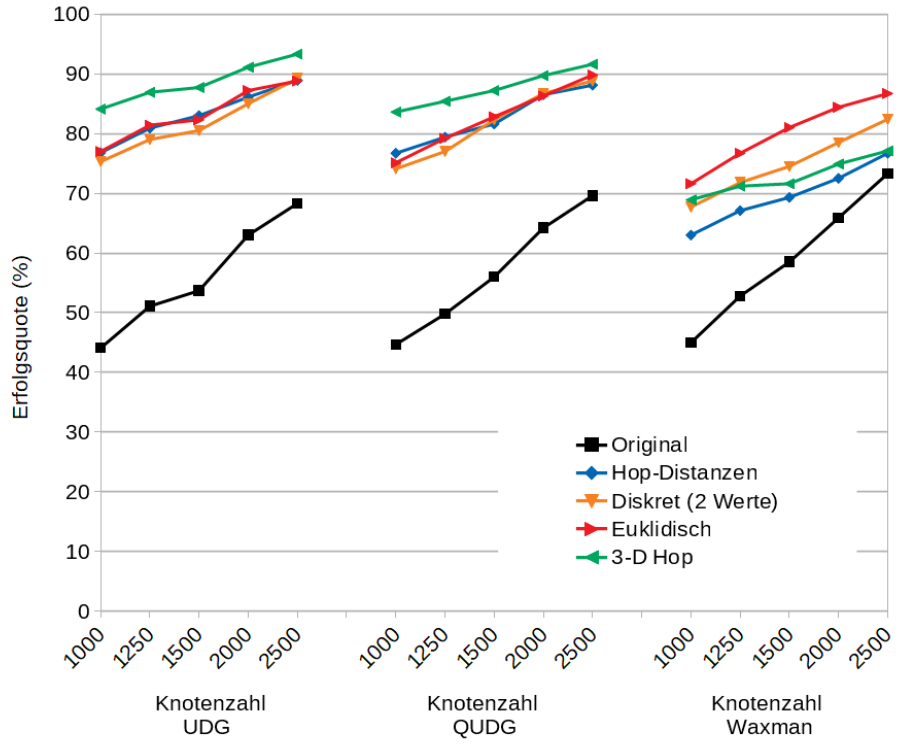

Abbildung 3.18: Erfolgsquoten in Prozent für Greedy-Routing bei ungleichmäßiger Verteilung der Knoten auf einem Gebiet der Größe 1000×1000. Anzahl der Zentren: 25, Streuweite 100.

gen Verteilung näher zu untersuchen. Insbesondere stellt sich die Frage, ob man bei einer Reduktion der ungleichmäßig verteilten Graphen auf die jeweiligen Verteilzentren eine sinnvolle Vergleichsmöglichkeit zum vorigen Fall (also den Graphen mit einer gleichmäßigen Verteilung der Knoten) erhält. Dabei ergibt sich ein technisches Problem aus der Tatsache, dass die Teilgraphen, die von den Knoten induziert werden, die um ein Zentrum herum verteilt werden (vgl. Algorithmus 2), nicht notwendigerweise zusammenhängend sind. Diese Tatsache ändert sich auch dann nicht, wenn wir die Knoten nach der Verteilung dem jeweils nächstliegenden Zentrum zuordnen. Es reicht also nicht, wenn für jedes Zentrum einen Repräsentanten wählen. Stattdessen müssen wir für jede Zusammenhangskomponente dieser so induzierten Teilgraphen einen Repräsentanten wählen.

Definition 25 (Zentrumsgraph) Sei $G = (V, E)$ ein Graph, $S \subset \mathbb{R}^2$ eine endliche Punktmenge und $c : V \to S$ eine Zuordnung der Knoten zu den

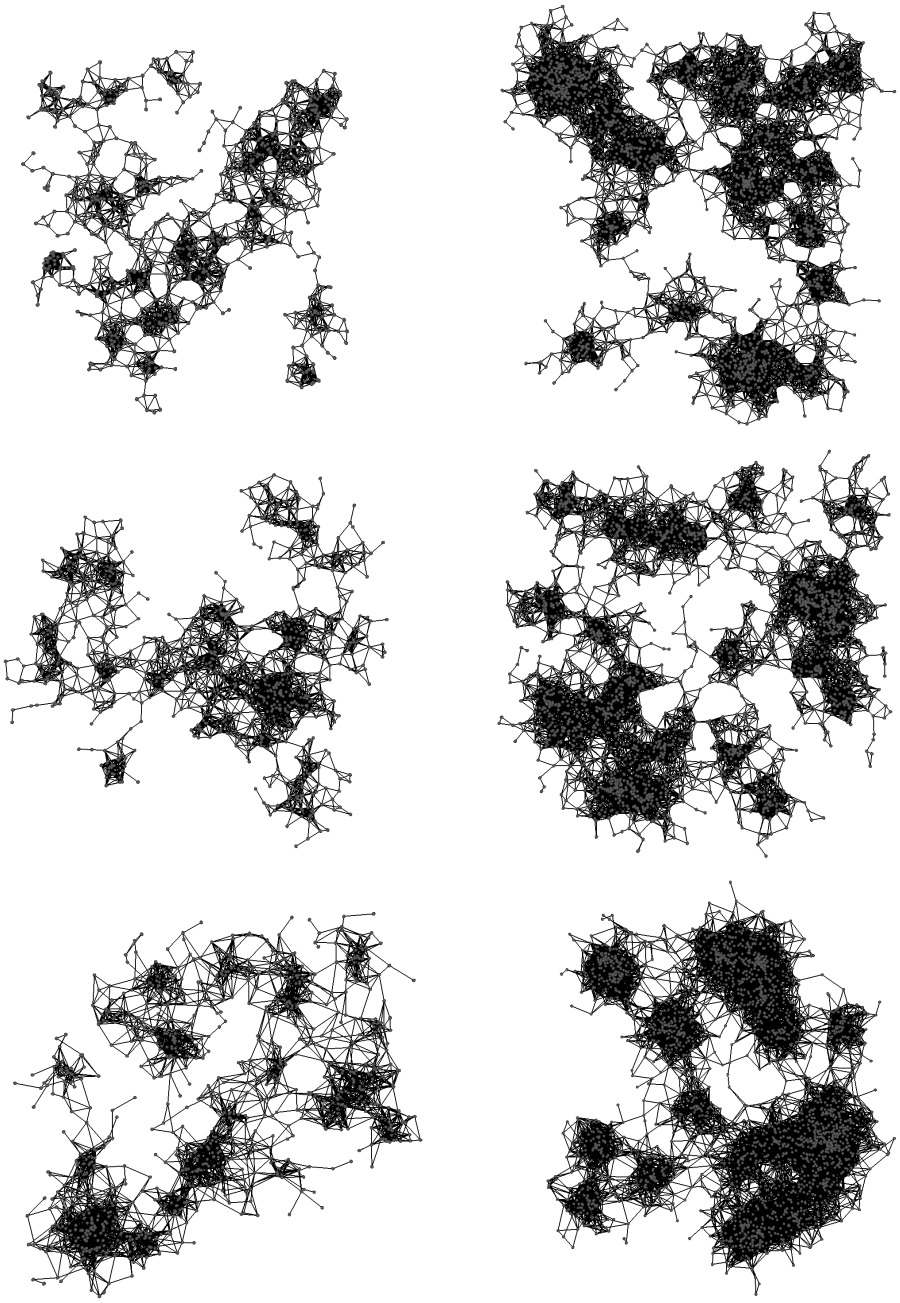

Abbildung 3.19: Einige Beispiele für die getesteten Netzwerke mit $n = 1000$ (links) und $n = 2500$ (rechts) bei Verteilung über 25 Zentren mit einer Streuweite von 100. Oben Graphen mit Unit-Disk-Kanten ($r = 50$), in der Mitte $1/\sqrt{2}$-Quasi-Unit-Disk-Kanten ($r = 58$), unten Kanten im Waxman-Modell ($r = 93$).

Punkten in S.

Weiter sei $G_S = (V, E_S) \subset G$ der Teilgraph von G, der nur die Kanten $\{u, v\} \in E$ enthält mit $c(u) = c(v)$, und $G_{S,1} = (V_{S,1}, E_{S,1}), \ldots G_{S,k} = (V_{S,k}, E_{S,k})$ die Zusammenhangskomponenten von G_S.

Der *Zentrumsgraph* $G_Z(G, S, c) = (V_Z, E_Z)$ ist wie folgt definiert. Die Knotenmenge V_Z enthält für jede Zusammenhangskomponente von G_S genau einen Repräsentanten, und eine Kante zwischen zwei Knoten genau dann, wenn die zugehörigen Zusammenhangskompenten von G_S in G benachbart sind.

$$\begin{aligned} V_Z &:= \{v_1, \ldots v_k\} \\ E_Z &:= \{\{v_i, v_j\} \mid \exists u \in V_{S,i}, \exists v \in V_{S,j} \text{ mit } \{u, v\} \in E\} \end{aligned}$$

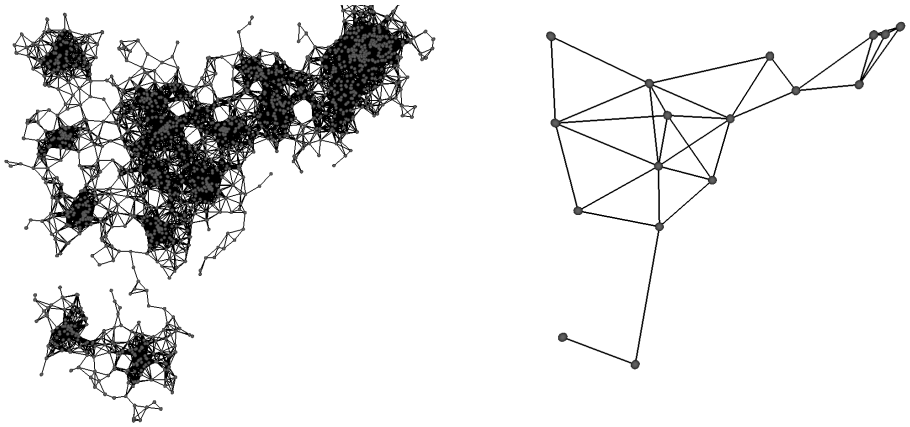

Abbildung 3.20: Beispiel für einen Graphen und einen zugehörigen Zentrumsgraphen. Die Punktmenge S ist die, die in Algorithmus 2 für den Aufbau des Graphen benutzt wurde, die Zuordnung c ordnet hier jedem Knoten das nächstliegende Zentrum zu.

An dem Beispiel in Abbildung 3.20 wird aber direkt klar, dass ein näherer Vergleich der Erfolgsquoten im Originalgraphen mit dem Zentrumsgraphen wenig zielführend ist. Der Zentrumsgraph ist so stark vereinfacht, dass die Multidimensionale Skalierung eine Auslegung erzeugt, die eine 100%ige Erfolgsquote beim Greedy-Routing ermöglicht, während im Originalgraphen es doch einige Sackgassen gibt.

Knotenverteilung und Kantenaufbau mit Hindernissen

Die Verteilung über Karten und die Verbindung nur bei Sichtkontakt testen wir an einem Straßenzug in Berlin Schöneberg (52°30'N, 13°21'E, Abbildung 3.21). Die Knoten können nur auf unbebauten Stellen platziert werden, Gebäude dienen als Hindernisse.

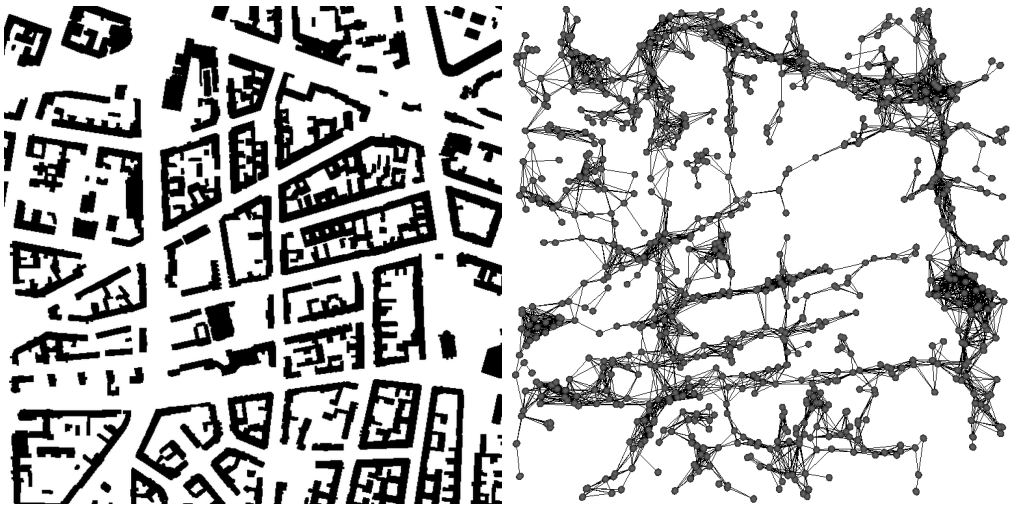

Abbildung 3.21: Die verwendete Karte (links) und ein Netzwerk, bei dem nur Knoten auf den weißen Gebieten erlaubt sind, und Kanten keine schwarzen Gebiete kreuzen dürfen.

Bei dieser Verteilung ergibt eine Unterteilung in die unterschiedlichen Kantenmodelle nur noch wenig Sinn, da die Kanten durch die Einschränkung „nur Sichtkontakt“ bereits stark von denen eines eingebetteten Unit-Disk-Graphen abweichen.

Bei der Betrachtung von Abbildung 3.22 ist zu beachten, dass die Knotenzahl n zwar deutlich höher ist als in den bisherigen Tests. Allerdings werden hier auch viele Knoten verworfen - nämlich dann, wenn sie durch Zufall auf in einem ungültigen Gebiet landen. Die Größe der tatsächlich betrachteten Netzwerke liegt auch hier wieder im Bereich von 500 bis 2000 Knoten.

Außerdem fällt auf, dass eine Erhöhung der Knotenzahl zwar durchaus einen recht deutlichen Einfluss auf die Erfolgsquote beim Routen mit den tatsächlichen Koordinaten hat (wenn auch auf generell sehr niedrigem Niveau), aber nicht unbedingt, wenn man mit den den rekonstruierten Koordinaten arbeitet. Verwendet man für MDS nur Hop-Distanzen, hat die Knotenzahl fast keinen Einfluss auf das Ergebnis - weder in der zweidimensionalen, noch in der dreidimensionalen Auslegung.

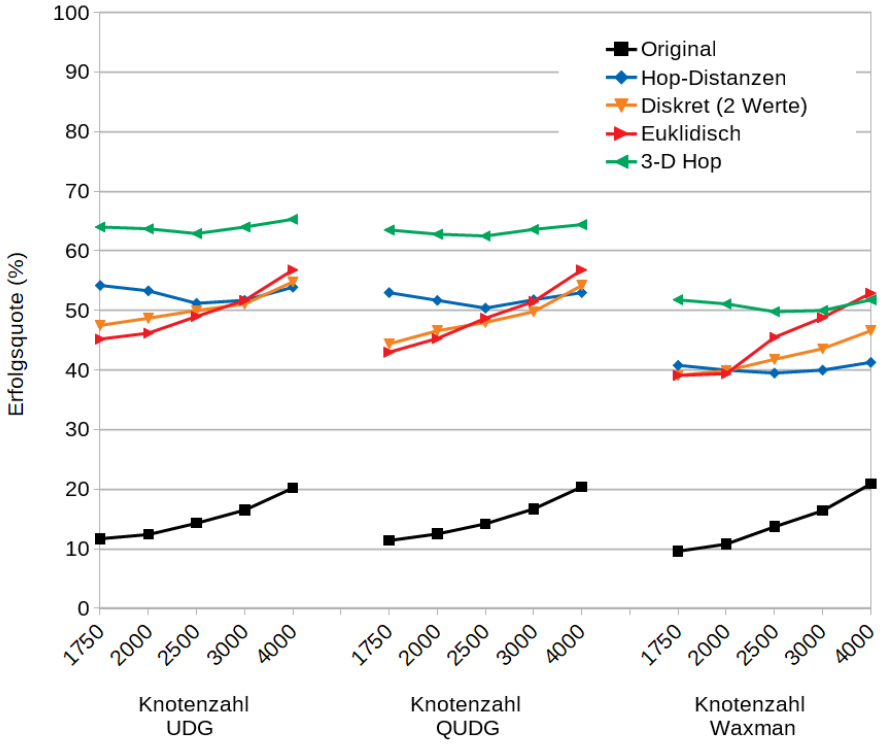

Abbildung 3.22: Erfolgsquoten beim Greedy-Routing bei Knotenverteilung mit Hindernissen, die durch die Karte in Abbildung 3.21 definiert werden. Kanten, sofern nicht durch Hindernisse unterbunden, mit UDG (links), Q-UDG (Mitte) und Waxman-Kanten (rechts).

Bei einer Abschätzung der Kantenlängen lässt sich zwar eine Verbesserung der Quoten mit steigenden Knotenzahlen beobachten. Jedoch sorgt bei kleinen Knotenzahlen die Abschätzung der Kantenlängen für schlechtere Ergebnisse als die Verwendung von Hopdistanzen. Insgesamt betrachtet ist eine Abschätzung der Kantenlängen hier nicht nützlich.

Die Auslegung in drei Dimensionen sorgt durchgehend für bessere Ergebnisse.

Beobachtung 26 *Bei einer Verteilung der Knoten im Straßennetz reichen Hopdistanzen aus für eine sehr deutliche Verbesserung der Erfolgsquote. Eine Abschätzung der Kantenlänge ist hier nicht hilfreich. Eine Auslegung in mehr Dimensionen verbessert die Erfolgsquote.*

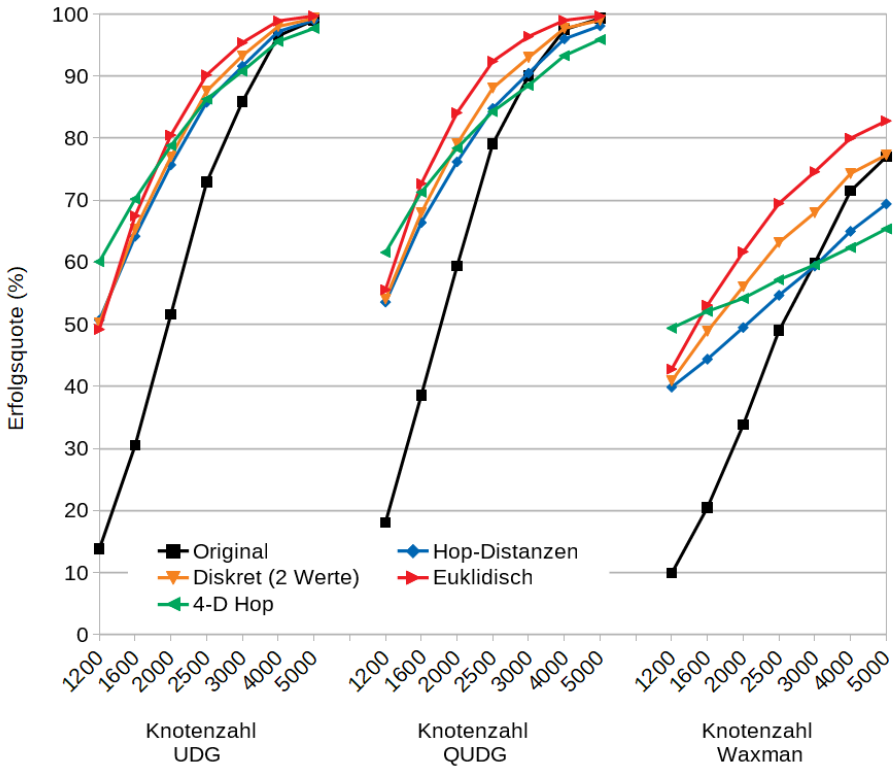

Abbildung 3.23: Erfolgsquoten beim Greedy-Routing bei 3D-Netzen auf einem Gebiet der Größe $1000 \times 1000 \times 1000$ mit UDG (links), Q-UDG (Mitte) und Waxman-Kanten (rechts)

Dreidimensionale Netzwerke

In einigen Anwendungsfällen sind auch dreidimensionale Sensornetzwerke denkbar - zum Beispiel in Unterwassernetzwerken. Auch auf solche Netzwerke lässt sich die Multidimensionale Skalierung anwenden. Dann ist natürlich auch eine dreidimensionale Auslegung sinnvoll und ggf. die Frage, ob eine vierte Dimension bei den virtuellen Koordinaten eine Verbesserung bringen kann.

Die Tendenzen der Erfolgsquoten sind hier die gleichen wie bei zweidimensionalen Netzen. Je wilder die Kantenlängen variieren, desto stärker wird die Verbesserung durch eine Abschätzung der Kantenlängen. Eine Erhöhung der Dimension bringt bei geringem Knotengrad gewisse Vorteile, bei hohem Knotengrad ist sie nicht zu empfehlen.

Beobachtung 27 *Bei Anwendung der Multidimensionalen Skalierung auf*

Netzwerke mit einer ursprünglich dreidimensionalen Auslegung lassen sich die gleichen Effekte beobachten, die auch bei zweidimensionalen Netzwerken beobachtet werden können.

Unvollständige Distanzmatrizen

Bisher haben wir für die Multidimensionale Skalierung immer alle kürzesten Wege berechnet (All-Pairs-Shortest-Path) und daraus eine vollständige Distanzmatrix Δ erstellt, die die Eingabe für das SMACOF-Verfahren bildet (vgl. Algorithmus 3). Die Gewichte w_{ij} werden allesamt auf $w_{ij} = 1$ gesetzt, so dass die Länge eines jeden Weges gleichberechtigt in die Bewertung der Auslegung eingeht. Eine Frage ist noch, ob für gute Ergebnisse tatsächlich *alle* kürzesten Wege bekannt sein müssen, oder ob es ausreicht, diese nur zu einem Teil der Knoten zu kennen.

Für diese Testreihen wurde in der Distanzmatrix ein gewisser Anteil der Spalten und Zeilen markiert, und nur diese markierten Werte wurden dann in der Berechnung des Stresswertes für die Multidimensionale Skalierung verwendet. Die übrigen Distanzwerte wurden nicht weiter verwendet.

Dabei zeigt sich in den Testläufen, dass oft auch deutlich weniger Information ausreichend ist, um noch brauchbare Ergebnisse zu erzielen. Es hat sich außerdem herausgestellt, dass bei einer ausgedünnten Distanzmatrix eine geschickt gewählte Startkonfiguration (vgl. Abschnitt 3.3.3) die Ergebnisse sehr deutlich verbessern kann. Während bei den vorangegangenen Testreihen die Wahl der Startkonfiguration nur einen marginalen Einfluss auf die Erfolgsquoten hatte, spielt sie hier eine ganz entscheidende Rolle - besonders bei stark ausgedünnten Distanzmatrizen.

Ohne eine geschickte Initialisierung der Knotenpositionen kommt es bei dieser Ausdünnung sehr oft dazu, dass sich das Netzwerk nicht korrekt in die Ebene entfaltet, und es zu den besprochenen Verknotungs-Effekten kommt (vgl. Abbildung 3.7), wodurch das Greedy-Routing häufiger fehlschlägt. Beispielhaft für diesen Effekt sind in Abbildung 3.24 die unterschiedlichen Erfolgsquoten bei Unit-Disk-Graphen aufgelistet. Besonders bei sehr stark ausgedünnten Distanzmatrizen (Verwendung von nur 5% bis 10% aller Wege) ist eine sehr deutliche Steigerung der Erfolgsquoten zu verzeichnen, wenn die Startkonfiguration gezielt gewählt wird.

In Abbildung 3.25 sind schließlich die Ergebnisse für die drei bisher auch untersuchten Kantenmodelle aufgetragen, wenn nur die kürzesten Wege zu einem Bruchteil der Knoten bekannt sind. Insgesamt zeigt sich, dass bei ei-

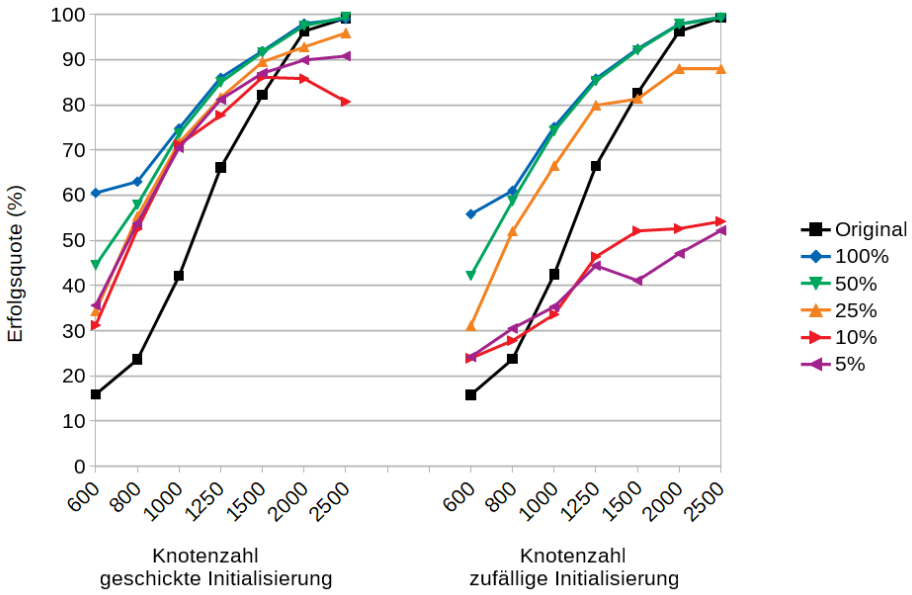

Abbildung 3.24: Erfolgsquoten beim Greedy-Routing bei gleichmäßiger Verteilung der Knoten auf einem Gebiet der Größe 1000×1000. Rekonstruktion über Hop-Distanzen mit unvollständigen Distanzmatrizen. Kantenmodell UDG. Links mit einer geschickter gewählten Startkonfiguration, rechts mit zufälliger Initialisierung.

nem Ausdünnen der Distanzmatrix zwar etwas schlechtere Ergebnisse erzielt werden, insgesamt die Erfolgsquoten aber in vielen Fällen immer noch deutlich über dem Wert liegen, der mit den tatsächlichen Koordinaten erreicht wird. Nur bei sehr dichten Graphen werden die Quoten im Vergleich schlechter.

Beobachtung 28 *Bei unvollständigen Distanzmatrizen ist es empfehlenswert, die Startkonfiguration durch eine schnelle Vorab-Analyse des Graphen geschickt zu wählen. Bei zunehmender Ausdünnung der Matrizen sinken zwar auch damit die Erfolgsquoten, bleiben aber insgesamt auf einem hohen Niveau und liegen - abgesehen von sehr dichten Graphen - weiterhin über den Erfolgsquoten beim Routen mit den tatsächlichen Koordinaten.*

Die Auswahl der Ankerknoten, zu denen die kürzesten Wege ermittelt werden, hat dabei ebenfalls noch Einfluss auf die Qualität der Rekonstruktion. Gerade dann, wenn nur die Distanzen zu wenigen Knoten berücksichtigt werden sollen, ist es empfehlenswert, diese Knoten nicht zufällig auszuwählen, sondern gezielt. Dabei sollte eine gleichmäßige Verteilung dieser Knoten im

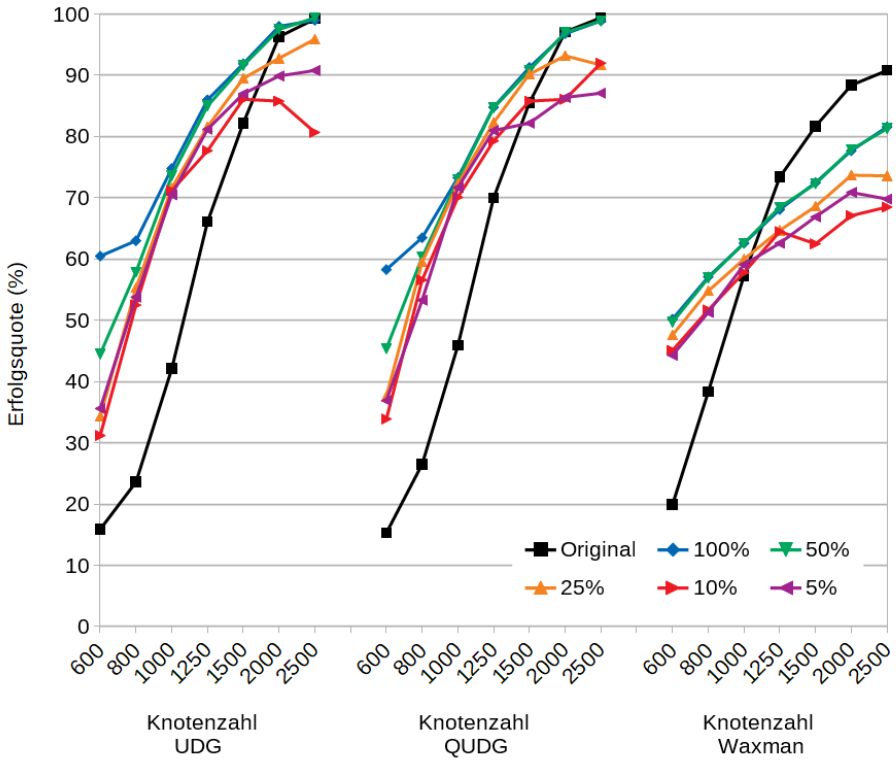

Abbildung 3.25: Erfolgsquoten beim Greedy-Routing bei gleichmäßiger Verteilung der Knoten auf einem Gebiet der Größe 1000×1000. Rekonstruktion über Hop-Distanzen mit unvollständigen Distanzmatrizen. Kantenmodell im Graph wie sonst auch UDG (links), Q-UDG (Mitte) und Waxman (rechts)

Netzwerk angestrebt werden. Die ersten beiden Knoten können dabei so gewählt werden, wie wir auch die beiden speziellen Knoten für die Startkonfiguration wählen. Die folgenden Knoten werden dann so gewählt, dass der minimale Abstand des nächsten Knoten zu allen bereits gewählten Knoten maximal wird (vgl. Algorithmus 7).

Eine ähnliche Auswahl von Ankerknoten werden wir auch im nächsten Kapitel bei der Randerkennung benutzen. Allerdings werden wir die Distanzwerte d anders wählen, nach der wir den jeweils nächsten Ankerknoten bestimmen.

Algorithmus 7 : Auswahl von Ankerknoten für MDS mit ausgedünnter Distanzmatrix

Eingabe : Ein Graph $G = (V, E)$
zwei Knoten $u_1, u_2 \in V$ mit großem Abstand (Alg. 5)
gewünschte Anzahl k an Ankerknoten
Ausgabe : gleichmäßig verteilte Knotenmenge $A \subset V$

```
1 Initialisiere A = {u_1, u_2}
2 Bestimme d(u) = min{dist(u_1, u), dist(u_2, u)} für alle u ∈ V
3 while |A| < k do
4     wähle v mit d(v) maximal
5     Füge v zu A hinzu
6     Setze d(u) = min{d(u), dist(v, u)} für alle u ∈ V
7 end
```

3.5.2 Tabellen mit den Erfolgsquoten

Im Folgenden als Ergänzung zu den Grafiken einige Tabellen mit den ermittelten Erfolgsquoten. Die Daten sind hier ergänzt durch Angaben zum durchschnittlichen Knotengrad $\deg(G)$ der getesteten Graphen, sowie die mittlere Distanz $\delta(p)$ zwischen zwei Knoten in diesen Graphen. Außerdem finden sich in den Tabellen die Werte für eine Rekonstruktion der Koordinaten mit diskreten Kantengewichten in 4 Abstufungen (ω_4^D), die in den Grafiken aus Gründen der Übersichtlichkeit weggelassen wurden.

	n	$\deg(G)$	$\delta(p)$	orig.	ω^H	ω_2^D	ω_4^D	ω^E	3D
UDG	600	4.7	22.0	15.9	60.5	55.0	49.6	48.8	71.5
$r = 50$	800	6.1	18.1	23.6	63.0	61.3	60.4	59.4	72.3
	1000	7.5	15.7	42.1	74.8	77.4	77.0	76.3	80.1
	1250	9.3	14.5	66.1	86.0	87.6	88.8	89.3	86.4
	1500	11.2	13.8	82.2	91.9	93.6	94.4	95.0	90.7
	2000	15.0	13.2	96.3	98.0	98.3	98.6	99.1	95.4
	2500	18.8	12.8	99.2	99.0	99.2	99.4	99.6	97.6
QUDG	600	4.7	20.2	15.3	58.3	49.3	47.5	47.4	69.8
$r = 58$	800	6.1	16.2	26.5	63.5	61.9	64.3	63.5	71.5
	1000	7.5	14.2	45.9	73.3	75.6	78.7	78.3	78.3
	1250	9.4	13.1	70.0	84.8	87.4	89.8	90.9	84.2
	1500	11.3	12.6	85.5	91.3	94.0	95.8	95.1	87.9
	2000	15.1	11.9	97.1	96.8	97.9	98.6	99.2	93.2
	2500	18.9	11.5	99.4	98.9	99.4	99.6	99.6	95.9
Waxman	600	4.6	11.6	19.9	50.2	52.9	54.6	55.0	59.6
$r = 93$	800	6.0	10.0	38.3	57.1	62.9	68.0	69.3	62.0
	1000	7.5	9.2	57.3	62.6	71.1	76.3	78.3	63.7
	1250	9.4	8.6	73.4	68.1	77.5	85.1	82.4	67.2
	1500	11.3	8.3	81.7	72.5	81.1	85.5	87.9	68.9
	2000	15.1	7.9	88.4	77.7	85.2	88.8	90.6	73.4
	2500	18.8	7.6	90.8	81.6	87.3	90.2	91.9	76.7

Tabelle 3.3: Erfolgsquoten in Prozent für Greedy-Routing bei gleichmäßiger Verteilung der Knoten auf einem Gebiet der Größe 1000×1000.

	s	σ	$\deg(G)$	$\delta(p)$	orig.	ω^H	ω_2^D	ω_4^D	ω^E	3D
UDG	50	75	18.6	15.4	46.8	77.0	76.4	76.4	76.4	85.7
$r = 50$	25	100	21.0	14.9	53.6	80.5	79.7	79.9	80.0	87.4
	15	150	19.1	14.1	60.2	81.4	81.5	81.9	81.9	87.6
QUDG	50	75	18.5	13.5	47.9	77.7	76.6	79.0	77.6	84.9
$r = 58$	25	100	21.1	13.1	55.3	78.9	79.5	80.6	77.6	86.1
	15	150	19.2	13.1	62.4	80.8	80.9	81.2	80.8	86.4
Waxman	50	75	16.9	8.8	55.4	67.0	73.5	76.9	78.7	70.4
$r = 93$	25	100	18.7	8.8	57.5	68.0	72.7	76.5	78.0	71.9
	15	150	17.7	8.4	63.8	68.6	73.1	76.6	79.5	71.0

Tabelle 3.4: Erfolgsquoten beim Greedy-Routing bei ungleichmäßiger Knotenverteilung. Parameter s gibt die Anzahl der Zentren an, σ die Streuweite. Knotenzahl $n = 1500$.

	n	$\deg(G)$	$\delta(p)$	orig.	ω^H	ω_2^D	ω_4^D	ω^E	3D
UDG	1000	14.5	14.8	44.1	76.8	75.3	75.1	77.0	84.1
$r = 50$	1250	17.6	15.0	51.1	80.9	79.0	79.2	81.4	86.9
	1500	20.9	14.2	53.7	83.0	80.5	82.3	82.3	87.7
	2000	27.5	13.9	63.0	86.1	85.0	87.5	87.2	91.1
	2500	34.9	13.6	68.3	88.9	89.2	89.1	88.8	93.3
QUDG	1000	14.4	13.6	44.7	76.7	74.1	75.5	75.1	83.6
$r = 58$	1250	17.7	13.5	49.8	79.4	77.0	79.5	79.2	85.4
	1500	21.8	12.8	56.0	81.6	82.3	82.4	82.8	87.2
	2000	27.3	12.9	64.2	86.5	86.7	87.1	86.3	89.7
	2500	34.2	12.2	69.6	88.1	88.8	90.5	89.8	91.6
Waxman	1000	12.9	9.5	45.0	63.0	67.7	70.4	71.6	68.9
$r = 93$	1250	15.9	9.1	52.8	67.1	71.8	74.1	76.7	71.2
	1500	18.8	8.8	58.5	69.3	74.5	77.3	81.0	71.6
	2000	24.8	8.5	65.9	72.5	78.5	82.0	84.4	74.9
	2500	32.0	8.0	73.3	76.7	82.4	85.3	86.7	77.1

Tabelle 3.5: Erfolgsquoten in Prozent für Greedy-Routing bei ungleichmäßiger Verteilung der Knoten auf einem Gebiet der Größe 1000×1000. Anzahl der Zentren: 25, Streuweite 100.

	n	$\deg(G)$	$\delta(p)$	orig.	ω^H	ω_2^D	ω_4^D	ω^E	3D
UDG	1750	7.5	25.4	11.7	54.2	47.5	45.5	45.2	64.0
$r = 50$	2000	8.4	22.9	12.4	53.3	48.7	46.9	46.2	63.7
	2500	10.2	20.5	14.3	51.2	50.0	48.3	49.0	62.9
	3000	12.0	18.0	16.5	51.7	51.1	51.3	51.7	64.0
	4000	15.7	16.3	20.2	53.9	54.8	55.8	56.8	65.3
QUDG	1750	7.4	23.0	11.4	53.0	44.4	44.3	43.0	63.5
$r = 58$	2000	8.2	20.9	12.5	51.7	46.6	46.0	45.3	62.8
	2500	10.0	18.1	14.2	50.4	48.0	48.7	48.7	62.5
	3000	11.8	16.4	16.7	51.8	49.8	51.5	51.5	63.6
	4000	15.5	15.0	20.4	53.0	54.2	56.3	56.8	64.4
Waxman	1750	6.0	15.8	9.6	40.8	39.1	39.0	39.1	51.8
$r = 93$	2000	6.7	14.5	10.8	40.0	39.9	40.3	39.4	51.1
	2500	8.1	12.7	13.7	39.5	41.8	42.8	45.5	49.8
	3000	9.6	11.8	16.4	40.0	43.6	46.1	48.8	50.0
	4000	12.6	10.9	20.9	41.3	46.6	50.5	52.9	51.8

Tabelle 3.6: Erfolgsquoten beim Greedy-Routing bei Knotenverteilung mit Hindernissen, die durch die Karte in Abbildung 3.22 definiert werden.

	n	$\deg(G)$	$\delta(p)$	orig.	ω^H	ω_2^D	ω_4^D	ω^E	4D
UDG	1200	4.6	13.9	13.8	50.7	50.2	50.3	49.2	60.1
$r = 100$	1600	6.0	11.6	30.5	64.2	65.3	66.6	67.5	70.2
	2000	7.4	10.7	51.6	75.7	77.0	79.3	80.5	78.8
	2500	9.3	10.0	72.9	85.8	87.6	89.6	90.2	86.3
	3000	11.2	9.6	85.9	91.6	93.3	94.7	95.4	90.8
	4000	14.9	9.1	96.5	97.2	98.0	98.6	98.9	95.6
	5000	18.6	8.8	99.0	99.0	99.3	99.6	99.7	97.7
QUDG	1200	4.8	12.1	18.1	53.6	54.1	55.2	55.6	61.6
$r = 116$	1600	6.2	10.3	38.6	66.4	68.0	71.5	72.7	71.3
	2000	7.8	9.5	59.4	76.2	79.1	82.7	84.1	78.4
	2500	9.7	8.9	79.1	84.8	88.1	91.1	92.4	84.3
	3000	11.7	8.6	90.0	90.5	93.0	95.4	96.4	88.5
	4000	15.6	8.1	97.5	96.0	97.7	98.7	99.0	93.3
	5000	19.5	7.9	99.3	98.1	99.0	99.6	99.7	95.9
Waxman	1200	3.7	10.8	9.9	39.9	40.9	42.0	42.8	49.4
$r = 150$	1600	4.8	9.2	20.5	44.4	48.9	51.7	53.1	52.1
	2000	5.9	8.4	33.8	49.5	56.1	59.3	61.7	54.2
	2500	7.3	7.8	49.0	54.7	63.2	66.3	69.5	57.2
	3000	8.8	7.4	59.9	59.4	68.0	72.0	74.6	59.6
	4000	11.6	7.0	71.5	65.0	74.3	77.5	80.0	62.4
	5000	14.7	6.7	77.0	69.4	77.3	80.4	82.8	65.4

Tabelle 3.7: Erfolgsquoten in Prozent für Greedy-Routing bei gleichmäßiger Verteilung der Knoten auf einem Gebiet der Größe $1000 \times 1000 \times 1000$.

	n	deg(G)	$\delta(p)$	100%	50%	25%	10%	5%	original
UDG	600	4.7	22.0	60.5	44.5	34.3	31.2	35.6	15.9
$r = 50$	800	6.1	18.1	63.0	57.9	55.3	52.5	53.8	23.6
	1000	7.5	15.7	74.8	73.6	71.6	71.2	70.5	42.1
	1250	9.3	14.5	86.0	85.0	81.6	77.7	81.2	66.1
	1500	11.2	13.8	91.9	91.6	89.5	86.1	87.0	82.2
	2000	15.0	13.2	98.0	97.5	92.8	85.8	89.9	96.3
	2500	18.8	12.8	99.0	99.4	95.9	80.7	90.8	99.2
QUDG	600	4.7	20.2	58.3	45.4	37.6	33.9	36.9	15.3
$r = 58$	800	6.1	16.2	63.5	60.3	59.5	56.6	53.3	26.5
	1000	7.5	14.2	73.3	73.0	72.3	70.1	71.7	45.9
	1250	9.4	13.1	84.8	84.7	82.3	79.3	81.0	70.0
	1500	11.3	12.6	91.3	90.8	90.2	85.8	82.2	85.5
	2000	15.1	11.9	96.8	97.0	93.2	86.1	86.4	97.1
	2500	18.9	11.5	98.9	98.9	91.7	92.0	87.1	99.4
Waxman	600	4.6	11.6	50.2	49.7	47.6	45.1	44.3	19.9
$r = 93$	800	6.0	10.0	57.1	56.9	54.8	51.7	51.3	38.3
	1000	7.5	9.2	62.6	62.6	60.0	57.8	59.1	57.3
	1250	9.4	8.6	68.1	68.5	64.7	64.5	62.6	73.4
	1500	11.3	8.3	72.5	72.3	68.6	62.5	66.9	81.7
	2000	15.1	7.9	77.7	77.9	73.7	67.1	70.9	88.4
	2500	18.8	7.6	81.6	81.3	73.6	68.5	69.8	90.8

Tabelle 3.8: Erfolgsquoten beim Greedy-Routing bei gleichmäßiger Verteilung der Knoten auf einem Gebiet der Größe 1000×1000. Rekonstruktion über Hop-Distanzen mit unvollständigen Distanzmatrizen.

Kapitel 4

Randerkennung

Sowohl bei der Rekonstruktion einer Einbettung, die möglichst nahe an die tatsächliche herankommen soll, als auch beim Routen mit den rekonstruierten Koordinaten ist aufgefallen, dass „Löcher“ im Netzwerk störend wirken. Solche Bereiche ohne Knoten im Netzwerk sorgen dafür, dass die Abschätzung der Entfernung zweier Knoten über einen kürzesten Weg unter Umständen nicht besonders gut ist. Nämlich dann, wenn der Kantenzug in der Einbettung stark von der direkten Verbindung durch eine gerade Linie abweicht.

Die Frage, die wir im Folgenden untersuchen wollen, lautet, wie diese Bereiche erkannt werden können, und ob die Kenntnis darüber sinnvoll genutzt werden kann.

4.1 Einleitung

Zunächst zwei Beispiele, um das Problem etwas zu veranschaulichen. In Abbildung 4.1 sehen wir links einen Unit-Disk-Graphen, in dessen Einbettung in der Mitte sehr deutlich ein größerer Bereich ohne Knoten erkennbar ist. Eine solche Situation entspricht dem, was man sich wohl unter einem „Loch“ vorstellt.

In dem gestörten Gittergraphen daneben ist das für die Einbettung noch störendere Loch nicht auf den ersten Blick zu erkennen. Störender für die Einbettung ist dieses Loch hauptsächlich deswegen, weil hier die Diskrepanz zwischen euklidischer Distanz in der Einbettung und der Distanz im Graphen einiger Knotenpaare deutlich größer ist. Einige Knoten liegen in der Einbettung sehr nahe beieinander, haben aber dennoch im Graphen eine recht große

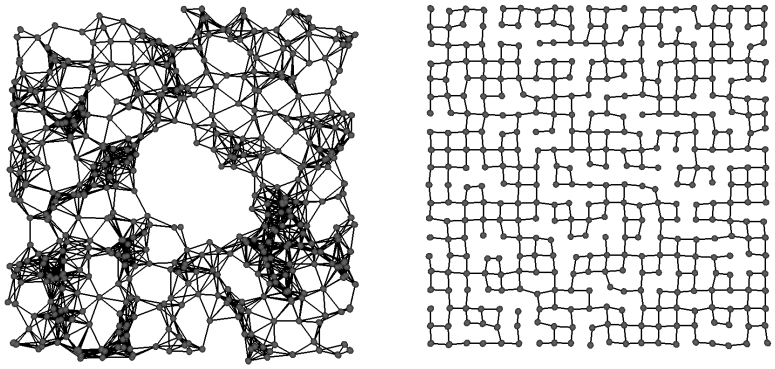

Abbildung 4.1: Ein eingebetteter Unit-Disk-Graph mit einem offensichtlichen Loch in der Mitte (links) und einen weiteren mit einem nicht ganz so offensichtlichen Loch (rechts, das Loch befindet sich in der rechten Hälfte ungefähr in der Mitte).

Distanz. Und auch wenn dieses Loch nicht so sehr der Anschauung entspricht, hat diese Struktur einen enormen Einfluss auf die Rekonstruktion der Koordinaten mittels Multidimensionaler Skalierung.

Die Erkennung dieser Löcher ist ein häufig diskutiertes Problem, mit teils sehr unterschiedlichen Ansätzen. In [FGG06] werden die Positionen der Knoten als gegeben angenommen. Die Bestimmung der Löcher läuft über geometrische Verfahren (Voronoi-Diagramme bzw. Delaunay-Triangulierung). Sie ist dabei interessanterweise auch verteilt möglich, obwohl in der Delaunay-Triangulierung sehr lange Kanten enthalten sein können, die nicht ohne Weiteres nur durch lokale Informationen bestimmt werden können.

Bei [Fun05] wird das Netzwerk von einigen ausgewählten Knoten geflutet, was im Grunde nichts anderes ist als ein verteilter Breitensuchdurchlauf. Alle Knoten im Netzwerk erhalten dadurch eine Distanzinformation zu diesen sogenannten *Beacons*. Im kontinuierlichen Fall laufen dadurch gewissermaßen Wellen durch das Netzwerk, die an den Löchern unterbrochen werden. Im diskreten Fall (den man bei Graphen ja hat) gibt es bei ausreichender Knotendichte und Knotengrad immer noch ähnliche Effekte. Über eine Analyse der Zusammenhangskomponenten der von den Knoten mit gleicher Distanz zu einem solchen Beacon induzierten Teilgraphen lassen sich Randknoten von Löchern identifizieren.

Auch in [WGM06] ist die Basis der Erkennung eine Breitensuche. Hier werden jedoch erst sogenannte *cut pairs* identifiziert. Das sind benachbarte Knotenpaare (u, v), deren Distanz zum ersten gemeinsamen Vorfahr $LCA(u, v)$ im

Breitensuchbaum eine Distanz von mindestens δ_1 hat. Als zweite Bedingung sollen die daran beteiligten Äste zwischendurch weit genug auseinander sein, d.h. die maximale Distanz zwischen zwei Knoten in diesen beiden Ästen soll mindestens δ_2 betragen.

Diese Verfahren funktionieren sehr gut, wenn der durchschnittliche Knotengrad sehr hoch ist und die zu erkennenden Löcher auch recht groß sind. Auf dünnen Netzwerken, oder auch zur Erkennung kleinerer Löcher im Netzwerk sind sie oft nicht sonderlich gut geeignet.

Andere Verfahren setzen auch explizit einen Unit-Disk-Graphen voraus, oder wenigstens einen q-QUDG mit einem nicht zu kleinen Parameter q. Ein Beispiel ist [KFPF06]. Dort werden bestimmte lokale Strukturen im Netzwerk gesucht (sogenannte *flowers*), von denen man einige Knoten sicher als „innere Knoten“ identifizieren kann, die nicht am Rand bzw. an einem Loch liegen. Diese Strukturen werden in einem zweiten Schritt erweitert, so dass mehr und mehr Knoten dem Inneren des Netzwerks zugeordnet werden können, bis schließlich nur die Randknoten übrig bleiben.

Wir betrachten hier eine Methode, die auch auf Graphen mit geringem durchschnittlichen Knotengrad gut funktioniert, und die auch nicht voraussetzt, dass der Graph ein (Quasi-)Unit-Disk-Graph ist.

4.2 Erkennung von Löchern durch Aufteilung von Kreisen

Die Grundidee des hier vorgestellten Verfahrens ist es, mit einem großen Kreis zu starten, und diesen nach und nach wie eine Schlinge zusammenzuziehen bzw. in kleinere Kreise aufzuteilen, bis am Ende nur Kreise übrigbleiben, die nicht weiter verkleinert werden können. Diese Kreise enthalten dann jeweils ein anschauliches „Loch“.

Definition 29 (distanzerhaltend) Sei $G = (V, E)$ ein Graph und $G' = (V', E')$ ein Teilgraph von G. Der Teilgraph G' heißt *distanzerhaltend*, wenn für jedes Knotenpaar (u, v) die Länge eines kürzesten Weges von u nach v in G identisch ist mit der Länge eines kürzesten Weges von u nach v in G', also

$$\text{dist}_G(u, v) = \text{dist}_{G'}(u, v) \text{ für alle } u, v \in V'$$

Wir versuchen nun, solche distanzerhaltenden Kreise in G zu finden.

Dafür suchen wir zunächst einen „sehr großen“ Kreis, der möglichst das gesamte Gebiet umfasst. Dabei ist das Maß für die Größe des Kreises nicht (wie sonst üblich in der Graphentheorie) die Anzahl der Kanten auf dem Kreis, sondern eher die Fläche, die der Kreis in dem ausgelegten Graphen einschließt. Anschließend teilen wir diesen Kreis auf, indem zwischen zwei gegenüberliegenden Knoten eine „Abkürzung“ gesucht wird. Die daraus entstehenden zwei kleineren Kreise werden rekursiv weiter geteilt, bis ein distanzerhaltender Kreis gefunden wurde, oder die Größe des Kreises unter eine vorher festgelegte Schranke fällt.

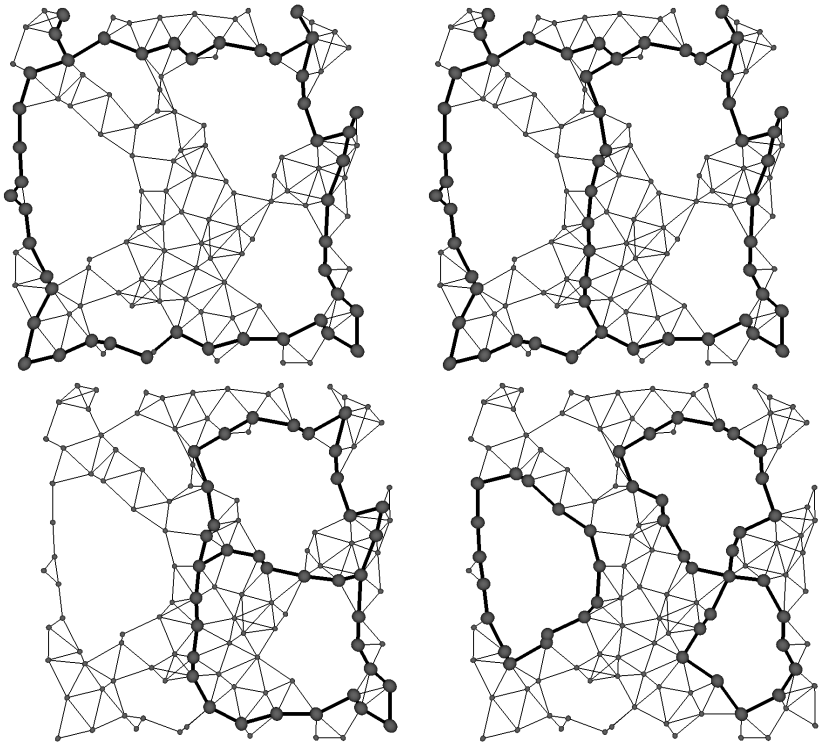

Abbildung 4.2: Grundidee des Verfahrens: Ein initialer großer Kreis wird in kleinere Kreise aufgeteilt, die rekursiv weiter aufgeteilt werden, bis am Ende nur die „Löcher“ übrig bleiben (unten rechts).

4.2.1 Bestimmung des initialen Kreises

Ob alle Löcher in dem Graphen gefunden werden, hängt davon ab, ob der initiale Kreis, mit dem das Verfahren startet, auch alle Löcher umschließt. Ideal wäre es, wenn dieser Kreis entlang des äußeren Randes des Netzwer-

kes verlaufen würde. Diesen äußeren Rand zu erkennen ist jedoch Teil des Problems, das wir hier zu lösen versuchen.

Wir verfolgen hier den Ansatz, zunächst einige Stützknoten zu finden, die dann zu einem Kreis verbunden werden. Je nach Aufbau des Graphen sind dabei unterschiedliche Heuristiken zielführend. Eine, die sich in vielen Fällen als sehr brauchbar herausgestellt hat, ist in Algorithmus 8 zusammengefasst.

Das ist eine Erweiterung der Methode, die wir auch im letzten Kapitel verwendet haben - die ersten beiden Stützknoten sind genau die, die wir bei der Multidimensionalen Skalierung für die Initialisierung der virtuellen Koordinaten benutzt haben. Die weiteren Stützknoten wählen wir aber nicht wie in Algorithmus 7 basierend auf der minimalen Distanz zu den bereits gewählten Knoten, sondern basierend auf der Summe der Distanzen zu eben diesen Knoten.

Die Suche nach weiteren Knoten wird abgebrochen, wenn dabei ein bereits gewählter Knoten oder ein Nachbar eines bereits gewählten Knotens ausgewählt werden würde.

Diese Knotenwahl, und auch das Abbruchkriterium mögen auf den ersten Blick merkwürdig erscheinen, funktionieren aber auf vielen Netzwerken sehr gut. Es werden dabei auch tatsächlich nur Knoten am Rand gewählt, und keine Knoten im Inneren des Netzwerkes. Das lässt sich auch theoretisch mit einer Aussage aus der Geometrie begründen, die in einer allgemeineren Fassung in [Bau58] bewiesen wird.

Lemma 30 *Sei $P = (p_1, \ldots, p_n) \subset \mathbb{R}^2$ ein konvexes Polygon und $D : P \to \mathbb{R}$ die Funktion, die jedem Punkt $p \in P$ die Summe der Abstände von p zu den Eckpunkten des Polygons zuordnet:*

$$D(p) = \sum_{i=1}^{n} \|p - p_i\|$$

Dann gibt es ein $1 \leq k \leq n$, so dass für alle Punkte $p \in P$ gilt:

$$D(p) \leq D(p_k)$$

Beweis. Betrachtet man die Funktion D nur auf den endlich vielen Eckpunkten $p_1, \ldots, p_n$ des Polygons, dann gibt es offensichtlich ein k mit $D(p_j) \leq D(p_k)$ für alle $1 \leq j \leq n$. Weiter lässt sich ein Punkt p innerhalb des Polygons

Algorithmus 8 : Bestimmung eines großen Kreises in G.

Eingabe : Ein Graph $G = (V, E, p)$
Ausgabe : Ein Kreis C in G

`/* Stützknoten` P `des Kreises finden */`
Initialisiere eine Menge P von Knoten
Wähle einen zufälligen Knoten $w \in V$
Wähle einen Knoten $p \in V$ mit $\mathrm{dist}(p, w)$ maximal
while $P \cap N(p) = \emptyset$ **do**
    Setze $P = P \cup p$
    Wähle ein p' mit $\sum_{p \in P} \mathrm{dist}(p, p')$ maximal
    Setze $p = p'$
end
`/* Kreis` C `aus den Stützknoten bilden */`
Wähle ein Knotenpaar $c_l, c_r \in P$ mit $\mathrm{dist}(c_l, c_r)$ minimal
Setze $C = \mathrm{kw}(c_l, c_r)$; `/*` $\mathrm{kw}(u, v)$`: Kürzester Weg von` u `nach` v `*/`
while *Kreis* C *enthält nicht alle Knoten aus* P **do**
    Bestimme $p_l \in P \setminus C$ mit $\mathrm{dist}(p_l, c_l)$ minimal
    Bestimme $p_r \in P \setminus C$ mit $\mathrm{dist}(p_r, c_r)$ minimal
    if $\mathrm{dist}(p_l, c_l) \leq \mathrm{dist}(p_r, c_r)$ **then**
        Setze $C = \mathrm{kw}(p_l, c_l) + C$
        Setze $c_l = p_l$
    else
        Setze $C = C + \mathrm{kw}(c_r, p_r)$
        Setze $c_r = p_r$
end
Setze $C = C + \mathrm{kw}(c_r, c_l)$

als Konvexkombination der Eckpunkte schreiben

$$p = \sum_{j=1}^{n} \lambda_j p_j$$

mit $\sum_{j=1}^{n} \lambda_j = 1$ und $0 \leq \lambda_j$. Damit folgt

$$\begin{aligned} D(p) &= \sum_{i=1}^{n} \|p - p_i\| = \sum_{i=1}^{n} \Big\| \sum_{j=1}^{n} \lambda_j p_j - p_i \Big\| \\ &= \sum_{i=1}^{n} \Big\| \sum_{j=1}^{n} \lambda_j (p_j - p_i) \Big\| \qquad (\textstyle\sum \lambda_j = 1) \end{aligned}$$

$$
\begin{aligned}
&\leq \sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_j \|p_j - p_i\| && \text{(Dreiecksungleichung)} \\
&= \sum_{j=1}^{n}\sum_{i=1}^{n} \lambda_j \|p_j - p_i\| = \sum_{j=1}^{n} \lambda_j \sum_{i=1}^{n} \|p_j - p_i\| && \text{(Umsortieren)} \\
&= \sum_{j=1}^{n} \lambda_j D(P_j) \leq \sum_{j=1}^{n} \lambda_j D(P_k) = D(p_k)
\end{aligned}
$$

□

Natürlich gilt diese Aussage nicht exakt so im diskreten Fall. Aber dennoch lässt sich in sehr vielen Fällen beobachten, dass erstens sämtliche so gefunden Stützknoten in den Ecken des Netzwerkes befinden und zweitens, dass der Selektionsprozess sehr schnell abbricht, weil man wieder bei einem bereits markierten Knoten oder einem seiner Nachbarn landet.

In der zweiten Phase werden diese Stützknoten zu einem Kreis verbunden. Wenn dieser Kreis über die eingebetteten Knoten eine möglichst große Fläche einfassen soll, dann muss die Länge des Kreises möglichst klein sein. Das bedeutet, dass wir hier eine Variante des NP-vollständigen Travelling-Salesman-Problems haben.

Zwar ist das bei den Instanzgrößen, mit denen wir es hier zu tun haben (in der Regel weniger als 10 Knoten), auch sehr schnell exakt lösbar, indem einfach alle möglichen Kreise durchgetestet werden. Aufgrund der besonderen Wahl der Knoten ist aber eine einfache Nächster-Nachbar-Heuristik wie sie in Algorithmus 8 skizziert ist, vollkommen ausreichend.

Der initiale Kreis über die Stützknoten $P \subset V$ wird stückweise aufgebaut, ausgehend von einem Paar $p_l, p_r \in P$ mit minimaler Distanz. Ein kürzester Weg zwischen diesem Knotenpaar bildet das erste Teilstück des aufzubauenden Kreises. Dieses wird stückweise erweitert, indem ein verbleibender Stützknoten mit minimaler Distanz zu einem Ende des Weges gewählt, und der Weg entsprechend erweitert wird. Abschließend wird dieser Weg über alle Stützknoten durch einen kürzesten Weg zwischen den beiden Endknoten zu einem Kreis geschlossen.

Für die Laufzeitabschätzung ist die Anzahl der Stützknoten von Bedeutung. In den meisten Graphen mit einer zufälligen Knotenverteilung auf einem rechteckigen Gebiet werden deutlich weniger als zehn Knoten gefunden, unabhängig von der Knotenzahl im Graphen. In dichten Graphen auf einem rechteckigen Gebiet werden meistens vier Knoten gefunden, was genau der Anschauung entspricht.

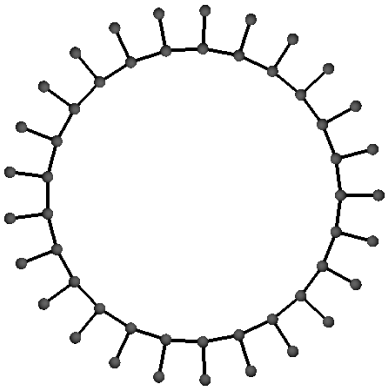

Abbildung 4.3: Ein Graph, für den $\mathcal{O}(n)$ viele Stützknoten für den initialen Kreis gefunden werden, bis das Verfahren abbricht.

Im Worst-Case werden jedoch $\mathcal{O}(|V|)$ viele Knoten gefunden, bis das Verfahren abbricht. Ein Beispiel dafür ist in Abbildung 4.3 zu sehen. In diesem Graphen werden alle Knoten mit Grad 1 auf dem äußeren Ring markiert. Der initiale Kreis ist dann kein einfacher Kreis und enthält alle Knoten des Graphen. Die Knoten auf dem inneren Kreis werden dabei alle doppelt besucht.

Für die Abschätzung der Laufzeit nehmen wir später an, dass das Verfahren auch dann abgebrochen wird, wenn eine vorher festgelegte konstante Anzahl an Stützknoten erreicht wird.

4.2.2 Bestimmung der Löcher

Für diesen initialen Kreis C im Graphen G wird im nächsten Schritt überprüft, ob er distanzerhaltend ist, d.h. ob $\text{dist}_C(u, v) = \text{dist}_G(u, v)$ für alle $u, v \in C$ gilt.

Dafür müssen aber nicht alle $\mathcal{O}(|C|^2)$ viele Knotenpaare getestet werden. Es reicht, gegenüberliegende Knotenpaare zu überprüfen.

Lemma 31 *Sei $G = (V, E)$ ein ungewichteter Graph und*

$$C = (c_0, c_1, \ldots, c_{n-1}, c_n = c_0)$$

ein Kreis in G. Wenn der Kreis C nicht *distanzerhaltend ist, dann existiert ein $0 \leq k < n$ und ein $k' = (k + \lfloor \frac{n}{2} \rfloor) \bmod n$ mit*

$$\text{dist}_C(u_k, u_{k'}) > \text{dist}_G(u_k, u_{k'})$$

Beweis. Für den Beweis benötigen wir eine Bezeichnung für die Laufrichtung im Kreis C, um für zwei Knoten c_i und c_j die beiden möglichen einfachen Wege zu unterscheiden. Seien also (alle Indizes modulo n)

$$\begin{aligned} C^+(i,j) &:= (c_i, c_{i+1}, \ldots, c_{j-1}, c_j) \\ C^-(i,j) &:= (c_i, c_{i-1}, \ldots, c_{j+1}, c_j) \end{aligned}$$

die Wege von c_i nach c_j im Uhrzeigersinn bzw. entgegengesetzt des Uhrzeigersinnes.

Wenn der Kreis nicht distanzerhaltend ist, dann existiert ein Paar $0 \leq i, j < n$ mit

$$\text{dist}_C(u_i, u_j) > \text{dist}_G(u_i, u_j).$$

Falls $j = (i + \lfloor \frac{n}{2} \rfloor) \bmod n$ oder $(j + \lfloor \frac{n}{2} \rfloor) \bmod n = i$, dann folgt die Aussage direkt.

Sei nun ohne Beschränkung der Allgemeinheit $C^+(i,j) < C^-(i,j)$ (ansonsten vertausche i und j), und damit auch $j \leq i + \lfloor \frac{n}{2} \rfloor$

Dann gilt

$$\begin{aligned} \text{dist}_C(u_i, u_{i+\lfloor \frac{n}{2} \rfloor}) &= \left|(C^+(i,j)\right| + \left|C^+(j, i + \lfloor \frac{n}{2} \rfloor)\right| \\ &= \text{dist}_C(u_i, u_j) + \text{dist}_C(u_j, u_{i+\lfloor \frac{n}{2} \rfloor}) \\ &> \text{dist}_G(u_i, u_j) + \text{dist}_C(u_j, u_{i+\lfloor \frac{n}{2} \rfloor}) \\ &\geq \text{dist}_G(u_i, u_j) + \text{dist}_G(u_j, u_{i+\lfloor \frac{n}{2} \rfloor}) \\ &= \text{dist}_G(u_i, u_{i+\lfloor \frac{n}{2} \rfloor}) \end{aligned}$$

□

Für Graphen mit Kantengewichten gilt diese Aussage in dieser Form im Allgemeinen nicht, da dort die Gleichheit in der jeweils ersten Zeile der Beweisführung nicht gelten muss.

Das gilt auch schon für eingebettete Unit-Disk-Graphen und euklidische Kantengewichte, also nicht nur für ganz beliebig gewichtete Kanten. Betrachten wir dazu den Graphen in Abbildung 4.4. Der fett markierte Kreis ist offensichtlich *nicht* distanzerhaltend. Die längeren Kanten auf dem Kreis im oberen Bereich haben die Länge 1, die kürzeren die Länge $0.5 + \varepsilon$ für beliebig kleines $\varepsilon > 0$.

Für Hopdistanzen läuft ein kürzester Weg auf dem Kreis von u_i nach v über Knoten u_j und kann entsprechend abgekürzt werden.

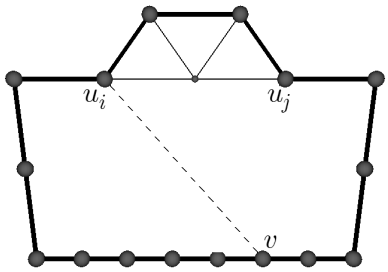

Abbildung 4.4: Die Methode, nur gegenüberliegende Knoten zu testen, kann mit Kantengewichten fehlschlagen.

Mit euklidischen Distanzen ist jedoch der Weg auf dem Kreis von u_i nach v entgegen des Uhrzeigersinnes kürzer, und auch kürzer als die mögliche Abkürzung. Entsprechendes gilt auch für alle andere Knotenpaare mit einer Hopdistanz von $\frac{|C|}{2}$. Ein kürzester Weg zwischen solchen Knoten verläuft nie über die gesuchte Abkürzung, so dass diese Abkürzung nicht gefunden wird und der Kreis fälschlicherweise als distanzerhaltend erkannt wird.

Wenn man jedoch „gegenüberliegende Knotenpaare“ anders definiert, dann gilt die Aussage auch für Graphen mit positiven Kantengewichten.

Lemma 32 *Sei $G = (V, E)$ ein gewichteter Graph mit Kantengewichtsfunktion $f : E \to \mathbb{R}$ und $f(e) > 0$ für alle $e \in E$. Sei $C = (c_0, c_1, \ldots, c_{n-1}, c_n = c_0)$ ein Kreis in G und $|C|$ die Länge des Kreises. Weiter sei für einen Knoten c_k auf C der Knoten $c_{k'}$ definiert durch die Bedingungen*

$$C^+(k, k') \leq \frac{|C|}{2} \text{ und}$$
$$C^+(k, k' + 1 \bmod n) > \frac{|C|}{2}.$$

Wenn der Kreis C nicht *distanzerhaltend ist, dann existiert ein $0 \leq k < n$ mit*

$$\mathrm{dist}_C(u_k, u_{k'}) > \mathrm{dist}_G(u_k, u_{k'})$$

Der Beweis läuft dann völlig analog zu dem einfachen Fall.

Solche Fälle wie in dem Beispiel sind aber sehr konstruiert und treten in den allermeisten Netzwerken nicht auf. In der Implementierung wurde bei Verwendung von Kantengewichten (euklidische Distanzen oder in Stufen diskret abgeschätzte euklidische Distanzen) nur die einfache Variante verwendet, und die Ergebnisse zeigen, dass dies praktisch immer ausreicht.

Tatsächlich funktioniert das hier vorgestellte Verfahren mit euklidischen Distanzen deutlich besser - zumindest dann, wenn man die Güte des Verfahrens daran misst, wie „schön“ die sichtbaren Löcher markiert werden, trotz der theoretisch möglichen Fehler bei der Betrachtung nur gegenüberliegender Knotenpaare.

Algorithmus 9 : Kreis aufteilen

Eingabe : Graph $G = (V, E)$
Kreis $C = (c_0, \ldots, c_{n-1}, c_n = c_0)$ in G
Abbruchgrenze Treshold

```
1  Procedure TeileKreis(Kreis C)
2      for i = 0 ... n − 1 do
3          Berechne DC_i := dist_C(c_i, c_{(i+⌊n/2⌋) mod n});
4          Berechne DG_i := dist_G(c_i, c_{(i+⌊n/2⌋) mod n});
5      end
6      if DC_i ≤ DG_i ∀i then
7          gebe C als distanzerhaltenden Kreis aus;
8      else
9          Wähle ein i mit DC_i > DG_i;
           /* Bilde zwei neue Kreise (alle Indizes mod n)      */
10         Setze C_1 := kw_G(c_i, c_{(i+⌊n/2⌋)}) + C^+(i + ⌊n/2⌋ + 1, i)
11         Setze C_2 := kw_G(c_{(i+⌊n/2⌋)}, c_i) + C^+(i + 1, i + ⌊n/2⌋)
           /* rekursiv fortführen                              */
12         if |C_1| > Treshold then Teilekreis(C1);
13         if |C_2| > Treshold then Teilekreis(C2);
14     end
15 end
```

Das Prinzip des Verfahrens ist in Algorithmus 9 zusammengefasst. Über den zusätzlichen Parameter *Treshold* kann dabei gesteuert werden, ab welcher Größe ein distanzerhaltender Kreis als Loch ausgegeben werden soll.

Die Wahl in Zeile 9 wird dabei so durchgeführt, dass eine möglichst gute Abkürzung gefunden wird, d.h. die Differenz $DC_i - DG_i$ sollte möglichst groß sein.

4.3 Laufzeit

So wie der Algorithmus selbst in mehreren Phasen abläuft, muss auch die Abschätzung der Laufzeit gestaffelt erfolgen.

Im Folgenden sei für einen Graphen $G = (V, E)$ die Knotenzahl $|V| = n$ und die Kantenzahl $|E| = m$.

Initialisierung

Im ersten Schritt werden einige Stützknoten gesucht. Als Auswahlkriterium wird die Distanz der Knoten zu den bereits gewählten Knoten benötigt. Die Berechnung der Distanzen geschieht über die üblichen Verfahren für das Single-Source-Shortest-Path Problem (SSSP).

Die Laufzeit für SSSP liegt mit Dijkstras Algorithmus in $\mathcal{O}(m + n \log n)$. Anschließend ist ein Knoten mit maximaler Distanz zu den bisher gefundenen in Linearzeit $\mathcal{O}(n)$ bestimmbar. In den für uns interessanten Netzwerken können wir weiter annehmen, dass die Kantenzahl linear in der Anzahl der Knoten ist, so dass sich die Abschätzung auf $\mathcal{O}(n \log n)$ vereinfacht.

Da wir weiterhin die Zahl der Stützknoten explizit auf eine konstante Zahl beschränken, bleibt es bei $\mathcal{O}(n \log n)$ für die Suche nach allen Stützknoten. Der Aufbau des Kreises über all diese Stützknoten geht ebenfalls in $\mathcal{O}(n)$, wenn bei der Ermittlung der Distanzen auch die Wege gespeichert werden.

Die Laufzeit für den Aufbau des initialen Kreises liegt somit in $\mathcal{O}(n \log n)$.

Aufteilungsphase

Die Abschätzung der Laufzeit des Verfahrens kann sehr unübersichtlich werden, wenn man sie sehr detailliert durchführen will. Wir belassen es hier bei einer recht groben Abschätzung.

Die Laufzeit der Funktion *TeileKreis* in Algorithmus 9 wird dominiert durch das Finden der kürzesten Wege in G bzw. der Berechnung der Distanzen DG_i. Die Distanzen DC_i auf dem Kreis C sind insgesamt in $\mathcal{O}(|C|)$ berechenbar, da aus DC_i der nächste Wert DC_{i+1} in konstanter Zeit ermittelt werden kann. Für die Abschätzung nehmen wir an, dass alle kürzesten Wege im Vorfeld bestimmt wurden. Tatsächlich kommt man mit weniger aus, und zusätzlich kann man die Suche nach einem kürzesten Weg in G abbrechen, wenn der jeweilige Breitensuche-Baum eine Tiefe von $|C|/2$ erreicht hat. Denn dann ist

bereits klar, dass es in G keinen kürzeren Weg geben kann als in C. Als grobe Abschätzung nehmen wir dennoch (wieder unter der Annahme $|E| \in \mathcal{O}(n)$) einen Aufwand an von $\mathcal{O}(n^2 \log n)$.

Alle übrigen Operationen in Algorithmus 9 sind in Zeit $\mathcal{O}(n)$ möglich.

Zusammen mit einer Rekursionstiefe in $\mathcal{O}(\log n)$ bleibt dann neben der Bestimmung der kürzesten Wege in G ein Aufwand von $\mathcal{O}(n \log n)$.

Insgesamt liegt damit die Laufzeit des Verfahrens in $\mathcal{O}(n^2 \log n)$.

4.4 Probleme und Grenzen des Verfahrens

Wie andere Verfahren zur Randerkennung gibt es auch hier Schwachstellen und Probleme. Je nach Anwendungsfall lassen sich diese wegdiskutieren, oder sorgen auf der anderen Seite dafür, dass das Verfahren nicht sinnvoll anwendbar ist.

Distanzerhaltende Kreise und anschauliche Löcher

Das Verfahren beginnt mit einem zufällig gewählten Knoten, von dem aus der erste Eckknoten des initialen Kreises gesucht wird. Wie der initiale Kreis aussieht, hängt dann auch von der Wahl jenes ersten Zufallsknoten ab. Trotzdem sollten ja - sofern der initiale Kreis alle anschaulichen Löcher im Graphen umfasst - jedes Mal all diese Löcher gefunden werden.

Bei den Tests der Implementierung ist jedoch aufgefallen, dass bei Verwendung von Hopdistanzen dies nicht immer der Fall ist. Es gibt Graphen, bei denen je nach Wahl des ersten Knotens einige Löcher gefunden werden, und dann bei anderer Wahl wieder nicht. Das ist jedoch kein direkter Fehler im Verfahren, sondern eher eine Schwäche in der Definition von „Loch“ über distanzerhaltende Kreise. Eigentlich noch nicht einmal das, sondern eine Auswirkung des Problems, dass die anschauliche Definition eines Loches im ausgelegten Graphen von der Auslegung abhängt.

Betrachten wir dazu den Graphen in Abbildung 4.5 mit zwei verschiedenen Unit-Disk-Einbettungen. In der linken Einbettung wird das anschaulich gut erkennbare Loch von einem distanzerhaltenden Kreis der Länge $|C| = 6$ umfasst, in der rechten jedoch von einem Kreis der Länge $|C| = 7$, der ebenfalls distanzerhaltend ist.

Abhängig vom Verlauf des Verfahrens wird bei der Aufteilung des initialen

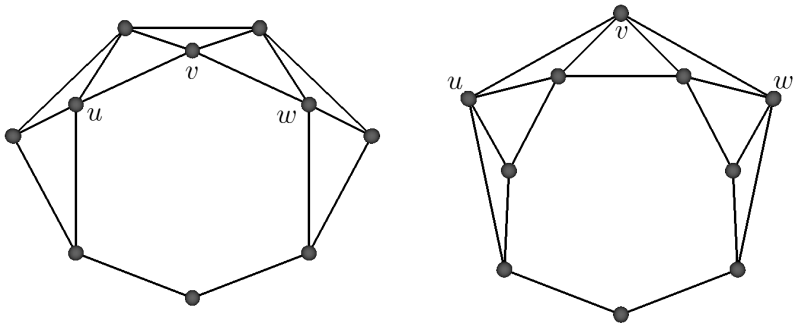

Abbildung 4.5: Ein Graph mit zwei verschiedenen Unit-Disk-Einbettungen und zwei distanzerhaltenden Kreisen unterschiedlicher Länge.

Kreises mal der Kreis mit sieben Knoten, und mal der mit sechs Knoten gefunden. Dementsprechend wird bei einer angegebenen Mindestgröße für die gesuchten Löcher an dieser Stelle des Graphen mal ein Treffer ausgegeben, und mal nicht.

Wenn man sich nicht nur auf Unit-Disk-Graphen beschränkt, verschärft sich das Problem noch etwas. Bei Verwendung von euklidischen Distanzen tritt dieses Problem nicht auf.

Ausbuchtungen am Rand und Löcher innerhalb distanzerhaltender Kreise

Ein weiteres Problem tritt auf, wenn das Netzwerk nahe des äußeren Randes Löcher hat. Diese werden manchmal nicht von dem ersten Kreis umschlossen und werden dann natürlich auch nicht durch die Suche nach Abkürzungen gefunden. Ein Beispiel dafür ist in Abbildung 4.6 (links) zu sehen.

Ein anderes Problem tritt auf, wenn das Netzwerk große distanzerhaltende Kreise enthält, und im inneren dieser Kreise ein Teilgraph liegt, der ebenfalls noch durchlöchert ist.

Beide Probleme lassen sich teilweise beheben, indem während der Aufteilung des großen initialen Kreises alle besuchten Knoten markiert werden. Anschließend werden in den Zusammenhangskomponenten des durch die nicht markierten Knoten induzierten Teilgraphen weitere Löcher gesucht.

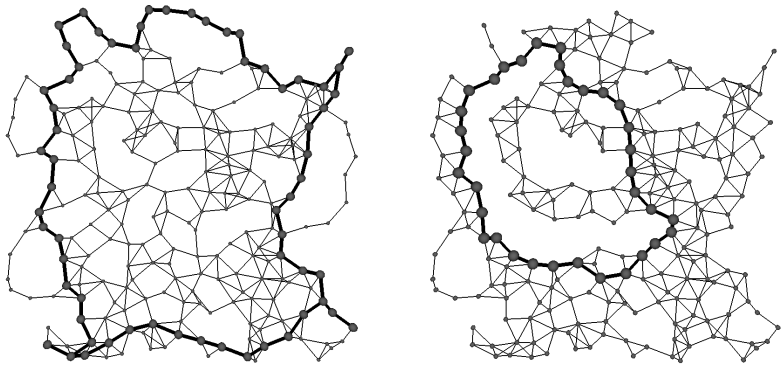

Abbildung 4.6: Ein Netzwerk mit Löchern am Rand, die nicht alle vom initialen Kreis umschlossen werden (links), und ein großer distanzerhaltender Kreis, der ein Loch enthält (rechts).

4.5 Möglicher Nutzen für die Multidimensionale Skalierung

Zur Erinnerung (vgl. Definition 15): Bei der Multidimensionalen Skalierung, über die wir aus der Graphstruktur eine sinnvolle Einbettung des Graphen ermitteln, minimieren wir den Stress $\sigma_{\Delta,W}$ definiert als

$$\sigma_{\Delta,W}(X) \ := \ \sum_{i<j} w_{ij}(\delta_{ij} - \|x_i - x_j\|)^2.$$

Dabei ist $\Delta = (\delta_{ij})$ eine Distanz- bzw. Ähnlichkeitsmatrix, und $W = (w_{ij})$ eine Matrix mit Gewichten, die die Vertrauenswürdigkeit der Distanzinformationen angibt.

Im vorigen Kapitel wurden für die Gewichte nur $w_{ij} = 1$ oder $w_{ij} = 0$ gewählt - je nachdem, ob der entsprechende Distanzwert bekannt war oder eben nicht. Zulässig sind aber beliebige Werte $0 \leq w_{ij} \leq 1$.

Motivation für die Erkennung der Löcher ist hier, die Information, dass ein Knoten am Rand eines Loches liegt, dafür zu nutzen, die Vertrauenswürdigkeit der Wege, die über diesen Knoten laufen, herabzusetzen.

Sei $c \in V$ ein Knoten in einem eingebetteten Graphen $G = (V, E, p)$, und Knoten c liegt auf einem distanzerhaltenden Kreis in G. Seien $u_1, u_2 \in V$ zwei weitere Knoten, und ein kürzester Weg zwischen u_1 und u_2 enthält den Knoten c. Dann besteht die Möglichkeit, dass dieser Weg im eingebetteten Graphen an Knoten c einen Knick macht, wodurch die ermittelte Weglänge

zwischen u_1 und u_2 weniger stark mit der euklidischen Distanz $\|p(u_1)-p(u_2)\|$ korreliert.

Es erscheint also sinnvoll, dieser Weglänge nicht das gleiche Vertrauen entgegenzubringen wie einer anderen Weglänge, die nicht durch ein Loch abgelenkt zu sein scheint. Das ließe sich dadurch erreichen, dass in diesem Beispiel $w_{1,2}$ (und natürlich auch $w_{2,1}$) auf einen Wert kleiner als 1 gesetzt wird.

Bei genauerer Betrachtung jedoch führt das zu nichts. Stichprobenartige Tests zeigen sehr klar, dass dies keine Verbesserung bringt, sondern im Gegenteil eine deutliche Verschlechterung, wenn es um die Erfolgsquoten beim Greedy-Routing geht.

Der Grund dafür ist, dass man aus der Existenz eines Randknotens c auf einem kürzesten Weg zwischen u_1 und u_2 eben nicht darauf schließen kann, wie der Weg in dem eingebetteten Graph verläuft. Auch dann nicht, wenn mehrere Randknoten in Folge auf einem kürzesten Weg liegen. Egal, welche Heuristik man hier wählt - es gibt immer Fälle, in denen genau das Gegenteil von dem erreicht wird, was eigentlich gewollt ist.

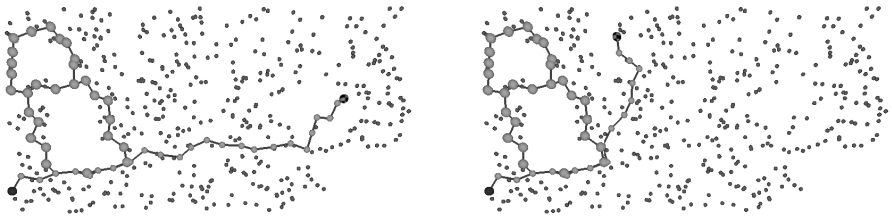

Abbildung 4.7: Kürzeste Wege entlang von Löchern geben keine sichere Auskunft über die Zuverlässigkeit der Abschätzung der Distanz.

Dabei ist es unerheblich, ob das Ziel ist, eine Einbettung zu finden, die (wahrscheinlich) näher an die tatsächliche Einbettung heranreicht, oder ob wir erreichen wollen, dass Greedy-Routing auf dieser Einbettung besser funktioniert. Für beide Ziele ist eine Herabsetzung der w_{ij}-Werte für kürzeste Wege mit Randknoten zwischen u_i und u_j keine sinnvolle Idee. Zumindest nicht ohne weitere Analyse der Graphen, die dann aber höhere Anforderungen an Knoten- bzw. Kanten-Dichte stellt.

Das hier vorgestellte Verfahren bleibt aber eine gute Methode zur Erkennung von nicht erfassten Bereichen innerhalb des Zielgebietes, das von von dem Sensornetzwerk abgedeckt werden soll. Für die Verbesserung von Greedy-Routing, oder gar einer garantierten Zustellung einer Nachricht zum Ziel ist es jedoch nicht geeignet. Damit befassen wir uns im nächsten Kapitel.

Kapitel 5

Garantiertes Routen durch Hierarchische Bipartition

In Kapitel 3 haben wir mit Hilfe der rekonstruierten Koordinaten die Erfolgsquote für Greedy-Routing verbessern können. Eine Garantie dafür, dass eine Nachricht über Greedy-Routing mit den MDS-Koordinaten am Ziel ankommt, gibt es aber nicht. Auch wenn man die Anzahl der Sackgassen, in die Greedy-Routing immer wieder mal hineinläuft, durch MDS deutlich reduzieren kann, lassen sich diese dadurch nie ganz eliminieren.

Für eine sichere Übertragung einer Nachricht vom Start- zum Zielknoten sind andere Routing-Protokolle erforderlich. Im ersten Kapitel wurden einige Verfahren kurz angerissen (z.B. *Facettenrouting*, *VCap*, *Beacon-Vector-Routing*, oder Baum-basierte Verfahren wie *HECTOR* oder *LTP*).

All diese Verfahren können mit Greedy-Routing kombiniert werden, indem sie als Ausweichprotokolle genutzt werden, wenn man mit dem Greedy-Verfahren in eine Sackgasse hineingelaufen ist. Wenn mit dem Ausweichprotokoll ein Knoten erreicht ist, der näher am Ziel liegt als der Knoten, an dem das Greedy-Routing zuvor stoppte, kann wieder auf greedy umgeschaltet werden.

In diesem Kapitel stellen wir eine neue Methode dafür vor, die auch bereits im Rahmen der *International Conference on Wireless Networks* veröffentlicht und dort von mir präsentiert wurde [GHW12] bzw. [GHW13]. Für die hier vorliegende Arbeit wurden jedoch unter anderem die experimentellen Auswertungen angepasst bzw. neu durchgeführt. So wird hier beispielsweise das einfache *landmark-based* Routing-Protokoll aus dem Paper ersetzt durch Routing auf den MDS-Koordinaten, dem wir uns in Kapitel 3 gewidmet

haben. Des Weiteren wird das Verfahren hier detaillierter und mit einigen Modifikationen untersucht.

Hierarchisches Bipartitions-Routing (HBR) verwendet Ankerknoten zur Bestimmung von virtuellen Koordinaten für die Knoten im Netzwerk. Im Gegensatz zum Beacon-Vector-Routing und ähnlichen Ansätzen ist die Eindeutigkeit der Koordinaten direkt durch das Konzept gegeben. In bestimmten Fällen kann bei HBR aus Gründen der Performance auch auf Eindeutigkeit der virtuellen Koordinaten verzichtet werden. Dann jedoch ist sichergestellt, dass alle Knoten mit gleicher Koordinate Nachbarn eines der Ankerknoten sind, und dadurch das Routing (mit einer kleinen Anpassung) dennoch garantiert zum Ziel führt.

5.1 Bestimmung der virtuellen Koordinaten

Im Folgenden betrachten wir wieder gewichtete Graphen $G = (V, E)$ mit einer Kantengewichtsfunktion $\omega : E \to \mathbb{R}_{>0}$.

Für die Bestimmung der virtuellen Koordinaten der Knoten in einem Graphen $G = (V, E)$ für HBR starten wir mit ganz ähnlich wie bei dem Verfahren zur Randbestimmung aus dem vorigen Kapitel. Der Unterschied ist nur, dass nicht beliebig viele Knoten gesucht werden, sondern nach zwei Knoten abgebrochen wird.

Zunächst wird also ein zufälliger Knoten $w \in V$ gewählt, dann ein Ankerknoten $x_0 \in V$ mit maximaler Distanz zu w, und zuletzt ein zweiter Ankerknoten $x_1 \in V$ mit maximaler Distanz zu x_0.

Anschließend wird das Netzwerk in zwei Teile aufgeteilt. Alle Knoten u mit

$$\text{dist}^\omega(u, x_0) \leq \text{dist}^\omega(u, x_1) \text{ bekommen die virtuelle Adresse 0,}$$

und jeder Knoten u mit

$$\text{dist}^\omega(u, x_0) > \text{dist}^\omega(u, x_1) \text{ bekommt die virtuelle Adresse 1.}$$

Die Knoten mit Adresse 0 induzieren den Teilgraphen $G_0 = (V_0, E_0)$, und die Knoten mit Adresse 1 entsprechend $G_1 = (V_1, E_1)$. Anschließend werden diese beiden Teilgraphen mit dem gleichen Verfahren weiter aufgeteilt.

Dass die weitere Aufteilung sinnvoll möglich ist, zeigt folgendes kleines Lemma:

Lemma 33 *Wenn der Graph G zusammenhängend ist, dann sind auch die Teilgraphen G_0 und G_1 zusammenhängend.*

Beweis. Sei v ein Nachbar von Knoten u auf einem kürzesten Weg von u nach x_0. Dann gilt offensichtlich

$$\begin{aligned} & \omega(u,v) + \text{dist}^\omega(v,x_0) = \text{dist}^\omega(u,x_0) \\ \Leftrightarrow \quad & \text{dist}^\omega(v,x_0) = \text{dist}^\omega(u,x_0) - \omega(u,v) \end{aligned}$$

Wenn $u \in V_0$, dann gilt $\text{dist}^\omega(u,x_0) \leq \text{dist}^\omega(u,x_1)$, also

$$\text{dist}^\omega(v,x_0) \leq \text{dist}^\omega(u,x_1) - \omega(u,v)$$

Mit $\text{dist}^\omega(u,x_1) \leq \omega(u,v) + \text{dist}^\omega(v,x_1)$ gilt dann

$$\begin{aligned} \text{dist}^\omega(v,x_0) &\leq \omega(u,v) + \text{dist}^\omega(v,x_1) - \omega(u,v) \\ &= \text{dist}^\omega(v,x_1) \end{aligned}$$

Damit haben alle Knoten auf einem kürzesten Weg von $u \in V_0$ nach x_0 die virtuelle Adresse 0, und damit ist G_0 zusammenhängend. Der Zusammenhang von G_1 folgt ganz analog. □

Für die sukzessive weitere Aufteilung werden die beiden Teilnetzwerke getrennt betrachtet. Für jede virtuelle Adresse $\alpha \in \{0,1\}$ wählen wir je zwei Ankerknoten $x_{\alpha\cdot 0}$ und $x_{\alpha\cdot 1}$ in G_α. Dabei bezeichnet $\alpha \cdot 0$ die Erweiterung von α um ein weiteres Symbol 0. Der Knoten $x_{\alpha\cdot 0}$ ist ein Knoten mit maximaler Distanz zu x_α in G_α, und $x_{\alpha\cdot 1}$ ein Knoten mit maximaler Distanz zu $x_{\alpha\cdot 0}$ in G_α. Alle Knoten in G_α bekommen entsprechend ihrer Distanz zu den beiden neuen Ankerknoten $x_{\alpha\cdot 0}$ und $x_{\alpha\cdot 1}$ die virtuelle Adresse $\alpha\cdot 0$ bzw. $\alpha \cdot 1$. Zuletzt wird das Teilnetz G_α weiter unterteilt in $G_{\alpha\cdot 0}$ und $G_{\alpha\cdot 1}$.

Die Wahl der Ankerknoten und die weitere Aufteilung des Netzwerkes läuft also völlig analog zur ersten Aufteilung mit dem einzigen Unterschied, dass kein initialer Hilfsknoten w gewählt wird, sondern der vorher bestimmte Ankerknoten im entsprechenden Teilgraphen diese Rolle übernimmt.

Diese Aufteilung wird fortgeführt, bis alle Teilnetze nur noch aus einem einzelnen Knoten bestehen, und somit jeder Knoten durch seine virtuelle Adresse eindeutig bestimmt ist. Wendet man Lemma 33 induktiv an, dann folgt daraus der Zusammenhang aller Teilnetze G_α, die während der Berechnung der virtuellen Koordinaten verwendet werden.

Eine Übersicht ist in Algorithmus 10 gegeben.

Algorithmus 10 : Bestimmung der virtuellen Koordinaten bei HBR

Eingabe : Ein Graph $G = (V, E)$
Ausgabe : eine virtuelle Einbettung der Knoten für
$p_{HBR} : V \to \{0, 1\}^*$

```
1  Initialisiere p_HBR(u) = λ für alle u ∈ V;
   /* λ: leeres Wort                                          */
2  Wähle zufälligen Knoten w ∈ V;
3  TeileNetzwerk(G, w)
4  Procedure TeileNetzwerk(Teilgraph G = (V, E), Knoten w ∈ V)
5      Wähle einen Knoten x_0 mit dist^ω_G(w, x_0) maximal;
6      Wähle einen Knoten x_1 mit dist^ω_G(x_0, x_1) maximal;
7      Initialisiere zwei leere Knotenlisten V_0 und V_1;
8      for alle u ∈ V do
9          if dist^ω_G(u, x_0) ≤ dist^ω_G(u, x_1) then
10             Setze p_HBR(u) = p_HBR(u) · 0;
11             Setze V_0 = V_0 ∪ {u};
12         else
13             Setze p_HBR(u) = p_HBR(u) · 1;
14             Setze V_1 = V_1 ∪ {u};
15         end
16     end
17     if |V_0| > 1 then
18         Erzeuge Teilgraph G_0, der durch V_0 induziert wird;
           TeileNetzwerk(G_0, x_0);
19     if |V_1| > 1 then
20         Erzeuge Teilgraph G_1, der durch V_1 induziert wird;
           TeileNetzwerk(G_1, x_1);
21 end
```

5.1.1 Routing-Protokoll

Das Routing-Protokoll ist recht einfach. In einem Satz salopp zusammengefasst ist es: „Solange der Zielknoten einer Nachricht in der andern Hälfte des Netzwerkes liegt, schicke die Nachricht erstmal in Richtung der richtigen Hälfte."

Genauer: Nehmen wir an, wir möchten eine Nachricht von einem Knoten s zu einem Knoten t schicken. Wenn die virtuelle Adresse von s mit einer 0 beginnt, und die virtuelle Adresse von t mit einer 1, dann befindet sich s im Teilnetzwerk G_0 und t in G_1. Die Nachricht muss also von Knoten s in Richtung des Knotens x_1 weitergeleitet werden. Die Nachricht wird solange weiter in Richtung x_1 weitergeleitet, bis ein Knoten im Teilnetzwerk G_1 erreicht wird. Dann wird die Nachricht innerhalb von G_1 weitergeleitet, indem die zweite Koordinate des Zielknotens verwendet wird, bis die Nachricht letztlich am Ziel ankommt.

Die Information, an welchen Knoten die Nachricht konkret weitergeleitet werden muss, um den Zielknoten t zu erreichen, kann während der Aufteilungsphase des Netzwerkes bestimmt werden. Wenn die Distanzen aller Knoten zu den neu gewählten Ankerknoten bestimmt werden, dann werden dabei auch Wege zu diesen Knoten aufgebaut. Ob das nun global über eine Breitensuche erfolgt (bzw. einen Kürzesten-Wege-Algorithmus, der auch Kantengewichte berücksichtigen kann), oder verteilt über ein Fluten des Netzwerkes ist dabei unerheblich. Auf jeden Fall erhält jeder Knoten in G die Information, welcher seiner Nachbarknoten auf einem kürzesten Weg zu x_0 bzw. x_1 liegt.

Allgemeiner: Sei $\alpha \cdot d_u \cdot \alpha_u$ die virtuelle Adresse von Knoten u und $\alpha \cdot d_t \cdot \alpha_t$ die Adresse des Zielknotens t mit $\alpha, \alpha_u, \alpha_t \in \{0,1\}^*$, $d_u, d_t \in \{0,1\}$ und $d_u \neq d_t$. Die Symbole links von d_u bzw. d_t stimmen also in den Adressen von u und t überein.

Wenn $d_u = 0$ und $d_t = 1$, dann wird die Nachricht in Richtung des Ankerknotens $x_{\alpha \cdot 1}$ weitergeleitet, ohne das zusammenhängende Teilnetz G_α dabei zu verlassen. Im Fall $d_u = 1$ und $d_t = 0$ wird die Nachricht entsprechend in Richtung von Ankerknoten $x_{\alpha \cdot 0}$ geschickt. Der jeweils richtige Nachbarknoten für diese Weiterleitung kann während der Aufteilung des Teilnetzes G_α bestimmt werden.

Ein induktives Argument zeigt sofort, dass bei HBR die Zustellung einer Nachricht garantiert ist.

Folgerung 34 *Eine Nachricht, die mit HBR verschickt wird, erreicht immer den Zielknoten.*

Algorithmus 11 : Routing-Protokoll für HBR an Knoten u

Eingabe : Eine Nachricht M für Zielknoten t mit Adresse $p_{HBR}(t)$

if $p_{HBR}(u) = p_{HBR}(t)$ **then**
    `/* Nachricht hat Ziel erreicht, keine weiteren Schritte notwendig */`
if $v \in N(u)$ *mit* $p_{HBR}(v) = p_{HBR}(t)$ **then**
    Sende Nachricht zu Knoten v;
else
    Bestimme längstes gemeinsames Präfix α der virtuellen Adressen mit
    $p_{HBR}(u) = \alpha \cdot d_u \cdot \alpha_u$ und $p_{HBR}(t) = \alpha \cdot d_t \cdot \alpha_t$
    if $d_t = 1$ **then**
        Leite Nachricht in Richtung $x_{\alpha \cdot 1}$;
    else
        Leite Nachricht in Richtung $x_{\alpha \cdot 0}$;
    end
end

5.1.2 Verteilte Berechnung der Adressen

Im Gegensatz zur multidimensionalen Skalierung, bei der eine zentrale Berechnung der Koordinaten nötig ist, lässt sich die Bestimmung der Koordinaten bei HBR auch leicht verteilt berechnen.

Zur Bestimmung des zufälligen Startknoten w kann ein Algorithmus zur Lösung des *Leader-Election-Problems* (Übersicht z.B. in [HKP$^+$05]) herangezogen werden. Das ausschlaggebende Kriterium für die Wahl könnten hier die eindeutigen Identifikationsnummern der Knotens sein. Alternativ kann dieser Knoten auch von außen bestimmt werden. Der so zum *Leader* ernannte Knoten flutet dann das Netzwerk. Die flutende Nachricht enthält einen Hopcounter (oder einen anderen Distanzwert, je nach gewähltem Modell des Kantengewichtes), der bei der Weiterleitung im Netzwerk um die jeweiligen Kantengewichte erhöht wird. Jeder Knoten im Netzwerk erhält so Kenntnis über seine Distanz zu Knoten w.

Für die Bestimmung der Ankerknoten x_0 und x_1 kann wieder ein beliebiger Leader-Election-Algorithmus verwendet werden. Auswahlkriterium ist für die Wahl von x_0 die Distanz zu Knoten w (bzw. für x_1 die Distanz zu dem dann bekannten x_0). Da diese Werte aber nicht zufällig im Netz verteilt sind, lässt sich hier eine deutliche Reduzierung der versendeten Nachrichten

erreichen. Jeder Knoten, der sich nicht am Rand des Netzwerkes befindet, hat mindestens einen Nachbarknoten, der eine höhere Distanz zu Knoten w hat, als er selbst. Durch eine Abfrage der Distanzwerte der Nachbarknoten können somit schon sehr viele Knoten als mögliche Leader ausgeschlossen werden.

Erkennt sich ein Knoten dadurch als Kandidat für die Leader-Rolle, schickt er seinen Distanzwert und seine ID an Knoten w, der dann die weitere Auswahl übernimmt und im Anschluss den Knoten mit dem höchsten Distanzwert als Leader bestimmt und darüber benachrichtigt.

Der Routingweg von den Kandidaten zurück zu w kann während des Flutprozesses von w gespeichert werden, und der Weg von Knoten w zum noch zu bestimmenden Ankerknoten x_0 während des Übertragens der Kandidatenliste. Dabei ist nur konstanter Speicherplatz nötig, unabhängig von der Zahl der Kandidaten. Wenn ein Knoten mehrere Kandidatenanfragen weiterleiten muss, dann muss er nur den nächsten Knoten auf dem Weg zu dem Knoten mit der größten Distanz speichern. Alle anderen Routing-Informationen können wieder verworfen werden, da sie nicht weiter benötigt werden.

Ist Knoten x_0 bestimmt, flutet er das Netzwerk, so wie zuvor Knoten w. Die dadurch ermittelbaren Distanzwerte zu x_0 dienen in der zweiten Phase als Auswahlkriterium für die zweite Runde zur Leaderwahl, in der Knoten x_1 bestimmt wird.

In den Subnetzen, die dann erstellt und weiter aufgeteilt werden, übernehmen Knoten x_0 bzw. Knoten x_1 die Rolle des Knotens w im ursprünglichen Netzwerk.

5.1.3 Kosten für den Aufbau

Für die Aufteilung eines Netzwerkes in zwei Subnetze mit dieser Methode wird zweimal eine Leader-Election durchgeführt. Damit ein Knoten seinen Status als Kandidat für diese Rolle feststellen kann, muss er im Worst-Case all seine Nachbarknoten kontaktieren, um deren Distanz zum Koordinierungsknoten mit der eigenen zu vergleichen. Im besten Fall ist dabei schon nach einem Vergleich eine größere Distanz gefunden, wodurch sich die Vergleiche mit den anderen Nachbarn erübrigen.

Für das abschließende Fluten muss abermals jeder Knoten all seine Nachbarn kontaktieren (mit Ausnahme des Knotens, von dem die Flut-Nachricht erhalten wurde).

Hinzu kommt die zielgerichtete Kommunikation zwischen den Kandidaten und dem Koordinierungsknoten w für den neu zu bestimmenden Ankerknoten. Dieser Aufwand fällt insgesamt aber kaum ins Gewicht und ist zu vernachlässigen.

Wenn die Kandidatenrolle über die maximale Distanz zum Koordinierungsknoten in der k-Hop-Nachbarschaft ($k > 1$) weiter abgesichert werden soll, sind zusätzliche $k-1$ Runden erforderlich. Ein vorzeitiger Abbruch ist dabei erst in der letzten Runde möglich.

Damit folgt:

Lemma 35 *Wenn für den Aufbau der HBR-Struktur die Kandidatenauswahl in der k-Hop-Nachbarschaft abgesichert wird, dann ist in jedem Aufteilungsschritt an jedem Knoten ein Kommunikationsaufwand zwischen $2k \cdot \deg(G)$ und $2(k+1) \cdot \deg(G)$ zu erwarten.*

Ein weiteres Aussieben der Kandidatenliste durch Überprüfung der k-Hop-Nachbarschaft ist nur auf den ersten Blick eine gute Idee. Tatsächlich erhöht das den Kommunikationsbedarf insgesamt in den meisten Fällen sehr deutlich, bringt aber keinen weiteren Nutzen.

Jeder Kandidat x' verursacht durch das Senden seiner Distanz und ID zu Knoten w insgesamt nur einen Kommunikationsaufwand, der der Länge des kürzesten Weges von x' zu w entspricht. Bei einem Netzwerk, das ein quadratisches Gebiet überdeckt, liegt die Länge eines längsten kürzesten Weges in $\mathcal{O}(\sqrt{|V|})$. Für die Überprüfung der k-Hop-Nachbarschaft hingegen muss jeder Knoten mindestens $(k-1)$-mal all seine Nachbarn kontaktieren, was einen Aufwand von $\mathcal{O}\big(k \cdot |V| \cdot \deg(G)\big)$ verursacht. Daraus folgt, dass sich die Überprüfung der k-Hop Nachbarschaft nur dann lohnt, wenn durch diese Überprüfung mindestens $\mathcal{O}(k \cdot \deg(G) \cdot \sqrt{|V|})$ viele Kandidaten ausgeschlossen werden können. Es lassen sich Beispiele dafür konstruieren (z.B. der Graph in Abbildung 4.3 aus dem vorigen Kapitel), aber in zufällig erstellten Graphen kommen solche Fälle praktisch nicht vor, wie wir in der experimentellen Auswertung sehen werden.

5.1.4 Länge der Adressen

Die Größe $|\alpha|$ der virtuellen Adressen α der Knoten hängt von der Anzahl der Aufteilungsschritte ab, bis die Teilgraphen G_α nur noch aus einzelnen Knoten bestehen. Dann ist sicher gestellt, dass jeder Knoten eindeutig durch seine virtuelle Adresse identifiziert werden kann.

Wenn wir ungewichtete Graphen betrachten, dann besteht das Worst-Case-Szenario für die Länge der Adressen in einem vollständigen Graphen - also einem Graphen, in dem jeder Knoten mit allen anderen durch eine Kante verbunden ist. In einem vollständigen Graphen mit n Knoten werden $n-1$ viele Aufteilungsschritte benötigt, und damit ist dann auch die Länge der virtuellen Adressen $n-1$.

Bei gewichteten Graphen gibt es ein weiteres Worst-Case-Szenario, nämlich einen Weg mit exponentiell ansteigenden Kantengewichten. Sei $G = (V, E)$ ein Weg mit $V = \{u_0, \ldots, u_{n-1})$ für ein $n \in \mathbb{N}$ und Kantengewichtsfunktion $\omega\big(\{u_i, u_{u+1}\}\big) = 2^i$. Dann sieht man schnell, dass auch in diesem Fall die Länge der Adressen die Anzahl n der Knoten erreicht, da in jedem Aufteilungsschritt der Teilgraph $G_{\alpha\cdot 0}$ nur aus je einem Knoten besteht.

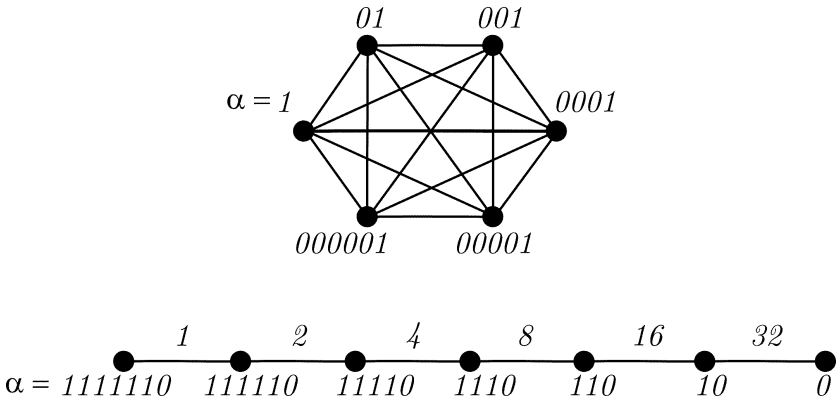

Abbildung 5.1: Worst-Case-Szenarios für die Adresslänge bei HBR.

Vermeidung des Worst-Case-Szenarios für ungewichtete Kanten

Einige dieser Situationen lassen sich vermeiden, wenn der Aufteilungsprozess gestoppt wird, wenn im Teilnetz G_α alle Knoten mit dem Ankerknoten x_α verbunden sind, also

$$G_\alpha = (V_\alpha, E_\alpha) \text{ mit } E_\alpha \supseteq \big\{\{x_\alpha, v\} \mid v \in V_\alpha, v \neq x_\alpha\big\}$$

Diese Eigenschaft von G_α ist je nach Graph und verwendeter Kostenfunktion mehr oder weniger direkt während der Aufteilungsphase erkennbar. Bei Hopdistanzen reicht es, nach der Bestimmung des nächsten Ankerknotens $x_{\alpha\cdot 0}$ die Existenz der Kante $\{x_\alpha, x_{\alpha\cdot 0}\}$ zu überprüfen. Ist diese vorhanden, dann sind alle Knoten in G_α mit x_α verbunden, da $x_{\alpha\cdot 1}$ mit am weitesten von x_α entfernt ist.

Ebenso lässt sich das bei Unit-Disk-Graphen mit Kostenfunktionen überprüfen, die monoton mit der Entfernung wachsen, also

$$\|p_{\mathbb{R}^2}(u_1) - p_{\mathbb{R}^2}(v_1)\| \leq \|p_{\mathbb{R}^2}(u_2) - p_{\mathbb{R}^2}(v_2)\| \Longrightarrow \delta^{\omega}(u_1, v_1) \leq \delta^{\omega}(u_2, v_2).$$

Für Graphen mit weniger restriktiven Kantenmodellen, bei denen aus der Existenz einer langen Kante nicht auf die Existenz kürzerer Kanten geschlossen werden kann, ist ein wenig mehr Aufwand nötig. Beispielsweise eine Suche nach einem zweiten Ankerknoten $x'_{\alpha\cdot 1}$ unter Verwendung von Hopdistanzen, falls der eigentliche Ankerknoten $x_{\alpha\cdot 1}$ ein Nachbar von x_α ist.

Dadurch wird zwar die Adressgröße in Graphen mit großen Cliquen reduziert, jedoch muss dann in den verschickten Nachrichten auch die eindeutige ID des Zielknotens enthalten sein, um eine sichere Zustellung einer Nachricht garantieren zu können. Das Routing-Protokoll muss dann erweitert werden wie in Algorithmus 12 beschrieben.

Algorithmus 12 : erweitertes Routing-Protokoll für HBR an Knoten u

Eingabe : Eine Nachricht M für Zielknoten t mit Adresse $p_{HBR}(t)$ und eindeutiger ID $ID(t)$

```
1 if p_HBR(u) = p_HBR(t) then
2     if ID(u) = ID(t) then
          /* Nachricht hat Ziel erreicht                    */
3     else
4         Setze α := p_HBR(t)
5         Schicke M an x_α
6     end
7 else
      /* wie in Algorithmus 11                              */
8 end
```

Benötigter Speicherplatz

Für das Routen muss ein Knoten seine eigene ID kennen, seine virtuelle Adresse, die Routingtabelle und ein paar temporäre Daten wie die ω-Distanzen zu den neuen Ankerknoten $G_{\alpha\cdot 0}$ und $G_{\alpha\cdot 1}$ während der Aufteilung von Teilnetzwerk G_α. Nach der Aufteilung von G_α sind weder die Distanzen zu diesen Ankerknoten weiter nötig, noch die IDs dieser Knoten. Es reicht für jedes i, $1 \leq i \leq |\alpha|$ zu wissen, zu welchem Nachbarknoten eine Nachricht

weitergeleitet werden muss, wenn die eigene virtuelle Adresse und die des Zielknoten auf den ersten $i-1$ Positionen übereinstimmen, und auf Position i abweichen. Ein Knoten u mit Knotengrad $\deg(u)$ benötigt dann nur $|\alpha| \cdot \log_2 \deg(u)$ Bits zur Speicherung der Routingtabelle.

5.2 Länge der Routingwege

Während bei der Multidimensionalen Skalierung die erhöhte Erfolgsquote für das Greedy-Routing interessant war, ist es hier die Länge der Routingwege. Da die erfolgreiche Zustellung einer Nachricht bei HBR garantiert ist, stellt sich hier die Frage, wie gut diese Wege im Vergleich zu einem kürzesten Weg sind.

Bezeichnung 36 Sei $G = (V, E)$ ein Graph und $u_1, u_2 \in V$ zwei Knoten und p ein Weg von u nach v. Sei weiter $\text{dist}(u, v)$ die Länge eines kürzesten Weges zwischen u und v und $\delta(p)$ die Länge von p. Der Quotient

$$\frac{\delta(p)}{\text{dist}(u, v)}$$

heißt *Streckfaktor* von p. Den Wert

$$\frac{\delta(p) - \text{dist}(u, v)}{\text{dist}(u, v)}$$

bezeichnen wir als *Overhead*.

Satz 37 *Bei Verwendung von HBR kann der Streckfaktor beliebig groß werden, d.h. er kann bis auf $\mathcal{O}(|V|)$ anwachsen.*

Ein Beispiel dafür ist in Abbildung 5.2 gegeben. Der kürzeste Weg von s nach t hat die Länge 2, bei der gegebenen Aufteilung durchläuft der von HBR benutzte Weg jedoch den halben Graphen.

Solche Fälle sind aber konstruiert und treten in zufälligen Netzwerken nur sehr selten auf - wenn überhaupt. Leider gilt aber nicht der Umkehrschluss, dass mit zunehmender Dichte des Graphen der Streckfaktor beliebig nahe an 1 herankommt.

Beobachtung 38 *Die Verwendung von HBR führt bei sehr dichten Graphen nicht unbedingt dazu, dass der Streckfaktor beliebig klein wird.*

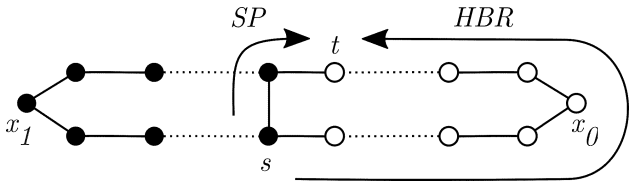

Abbildung 5.2: Worst-Case-Szenario für den Streckfaktor bei HBR.

Betrachten wir eine Situation, wie sie idealisiert in Abbildung 5.3 angedeutet ist. Bei der Bestimmung der HBR-Koordinaten in einem sehr dichten Graphen, der in einem quadratischen Gebiet ausgelegt ist, werden zwei Knoten in gegenüberliegenden Ecken als erste Ankerknoten x_0 und x_1 ausgewählt. In den beiden entstehenden Teilnetzen werden jeweils Knoten an den beiden verbleibenden Eckpunkten als Ankerknoten x_{00}, x_{01}, x_{10} und x_{11} gewählt. Routet man nun mit HBR von x_0 zu x_{10}, dann wird der Weg erst entlang der Diagonalen ins Zentrum des quadratischen Gebietes laufen, und von dort entlang der anderen Diagonale in die Zielecke. Damit hat der von HBR gewählte Weg einen Streckfaktor von näherungsweise $\sqrt{2}$, wenn wir euklidische Distanzen als Maß verwenden.

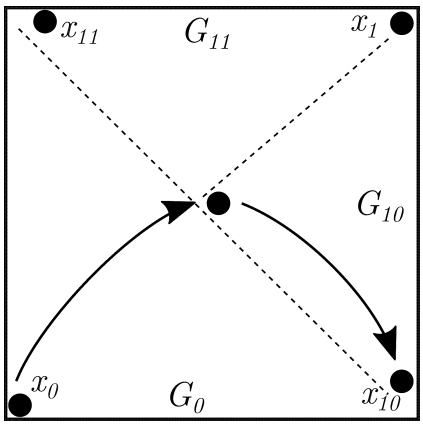

Abbildung 5.3: Routingweg mit einem Streckfaktor von ungefähr $\sqrt{2}$ in sehr dichten Graphen.

5.3 Erweiterungen des Verfahrens

Wenn wir ein recht dichtes Netzwerk auf einem näherungsweise quadratischem Gebiet betrachten, dann fällt bei der Aufteilung durch HBR auf,

dass die ersten beiden Teilnetze kein quadratisches Gebiet abdecken, sondern näherungsweise zwei rechtwinklig gleichschenklige Dreiecke bilden. Angesichts dessen, wie wir die beiden Ankerknoten x_0 und x_1 wählen, ist das kaum verwunderlich. Diese beiden Knoten liegen in gegenüberliegenden Ecken des Quadrates, und die Aufteilung des Netzwerkes erfolgt dann entlang der Diagonalen durch die beiden anderen Ecken. Diese beiden dreieckigen Teilnetze werden dann in den weiteren Schritten in kleinere Dreiecke aufgeteilt (vgl. Abbildung 5.4).

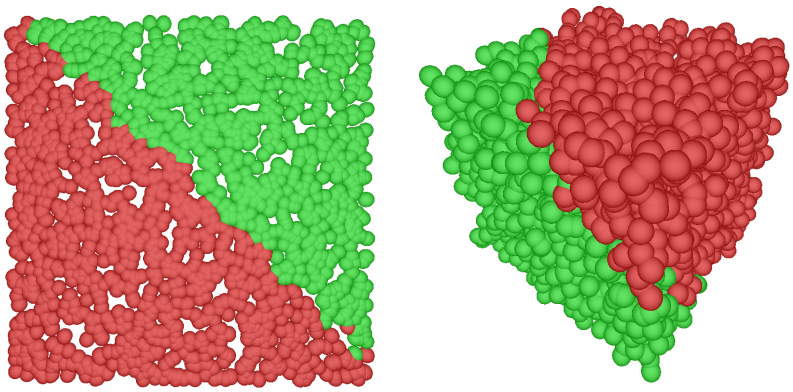

Abbildung 5.4: Aufteilung in zwei Teilnetze bei einem quadratisch (links) bzw. würfelförmig ausgelegten Netzwerk. Für eine klarere Visualisierung wurden die Kanten in dem Graphen ausgeblendet und die Knoten stark vergrößert dargestellt. Der Radius der Kugeln entspricht aber nicht dem r im UDG-Modell, und auch nicht $\frac{1}{2}r$, d.h. von der Überlappung (bzw. der Nicht-Überlappung) zweier Kugeln lässt sich nicht auf die Existenz einer Kante schließen

Im dreidimensionalen Fall wird die Situation noch etwas unübersichtlicher. Die beiden Ankerknoten x_0 und x_1 liegen an diagonal gegenüberliegenden Ecken. Betrachten wir die Ebene E_0, die durch gleiche Abstände zu diesen beiden Knoten definiert wird

$$E_0 := \{q \in \mathbb{R}^3 \mid \|q - p(x_0)\| = \|q - p(x_1)\|\},$$

dann teilt diese Ebene ein würfelförmig ausgelegtes Netzwerk in zwei etwa gleich große Teile. Die Schnittfläche zwischen Würfel und Ebene ist ein regelmäßiges Sechseck mit Eckpunkten auf der Mitte der Kanten, die nicht an $p(x_0)$ bzw. $p(x_1)$ angrenzen. Wir erhalten nach dem ersten Schritt also annähernd zwei Körper mit je zehn Ecken, die mit einer Würfelform nichts mehr gemein haben.

Man kann nun der Ansicht sein, dass ein solcher erster Aufteilungsschritt nicht besonders sinnvoll ist, weil er die ursprüngliche Form des Netzwerkes verändert. Man könnte das verhindern, indem man die beiden Ankerknoten anders wählt. Zum Beispiel so, dass nicht zwei Eckpunkte benutzt werden, sondern zwei Punkte mittig auf gegenüberliegenden Kanten des quadratischen Gebietes. Dann sollten im ersten Aufteilungsschritt zwei rechteckige Teilnetze entstehen, aus denen dann im nächsten Schritt je zwei kleinere quadratische Teilnetze gebildet werden. Im dreidimensionalen dann entsprechend zwei Punkte mittig auf gegenüberliegenden Seitenflächen.

Wirklich sinnvoll ist diese sehr spezielle Vorgehensweise aber auch nicht. Wenn das Netzwerk nicht annähernd quadratisch (bzw. würfelförmig) ist, dann läuft diese Methode ins Leere und erzeugt anderweitig wenig intuitiv sinnvolle Aufteilungen.

Stattdessen betrachten wir im Folgenden eine Idee nach dem Prinzip „d Ecken, d Teilnetze“. Intuitiv scheint dass doch recht sinnvoll zu sein, und es lässt sich sehr gut auf dreidimensional ausgelegte Netzwerke erweitern.

Das grundlegende Verfahren zur Bestimmung der Eckpunkte eines Netzwerkes haben wir auch schon in den vorherigen Kapiteln benutzt. Bei der Bestimmung einer guten Startkonfiguration für die Multidimensionale Skalierung in Algorithmus 5 ebenso wie im Kapitel zur Randerkennung bei der Suche nach einem großen initialen Kreis, der möglichst das gesamte Netzwerk umfasst. Diese Methode lässt sich auch hier verwenden, um bis zu d viele Ankerknoten zu finden, um dann das Netzwerk in bis zu d viele Teilnetze aufzuteilen.

In 5.5 sind einige Beispiele für mehr als zwei Aufteilungen aufgeführt. Man erkennt recht klar, dass im zweidimensionalen Fall die Aufteilung in vier quadratische Teilnetze recht gut funktioniert. Bei vier Aufteilungen entstehen aus dem Würfel vier Prismen mit dreieckiger Grundfläche. Bei acht Aufteilungen entstehen näherungsweise acht kleinere Würfel.

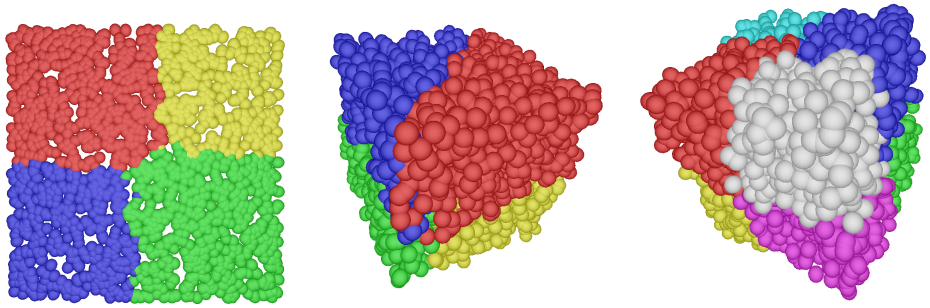

Abbildung 5.5: Aufteilung in vier bzw. acht Teilnetze.

5.3.1 Kosten und Speicherplatzbedarf des erweiterten Verfahrens

Während der Aufbauphase werden nicht mehr nur 2 Ankerknoten gewählt, sondern $d \geq 2$ viele Ankerknoten. Analog zur Analyse der Kosten bei der Aufteilung in $d = 2$ Teilnetze gilt also:

Lemma 39 *Wenn für den Aufbau der HBR-Struktur die Kandidatenauswahl in der k-Hop-Nachbarschaft abgesichert wird, dann ist bei einer Aufteilung in d Teilnetze in jedem Aufteilungsschritt an jedem Knoten ein Kommunikationsaufwand zwischen* $d \cdot k \cdot \deg(G)$ *und* $d(k+1) \cdot \deg(G)$ *zu erwarten.*

An den Worst-Case-Szenarien ändert sich nichts. Das heißt, auch mit dem erweiterten Verfahren können $\mathcal{O}(|V|)$ viele Aufteilungsschritte benötigt werden, und somit gilt auch hier für die Länge der Adressen im Worst-Case $|\alpha| = |V|$. Zu beachten ist, dass die Suche nach weiteren Ankerknoten (und somit die Suche nach weiteren Aufteilungen) abgebrochen wird, wenn ein neu gefundener Ankerknoten in der Nachbarschaft eines bereits existierenden Ankerknoten liegt. Damit wird ein vollständiger (Teil-)Graph für $d > 2$ genauso aufgeteilt wie bei $d = 2$ Aufteilungen.

Wenn wir pro Iteration nicht in zwei, sondern in mehr als zwei Teilnetze aufteilen, kommen wir nicht mit einem Bit pro Ebene für die Adresse aus, sondern wir benötigen $\log_2(d)$ viele Bits dafür. Der Speicherplatzbedarf für die Adressen steigt also um den Faktor $\log_2(d)$ an.

Es ist aber im Mittel zu erwarten, dass das Verfahren mit d Teilnetzen pro Ebene nach weniger Aufteilungsschritten zum Ende kommt. Wenn $d = 2^j$, dann ist sogar die Hoffnung begründet, dass das Verfahren um den Faktor $1/j$ weniger Iterationen benötigt. Damit wird der Bedarf von j Bits pro Ebene ausgeglichen, so dass sich der Platzbedarf zur Speicherung der Adressen insgesamt nicht (wesentlich) verändert.

In der experimentellen Auswertung werden wir betrachten, wie sich das in den einzelnen Testszenarien auswirkt.

5.4 Experimentelle Auswertung

Wir betrachten zunächst die Kosten für den Aufbau der HBR-Koordinaten. Daran anschließend die Länge der dabei bestimmten Adressen, und schließlich die Länge der Routingwege im Vergleich zu den optimalen Wegen.

Wie auch im Kapitel zur Bestimmung virtueller Koordinaten mittels Multidimensionaler Skalierung testen wir in jedem Testlauf 100 zufällig erstellte Graphen mit jeweils 1000 zufällig ausgewählten Start- und Zielknoten.

Als Kantengewichte werden hier zunächst ausschließlich Hopdistanzen verwendet.

5.4.1 Aufbau der Adressen

Für den Aufbau der HBR-Struktur beschränken wir uns bei der experimentellen Auswertung meist auf Unit-Disk-Graphen. Die Positionierung der Knoten spielt bei der Aufteilung in Teilnetze keine Rolle. Bei der Länge der Adressen spielt das Graphmodell eine gewisse Rolle, da es bei Unit-Disk-Graphen eher zur Bildung von Cliquen kommt als im Quasi-Unit-Disk-Graphen oder gar im Waxman-Modell. Das kommt besonders dann zum Tragen, wenn wir die Aufteilung in weitere Teilnetze abbrechen, wenn in einem Teilnetz alle Knoten mit dem Ankerknoten aus der vorigen Aufteilungsstufe benachbart sind.

Überprüfung der k-Hop-Nachbarschaften

Zunächst wollen wir an einigen Stichproben zeigen, dass die Überprüfung einer größeren Nachbarschaft wie erwartet tatsächlich höhere Kosten verursacht. Es reicht daher für einen Knoten vollkommen aus, nur die direkte Nachbarschaft zu überprüfen, wenn es darum geht, ob er als Kandidat für den nächsten Ankerknoten in Frage kommt oder nicht. Tatsächlich lässt sich das in allen getesteten Varianten so beobachten.

In Tabelle 5.1 sind für Unit-Disk-Graphen die Kosten für den Aufbau der HBR-Koordinaten aufgelistet, wenn der Kandidatencheck in der k-Hop Nachbarschaft, $k = 1, 2, 3$ durchgeführt wird. Die Tabelle beschränkt sich auf einige wenige verschiedene Knotenzahlen n, da sich zwar mit steigendem Knotengrad die absolute Anzahl der verschickten Nachrichten pro Knoten erhöht, aber der Kommunikationsaufwand normiert auf den Knotengrad sich nicht oder nur unwesentlich ändert.

Die einzelnen Spaltenblöcke geben verschiedene Maße für die Kosten an.

- Die ersten drei Spalten geben die durchschnittliche Anzahl von Nachrichten an, die jeder Knoten während der Aufbauphase verschickt.
- Der zweite Block gibt die Anzahl der Runden an, d.h. wie oft jeder Kno-

n	deg(G)	Kosten pro Knoten $k=1,2,3$			Runden $k=1,2,3$			1. Iteration $k=1,2,3$		
zweidimensionale Verteilung der Knoten										
1000	7.5	173	249	366	22.9	33.1	48.6	2.6	3.9	5.8
2500	18.8	392	613	916	20.8	32.6	48.6	2.3	3.9	5.9
5000	37.6	787	1280	1913	20.9	34.0	50.8	2.2	3.9	5.9
Verteilung mit Hindernissen (Straßen)										
2000	8.4	206	303	441	24.5	35.9	52.8	2.7	4.1	6.0
4000	15.8	388	594	883	24.4	37.5	55.8	2.5	4.0	5.9
7000	27.1	675	1061	1583	24.8	39.0	58.3	2.4	4.0	5.9
dreidimensionale Verteilung										
2000	7.5	179	255	374	23.8	34.0	49.7	2.6	3.9	5.8
4000	14.9	317	483	716	21.2	32.4	48.0	2.4	3.9	5.9
7000	26.1	531	843	1256	20.3	32.2	48.1	2.3	3.9	5.9

Tabelle 5.1: Anzahl der verschickten Nachrichten zum Aufbau der HBR-Struktur bei Unit-Disk-Graphen mit zwei Aufteilungen pro Stufe. Die Spalten „Runden“ geben die Anzahl der verschickten Nachrichten normiert auf die Anzahl der Nachbarn an. Die letzten 3 Spalten geben den Kommunikationsaufwand in Runden für den ersten Aufteilungsschritt an.

ten all seine Nachbarn kontaktieren muss. Dieser Wert ergibt sich aus der Anzahl der insgesamt verschickten Nachrichten, normiert auf den Knotengrad. Abweichungen in der Tabelle kommen durch Rundungen zustande.

- Die letzten drei Spalten geben die Anzahl der Runden während der ersten Aufteilungsphase an. Nach der Analyse im vorigen Abschnitt ist hier ein Wert $2k$ und $2(k+1)$ zu erwarten.

Man erkennt in der Tabelle deutlich, dass ein stärkeres Aussieben der Kandidatenliste den Kommunikationsaufwand sehr stark erhöht - ganz egal, wie viele Knoten die Graphen enthalten und wie diese verteilt sind.

Die Anzahl der „Runden“ ist dabei nicht gleichzusetzen mit der Länge der entstehenden HBR-Koordinaten. Mit jeder weiteren Aufteilung nimmt der für die Kandidatensuche (und das Fluten) relevante durchschnittliche Knotengrad immer mehr ab. Denn für die weitere Aufteilung müssen die Knoten nicht mehr mit denjenigen Nachbarn kommunizieren, die sich in einem anderen Teilnetz befinden.

Beim Aufbau der HBR-Koordinaten gibt es mit steigender Knotenzahl der Graphen zwei gegenläufige Effekte, was den Kommunikationsaufwand angeht. Zum einen wird mit steigender Knotenzahl die Anzahl der notwen-

digen Aufteilungsschritte höher, bis jeder Knoten eine eindeutige Adresse erhält. Das ist besonders bei sehr dichten Graphen und den dabei häufiger auftretenden größeren Cliquen der Fall. Das erhöht den auf den Knotengrad normierten Aufwand für dichte Graphen. Zum anderen muss für die Kandidatensuche bei weniger dichten Graphen ein höherer Anteil der Nachbarn kontaktiert werden, bis sich ein Knoten als Kandidat ausschließen kann. Das erhöht den auf den Knotengrad normierten Aufwand für weniger dichte Graphen. In einigen Fällen ist der erste Effekt stärker, in anderen dominiert der zweite Effekt. Damit sind die unterschiedlichen Tendenzen in den Spalten „Runden“ in Tabelle 5.1 zu erklären.

Außerdem lässt sich an den Testergebnissen ablesen, dass die durchschnittliche Anzahl der Runden in der ersten Aufteilungsphase wie erwartet zwischen $2k$ und $2(k+1)$ liegt. Tatsächlich liegt sie bei Überprüfung der 2- und 3-Hop-Nachbarschaft sogar leicht unter dem Wert $2k$. Das ist dadurch zu erklären, dass bei den meisten Knoten schon während der Überprüfung der 1- bzw. 2-Hop-Nachbarschaft die Kandidatur für den nächsten Ankerknoten ausgeschlossen werden kann. Es ist also im Anschluss keine weitere Anfrage nötig. Für das abschließende Fluten hingegen benötigt jeder Knoten (mit Ausnahme des neu bestimmten Ankerknotens) im Durchschnitt nur $\deg(G)-1$ viele Nachrichten, da eine Nachricht an den Knoten, von dem eine Flut-Nachricht erhalten wurde, nicht sinnvoll ist. Insgesamt landet man dadurch bei einem Wert, der minimal unter der Abschätzung liegt.

Aufteilung in zwei oder mehr Teilnetze

Als nächstes betrachten wir, wie stark die Aufteilung in zwei oder mehr Teilnetze Einfluss auf den Kommunikationsaufwand hat. Hierbei müssen ja in jedem Aufteilungsschritt mehr Ankerknoten gesucht werden, was den Kommunikationsaufwand erst einmal deutlich erhöht. Auf der anderen Seite werden dann weniger Aufteilungsschritte benötigt, was den Aufwand wieder verringert.

Die Ergebnisse für Unit-Disk-Graphen sind in Abbildung 5.6 zusammengefasst. Bei mehr als zwei Aufteilungen steigt der Kommunikationsaufwand an, was nicht sonderlich verwunderlich ist. Allerdings nicht so stark, wie man zunächst vermuten würde. Insgesamt erhöht sich der Aufwand nur um ungefähr 25% bzw. 50%, wenn statt zwei Teilnetze vier oder acht Teilnetze pro Iteration gesucht werden.

Bei Betrachtung der Anzahl der Runden in der ersten Iteration fällt zudem auf, dass bei zweidimensional ausgelegten Netzwerken mit vielen Knoten bei

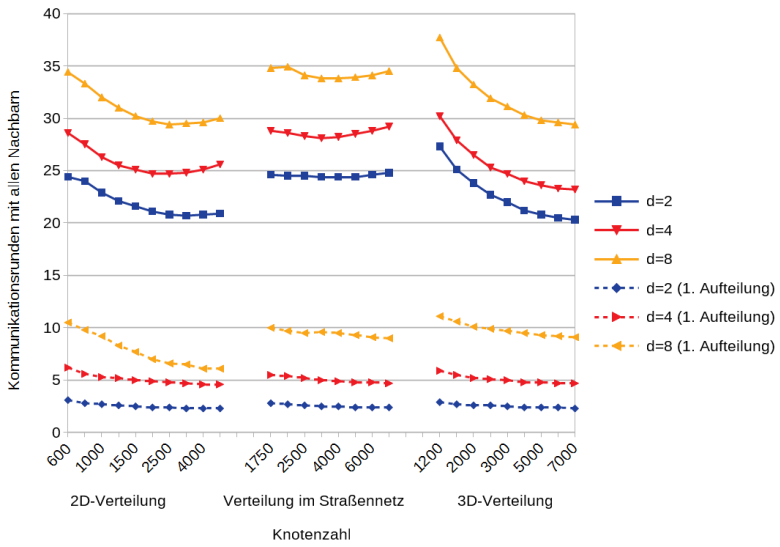

Abbildung 5.6: Anzahl der verschickten Nachrichten zum Aufbau der HBR-Struktur bei Unit-Disk-Graphen mit $d = 2, 4, 8$ Aufteilungen pro Stufe. Die Anzahl der Nachrichten ist normiert auf den Knotengrad - sie gibt also an, wie oft ein Knoten all seine Nachbarn kontaktieren muss.

$d = 8$ Aufteilungen der Kommunikationsaufwand unter den erwarteten 8 bis 16 Kommunikationsrunden mit den Nachbarn liegt. Das ist durch die Tatsache zu erklären, dass die Suche nach weiteren Ankerknoten abgebrochen wird, wenn der nächste potentielle Ankerknoten mit einem bereits gewählten Ankerknoten benachbart ist. Das bedeutet, das Netzwerk wird oft nicht in 8 Teilnetze aufgeteilt, sondern in weniger Teilnetze.

Bei einer Verteilung der Knoten im Straßennetz und in einer dreidimensionalen Auslegung tritt dieser Effekt nicht in so einem starken Maß auf. Hier werden (zumindest im ersten Aufteilungsschritt) ausreichend viele Ankerknoten gefunden, um das Netzwerk sinnvoll in die gewünschte Anzahl von Teilnetzen aufzuteilen.

Die Ergebnisse für Quasi-Unit-Disk-Graphen und im Waxman-Modell weichen nur unerheblich von den Werten für Unit-Disk-Graphen ab und liefern keine neuen Erkenntnisse.

5.4.2 Länge der Adressen

Im Folgenden betrachten wir die Länge der entstehenden Adressen unter gewissen Parametern.

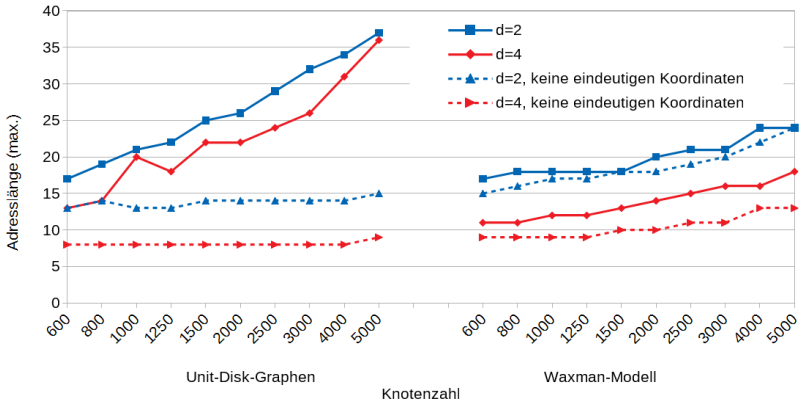

Abbildung 5.7: Maximale Länge der HBR-Adressen bei Unit-Disk-Graphen ($r = 50$, links) und Waxman-Graphen ($r = 93$, rechts) bei zweidimensional ausgelegten Knoten.

In Abbildung 5.7 sind die Adresslängen für zweidimensional ausgelegte Netzwerke mit Kanten im Unit-Disk- und Waxman-Modell angegeben. Die Werte geben die Länge der längsten Adresse aller Knoten in allen getesteten Graphen an. Die durchschnittliche Länge der längsten Adresse in den getesteten Graphen ist durchgehend ca. $15 - 20\%$ geringer. Es handelt sich bei diesen Maximalwerten also nicht um krasse Ausreißer bei einzelnen Graphen, sondern um einen üblichen Wert. Die mittleren und kürzesten Längen sind in den Tabellen am Ende des Kapitels angegeben. Die kürzesten Längen geben dabei *nicht* die kürzeste Länge der Adresse eines einzelnen Knotens in einem der getesteten Graphen an, sondern geben an, dass es einen Graphen in der Testreihe gab, bei dem man für alle Knoten mit dieser Adresslänge ausgekommen ist. Die tatsächlich kürzeste Knotenadresse α ist in vielen Fällen (besonders bei $d = 4$ und $d = 8$ Aufteilungen) $|\alpha| = 1$ oder $|\alpha| = 2$, da es immer wieder einzelne Graphen gibt, bei denen schon im ersten Aufteilungsschritt ein Teilnetz aus einem einzelnen Knoten (oder ein paar ganz wenigen) besteht.

Bei Unit-Disk-Graphen fällt auf, dass es bei einem vollständigen Aufbau der HBR-Koordinaten zu einem sehr starken Anstieg der Adresslängen mit steigender Knotenzahl kommt. Das ist vor allem durch die häufigen größeren Cliquen zu erklären, deren Auflösung in eindeutige Adressen den Worst-Case bedeuten. Das Argument wird dadurch bestärkt, dass bei einem vorzeitigen Abbruch des Aufbaus der HBR-Koordinaten die maximale Adresslänge kaum anwächst.

Im Waxman-Modell fällt der Anstieg der Adresslängen bei vollständiger

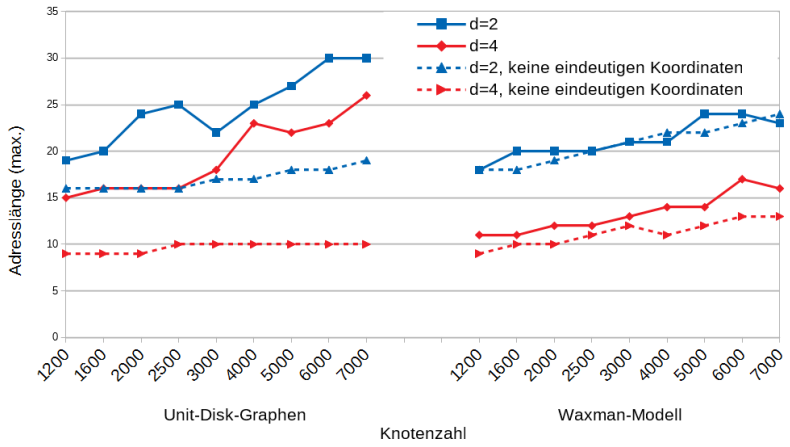

Abbildung 5.8: Maximale Länge der HBR-Adressen bei Unit-Disk-Graphen ($r = 100$, links) und Waxman-Graphen ($r = 150$, rechts) bei dreidimensional ausgelegten Knoten.

Auflösung der Adressen deutlich moderater aus, da hier seltener größere Cliquen in den Graphen entstehen - es fehlen immer wieder mal einige Kanten zwischen nahe beieinanderliegenden Knoten. Auf der anderen Seite kann man hier nicht so sehr von einem vorzeitigen Abbruch der Koordinatenbestimmung profitieren.

Bei dreidimensionalen Netzwerken (Abbildung 5.8) sehen die absoluten Werte ein wenig anders aus, in der Tendenz sind hier aber die gleichen Effekte zu beobachten.

5.4.3 Länge der Routingwege

Da bei HBR die Zustellung einer Nachricht zum Ziel garantiert ist, betrachten wir hier nicht die Erfolgsquoten der Zustellung, sondern die Länge des Weges, auf dem die Nachricht durch das Netzwerk durchgeleitet wurde. Wir vergleichen das Verhalten verschiedener Routing-Protokolle.

Zum einen verwenden wir ausschließlich die HBR-Koordinaten für das Routing. Andererseits benutzen wir Greedy-Routing mit HBR als Ausweichprotokoll. Das heißt, dass HBR nur dann zum Einsatz kommt, wenn das Greedy-Routing fehlschlägt. Die Nachricht wird dabei so lange mit dem Ausweichprotokoll weitergeleitet, bis sie einen Knoten erreicht, der näher am Ziel liegt als der Knoten, bei dem das Greedy-Routing zuletzt in einer Sackgasse landete. Dann wird wieder auf Greedy-Routing gewechselt.

Die Zustellung der Nachricht mit diesem Misch-Verfahren ist offensichtlich ebenfalls garantiert, da nach einem Fehlschlag durch Greedy-Routing der nächste Startpunkt für Greedy näher am Ziel liegt. Es ist also nicht möglich, dass das Routingverfahren in eine Schleife läuft, die den Zielknoten nie erreicht.

Zusätzlich vergleichen wir HBR als Ausweichprotokoll mit einem Verfahren, das entlang eines kürzesten Weges (SP) aus Sackgassen herausführt. Es ist offensichtlich, dass dieses Ausweichprotokoll in der Praxis oft nicht anwendbar ist - wenn der kürzeste Weg bekannt ist, dann benötigt man keine weiteren Routing-Protokolle wie HBR oder das unsichere Greedy-Routing. Wir verwenden SP als Vergleichsprotokoll zu HBR unter der Annahme, dass SP im Allgemeinen eine sinnvolle Strategie ist, um aus einer Sackgasse herauszukommen.

Bemerkung 40 *Das ist aber nicht unbedingt die optimale Strategie. Es kann Fälle geben, in denen der Wechsel zurück zum Greedy-Protokoll an einem Knoten entlang eines kürzesten Weges erneut in eine lange Sackgasse führt, wo hingegen ein Ausweichen aus der Sackgasse entlang eines anderen Weges und Wechsel zum Greedy-Protokoll an einem anderen Knoten direkt zum Ziel führen könnte (vgl. Abbildung 5.9).*

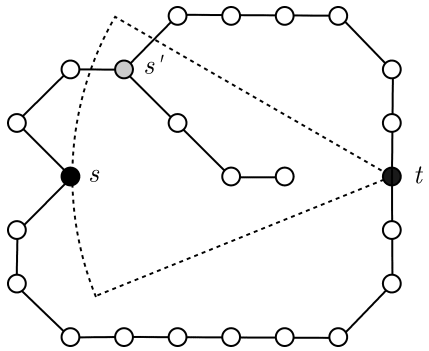

Abbildung 5.9: Von Knoten s aus geht es mit Greedy-Schritten nicht weiter in Richtung des Zielknotens t. Beim Ausweichen entlang eines kürzesten Weges (oben herum) wird bei Knoten s' zurück auf Greedy gewechselt, wodurch man in eine weitere Sackgasse hineinläuft. Das Ausweichen entlang eines etwas längeren Weges (unten herum) würde insgesamt zu einer kürzeren Routingstrecke führen.

Wir betrachten also folgende fünf Routing-Protokolle:

- **HBR**. Das Routing wird vollständig über die HBR-Koordinaten durchgeführt.

- **GEO-HBR**. Greedy-Routing mit den tatsächlichen Koordinaten der Knoten und HBR als Ausweichprotokoll.
- **MDS-HBR**. Wie GEO-HBR, jedoch wird Greedy-Routing auf den vorher ermittelten virtuellen Koordinaten durch MDS durchgeführt.
- **GEO-SP**. Greedy-Routing mit den tatsächlichen Koordinaten und SP als Ausweichprotokoll.
- **MDS-SP**. Wie GEO-SP, jedoch wird Greedy-Routing auf den vorher ermittelten virtuellen Koordinaten durch MDS durchgeführt.

Wir vergleichen diese fünf Varianten bzgl. der gerouteten Weglängen in Bezug zu einem kürzesten Weg. Wie auch schon in Abschnitt 3.5 basieren die Werte auf der Auswertung von jeweils 100 Graphen mit je 1000 zufällig ausgewählten Wegen.

Für den Overhead der jeweiligen Verfahren betrachten wir die Summe der Kosten der Wege von den 1000 Startknoten zu den entsprechenden Zielknoten in allen 100 Netzwerken. Seien $C(HBR)$ und $C(SP)$ die Gesamtkosten bei Verwendung von HBR bzw. eines kürzesten-Wege-Algorithmus. Der durchschnittliche Overhead für HBR ist dann definiert durch

$$\frac{C(HBR) - C(SP)}{C(SP)}.$$

Entsprechend ist der Overhead für die anderen Routing-Protokolle definiert.

Zufällige Verteilung der Knoten

In den theoretischen Betrachtungen zu den Längen der Routingwege wurde bereits klar, dass bei Verwendung von HBR nicht damit zu rechnen ist, dass der Overhead bei zunehmend dichten Graphen gegen 0 läuft. Es ist sogar im Gegenteil zu beobachten, dass der Overhead mit zunehmender Dichte der Graphen zunimmt.

In Abbildung 5.10 ist der Overhead für die hier getesteten Routingverfahren für zufällig ausgelegte Graphen in der Ebene aufgetragen. Die Verfahren, die im Normalfall greedy arbeiten, werden mit zunehmender Knotendichte besser. Das ist vor allem dadurch zu erklären, dass die Erfolgsquote für diese Verfahren bei hohem Knotengrad fast bei 100% liegt, und in diesen Fällen der Greedy-Weg nur unwesentlich vom optimalen Weg abweicht. HBR als Ausweichprotokoll ist dabei ein wenig schlechter als das Ausweichen entlang eines kürzesten Weges, wobei der Unterschied bei Greedy-Routing auf den MDS-Koordinaten deutlich geringer ausfällt und zu vernachlässigen ist.

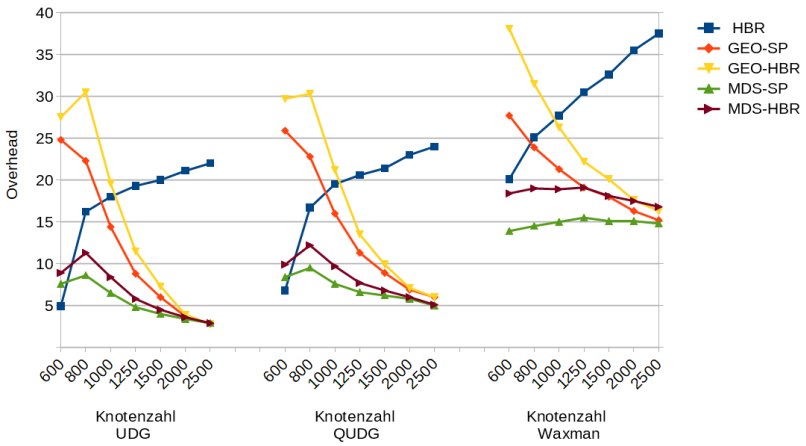

Abbildung 5.10: Overhead beim Routen mit HBR auf zweidimensionalen Netzwerken im UDG- , QUDG- und Waxman-Modell.

HBR als alleiniges Routing-Protokoll ist sehr gut für dünne Graphen geeignet. Allerdings steigt der Overhead schnell recht stark an, wenn die Graphen ein wenig dichter werden und es dann zwischen zwei weiter entfernten Knoten mehr als „nur einen Weg“ gibt. Die Graphen mit 600 Knoten liegen in unserer Testumgebung am unteren Limit des sinnvoll testbaren, d.h. es kommt regelmäßig vor, dass der Graph bei Herausnahme nur einer einzelnen Kante in zwei große Zusammenhangskomponenten zerfällt. Über dieses Nadelöhr laufen dann alle Wege zwischen zwei Knoten in unterschiedlichen Bereichen des Graphen, so dass zwischen einem optimalen Weg und der HBR-Routingstrecke kein großer Unterschied bestehen kann.

Bei Graphen im Waxman-Modell ist recht deutlich ein größerer Overhead zu beobachten, wenn ausschließlich mit den HBR-Koordinaten geroutet wird. Eine Erklärung dafür ist das Verhalten des Routings auf dem letzten Teil des Routing-Weges. Bei Unit-Disk-Graphen (und auch leicht eingeschränkt bei Quasi-Unit-Disk-Graphen) existiert eine Kante zum Zielknoten, wenn man nur nahe genug an den Knoten herankommt. Ein Routen entlang der HBR-Struktur bis auf die letzte Koordinaten-Ebene ist dann nicht notwendig. Im Waxman-Modell fehlen oft auch Kanten zwischen Knoten, die sehr nahe beieinanderliegen. Dann kommt es auf der „letzten Meile“ vom Start- zum Zielknoten gelegentlich zu größeren Umwegen, die sich dann negativ auf den durchschnittlichen Overhead auswirken.

Beobachtung 41 *Bei zweidimensional ausgelegten Netzwerken liefert HBR als alleinstehendes Routing-Protokoll sehr gute Ergebnisse auf dünnen Netz-*

werken. Bei dichteren Netzen empfiehlt sich HBR als Ausweichprotokoll für Greedy-Routing, wobei der durchschnittliche Overhead bei Verwendung von MDS-Koordinaten in den meisten Fällen geringer ist als bei Verwendung der tatsächlichen Koordinaten, was in der höheren Erfolgsquote dieser Greedy-Variante begründet liegt.

Verteilung der Knoten mit Hindernissen

Als zweites betrachten wir den Overhead bei einer Verteilung der Knoten mit Hindernissen - hier verwenden wir wieder den Kartenausschnitt aus der Berliner Innenstadt. Auffallend ist hier (vgl. Abbildung 5.11, links) vor allem der wesentlich größere Unterschied zwischen den hybriden Verfahren die auf tatsächlichen bzw. auf den MDS-Koordinaten beruhen. Auch hier lässt sich das im Wesentlichen durch die Erfolgsquoten des Greedy-Verfahrens erklären. Für den Overhead kommt als zusätzlicher ungünstiger Faktor hinzu, dass die Sackgassen hier auch bei hoher Knotendichte häufig auftreten, und dann oft auch mehrere Hops lang sind.

HBR alleine und Greedy-MDS arbeiten insgesamt auf einem vergleichbaren Level. Greedy-Routen mit den tatsächlichen Koordinaten ist hier deutlich schlechter - was nach den Testreihen im Kapitel zur Multidimensionalen Skalierung aber auch nicht weiter verwundert.

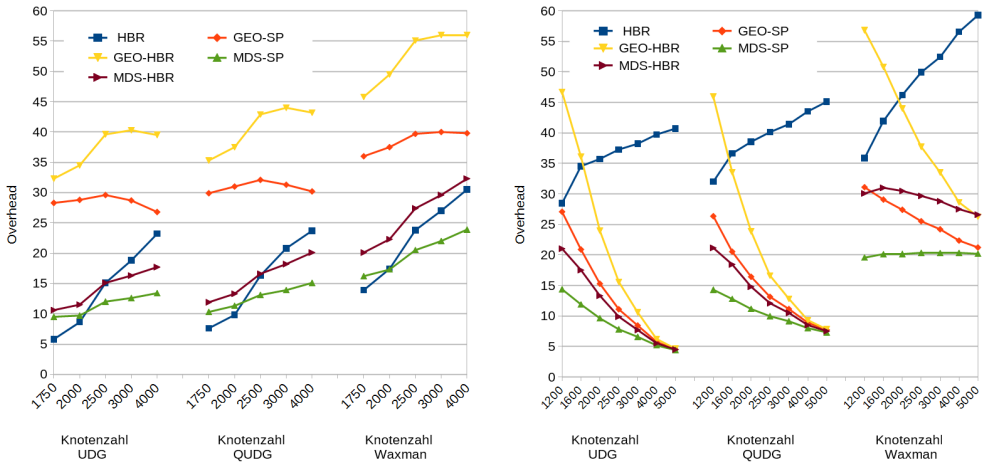

Abbildung 5.11: Overhead beim Routen mit HBR auf zweidimensionalen Netzwerken mit Hindernissen (links) und 3D-Netzwerken (rechts)

Beobachtung 42 *Bei einer Verteilung der Knoten in einem Straßennetz arbeiten HBR als alleinstehendes und HBR als Ausweichprotokoll auf ei-*

nem ähnlichen Niveau, wenn im Greedy-Teil die MDS-Koordinaten verwendet werden.

Dreidimensionale Verteilung der Knoten

Das Verhalten des Overheads der betrachteten Routing-Verfahren ist bei dreidimensionalen Netzwerken (Abbildung 5.11, rechts) ähnlich zu der zufälligen zweidimensionalen Verteilung, allerdings generell auf einem höheren Niveau, d.h. der Overhead ist größer.

Für diese Erhöhung sind zwei Effekte verantwortlich, weswegen ein direkter Vergleich der Werte zwischen 2D und 3D nur eingeschränkt sinnvoll ist. Der Hauptpunkt ist, dass wir bei der Wahl der Senderadien der Knoten für die unterschiedlichen Testreihen einen ähnlichen Knotengrad fixiert haben. Dadurch, dass der Senderadius der Knoten im dreidimensionalen deutlich höher gewählt wird, sinkt die durchschnittliche Hopdistanz zwischen zwei Knoten. Das hat dann zur Folge, dass ein kleiner Fehler von nur einem Hop einen stärkeren Einfluss auf den relativen Wert des Overheads hat.

Der andere Effekt ist der, der uns im Abschnitt 5.3 dazu motiviert hat, die HBR-Methode auf mehr als zwei Aufteilungen zu erweitern. Daher betrachten wir im folgenden Abschnitt, wie sich der Overhead entwickelt, wenn wir die Netzwerke in mehr als zwei Teilnetze aufteilen.

Beobachtung 43 *Bei einer dreidimensionalen Auslegung lassen sich ganz ähnliche Effekte beobachten wie bei einer zweidimensionalen Auslegung der Knoten. Der Overhead liegt hier aber generell auf einem höheren Niveau.*

Overhead bei mehr als zwei Aufteilungen

Da HBR für den gesamten Overhead bei den hybriden Verfahren insgesamt nur einen geringen Einfluss hat, wenn die Erfolgsquoten mit wachsender Dichte der Netzwerke steigen, beschränken wir uns im Folgenden auf das Verhalten von HBR als alleiniges Protokoll.

In den Abbildungen 5.12 und 5.13 ist der Overhead für HBR in den drei getesteten Varianten dargestellt, jeweils mit einer Aufteilung auf $d = 2, 4, 8$ Teilnetze pro Schritt.

Es zeigt sich, dass die „intuitiv richtige“ Anzahl von Aufteilungen pro Stufe die besten Werte für das Routing liefert. Bei zweidimensional ausgelegten Netzen sind vier Aufteilungen besser als zwei, bei $d = 8$ ist jedoch wieder

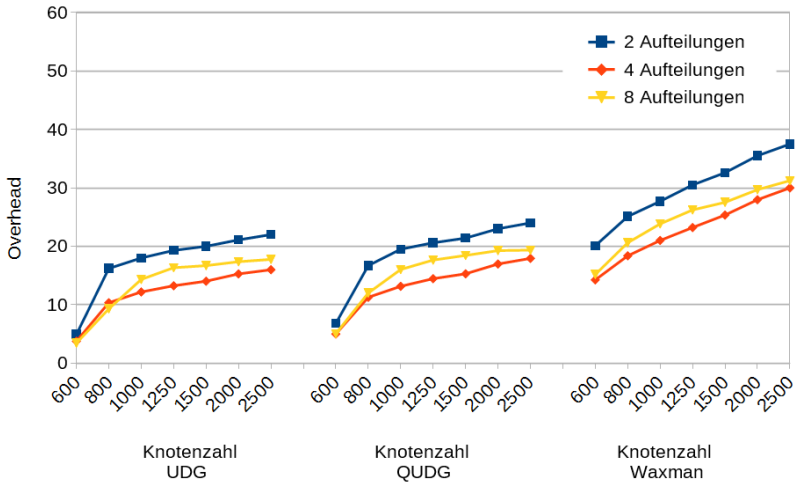

Abbildung 5.12: Overhead beim Routen mit HBR mit unterschiedlich vielen Aufteilungen pro Iteration auf zweidimensionalen Netzwerken im UDG-, QUDG- und Waxman-Modell. Kantengewicht: Hop.

eine Verschlechterung des Overheads zu beobachten. Bei einer dreidimensionalen Auslegung sinkt der Overhead bei acht Aufteilungen noch weiter, und man erreicht dort noch bessere Ergebnisse als mit zwei Aufteilungen. Eine weitere Erhöhung der Anzahl der Aufteilungen bei 3D-Netzen (nicht in der Abbildung) hat kaum weiteren Einfluss auf den Overhead. Das liegt vor allem auch daran, dass hier auch bei wenig dichten Graphen nur selten mehr als acht Ankerknoten mit der hier verwendeten Methode gefunden werden.

Bei einer Verteilung der Knoten im Straßennetz werden mit acht Aufteilungen ebenfalls die besten Ergebnisse erzielt. Jedoch ist der der Effekt nicht ganz so ausgeprägt wie bei den 3D-Netzwerken.

Beobachtung 44 *Die „intuitiv richtige“ Anzahl von Aufteilungen pro Iterationsschritt liefert die besten Ergebnisse. Bei gleichmäßig zweidimensional ausgelegten Netzwerken sind das $d = 4$ Aufteilungen, bei dreidimensionalen $d = 8$ Aufteilungen. Bei der Knotenverteilung im Straßennetz ist zwischen vier und acht Aufteilungen nur ein unwesentlicher Unterschied zu beobachten.*

Verwendung anderer Kantengewichte

Wie bei der Multidimensionalen Skalierung können auch für den Aufbau der HBR-Struktur andere Kantengewichte verwendet werden. In Abbildung 5.14 ist der Overhead für zwei- und dreidimensionale Netzwerke unter Verwendung

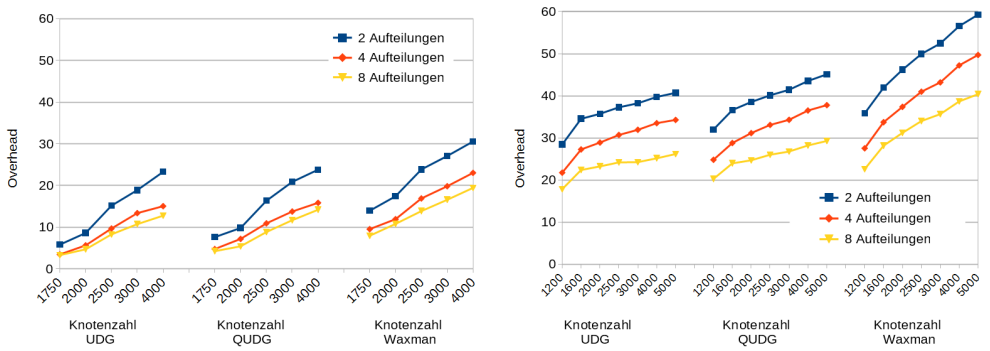

Abbildung 5.13: Overhead beim Routen mit HBR mit unterschiedlich vielen Aufteilungen pro Iteration auf zweidimensionalen Netzwerken im Straßennetz (links) und 3D-Netzwerken (rechts). Kantengewicht: Hop.

von euklidischen Distanzen aufgeführt. Im Vergleich zu Hop-Distanzen ist bei Unit-Disk-Graphen eine moderate Verbesserung des Overheads zu erkennen, bei Graphen im Waxman-Modell ist die Verbesserung deutlich stärker ausgeprägt.

Die Verbesserung im Waxman-Modell geht dabei zurück auf die hohe Varianz der Kantenlängen und die damit verbundene mögliche ungünstige Wahl von „kurzen“ Kanten bei Verwendung der HBR-Struktur, was die Länge der Routing-Wege stark verlängern kann, wenn diese in Hops gemessen werden. Ein weiterer Punkt sind die vorhin beschriebenen Effekte auf den letzten paar Hops, die bei Verwendung von euklidischen Distanzen nicht in diesem Maße auftreten.

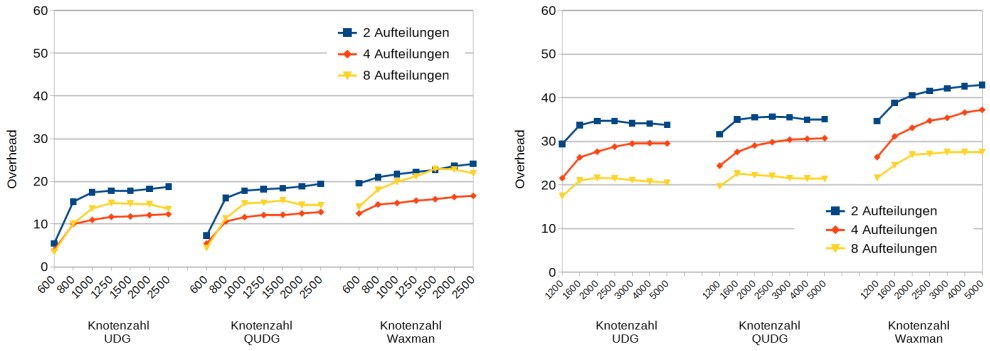

Abbildung 5.14: Overhead beim Routen mit HBR mit unterschiedlich vielen Aufteilungen pro Iteration bei 2D- (links) und 3D-Netzwerken (rechts). Kantengewicht: Euklidisch

Beobachtung 45 *Bei Verwendung euklidischer Kantengewichte steigt der Overhead abseits von sehr dünnen Graphen schneller auf ein höheres Niveau, bleibt dann aber bei weiter steigender Knotendichte nahezu konstant und steigt nicht weiter an. Dieser Effekt führt besonders bei Graphen im Waxman-Modell zu einer deutlichen Verbesserung des Overheads.*

5.4.4 Tabellen

Abschließend zu diesem Kapitel wieder eine tabellarische Auflistung der experimentellen Ergebnisse. Die Tabellen zu den Adresslängen sind dabei ergänzt um die durchschnittlichen und minimalen Werte.

Bei den Tabellen zum Overhead der Routingwege sind auch die Werte für HBR als Ausweichprotokoll bei $d = 4$ und $d = 8$ Aufteilungen angegeben, die aber wegen der generell hohen Erfolgsquote beim Greedy-Routing nur marginal von denen mit $d = 2$ Aufteilungen abweichen.

n	deg(G)	Kosten pro Knoten $d = 2, 4, 8$			Runden $d = 2, 4, 8$			1. Iteration $d = 2, 4, 8$		
zweidimensionale Verteilung der Knoten										
600	4.7	117	137	166	24.4	28.6	34.4	3.1	6.2	10.5
800	6.1	146	167	203	24.0	27.5	33.3	2.8	5.6	9.8
1000	7.5	173	198	242	22.9	26.3	32.0	2.7	5.3	9.2
1250	9.4	208	239	292	22.1	25.5	31.0	2.6	5.2	8.3
1500	11.2	244	283	340	21.6	25.1	30.2	2.5	5.0	7.7
2000	15.0	317	372	447	21.1	24.7	29.7	2.4	4.9	7.0
2500	18.8	392	465	553	20.8	24.7	29.4	2.4	4.8	6.6
3000	22.5	468	560	665	20.7	24.8	29.5	2.3	4.7	6.5
4000	30.1	625	757	890	20.8	25.1	29.6	2.3	4.6	6.1
5000	37.6	788	961	1129	20.9	25.6	30.0	2.3	4.6	6.1
Verteilung mit Hindernissen (Straßen)										
1750	7.5	184	217	262	24.6	28.8	34.8	2.8	5.5	10.0
2000	8.4	206	242	294	24.5	28.6	34.9	2.7	5.4	9.7
2500	10.2	252	288	350	24.5	28.3	34.1	2.6	5.2	9.5
3000	12.0	293	339	408	24.4	28.1	33.8	2.5	5.0	9.6
4000	15.8	388	446	534	24.4	28.2	33.8	2.5	4.9	9.5
5000	19.6	479	557	665	24.4	28.5	33.9	2.4	4.8	9.3
6000	23.4	576	674	797	24.6	28.8	34.1	2.4	4.8	9.1
7000	27.1	675	794	937	24.8	29.2	34.5	2.4	4.7	9.0
dreidimensionale Verteilung der Knoten										
1200	4.6	127	141	176	27.3	30.2	37.7	2.9	5.9	11.1
1600	6.0	152	168	210	25.1	27.9	34.8	2.7	5.5	10.6
2000	7.5	179	198	249	23.8	26.5	33.2	2.6	5.2	10.1
2500	9.3	212	237	298	22.7	25.3	31.9	2.6	5.1	9.9
3000	11.2	247	277	349	22.0	24.7	31.1	2.5	5.0	9.7
4000	14.9	317	358	453	21.2	24.0	30.3	2.4	4.8	9.5
5000	18.6	387	440	557	20.8	23.6	29.8	2.4	4.8	9.3
6000	22.3	460	523	661	20.5	23.3	29.6	2.4	4.7	9.2
7000	26.1	531	605	770	20.3	23.2	29.4	2.3	4.7	9.1

Tabelle 5.2: Anzahl der verschickten Nachrichten zum Aufbau der HBR-Struktur bei Unit-Disk-Graphen mit $d = 2, 4, 8$ Aufteilungen pro Stufe. Die Spalten „Runden“ geben die Anzahl der verschickten Nachrichten normiert auf die Anzahl der Nachbarn an. Die letzten 3 Spalten geben den Kommunikationsaufwand in Runden für den ersten Aufteilungsschritt an.

		jeweils mit $2, 4, 8$ Aufteilungen								
n	deg(G)	durchschn. Länge			min. Länge			max. Länge		
Unit-Disk-Graphen, $r = 50$										
600	4.8	14.4	10.8	10.4	13	9	9	17	13	14
		(11.2)	(6.9)	(6.5)	(10)	(6)	(6)	(13)	(8)	(7)
1000	7.5	17.3	13.4	12.9	15	11	11	21	20	16
		(12.1)	(7.1)	(6.8)	(11)	(7)	(6)	(13)	(8)	(8)
2000	15.0	21.5	18.2	18.2	19	16	16	26	22	24
		(12.3)	(7.2)	(7.0)	(11)	(7)	(7)	(14)	(8)	(8)
4000	30.0	28.9	25.8	25.6	26	23	23	34	31	31
		(12.8)	(7.6)	(7.3)	(12)	(7)	(7)	(14)	(8)	(9)
Quasi-Unit-Disk-Graphen, $r = 58$										
600	4.7	14.3	10.7	10.3	12	9	8	18	13	14
		(11.5)	(6.9)	(6.5)	(10)	(6)	(6)	(13)	(8)	(8)
1000	7.6	16.9	13.0	12.5	15	11	10	19	16	16
		(12.4)	(7.2)	(6.8)	(11)	(7)	(6)	(14)	(8)	(8)
2000	15.1	21.0	17.3	17.1	19	15	15	25	21	20
		(13.2)	(7.4)	(7.2)	(12)	(7)	(7)	(15)	(8)	(8)
4000	30.3	26.5	23.9	23.7	24	21	21	34	29	30
		(14.1)	(8.0)	(7.8)	(13)	(7)	(7)	(15)	(9)	(9)
Waxman, $r = 93$										
600	4.6	14.2	9.1	8.5	13	8	7	17	11	10
		(13.5)	(7.3)	(6.6)	(12)	(7)	(6)	(15)	(9)	(8)
1000	7.5	15.2	10.2	9.5	14	9	8	18	12	12
		(14.4)	(8.0)	(7.2)	(13)	(7)	(6)	(17)	(9)	(8)
2000	15.1	17.6	12.0	11.1	16	11	10	20	14	13
		(16.6)	(9.2)	(8.1)	(15)	(8)	(7)	(18)	(10)	(9)
4000	30.2	20.8	14.2	13.0	19	13	12	24	16	15
		(19.4)	(10.8)	(9.3)	(18)	(10)	(8)	(22)	(13)	(11)

Tabelle 5.3: Länge der Adressen, 2D, 2-4-8 splits. In Klammern der Wert bei vorzeitigem Abbruch des Aufbaus der HBR-Struktur.

		jeweils mit $2, 4, 8$ Aufteilungen								
n	deg(G)	durchschn. Länge			min. Länge			max. Länge		
Unit-Disk-Graphen, $r = 100$										
1200	4.6	16.3	11.6	10.3	14	10	9	19	15	15
		(14.2)	(8.0)	(7.0)	(13)	(7)	(6)	(16)	(9)	(8)
2000	7.5	18.1	13.3	12.5	17	12	10	24	16	16
		(14.8)	(8.2)	(7.1)	(14)	(8)	(7)	(16)	(9)	(8)
4000	14.9	21.4	16.8	15.6	20	15	13	25	23	20
		(15.7)	(8.9)	(7.3)	(15)	(8)	(7)	(17)	(10)	(8)
6000	22.4	24.4	19.9	18.7	22	18	16	30	23	22
		(16.3)	(9.1)	(7.7)	(15)	(9)	(7)	(18)	(10)	(9)
Quasi-Unit-Disk-Graphen, $r = 116$										
1200	4.8	16.3	11.0	10.2	14	10	9	20	13	12
		(14.7)	(8.0)	(7.0)	(13)	(7)	(6)	(17)	(9)	(8)
2000	7.8	17.8	12.6	11.8	16	11	10	21	14	16
		(15.3)	(8.3)	(7.1)	(14)	(8)	(7)	(17)	(9)	(8)
4000	15.6	20.8	16.0	14.7	19	14	12	24	19	18
		(16.6)	(9.2)	(7.7)	(16)	(9)	(7)	(18)	(10)	(9)
6000	23.4	23.1	18.4	17.0	21	17	16	27	22	21
		(17.3)	(9.6)	(8.1)	(16)	(9)	(7)	(19)	(11)	(9)
Waxman, $r = 150$										
1200	3.7	15.6	9.3	8.1	14	8	7	18	11	10
		(15.4)	(8.3)	(7.0)	(14)	(8)	(6)	(18)	(9)	(8)
2000	5.9	16.9	10.2	8.8	16	9	8	20	12	11
		(16.9)	(9.1)	(7.5)	(15)	(8)	(7)	(19)	(10)	(8)
4000	11.7	19.3	11.9	10.2	18	11	9	21	14	12
		(18.9)	(10.2)	(8.4)	(17)	(9)	(8)	(22)	(11)	(10)
6000	17.6	20.7	12.8	11.1	19	12	10	24	17	13
		(20.3)	(11.2)	(9.0)	(18)	(10)	(8)	(23)	(13)	(10)

Tabelle 5.4: Länge der Adressen, 3D, 2-4-8 splits. In Klammern der Wert bei vorzeitigem Abbruch des Aufbaus der HBR-Struktur.

n	HBR	GEO^{SP}	GEO^{HBR}	MDS^{SP}	MDS^{HBR}	γ GEO	γ MDS
Unit-Disk-Graphen, $r = 50$							
600	4.9	24.8	27.5	7.6	8.9	15.8	59.7
800	16.2	22.3	30.5	8.6	11.3	23.9	62.1
1000	18.0	14.4	19.6	6.5	8.4	42.9	75.3
1250	19.3	8.8	11.5	4.8	5.8	66.2	85.7
1500	20.0	6.0	7.3	4.0	4.5	82.7	92.4
2000	21.1	3.7	3.9	3.4	3.6	96.0	97.4
2500	22.0	2.8	2.8	2.9	2.9	99.2	99.2
Quasi-Unit-Disk-Graphen, $r = 58$							
600	6.8	25.9	29.7	8.4	9.9	15.4	57.7
800	16.7	22.8	30.3	9.5	12.2	26.0	62.1
1000	19.5	16.0	21.2	7.6	9.7	46.3	74.6
1250	20.6	11.3	13.5	6.6	7.7	70.2	84.7
1500	21.4	8.9	9.9	6.2	6.8	85.5	91.1
2000	23.0	6.9	7.1	5.8	6.0	97.0	96.5
2500	24.0	6.0	6.0	5.0	5.1	99.4	98.9
Waxman, $r = 93$							
600	20.1	27.7	38.1	13.9	18.4	20.1	50.5
800	25.1	23.9	31.5	14.5	19.0	38.6	57.0
1000	27.7	21.3	26.3	15.0	18.9	57.6	62.8
1250	30.5	19.1	22.2	15.5	19.1	73.6	67.8
1500	32.6	18.0	20.1	15.1	18.1	81.5	72.8
2000	35.5	16.3	17.6	15.1	17.5	88.3	77.9
2500	37.5	15.2	16.3	14.8	16.8	90.9	81.6

Tabelle 5.5: Durchschnittlicher Overhead in Prozent bei zufälliger Verteilung der Knoten in der Ebene und die Erfolgsquoten γ für reines Greedy-Routing. Aufbau der HBR-Koordinaten mit $d = 2$ Aufteilungen pro Schritt.

n	HBR		GEO^{SP}		GEO^{HBR}		MDS^{SP}		MDS^{HBR}	
	$d = 4$	$d = 8$	$d = 4$	$d = 8$	$d = 4$	$d = 8$	$d = 4$	$d = 8$	$d = 4$	$d = 8$
Unit-Disk-Graphen, $r = 50$										
600	3.7	3.4	24.8	25.2	26.8	26.8	7.6	7.9	8.3	8.6
800	10.3	9.2	22.1	23.0	26.7	27.1	8.4	8.6	10.0	10.0
1000	12.1	14.3	14.9	15.0	18.0	18.3	6.6	6.5	7.5	7.6
1250	13.2	16.3	8.9	9.0	10.3	10.8	5.0	4.9	5.5	5.6
1500	14.0	16.6	5.8	5.9	6.4	6.7	4.3	4.1	4.6	4.4
2000	15.2	17.3	3.6	3.7	3.8	3.8	3.5	3.4	3.7	3.5
2500	15.9	17.7	2.7	2.8	2.7	2.8	2.7	2.8	2.7	2.8
Quasi-Unit-Disk-Graphen, $r = 58$										
600	4.9	5.0	26.3	26.1	28.9	28.6	8.3	8.7	9.1	9.6
800	11.2	12.0	23.0	22.8	27.5	27.2	9.1	9.0	10.6	10.5
1000	13.1	16.0	15.9	15.9	18.7	19.1	7.5	7.5	8.5	8.6
1250	14.4	17.6	11.5	11.4	12.8	13.0	6.7	6.7	7.4	7.4
1500	15.2	18.4	9.0	9.1	9.6	9.8	5.9	6.1	6.2	6.5
2000	16.9	19.2	6.9	7.0	7.0	7.1	5.5	5.5	5.6	5.7
2500	17.9	19.3	6.0	6.0	6.0	6.0	5.1	5.0	5.1	5.1
Waxman, $r = 93$										
600	14.2	15.1	27.5	27.6	34.1	34.4	13.9	13.6	16.9	16.7
800	18.3	20.6	23.7	23.3	28.7	28.9	14.8	14.9	18.1	18.6
1000	21.0	23.8	21.2	21.3	24.6	25.4	15.1	15.1	18.2	18.5
1250	23.2	26.2	19.3	19.2	21.6	21.9	15.2	15.1	18.0	18.4
1500	25.3	27.5	18.0	17.9	19.8	19.8	15.2	15.4	17.7	18.4
2000	27.9	29.7	16.3	16.2	17.4	17.5	15.1	15.0	17.1	17.4
2500	30.0	31.2	15.2	15.2	16.1	16.2	14.9	14.9	16.7	16.9

Tabelle 5.6: Durchschnittlicher Overhead in Prozent bei zufälliger Verteilung der Knoten in der Ebene bei $d = 4$ und $d = 8$ Aufteilungen für die HBR-Koordinaten.

n	HBR	GEO^{SP}	GEO^{HBR}	MDS^{SP}	MDS^{HBR}	γ GEO	γ MDS
Unit-Disk-Graphen, $r = 50$							
1750	5.8	28.3	32.3	9.5	10.6	10.9	51.7
2000	8.6	28.8	34.5	9.7	11.5	12.4	53.0
2500	15.1	29.6	39.6	12.0	15.1	14.3	51.7
3000	18.8	28.7	40.3	12.6	16.3	16.4	51.7
4000	23.2	26.8	39.5	13.4	17.7	20.4	54.0
Quasi-Unit-Disk-Graphen, $r = 58$							
1750	7.6	29.9	35.3	10.3	11.9	10.5	50.4
2000	9.8	31.0	37.5	11.3	13.3	12.3	51.3
2500	16.3	32.1	42.9	13.1	16.6	14.3	50.6
3000	20.8	31.3	44.0	13.9	18.2	16.8	51.3
4000	23.7	30.2	43.2	15.1	20.1	20.9	52.9
Waxman, $r = 93$							
1750	13.9	36.0	45.8	16.2	20.1	9.3	40.0
2000	17.4	37.5	49.5	17.3	22.3	10.9	40.7
2500	23.8	39.7	55.1	20.5	27.4	13.7	39.3
3000	27.0	40.0	56.0	22.0	29.6	16.4	40.1
4000	30.5	39.8	56.0	23.9	32.3	20.8	41.5

Tabelle 5.7: Durchschnittlicher Overhead in Prozent bei Verteilung der Knoten mit Hindernissen und die Erfolgsquoten γ für reines Greedy-Routing. Aufbau der HBR-Koordinaten mit $d = 2$ Aufteilungen pro Schritt.

n	HBR		GEO^{SP}		GEO^{HBR}		MDS^{SP}		MDS^{HBR}	
	$d = 4$	$d = 8$	$d = 4$	$d = 8$	$d = 4$	$d = 8$	$d = 4$	$d = 8$	$d = 4$	$d = 8$
Unit-Disk-Graphen, $r = 50$										
1750	3.7	3.3	26.8	27.2	29.2	29.1	8.5	8.8	9.3	9.6
2000	5.6	4.6	28.0	28.6	31.5	31.2	9.7	9.9	10.8	11.0
2500	9.4	8.3	29.6	29.3	35.7	34.2	12.0	11.8	14.0	13.5
3000	11.8	10.0	28.7	29.3	35.8	35.1	12.7	12.9	15.4	15.1
4000	14.7	13.3	26.9	26.8	34.7	33.9	13.6	13.2	16.5	15.8
Quasi-Unit-Disk-Graphen, $r = 58$										
1750	4.9	3.6	29.5	29.1	32.8	31.2	9.6	9.5	10.7	10.3
2000	7.3	5.5	30.3	30.7	35.1	34.0	10.8	10.9	12.5	12.2
2500	10.6	8.8	32.1	32.3	38.8	37.6	13.2	13.2	15.5	15.3
3000	13.0	11.0	31.4	31.6	38.9	38.2	14.1	14.3	17.1	16.9
4000	15.7	14.0	30.2	30.2	38.3	37.6	15.1	15.1	18.4	18.2
Waxman, $r = 93$										
1750	9.8	8.0	35.6	35.6	42.2	40.8	15.5	15.5	18.6	18.1
2000	12.9	10.1	37.5	38.0	45.7	44.1	17.3	17.6	21.1	21.0
2500	17.3	13.5	39.6	39.5	50.5	47.5	20.5	20.4	25.9	25.0
3000	19.2	16.2	39.8	40.0	51.2	49.2	21.8	22.0	27.9	27.4
4000	22.9	19.1	39.9	40.0	52.0	49.8	23.8	23.6	30.6	29.6

Tabelle 5.8: Durchschnittlicher Overhead in Prozent bei Verteilung der Knoten mit Hindernissen bei $d = 4$ und $d = 8$ Aufteilungen für die HBR-Koordinaten.

n	HBR	GEO^{SP}	GEO^{HBR}	MDS^{SP}	MDS^{HBR}	γ GEO	γ MDS
Unit-Disk-Graphen, $r = 100$							
1200	28.4	27.0	46.7	14.3	21.0	13.9	50.6
1600	34.5	20.8	36.0	11.8	17.5	30.6	64.3
2000	35.7	15.2	23.9	9.6	13.3	52.0	75.7
2500	37.2	11.1	15.5	7.8	9.8	72.9	85.6
3000	38.2	8.4	10.5	6.5	7.6	86.1	91.8
4000	39.7	5.7	6.2	5.1	5.5	96.7	97.3
5000	40.7	4.5	4.6	4.4	4.5	99.1	99.1
Quasi-Unit-Disk-Graphen, $r = 116$							
1200	32.0	26.3	45.9	14.3	21.1	18.1	53.7
1600	36.6	20.5	33.5	12.7	18.4	39.1	65.5
2000	38.5	16.4	23.8	11.1	14.7	60.2	76.1
2500	40.1	13.1	16.6	9.9	12.0	79.1	85.0
3000	41.4	11.1	12.8	9.1	10.4	89.7	90.5
4000	43.5	8.8	9.3	8.0	8.5	97.5	95.9
5000	45.1	7.7	7.8	7.3	7.5	99.3	98.1
Waxman-Graphen, $r = 150$							
1200	35.8	31.1	56.8	19.5	30.0	9.7	39.8
1600	41.9	29.0	50.8	20.1	31.0	20.8	44.8
2000	46.2	27.4	44.0	20.1	30.4	34.0	49.9
2500	49.9	25.5	37.7	20.3	29.6	48.9	54.9
3000	52.4	24.2	33.5	20.2	28.8	59.5	59.2
4000	56.5	22.3	28.6	20.3	27.5	71.5	65.2
5000	59.2	21.2	26.2	20.1	26.6	77.0	69.2

Tabelle 5.9: Durchschnittlicher Overhead in Prozent bei gleichmäßiger Verteilung der Knoten in drei Dimensionen und die Erfolgsquoten γ für reines Greedy-Routing. Aufbau der HBR-Koordinaten mit $d = 2$ Aufteilungen pro Schritt.

n	HBR		GEO^{SP}		GEO^{HBR}		MDS^{SP}		MDS^{HBR}	
	$d=4$	$d=8$	$d=4$	$d=8$	$d=4$	$d=8$	$d=4$	$d=8$	$d=4$	$d=8$
Unit-Disk-Graphen, $r=100$										
1200	21.7	17.8	26.9	27.1	41.7	38.8	14.0	13.9	18.9	17.8
1600	27.2	22.3	20.7	20.7	32.2	29.9	11.6	11.8	15.9	15.2
2000	28.9	23.2	15.4	15.4	22.3	20.8	9.6	9.6	12.3	11.9
2500	30.7	24.1	11.0	11.0	14.5	13.6	7.7	7.7	9.3	9.0
3000	31.9	24.2	8.3	8.4	10.0	9.6	6.5	6.6	7.4	7.2
4000	33.5	25.1	5.7	5.7	6.1	6.0	5.2	5.1	5.4	5.4
5000	34.2	26.1	4.4	4.4	4.5	4.5	4.3	4.3	4.4	4.4
Quasi-Unit-Disk-Graphen, $r=116$										
1200	24.8	20.2	26.5	26.4	41.4	38.0	14.2	14.3	19.5	18.6
1600	28.8	23.9	20.7	20.8	30.6	28.8	12.6	12.5	16.8	15.8
2000	31.1	24.6	16.4	16.4	22.2	20.9	11.1	11.2	14.0	13.4
2500	33.0	26.0	13.1	13.1	15.9	15.2	9.9	10.0	11.6	11.3
3000	34.3	26.7	11.0	11.1	12.3	12.0	9.1	9.1	10.1	9.9
4000	36.5	28.2	8.9	8.8	9.2	9.1	8.0	8.0	8.4	8.3
5000	37.7	29.2	7.6	7.6	7.7	7.7	7.2	7.3	7.4	7.4
Waxman-Graphen, $r=150$										
1200	27.5	22.5	31.0	31.1	50.8	47.0	19.5	19.3	27.9	26.3
1600	33.7	28.1	29.2	29.0	46.3	43.1	20.0	20.0	28.8	27.6
2000	37.3	31.2	27.1	27.3	40.6	38.9	20.2	20.3	28.6	27.9
2500	40.9	34.0	25.3	25.4	35.4	34.1	20.2	20.2	28.1	27.3
3000	43.1	35.6	24.1	24.1	31.8	30.7	20.3	20.3	27.6	26.8
4000	47.2	38.6	22.3	22.3	27.7	27.0	20.2	20.3	26.6	26.0
5000	49.7	40.3	21.2	21.3	25.5	25.2	20.1	20.2	25.7	25.3

Tabelle 5.10: Durchschnittlicher Overhead in Prozent bei zufälliger Verteilung der Knoten in drei Dimensionen bei $d = 4$ und $d = 8$ Aufteilungen für die HBR-Koordinaten.

	HBR		
n	$d = 2$	$d = 4$	$d = 8$
UDG, $r = 50$			
600	5.4	3.9	3.4
800	15.2	10.0	10.1
1000	17.4	10.9	13.6
1250	17.8	11.7	14.9
1500	17.7	11.8	14.7
2000	18.2	12.1	14.6
2500	18.7	12.3	13.5
QUDG-Graphen, $r = 58$			
600	7.2	5.4	4.4
800	16.0	10.6	11.3
1000	17.8	11.6	14.8
1250	18.1	12.1	15.0
1500	18.3	12.1	15.5
2000	18.8	12.5	14.4
2500	19.4	12.8	14.4
Waxman-Graphen, $r = 93$			
600	19.5	12.4	14.0
800	20.9	14.5	18.1
1000	21.6	14.9	19.9
1250	22.2	15.4	21.1
1500	22.6	15.8	22.9
2000	23.5	16.3	22.8
2500	24.0	16.6	21.9

Tabelle 5.11: Durchschnittlicher Overhead für HBR in Prozent bei gleichmäßiger Verteilung der Knoten in zwei Dimensionen. Euklidische Kantengewichte.

	HBR		
n	$d = 2$	$d = 4$	$d = 8$
UDG, $r = 100$			
1200	29.3	21.5	17.4
1600	33.7	26.3	21.0
2000	34.6	27.6	21.6
2500	34.6	28.7	21.5
3000	34.1	29.4	21.1
4000	34.0	29.5	20.8
5000	33.7	29.5	20.5
QUDG-Graphen, $r = 116$			
1200	31.6	24.4	19.6
1600	34.9	27.5	22.5
2000	35.4	29.0	22.2
2500	35.6	29.8	22.0
3000	35.5	30.4	21.5
4000	34.9	30.5	21.4
5000	35.0	30.7	21.4
Waxman-Graphen, $r = 150$			
1200	34.6	26.3	21.6
1600	38.8	31.1	24.5
2000	40.5	33.1	26.9
2500	41.5	34.7	27.1
3000	42.1	35.4	27.5
4000	42.6	36.6	27.5
5000	42.9	37.2	27.5

Tabelle 5.12: Durchschnittlicher Overhead für HBR in Prozent bei gleichmäßiger Verteilung der Knoten in drei Dimensionen. Euklidische Kantengewichte.

Zusammenfassung

In dieser Arbeit haben wir uns mit der Fragestellung beschäftigt, wie gut verschiedene Koordinaten-Modelle für Routingverfahren auf Sensornetzwerken geeignet sind.

Im ersten Abschnitt hat sich herausgestellt, dass die Rekonstruktion der tatsächlichen Koordinaten mit den Verfahren der *Multidimensionalen Skalierung* für eine deutliche Steigerung der Erfolgsquoten bei Verwendung von Greedy-Routing sorgen kann.

Durch Modifikationen der Grundidee ist wahlweise eine genauere Positionierung der Knoten, oder eine weitere Verbesserung der Erfolgsquoten für Greedy-Routing möglich. Letzteres gilt besonders unter erschwerten Bedingungen, d.h. mit sehr unvollständigen Informationen über die Struktur des Netzwerkes. Ein gleichzeitiges Erreichen beider Ziele ist im Allgemeinen nicht möglich.

Durch eine schnell mögliche Vorab-Analyse lassen sich Knoten an den Eckpunkten eines Netzwerkes identifizieren. Werden die iterativ zu berechnenden virtuellen Knotenpositionen auf dieser Basis geschickt initialisiert, verringert sich die notwendige Anzahl von Iterationen zur Berechnung der Koordinaten deutlich. Ein weiterer nützlicher Effekt dabei sind seltener auftretende Fehlpositionierungen einzelner Knoten oder kleinerer Teile des Netzwerkes, wodurch sowohl die Güte der Einbettung verbessert wird, als auch ein positiver Einfluss auf das Greedy-Routing erreicht werden kann.

Bei der experimentellen Auswertung haben wir Wert darauf gelegt, die Verfahren nicht nur unter recht strengen formalen Bedingungen an die Netzwerke zu testen (d.h. Unit-Disk-Graphen und Quasi-Unit-Disk-Graphen), sondern auch mit einem weniger sicherem Kantenmodell, welches jedoch in vielen Fällen näher an die physikalisch bedingte Realität heranreichen sollte. Erfreulicherweise ist zu beobachten, dass die in dieser Arbeit verwendeten Verfahren auch unter in der Theorie schwieriger zu analysierenden Voraussetzungen sehr gut funktionieren und gute Ergebnisse liefern. Es gilt aber

auch hier, dass die Ergebnisse besser werden, wenn die einzelnen Knoten im Netzwerk konsistenter über eine Kante verbunden sind, wenn sie eine geringe Entfernung in der Ebene zueinander haben.

In Kapitel 4 wurde ein neues Verfahren untersucht, das Ränder und Löcher (also nicht von Sensorknoten abgedeckte Bereiche) erkennen kann. Dieses Verfahren kommt ebenfalls ohne strenge formale Anforderungen an die Netzwerke aus und liefert besonders dann noch gute Ergebnisse, wenn die Netzwerke insgesamt weniger dicht sind. Damit grenzt es sich klar von vielen anderen Verfahren ab, die einen hohen Knotengrad benötigen. Die Grenzen dieses Verfahrens beginnen dort, wo die intuitive Vorstellung von einem Loch im Netzwerk wegen einer zu variablen Kantenlänge nicht mit der Struktur des Graphen in Einklang zu bringen ist. Dieses Dilemma ist aber unter den gewählten sehr lockeren formalen Bedingungen nicht zu lösen.

Für die Verbesserung der Positionierung der Knoten lässt sich die Information über Randknoten jedoch nicht nutzen. Eine darauf basierende Heuristik führt in der Regel nicht zu einer Verbesserung, sondern zu einer Verschlechterung der Ergebnisse. Dieser Effekt deckt sich teilweise mit den Beobachtungen aus der experimentellen Auswertung der Multidimensionalen Skalierung, bei der unter einigen Parameter-Kombinationen ein Mehr an Information (wie eine Abschätzung der Kantenlänge) ebenfalls zu einer Verschlechterung der Ergebnisse geführt hat.

Im letzten Kapitel haben wir das *Hierarchische Bipartitions-Routing (HBR)* vorgestellt, das die Zustellung einer Nachricht zum Zielknoten garantiert. Es kann dabei wahlweise als alleinstehendes Routing-Protokoll verwendet werden, oder als Ausweichprotokoll für den Fall, dass Greedy-Routing fehlschlägt. Die für dieses Routing notwendigen Informationen können verteilt im Netzwerk bestimmt werden und müssen nicht aus einer globalen Sicht auf das Netzwerk extern ermittelt werden. Die Anzahl der Einträge in den Routingtabellen der einzelnen Knoten liegt im Mittel in $\mathcal{O}(\log |V|)$, ebenso die Länge der Adressen der einzelnen Knoten. Damit ist das Verfahren gut für Sensornetzwerke geeignet, die mit einem stark eingeschränktem Speicher zurecht kommen müssen.

Die experimentelle Auswertung zeigt, dass HBR besonders gut für dünne Netzwerke geeignet ist und dort Routingwege liefert, die nur minimal länger sind als ein optimaler Weg. Bei dichteren Netzwerken empfiehlt sich HBR als Ausweichprotokoll.

Eine Erweiterung der Grundidee, bei der keine Bipartition, sondern eine Aufteilung in mehr als zwei Teile erfolgt, kann die Routingwege weiter verbes-

sern, benötigt aber für den Aufbau auch einen höheren Kommunikationsaufwand. Die für die Güte der Routingwege optimale Anzahl an Aufteilungen entspricht dabei der anschaulich sinnvollen Aufteilung, also der Anzahl der „Ecken“ des ausgelegten Netzwerkes.

Anhang

Verwendete Manuskripte

In dieser Dissertation wurde folgende bereits veröffentlichte Arbeit in Teilen übernommen und für den Kontext dieser Dissertation angepasst:

> **Titel:** Hierarchical bipartition routing for delivery guarantee in sparse wireless ad hoc sensor networks with obstacles
>
> **Autorenliste:** Daniel Gaußmann, Stefan Hoffmann und Egon Wanke
>
> Veröffentlicht im Rahmen der *The 2012 International Conference on Wireless Networks*, vgl. [GHW12] bzw. [GHW13]

An dem Entwurf und der Analyse des dort vorgestellten Verfahrens war ich maßgeblich beteiligt. Die experimentelle Auswertung desselben für das Manuskript wurde zwecks gegenseitiger Kontrolle von Prof. Dr. Egon Wanke und mir selbst in zwei unabhängig voneinander erstellten Implementierungen durchgeführt.

Für die experimentelle Auswertung in dieser Dissertationsschrift wurde die Auswertung an den hier vorliegenden Kontext angepasst und erneut durchgeführt. Die Ausführungen aus dem Manuskript wurden hier ergänzt um eine detaillierte Betrachtung der Kosten für den Aufbau der Struktur, sowie durch die beschriebenen möglichen Modifikationen bzw. Erweiterungen des Verfahrens.

Literaturverzeichnis

[ACM96] D. Aingworth, C. Chekuri, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '96, pages 547–553, Philadelphia, PA, USA, 1996. Society for Industrial and Applied Mathematics.

[APM05] Ian F Akyildiz, Dario Pompili, and Tommaso Melodia. Underwater acoustic sensor networks: research challenges. *Ad hoc networks*, 3(3):257–279, 2005.

[ASSC02] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393 – 422, 2002.

[Bag05] Aline Baggio. Wireless sensor networks in precision agriculture. In *ACM Workshop on Real-World Wireless Sensor Networks (REALWSN 2005), Stockholm, Sweden*, volume 20. Citeseer, 2005.

[Bau58] Heinz Bauer. Minimalstellen von funktionen und extremalpunkte. *Archiv der Mathematik*, 9(4):389–393, 1958.

[BCG+04] Paul Boone, Edgar Chavez, Lev Gleitzky, Evangelos Kranakis, Jaroslav Opatrny, Gelasio Salazar, and Jorge Urrutia. Morelia test: Improving the efficiency of the gabriel test and face routing in ad-hoc networks. In *International Colloquium on Structural Information and Communication Complexity*, pages 23–34. Springer, 2004.

[BFLO06] Krists Boitmanis, Kārlis Freivalds, Pēteris Lediņš, and Rūdolfs Opmanis. Fast and simple approximation of the diameter and radius of a graph. In *Proceedings of the 5th International Conference on Experimental Algorithms*, WEA'06, pages 98–108,

Berlin, Heidelberg, 2006. Springer-Verlag.

[BG97] Ingwer Borg and Patrick Groenen. *Modern multidimensional scaling, theory and applications*. Springer, 1997.

[BK98] Heinz Breu and David G. Kirkpatrick. Unit disk graph recognition is np-hard. *Comput. Geom. Theory Appl.*, 9(1-2):3–24, 1998.

[BMPC08] K. Benkic, M. Malajner, P. Planinsic, and Z. Cucej. Using rssi value for distance estimation in wireless sensor networks based on zigbee. *Systems, Signals and Image Processing, 2008. IWSSIP 2008. 15th International Conference on*, 2008.

[BMSU01] Prosenjit Bose, Pat Morin, Ivan Stojmenović, and Jorge Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wirel. Netw.*, 7(6):609–616, November 2001.

[Boc] Sergey Bochkanov. ALGLIB. `https://www.alglib.net`.

[BRR08] Elizabeth A Basha, Sai Ravela, and Daniela Rus. Model-based monitoring for early warning flood detection. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 295–308. ACM, 2008.

[CCDU05] A. Caruso, S. Chessa, S. De, and A. Urpi. Gps free coordinate assignment and routing in wireless sensor networks. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 1, pages 150–160 vol. 1, March 2005.

[CMT07] Edgar Chávez, Nathalie Mitton, and Héctor Tejeda. Routing in wireless networks with position trees. In *International Conference on Ad-Hoc Networks and Wireless*, pages 32–45. Springer, 2007.

[Dij59] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[DPSJvL11] Josep DÃaz, Olli Pottonen, Maria Serna, and Erik Jan van Leeuwen. On the complexity of metric dimension. In *Proceedings of the 20th Annual European Conference on Algorithms*, 07 2011.

[FGG06] Qing Fang, Jie Gao, and Leonidas J. Guibas. Locating and bypassing holes in sensor networks. *Mob. Netw. Appl.*, 11(2):187–200, April 2006.

[FPW09] Roland Flury, Sriram V Pemmaraju, and Roger Wattenhofer. Greedy routing with bounded stretch. In *INFOCOM 2009, IEEE*, pages 1737–1745. IEEE, 2009.

[FRZ+05] Rodrigo Fonseca, Sylvia Ratnasamy, Jerry Zhao, Cheng Tien Ee, David Culler, Scott Shenker, and Ion Stoica. Beacon vector routing: Scalable point-to-point routing in wireless sensornets. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 329–342, Berkeley, CA, USA, 2005. USENIX Association.

[Fun05] Stefan Funke. Topological hole detection in wireless sensor networks and its applications. In *Proceedings of the 2005 Joint Workshop on Foundations of Mobile Computing*, DIALM-POMC '05, pages 44–53, New York, NY, USA, 2005. ACM.

[GG03] F. Gustafsson and F. Gunnarsson. Positioning using time-difference of arrival measurements. *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on*, 2003.

[GHS83] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees, 1983.

[GHW12] Daniel Gaußmann, Stefan Hoffmann, and Egon Wanke. Hierarchical bipartition routing for delivery guarantee in sparse wireless ad hoc sensor networks with obstacles. In *The 2012 International Conference on Wireless Networks*, pages 3–9. CSREA Press, 2012.

[GHW13] Daniel Gaußmann, Stefan Hoffmann, and Egon Wanke. Hierarchical bipartition routing for delivery guarantee in sparse wireless ad hoc sensor networks with obstacles. *CoRR*, abs/1307.1994, 2013.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[GL] Eric Grange and Mike Lischke. GLScene. `http://glscene.sourceforge.net`.

[GMG07] J Antonio Garcia-Macias and Javier Gomez. Manet versus wsn. In *Sensor networks and configuration*, pages 369–388. Springer, 2007.

[HHL88] Sandra M. Hedetniemi, Stephen T. Hedetniemi, and Arthur L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 1988.

[HKB99] Wendi Heinzelman, Joanna Kulik, and Hari Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. *Proceedings of the 6th Annual ACM International Conference on Mobile Computing and Networking*, 08 1999.

[HKP+05] J. Hromkovic, R. Klasing, A. Pelc, P. Ruzicka, and W. Unger. *Dissemination of Information in Communication Networks: Broadcasting, Gossiping, Leader Election, and Fault-Tolerance (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[HPJ02] Yih-Chun Hu, Adrian Perrig, and David B Johnson. Wormhole detection in wireless ad hoc networks. *Department of Computer Science, Rice University, Tech. Rep. TR01-384*, 2002.

[HSBH+09] Rebecca N Handcock, Dave L Swain, Greg J Bishop-Hurley, Kym P Patison, Tim Wark, Philip Valencia, Peter Corke, and Christopher J ONeill. Monitoring animal behaviour and environmental interactions using wireless sensor networks, gps collars and satellite remote sensing. *Sensors*, 9(5):3586–3603, 2009.

[HW12] Stefan Hoffmann and Egon Wanke. Metric dimension for gabriel unit disk graphs is np-complete. In *ALGOSENSORS*, 2012.

[JZ04] Xiang Ji and Hongyuan Zha. Sensor positioning in wireless ad-hoc sensor networks using multidimensional scaling. *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, 4:2652–2661, 2004.

[KFPF06] Alexander Kröller, Sándor P. Fekete, Dennis Pfisterer, and Stefan Fischer. Deterministic boundary recognition and topology extraction for large sensor networks. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 1000–1009, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics.

[KKJ+09] Roland Kays, Bart Kranstauber, Patrick Jansen, Chris Carbone, Marcus Rowcliffe, Tony Fountain, and Sameer Tilak. Camera traps as sensor networks for monitoring animal communities. In *Local Computer Networks, 2009. LCN 2009. IEEE 34th Con-*

ference on, pages 811–818. IEEE, 2009.

[KKP99] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: Mobile networking for smart dust. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, MobiCom '99, pages 271–278, New York, NY, USA, 1999. ACM.

[KMW04] Fabian Kuhn, Thomas Moscibroda, and Rogert Wattenhofer. Unit disk graph approximation. In *Proceedings of the 2004 Joint Workshop on Foundations of Mobile Computing*, DIALM-POMC '04, pages 17–23, New York, NY, USA, 2004. ACM.

[KNS06] Johnson Kuruvila, Amiya Nayak, and Ivan Stojmenovic. Progress and location based localized power aware routing for ad hoc and sensor wireless networks. *International Journal of Distributed Sensor Networks*, 2(2):147 – 159, 2006.

[KSU99] Evangelos Kranakis, Harvinder Singh, and Jorge Urrutia. Compass routing on geometric networks. In *IN PROC. 11 TH CANADIAN CONFERENCE ON COMPUTATIONAL GEOMETRY*, pages 51–54, 1999.

[MB64] G. Marsaglia and T. A. Bray. A convenient method for generating normal variables. *SIAM Review*, 6(3):260–264, 1964.

[MRSRS08] N. Mitton, T. Razafindralambo, D. Simplot-Ryl, and I. Stojmenovic. Hector is an energy efficient tree-based optimized routing protocol for wireless networks. In *Mobile Ad-hoc and Sensor Networks, 2008. MSN 2008. The 4th International Conference on*, pages 31–38, Dec 2008.

[MWV00] Navneet Malpani, Jennifer L Welch, and Nitin Vaidya. Leader election algorithms for mobile ad hoc networks. In *Proceedings of the 4th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 96–103. ACM, 2000.

[PE08] FJ Pierce and TV Elliott. Regional and on-farm wireless sensor networks for agricultural systems in eastern washington. *Computers and electronics in agriculture*, 61(1):32–43, 2008.

[PH07] Lilia Paradis and Qi Han. A survey of fault management in wireless sensor networks. *Journal of Network and systems management*, 15(2):171–190, 2007.

[Rea00] J. Rabaey and et al. Picoradio: Ad-hoc wireless networking of . . ., 2000.

[San05] Paolo Santi. Topology control in wireless ad hoc and sensor networks. *ACM computing surveys (CSUR)*, 37(2):164–194, 2005.

[SHK06] Byungrak Son, Yong-sork Her, and Jung-Gyu Kim. A design and implementation of forest-fires surveillance system based on wireless sensor networks for south korea mountains. *International Journal of Computer Science and Network Security (IJCSNS)*, 6(9):124–130, 2006.

[SL01] Ivan Stojmenovic and Xu Lin. Power-aware localized routing in wireless networks. *IEEE Trans. Parallel Distrib. Syst.*, 12(11):1122–1133, nov 2001.

[sLwScS07] Jin shyan Lee, Yu wei Su, and Chung chou Shen. A comparative study of wireless protocols: Bluetooth, uwb, zigbee, and wi-fi. In *in The 33rd Annual Conference of the IEEE Industrial Electronics Society (IECON*, pages 46–51, 2007.

[STR69] V. STRASSEN. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[Tou80] Godfried T. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, 12(4):261 – 268, 1980.

[Ume91] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(4):376–380, April 1991.

[VHW19] Duygu Vietz, Stefan Hoffmann, and Egon Wanke. Computing the metric dimension by decomposing graphs into extended biconnected components. In *International Workshop on Algorithms and Computation*, pages 175–187. Springer, 2019.

[Wax88] B.M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6:1917–1622, 1988.

[WGM06] Yue Wang, Jie Gao, and Joseph S.B. Mitchell. Boundary recognition in sensor networks by topological methods. In *Proceedings of the 12th Annual International Conference on Mobile Computing and Networking*, MobiCom '06, pages 122–133, New York, NY, USA, 2006. ACM.

[WX07] Chen Wang and Li Xiao. Sensor localization under limited measurement capabilities. *IEEE Network*, 12(4), 2007.

[WZ03] Roger Wattenhofer and Aaron Zollinger. Xtc: A practical topology control algorithm for ad-hoc networks. In *In 4th International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN*, 2003.

[ZZZZ08] Giovanni Zanca, Francesco Zorzi, Andrea Zanella, and Michele Zorzi. Experimental comparison of rssi-based localization algorithms for indoor wireless sensor networks. In *Proceedings of the Workshop on Real-world Wireless Sensor Networks*, REALWSN '08, pages 1–5, New York, NY, USA, 2008. ACM.

Danksagung

Abschließend möchte ich allen Personen danken, die mich auf dem Weg zur Promotion begleitet und unterstützt haben.

Zunächst Prof. Dr. Egon Wanke für die gute Betreuung während der letzten Jahre, sowie Duygu Vietz und Stefan Hoffmann für die vielen fachbezogenen und auch für die nicht ganz so fachbezogenen Gespräche.

Nicht vergessen möchte ich dabei die Mitarbeiter und Mitarbeiterinnen in den Sekretariaten für die Unterstützung bei der Erledigung von Formalitäten und der Überwindung der Bürokratie.

Für das Korrekturlesen geht ein Dank an meinen Bruder Fabian.

Und nicht zuletzt ein ganz besonderer Dank an meine Eltern Rosi und Herbert, sowie meine Tante Sigrid, die mich - ganz besonders in den letzten Jahren - immer in vielfältiger Art unterstützt haben.