UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Ein Reaktiver Algorithmus für Geografisches Clustering

Bachelorarbeit

zur Erlangung des Grades eines Bachelor of Science
im Studiengang Informationsmanagement

vorgelegt von

Julian Mosen

Erstgutachter:	Prof. Dr. Hannes Frey Institut für Informatik
Zweitgutachter:	MSc. Florentin Neumann Institut für Informatik

Koblenz, im Mai 2014

Erklärung

Hiermit bestätige ich, dass die vorliegende Arbeit von mir selbständig verfasst wurde und ich keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet–Quellen – benutzt habe und die Arbeit von mir vorher nicht in einem anderen Prüfungsverfahren eingereicht wurde. Die eingereichte schriftliche Fassung entspricht der auf dem elektronischen Speichermedium (CD-Rom).

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....

(Ort, Datum) (Unterschrift)

Zusammenfassung

Beaconlose geografische Routingverfahren für Unit-Disk Graphen basieren auf einer beaconlosen Strategie und bieten einen Ansatz zur Verbesserung von lokalen Routingverfahren für Quasi-Unit-Disk Graphen. Der Großteil der lokalen geografischen Routingverfahren für Quasi-Unit-Disk Graphen benötigt 2-lokale Nachbarschaftsinformation und verursacht einen Nachrichtenoverhead. Der in dieser Arbeit entwickelte Beaconlose Clustering Algorithmus zeigt, dass sich Nachrichtenoverhead und Energieverbrauch eines bestehenden nicht beaconlosen Verfahrens mittels beaconloser Strategie optimieren lassen. Der Beaconlose Clustering Algorithmus basiert auf einem geografischen Clustering und konstruiert eine lokale Sicht auf einen ausgedünnten Graphen mit einer konstanten Anzahl von Knoten pro Flächeneinheit. Neben der detaillierten Beschreibung des Algorithmus beinhaltet diese Arbeit einen Korrektheitsbeweis und eine Implementierung mit anschließender Simulation zur Untersuchung der Verbesserung des bestehenden Verfahrens.

Abstract

Existing algorithms using a beaconless strategy for geographic routing in Unit-Disk Graphs offer an approach for improving non-beaconless routing algorithms for Quasi-Unit-Disk Graphs. The majority of the aforementioned non-beaconless routing algorithms for Quasi-Unit-Disk Graphs are based on collecting 2-hop neighbourhood information. As shown by the Beaconless Clustering Algorithm developed in this thesis, a beaconless strategy can be used to enhance an existing non-beaconless algorithm by reducing message overhead and power consumption. The Beaconless Clustering algorithm is based on geographic clustering and constructs a sparse graph with constant number of vertices per unit area. The thesis at hand contains a detailed description of the algorithm, a proof of correctness and a simulation for presenting reachable improvements.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Drahtlose Ad-Hoc-Netze bieten eine Möglichkeit zur Kommunikation ohne feste Infrastruktur, indem ausgehend von einem Netzteilnehmer Nachrichten auf direktem Weg zu einem oder mehreren erreichbaren Nachbarn gesendet werden. Der Pfad, den eine Nachricht vom Sender bis zum Empfänger zurücklegt, besteht dabei aus einzelnen Geräten, den sogenannten *Hops*. Aufgrund der Tatsache, dass es sich um drahtlose Geräte handelt, deren Betriebszeit durch Akkukapazitäten begrenzt wird, ist ein effizienter Energieverbrauch von besonderer Bedeutung. Damit beim Routing einer Nachricht im drahtlosen Netz wenig Energie verbraucht wird sollte der gewählte Pfad aus möglichst wenigen Hops bestehen. Gleichzeitig darf der Abstand der einzelnen Hops untereinander nicht zu groß sein, da höhere Sendeleistungen mehr Energie benötigen. Die verwendeten Verfahren zum Routen in drahtlosen Ad-Hoc-Netzen lassen sich grundsätzlich in *proaktive* und *reaktive* Verfahren unterteilen. Die klassischen proaktiven Verfahren bestimmen in einer initialen Phase die gesamte Netztopologie, sodass für jedes Paar im Netz ein Routingpfad bekannt ist. Damit diese Routingpfade auch für mobile Geräte in dynamischen Netzen mit sich ändernder Topologie bestimmt werden können, ist eine kontinuierliche Aktualisierung notwendig. Mittels periodischen Kontrollnachrichten in Form von *Beacons* wird das gesamte Netz geflutet um die Netzstruktur zu überprüfen und zu erneuern. Dieses *Beaconing* hat zur Folge, dass ein permanenter Nachrichtenoverhead generiert wird. Um den damit einhergehenden Energieverbrauch zu reduzieren ermitteln *reaktive* Verfahren nur bei Bedarf die Netztopologie. Die Kombination der Reaktivität mit *lokalen geografischen Routingverfahren*, deren Routing-Entscheidungen einzig und allein auf der

Kenntnis der geografischen Positionen einer begrenzten Menge an Nachbarn basieren, ermöglicht es, die Anzahl der aktiv am Verfahren beteiligten Geräten und die insgesamt benötigte Energie im Netz zu verringern.

Zur Modellierung drahtloser Ad-Hoc-Netze eignet sich das Modell der *Unit-Disk Graphen* (UDG). Jeder Teilnehmer im Netz stellt einen Knoten des Unit-Disk Graphen mit zugehörigem Senderadius r dar. Zwei Knoten besitzen im Graph eine gemeinsame Kante, wenn deren euklidischer Abstand $d \leq r$ ist. Im Bereich der Unit-Disk Graphen existieren in der Literatur lokale reaktive Verfahren zur garantierten Nachrichtenauslieferung, die durch den Einsatz einer *beaconlosen Strategie* hinsichtlich der Gesamtanzahl versendeter Nachrichten besonders effizient sind. Im Gegensatz zu klassischen Routingverfahren kommen beaconlose Verfahren ohne Beaconing aus, indem ohne die Kenntnis der Nachbarschaft eine Nachricht via Broadcast ausgesendet wird. Alle Empfänger des Broadcasts starten einen Timer, sodass der Knoten der sich für die Weiterleitung der Nachricht eignet, den kürzesten Timer besitzt und bei dessen Ablauf die Nachricht mittels Broadcast wiederholt. Beim Eintreffen der wiederholten Nachricht brechen die übrigen Knoten ihre Timer ab.

Neben dem Unit-Disk Graph Modell, das starke Annahmen über einen exakten Senderadius in Form einer perfekten Scheibe trifft, gibt es das *Quasi-Unit-Disk Graph* Modell (QUDG), welches gemäß der Realität von unregelmäßigeren Sendebereichen ausgeht. Im Quasi-Unit-Disk Graph sind zwei Knoten definitiv mit einer Kante verbunden, wenn deren euklidischer Abstand d kleiner als ein Radius $r_{\min}$ ist. Für den Fall dass d größer als ein Radius $r_{\max}$ ist, existiert mit Sicherheit keine Kante zwischen beiden Knoten. Wenn $r_{\min} < d \leq r_{\max}$ gilt, ist die Existenz einer Kante ungewiss. Ferner gilt für die im Rahmen dieser Arbeit betrachteten Quasi-Unit-Disk Graphen ein in der Literatur gängiges Verhältnis von $\frac{r_{\min}}{r_{\max}} \geq \frac{1}{\sqrt{2}}$ [1], dessen Begründung an dieser Stelle zu weit ginge. Die Eigenschaften der Quasi-Unit-Disk Graphen führt zu der Problematik, dass effiziente beaconlose Verfahren zur garantierten Nachrichtenauslieferung im Unit-Disk Graphen nicht ohne Weiteres für Quasi-Unit-Disk Graphen adaptiert werden können. Stattdessen wurden in der Literatur bisher lediglich reaktive lokale Routingverfahren für Quasi-Unit-Disk Graphen präsentiert, die für eine garantierte Nachrichtenauslieferung grundsätzlich einen *planaren Graphen* benötigen. Die generelle Eigenschaft der *Planarität* wird von einem Graphen erfüllt, wenn sich in der zweidimensionalen Einbettung kein Kantenpaar überschneidet. Zur Detek-

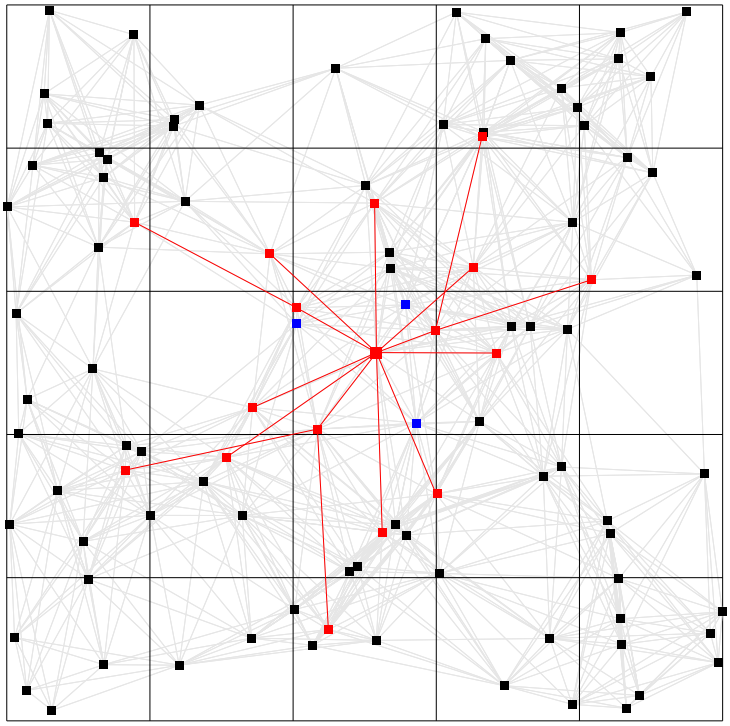


Abbildung 1.1: Das Ergebnis einer Ausführung des Beaconlosen Clustering Algorithmus im Unit-Disk Graphen. Die lokale Sicht auf den ausgedünnten Unit-Disk Graph besteht aus allen roten Knoten und Kanten. Blau gefärbte Knoten werden zwar am Verfahren beteiligt, sind jedoch kein Bestandteil des ausgedünnten Graphen. Alle schwarz dargestellten Knoten werden in keiner Form am BCA beteiligt.

tion der Kantenschnitten im Quasi-Unit-Disk Graph bedienen sich nahezu alle lokalen geografischen Routingverfahren der 2-lokalen Nachbarschaftsinformation und verursachen dadurch ein Nachrichtenaufkommen an dem alle Knoten der betroffenen Nachbarschaft beteiligt sind. Ein alternatives beaconloses Verfahren zur Bestimmung eines planaren Graphen könnte die Energieeffizienz durch einen verringerten Nachrichteneinsatz wesentlich verbessern.

Motiviert durch die Effizienzsteigerung bestehender lokaler Verfahren für Quasi-Unit-Disk Graphen, ist das Ziel dieser Arbeit einen Teilbeitrag zur beaconlosen Erweiterung eines bestehenden Verfahrens zur garantierten Nachrichtenauslieferung im Quasi-Unit-Disk Graph zu leisten. Das im folgenden Kapitel vorgestellte lokale geografische Routingverfahren von Lillis, Pemmaraju und Pirwani aus [11] verwendet zur Konstruktion eines planaren Graphen ein *geografisches Clustering*.

Das geografische Clustering beschreibt eine Methode zur *Topologiekontrolle*, mit der Knoten anhand ihrer geografischen Position im Raum auf ein *Cluster* abgebildet werden. Zusätzlich wird oftmals die unmittelbare Erreichbarkeit der Knoten eines Clusters untereinander vorausgesetzt. In einem Cluster werden die Rollen des *Clusterheads*, der *Brückenknoten* und der *Clustermember* vergeben. Ein Clusterhead verwaltet ausgehende Pfade zu anderen Clustern, die entweder direkt oder über einen Brückenknoten erreichbar sind. Die Konstruktion einer solchen Topologie mit geringstem Nachrichteneinsatz und einer minimalen Beteiligung zu bewerkstelligen ist die Aufgabe des in dieser Arbeit zu entwickelnden *Beaconlosen Clustering Algorithmus* (BCA). Dazu soll der BCA für einen gegebenen Graphen beaconlos eine *lokale Sicht* auf einen *ausgedünnten Graphen* generieren, der aus maximal konstant vielen Knoten pro Flächeneinheit besteht. Unter der Betrachtung eines *Connected Dominating Sets*, das an dieser Stelle nicht näher erläutert wird, entsprechen die im ausgedünnten Graphen enthaltenen Knoten den *Dominatoren* und die nicht enthaltenen den *dominierten Knoten*. Basierend auf geografischem Clustering und der beaconlosen Strategie bestimmt der BCA für einen Knoten u, der durch die Ausführung des Algorithmus zum Clusterhead wird, pro erreichbarem Cluster genau einen Knoten. Für u ist ein Cluster C_i genau dann erreichbar, wenn C_i einen 1-hop Nachbarn von u oder einen 1-hop Nachbarn der Brückenknoten aus dem Cluster von u enthält. Abbildung 1.1 zeigt das Ergebnis des BCA für einen Unit-Disk Graphen. Die Knoten des Graphen werden durch schwarze, blaue und rote Quadrate abgebildet. Graue und rote Verbindungslinien zwischen zwei Knoten repräsentieren Kanten im Graphen. Die zweidimensionale Ebene wird in 25 quadratische Kacheln unterteilt, sodass Knoten die in einer Kachel liegen zu einem Cluster zusammengefasst werden. Die aus dem BCA resultierende lokale Sicht auf den ausgedünnten Unit-Disk Graph besteht aus den rot gefärbten Knoten. Während rote und blaue Knoten durch das Versenden von Nachrichten aktiv am Verfahren beteiligt werden, bleiben schwarze Knoten gänzlich inaktiv.

Der Aufbau der Arbeit gestaltet sich wie folgt. In Kapitel 2 wird der Stand der Literatur ermittelt und ein bestehendes lokales Verfahren für Quasi-Unit-Disk Graphen ausgewählt, welches die effizienteste garantierte Nachrichtenauslieferung verspricht und sich besonders für die beaconlose Erweiterung eignet. Dazu wird in Kapitel 3 vorerst der BCA für Unit-Disk Graphen entwickelt und anschließend für Quasi-Unit-Disk Graphen erweitert, als auch hinsichtlich seiner Effizienz optimiert. Kapitel 4 zeigt mittels eines Korrektheitsbeweises, dass das

entwickelte Verfahren sowohl für Unit-Disk als auch für Quasi-Unit-Disk Graphen korrekt arbeitet. Auf die Implementierung beider Verfahren in Kapitel 5 folgt in Kapitel 6 eine Simulation zur Untersuchung der tatsächlichen Effizienz des entwickelten BCA im Vergleich zu einem bestehenden Verfahren. Das letzte Kapitel 7 fasst die Ergebnisse der Arbeit zusammen und gibt einen Ausblick für abschließende Arbeiten.

Kapitel 2

Literaturübersicht

In diesem Kapitel wird der aktuelle Stand der Literatur hinsichtlich lokaler Verfahren zur garantierten Nachrichtenauslieferung in Unit-Disk als auch in Quasi-Unit-Disk Graphen vorgestellt. Um eine garantierte Nachrichtenauslieferung zu gewährleisten setzen all diese Verfahren die Konstruktion eines planaren Subgraphen voraus. Während der Vorgang des Nachrichten Routings in vielen Verfahren auf dem Prinzip des *FACE*-Routings basiert, werden bei der planaren Subgraphkonstruktion verschiedene Ansätze verfolgt. Aus diesem Grund liegt das Hauptaugenmerk der im Folgenden betrachteten Arbeiten auf der Konstruktion planarer Subgraphen. Dazu werden im ersten Abschnitt zwei Verfahren für Unit-Disk Graphen beleuchtet, die den Einsatz der Beaconlosen-Strategie verfolgen. Da für Quasi-Unit-Disk Graphen bisher noch keine beaconlosen Verfahren existieren, werden im zweiten Abschnitt bestehende lokale Verfahren für Quasi-Unit-Disk Graphen vorgestellt.

2.1 Unit-Disk Graphen

Im Bereich der Unit-Disk Graphen existieren Verfahren, die zur Konstruktion eines lokalen planaren Subgraphen den Ansatz der Beaconless-Strategie verfolgen. Dabei verwenden viele Verfahren eine *Timer Funktion*, die dafür sorgt, dass aus der Menge von Empfängern einer Nachricht nur gezielte Kandidaten antworten. Empfängt ein Knoten eine Nachricht, wird anhand einer definierten Funktion ein Timer berechnet und gestartet. Beim Ablauf des Timers sendet der Knoten eine Antwort aus. Trifft vor Ablauf des Timers die Antwort eines anderen Knotens ein,

wird der Timer unterbrochen und keine eigene Antwort versendet. Solche Timer Funktionen werden häufig verwendet um aus der lokalen Nachbarschaft lediglich bestimmte Nachbarknoten kennenzulernen und das Nachrichtenaufkommen möglichst klein zu halten.

Rührup, Kalosha, Nayak und Stojmenović entwickeln in [12] das *Beaconless Forwarder Planarization* Verfahren (BFP) zur garantierten Nachrichtenauslieferung im Unit-Disk Graphen. Durch den kombinierten Einsatz einer Timer Funktion und einer *Select-and-Protest-Strategie* wird ein planarer Subgraph bestimmt, ohne auf die vollständige 1-hop Nachbarschaftsinformation zurückgreifen zu müssen. Dazu werden neben den Eigenschaften des *Gabriel Graphs* (GG) und des *Relative Neighbourhood Graphs* (RNG) auch ein eigens entwickelter *Circlunar Neighbourhood Graph* (CNG) betrachtet, der eine Kombination aus GG und RNG darstellt. Unter Verwendung einer dieser Grapheigenschaften werden Kanten des zugrunde liegenden Unit-Disk Graphen ausgewählt und zur Konstruktion eines planaren Subgraphen verwendet. Wird im Laufe des Verfahrens die Planarität durch eine hinzugefügte Kante verletzt, kann diese durch den Protest eines Knotens widerrufen werden. Das anschließende Routen einer Nachricht basiert dabei auf der Idee des *FACE-Routings*.

Um die Routingpfade möglichst kurz zu halten sind *Spanner* bei der Subgraphkonstruktion von besonderem Interesse, denn ein Spanner ist ein Subgraph indem der kürzeste Pfad zwischen zwei Knoten maximal um einen konstanten Faktor länger ist, als der kürzeste Pfad im zugrunde liegenden Graphen. Dabei wird, je nachdem ob der euklidisch oder topologisch kürzeste Pfad betrachtet wird, zwischen *topologischen Spannern* und *euklidischen Spannern* unterschieden.

In [2] zeigen Benter, Frey und Neumann, dass im Unit-Disk Graphen beaconlos ein planarer euklidischer Spanner bestimmt werden kann. Sie greifen die Select-Phase aus [12] auf und konstruieren lokal einen *Partial Delaunay Triangulation* Subgraph (PDT), der sowohl planar, als auch ein euklidischer Spanner ist. Um möglichst wenig Nachrichtenoverhead zu produzieren, wird für die PDT-Konstruktion ein Timer Mechanismus eingesetzt, der gewährleistet, dass im Verlauf des Verfahrens nur jene Knoten eine Nachricht versenden, die keine PDT-Eigenschaft verletzen. Nach Benter et al. eignet sich diese Methode zur lokalen Konstruktion eines PDTs besonders für Greedy-Recovery-Verfahren, um im besten Fall einen Recovery-Pfad zu bestimmen, der lediglich um einen konstanten Faktor länger ist, als der kürzeste Pfad im zugrunde liegenden Unit-Disk Graph.

2.2 Quasi-Unit-Disk Graphen

Bisher sind für Quasi-Unit-Disk Graphen in der Literatur keine beaconlosen Verfahren zur garantierten Nachrichtenauslieferung bekannt. Eine simple Adaptierung der beaconlosen Verfahren für Unit-Disk Graphen auf Quasi-Unit-Disk Graphen ist aufgrund der sich unterscheidenden Eigenschaften nicht ohne weiteres möglich. Während die bekannten Verfahren zur Konstruktionen von planaren Subgraphen mittels GG, PDT oder anderen Subgraphkonstruktionen im Unit-Disk Graphen mit 1-lokaler Nachbarschaftsinformation auskommen, ist für Selbiges im Quasi-Unit-Disk Graphen 2-lokale Nachbarschaftsinformation notwendig. Es existieren allerdings wissenschaftliche Arbeiten die sich mit lokalen Verfahren zur garantierten Nachrichtenauslieferung im Quasi-Unit-Disk Graphen beschäftigen, deren grundsätzlicher Aufbau den beaconlosen Verfahren für Unit-Disk Graphen entspricht. Unter Verwendung der 2-lokalen Nachbarschaftsinformation wird ein planarer Subgraph konstruiert, auf dem anschließend entsprechende Routingschritte ausgeführt werden.

Die Arbeit [1] von Barrière, Fraigniaud, Narayanan und Opatrny ist die erste, die sich mit geografischem Routing im Quasi-Unit-Disk Graph befasst und dazu die lokale Konstruktion eines zusammenhängenden planaren *Overlay-Graphen* untersucht. Das von ihnen entwickelte Verfahren berechnet in den zwei Teilschritten *completion* und *extraction* einen planaren lokalen Graphen und routet in einem dritten Schritt darüber. Im ersten Schritt erzeugt der ausführende Knoten durch das Hinzufügen von *virtuellen Kanten* zu 2-hop Nachbarn einen Supergraphen S. Im zweiten Schritt wird mittels GG-Konstruktion aus dem Supergraph S der planare zusammenhängende Overlay-Graph $GG(S)$ extrahiert. Der dritte Schritt besteht aus dem Routen der Nachricht in $GG(S)$ mittels *perimeter routing* [3]. Unter der Voraussetzung, dass der zugrunde liegende Quasi-Unit-Disk Graph zusammenhängend ist und die Eigenschaft $\frac{r_{\max}}{r_{\min}} \leq \sqrt{2}$ erfüllt, zeigen Berrière et al., dass ihr Verfahren eine garantierte Nachrichtenauslieferung ermöglicht. Für die korrekte Funktion des Verfahrens wird allerdings auch die volle 1-lokale und eine partielle 2-lokale Nachbarschaftsinformation vorausgesetzt.

Neben Barrière et al. beschäftigen sich Chavèz et al. in [4], ebenfalls motiviert durch lokales Routing, mit der lokalen Konstruktion eines zusammenhängenden planaren Subgraphen im Quasi-Unit-Disk Graph. Dazu wird unter Zuhilfenahme 2-lokaler Nachbarschaftsinformation von jedem Knoten ein *Minimal Spanning*

Tree (MST) mit dem Algorithmus von Kruskal berechnet und mittels Broadcast in der 1-hop Nachbarschaft bekannt gegeben. Mit diesen Informationen kann ein Knoten u einen lokalen Subgraphen konstruieren, indem für jede Kante zwischen u und einem Knoten v geprüft wird, ob die Kante uv sowohl im MST von u als auch im MST von v existiert. Ist das der Fall, beinhaltet der planare Subgraph von u ebenfalls die Kante uv. Ferner zeigen Chevèz et al., dass ein Graph der mit ihrem Verfahren konstruiert wurde planar, verbunden und durch einen maximalen Knotengrad beschränkt ist.

Kuhn, Wattenhofer und Zollinger greifen in [10] die Ergebnisse aus [1] auf und nutzen zur lokalen Konstruktion eines Subgraphen neben dem Konzept der virtuellen Kanten eine *Clustering*-Technik und das Konzept der Knotenaggregation, welches unter anderem von Frey und Görgen in [7] beschrieben wird. Dazu wird ein globales *Grid* aus regelmäßigen quadratischen Kacheln über dem Graph ausgebreitet, sodass Knoten aus ein und derselben Kachel ein *Cluster* bilden. Durch die Größe der Kacheln wird garantiert, dass Knoten alle aus ihrem Cluster versendeten Nachrichten mithören. Innerhalb eines Clusters wird ein Knoten als *Clusterhead* definiert. Alle übrigen Knoten eines Clusters sind entweder *Clustermember* oder *Connector Nodes*. Das Verfahren von Kuhn et al. bedient sich der Clustering-Technik und konstruiert einen *Dense Backbone Graph* G_{DBG}, in dem nur Kanten zwischen einem Clusterhead und seinen Clustermembern existieren oder zwischen einem Connector Node und seinem Clusterhead oder einem weiteren Connector Node. Durch das Hinzufügen von virtuellen Kanten zwischen Clusterheads, die über Connector Nodes verbunden sind, entsteht ein Graph $G^{(v)}_{DBG}$. Für $G^{(v)}_{DBG}$ wird mehrmals das Clustering-Grid verschoben und ein Subgraph berechnet, sodass die Vereinigung der Subgraphen den Graph $G^{(v)}_{BG}$ ergibt. Durch eine Substitution der virtuellen Kanten in $G^{(v)}_{BG}$ mit den Pfaden über Connector Nodes und das Hinzufügen aller Quasi-Unit-Disk Kanten wird G'_{BG} konstruiert. Zur Planarisierung diese Graphs wird das von Barrière et al. in [1] beschriebenen Verfahren mit anschließender GG-Konstruktion angewendet. Weiterhin zeigen Kuhn et al., dass bei der Verwendung von *GOAFR+* [9], einer erweiterten Variante von FACE-Routing, die Kosten des Routingpfades in $O(l^2)$ liegen, wenn l den Kosten des kürzesten Pfades entspricht.

Mit der Motivation im Quasi-Unit-Disk Graph lokales und effizientes Routing betreiben zu können, beschäftigen sich Lillis, Pemmaraju und Pirwani in [11] ebenfalls mit lokalen Verfahren zur Bestimmung eines planaren Graphen. Dabei

konzentrieren sie sich zuerst auf die Konstruktion eines topologischen Spanners und anschließend auf die Konstruktion eines euklidischen Spanners. Der topologische Spanner wird unter Verwendung einer Clustering-Technik und des Konzepts der *virtuellen Knoten* aus [10] bestimmt. Das Konzept der virtuellen Knoten beschreibt das Planarisieren eines Graphen indem Kantenschnitte durch virtuelle Knoten ersetzt werden. Lillis et al. stellen fest, dass die Verwendung von virtuellen Knoten zur Einführung beliebig vieler Knoten führen kann, wenn die Anzahl an Kantenschnitten im Quasi-Unit-Disk Graph nicht beschränkt wird. Zur Beschränkung der Anzahl an Kantenschnitten wird für einen Quasi-Unit-Disk Graph G mittels Clustering-Technik ein *Backbone Graph* G_B berechnet. Jedes Cluster C_i in G_B enthält neben einem Clusterhead, für jedes benachbarte Cluster C_j genau einen Connector Node, der eine Kante zu einem Knoten aus C_j besitzt und einen Pfad vom Clusterhead zum benachbarten Cluster als *bridge edge* repräsentiert. So wird die maximale Anzahl möglicher Schnitte einer Kante in G_B beschränkt. An Stelle der Kantenschnitte in G_B werden virtuelle Knoten eingefügt und es resultiert der planarisierte Backbone Graph $Virt(G_B)$. Für $Virt(G_B)$ wird ein Routing-Algorithmus vorgestellt, der Nachrichtenauslieferung garantiert und einen topologischen Spanner-Pfad bestimmt, dessen topologische Länge nur um einen konstanten Faktor länger ist als der kürzeste Pfad im zugrunde liegenden Quasi-Unit-Disk Graph. Basierend auf diesen Ergebnissen entwickeln Lillis et al. ein weiteres Verfahren zur Bestimmung von euklidischen Spanner-Pfaden. Dazu werden für den Eingangsgraph G mehrere planare Subgraphen mit verschobenem Clustering-Grid berechnet. Nachdem für ein Cluster mit dem Algorithmus von Wang und Li [13] ein planarer Subgraph berechnet wurde, wird das Grid verschoben und ein weiterer Subgraph berechnet. So entstehen pro Cluster drei planare Subgraphen, deren Vereinigung mit $Virt(G_B)$ den planaren Routing-Graph G_R ergeben. Auf diesen Graphen G_R wird ein erweiterter Routing-Algorithmus ausgeführt, der sich in zwei Fälle unterteilen lässt. Liegen Sender und Empfänger in einer der drei berechneten planaren Subgraphen in einem gemeinsamen Cluster, wird die Nachricht mit GOAFR+ in diesem Subgraphen geroutet. Liegen Sender und Empfänger in keinem der Subgraphen innerhalb eines Clusters, leitet der Sender die Nachricht an seinen Clusterhead weiter, von wo die Nachricht mittels GOAFR+ in G_R bis zu einem Knoten geroutet wird, der mit dem Empfängerknoten in mindestens einem der drei Subgraphen in einem gemeinsamen Cluster liegt. Von diesem Knoten aus wird die Nachricht über den entsprechen-

den Subgraphen zum Empfänger geroutet. Für diesen erweiterten Algorithmus zeigen Lillis et al., dass die Länge der resultierenden Routing-Pfade im Vergleich zum euklidisch beziehungsweise topologisch kürzesten Pfad im Quasi-Unit-Disk Graph nur um einen konstanten Faktor länger sind. Da diese lokale Konstruktion von G_B die Kenntnis der 2-lokalen Nachbarschaft benötigt, werden pro Knoten mindestens zwei Broadcasts versendet.

Guan nutzt in [8] die Idee der virtuellen Knoten, um ohne aufwändige Extraktion eines Subgraphen im Quasi-Unit-Disk lokal eine garantierte Nachrichtenauslieferung zu ermöglichen. Knoten in einem Graphen G beziehen ihre 2-lokale Nachbarschaftsinformation um Kantenschnitte in G zu detektieren und fügen für diese virtuelle Knoten ein. Aus G entsteht so ein *virtueller* planarer Graph auf dem der in [8] vorgestellte Algorithmus *Virtual Face Routing* angewendet wird. Dabei handelt es sich um eine für virtuelle planare Graphen erweiterte Version des bekannten FACE-Routings.

Die vorgestellten Verfahren für Quasi-Unit-Disk Graphen basieren bis auf das von Barrière et al., alle auf einer Einholung der vollständigen 2-hop Nachbarschaftsinformation. Das hat zur Folge, dass ein Knoten der nie an einer Nachrichtenauslieferung benötigt wird, trotzdem aktiv beteiligt wird. Wie sich der dadurch entstehende Nachrichtenoverhead im Unit-Disk Graph reduzieren lässt, haben [12] und [2] in Form der *Beaconless-Strategie* gezeigt. Möglicherweise lässt sich auch eines der lokalen Verfahren für Quasi-Unit-Disk Graphen hinsichtlich der Nachrichteneffizienz durch die Erweiterung um den beaconlosen Ansatz verbessern. Eine minimale Anzahl an Nachrichten im Verfahren von Lillis et al. aus [11] zu erzielen, ist aufgrund der euklidischen, beziehungsweise topologischen Eigenschaft des Routingpfads im Quasi-Unit-Disk Graph besonders erstrebenswert. Die Simulation in Kapitel 6 zeigt, dass der im Rahmen dieser Arbeit entwickelte und im Folgenden beschriebene Beaconlose Clustering Algorithmus für die Verbesserung des Verfahrens von Lillis et al. verwendet werden kann.

Kapitel 3

Verfahrensentwicklung

In diesem Kapitel wird der im Rahmen dieser Arbeit entwickelte Beaconlose Clustering Algorithmus vorgestellt. Nach einer detaillierten Betrachtung der Funktion des Verfahrens im Unit-Disk Graphen wird eine Erweiterung für Quasi-Unit-Disk Graphen vorgestellt. Im Anschluss werden drei Varianten zur Optimierung des BCA gezeigt.

3.1 Beaconloser Clustering Algorithmus

Gegeben sei ein Unit-Disk Graph G mit der Knotenmenge V, die in einer zweidimensionalen Ebene liegt. Diese zweidimensionale Ebene wird mit der Idee des geografischen Clusterings von Frey und Görgen aus [6] durch ein global bekanntes *Grid* in einzelne quadratische Kacheln unterteilt, sodass jeder Knoten aus V einer Kachel zugeordnet werden kann. Die Größe einer jeden Kachel wird so gewählt, dass die Diagonale einer Kachel höchstens dem Unit-Disk Radius entspricht. So wird gewährleistet, dass die Nachricht eines Knotens von allen übrigen Knoten der gleichen Kachel mitgehört werden kann. Je nach Position eines Knotens schneidet dessen Unit-Disk Radius unterschiedliche umliegende Kacheln. Abbildung 3.1 zeigt zwei Knoten u und v in einer gemeinsamen Kachel. Für u und v sind deren Unit-Disk Radien eingezeichnet, die genau der Länge der Diagonalen einer Kachel entsprechen. Es ist zu sehen, dass der Radius um u lediglich in die acht angrenzenden Kacheln reicht, wohingegen v insgesamt dreizehn umliegende Kacheln erreicht, die zum Teil in zweiter Reihe hinter den angrenzenden Kacheln liegen. Angenommen die Position von Knoten u innerhalb seiner Kachel

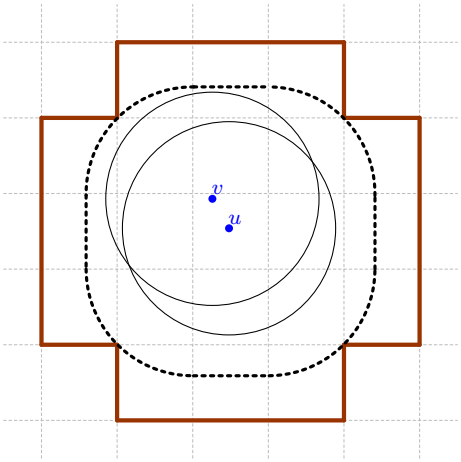


Abbildung 3.1: Erreichbare Kacheln der Unit-Disk Graph Knoten in einer gemeinsamen Kachel. Die rot umrandete Fläche zeigt die maximal erreichbaren Kacheln. Alle aus der Kachel von u und v erreichbaren Knoten liegen innerhalb des gepunkteten Bereichs.

sei unbekannt, so kann die Anzahl der erreichbaren Kacheln auf die 20 in Abbildung 3.1 markierten Kacheln beschränkt werden. Genauer betrachtet können Knoten, die eine gemeinsame Kante mit einem Knoten der Kachel von u und v haben, lediglich innerhalb des gepunkteten Bereichs in Abbildung 3.1 liegen.

Das Ziel des Beaconlosen Clustering Algorithmus ist es, ausgehend von einem beliebigen Knoten u, aus einer Kachel im Unit-Disk Graph, reaktiv und beaconlos genau einen Knoten für jede der zwanzig umliegenden Kachel kennen zu lernen, die entweder über u selbst oder einen Nachbarknoten v aus der Kachel von u erreichbar ist. Dabei können für die umliegenden Kacheln vier Fälle eintreten.

1. In der Kachel existiert mindestens ein Knoten w, ...
   (a) ... der eine gemeinsame Kante mit Knoten u hat.
   (b) ... der keine gemeinsame Kante mit u hat, jedoch mit einem Knoten v aus der Kachel von u.
   (c) ... der weder mit u noch mit einem Knoten v aus der Kachel von u eine gemeinsame Kante hat.
2. Es existiert kein Knoten w in der Kachel.

Für den Fall 1a soll der Knoten u nach der Ausführung des Verfahrens genau einen Knoten w aus der jeweiligen Kachel kennen lernen. Tritt Fall 1b ein,

soll u neben w auch den zugehörigen Knoten v kennenlernen, der eine gemeinsame Kante mit w besitzt. Wenn Knoten in einer Kachel existieren, für die Fall 1c oder 2 eintritt, bleiben diese für den ausführenden Knoten unbekannt. Das Verfahren unterteilt sich in zwei Phasen. In einer ersten Phase werden alle Kacheln gefunden, für die Fall 1a gilt und in einer zweiten Phase alle Kacheln für die Fall 1b vorliegt. Im folgenden werden Kacheln, für die bereits ein Knoten w kennengelernt wurde als *abgeschlossene Kacheln* bezeichnet und alle übrigen als *offene Kacheln*. Für die Kommunikation der Knoten untereinander werden vier Nachrichtentypen benötigt:

FirstRequest Enthält die Position des Senders und eine Liste von offenen Kacheln. Ist die Kachel des Empfängers in der Liste enthalten, ist die Nachricht von Relevanz.

SecondRequest Enthält die gleichen Informationen wie der FirstRequest, richtet sich allerdings nur an Knoten in der Kachel des Senders.

FirstResponse Die Antwort eines Knotens auf einen FirstRequest. Ein FirstResponse enthält lediglich die Positionsdaten des Senders, über die der Empfänger berechnet aus welcher Kachel der Sender stammt.

SecondResponse Die Antwort eines Knotens auf einen SecondRequest. Ein SecondResponse wird verwendet um dem Sender des SecondRequests mitzuteilen, dass in einer offenen Kachel ein Knoten gefunden wurde. Dazu enthält die Nachricht die Position des Senders und des ermittelten Knotens.

Knoten u sei ein beliebiger Knoten im Unit-Disk Graph, auf dem das hier beschriebene Verfahren ausgeführt wird. Mit Beginn der ersten Phase sendet u einen FirstRequest mit seiner Position und einer Liste aus, die alle erreichbaren Kacheln beinhaltet, ausgenommen der eigenen. Jeder Empfänger prüft daraufhin, ob seine Kachel in der Nachricht enthalten ist. Ist dies der Fall, ist er ein potentieller Knoten den u kennenlernen will. Damit u allerdings nur einen Knoten pro Kachel kennenlernt und auch nur eine Nachricht pro Kachel gesendet wird, treten die Knoten innerhalb einer Kachel in einen Wettstreit. Dazu berechnet jeder Knoten die euklidische Distanz d zum Mittelpunkt seiner Kachel und erstellt einen Timer t in Abhängigkeit von d, sodass der Knoten, dessen Distanz d am kleinsten ist, den kleinsten Timer t vorweist. Die Zeit von t ist dabei durch einen Maximalwert $t_{\max}$ nach oben beschränkt. Gleichzeitig starten die Knoten einer Kachel ihre Timer t. Ein Knoten v, dessen Timer t als erstes abläuft, gewinnt den Wettstreit in

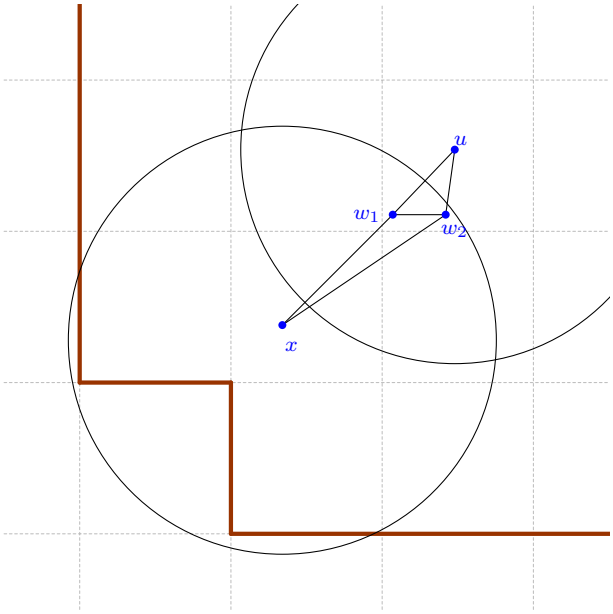


Abbildung 3.2: In Phase 2 lernt Knoten u mithilfe von w_1 Knoten x kennen. Knoten w_1 berechnet anhand einer Timerfunktion einen kürzeren Timer als w_2, sendet einen Broadcast aus auf den Knoten x antwortet und leitet dessen Existenz an u weiter. Knoten w_2 versendet keine Nachricht.

seiner Kachel und sendet eine Antwortnachricht mit seiner Positionsinformation aus. Diese Nachricht erfüllt zwei Funktionen. Zum einen erfährt Knoten u von der Existenz des Knotens v und berechnet anhand dessen Positionsinformation aus welcher Kachel die Antwort stammt. Zum anderen hören alle übrigen Knoten die am Wettstreit der Kachel von v beteiligt waren dessen Nachricht. Mit der darin enthaltenen Positionsinformation schlussfolgern die Empfänger, dass v den Wettstreit für sich entscheiden konnte und unterbrechen ihre Timer t. So bleibt v der einzige Knoten aus dessen Kachel, der eine Nachricht versendet und letztendlich für Knoten u bekannt ist. In allen Kacheln, die mindestens einen Knoten enthalten, der die anfängliche Nachricht von u empfangen hat, findet zeitlich parallel ein solcher Wettstreit statt, sodass u nach dem maximalen Zeitfenster $t_{\max}$ für jede dieser Kacheln genau einen Knoten kennengelernt hat.

Abbildung 3.2 zeigt ein Beispiel für die zweite Phase des Verfahrens in der u für alle offenen Kacheln genau einen Knoten x und einen seiner Nachbarn w aus der Kachel von u kennenlernt, wenn eine Kante wx im Unit-Disk Graph existiert. Durch die bekannten maximal erreichbaren Kacheln und die bereits abgeschlos-

senen Kacheln erstellt der ausführende Knoten u eine Liste, die alle noch offenen Kacheln beinhaltet und sendet eine SecondRequest-Nachricht mit der Position von u und der ersten Kachel der Liste aus. Die Empfänger der Nachricht wissen aufgrund des Nachrichtentyps SecondRequest und der darin enthaltenen Position von u, dass sich diese Nachricht nur an Knoten richtet, die in der Kachel von Knoten u liegen. Die in der Nachricht enthaltene Kachel stellt die temporäre *aktive Kachel* dar, für die es gilt einen Knoten x zu finden. Die Empfänger dieser Nachricht berechnen nun anhand ihrer eigenen Position, ob ihr Unit-Disk Radius bis in die aktive Kachel reicht. Alle Knoten die diese Bedingung erfüllen sind potentielle Kandidaten w, die eine gemeinsame Kante mit einem Knoten x aus der aktiven Kachel aufweisen könnten. Aus der Menge der Knoten muss nun ein geeigneter Kandidat hervorgehen, der zuerst prüft, ob sich ein Knoten x in seiner Reichweite befindet. Dazu berechnen die potentiellen Knoten je einen Timer t in Abhängigkeit ihrer Entfernung zu dem nächstgelegenen Punkt der aktiven Kachel und treten in einen Wettstreit indem sie ihre Timer gleichzeitig starten. Ein Knoten w dessen Timer t als erstes abläuft, besitzt den kleinsten euklidischen Abstand zur aktiven Kachel und darf als erstes prüfen, ob ein Knoten x erreichbar ist. Dazu versendet er einen FirstRequest mit der aktiven Kachel und seiner Position. Daraufhin erhöhen die übrigen Knoten des Wettstreits ihre Timer um den Zeitraum $t_{\max}$, damit w temporär der einzige Knoten bleibt, der die aktive Kachel untersucht. Die Empfänger-Knoten in der aktiven Kachel werden durch diese Nachricht, wie in der ersten Phase des Verfahrens beschrieben, ebenfalls zu einem Wettstreit aufgefordert, der eine FirstResponse-Nachricht eines antwortenden Knoten x hervorbringt. Um sicherzustellen, dass alle beteiligten Knoten der Kachel von u mithören, dass ein Knoten x gefunden wurde, versendet w einen SecondResponse, bestehend aus den Positionen von w und x. Dadurch erfährt Knoten u von der Existenz und der Position von Knoten w und x mit deren Kante wx und vermerkt die Kachel von x als abgeschlossene Kachel. Alle Knoten deren Timer t um $t_{\max}$ erweitert wurden beenden diesen beim Empfang des SecondResponse. Wenn Knoten w keinen Knoten der aktiven Kachel erreicht und somit keinen FirstResponse empfängt, versendet er auch keinen SecondResponse. Folglich laufen die Timer der übrigen Knoten unangetastet weiter, bis der Timer des nächsten Knoten abläuft und einen FirstRequest für die aktive Kachel aussendet. So werden im Falle keiner erreichbaren Knoten x der aktiven Kachel alle potentiellen Knoten w sukzessive abgearbeitet, bis ein Knoten gefunden wurde,

oder kein Knoten w mehr übrig ist. Der das Verfahren ausführende Knoten u vermerkt währenddessen alle versendeten FirstRequest-Nachrichten. Solange noch kein Knoten in der aktiven Kachel gefunden wurde hört Knoten u in zeitlichen Abständen die kleiner als zwei mal $t_{\max}$ sind jeweils einen FirstRequest seiner Nachbarknoten und schließt daraus, dass die aktive Kachel noch nicht von allen möglichen Knoten geprüft wurde. Wenn die Nachrichten ausbleiben, vermerkt Knoten u, dass der erreichbare Bereich der aktiven Kachel leer sein muss und schließt die Kachel ab. Nachdem eine Kachel abgeschlossen ist, fährt u mit der nächsten offenen Kachel fort und sendet einen SecondRequest mit deren Position aus. So werden alle offenen Kacheln sequentiell auf einen erreichbaren Knoten untersucht, bis die letzte Kachel abgeschlossen wurde und das Verfahren somit endet.

3.2 Adaption für Quasi-Unit-Disk Graphen

Dieser Abschnitt erläutert die Adaption des BCA von Unit-Disk zu Quasi-Unit-Disk Graphen. Gegeben sei ein Quasi-Unit-Disk Graph G mit der Knotenmenge V der zweidimensionalen Ebene und den Radien $r_{\min}$ und $r_{\max}$, sodass $\frac{r_{\min}}{r_{\max}} \geq \frac{1}{\sqrt{2}}$ gilt. Diese zweidimensionale Ebene wird ähnlich zu Abschnitt 3.1 in Kacheln unterteilt. Um allerdings weiterhin zu gewährleisten, dass ein Knoten jede Nachricht aus seiner Kachel empfängt, darf die Diagonale einer Kachel höchstens dem Radius $r_{\min}$ entsprechen. Das hat zur Folge, dass die maximale Reichweite $r_{\max}$ eines Knotens mindestens 15 und höchstens 22 seiner umliegenden Kacheln umfasst oder schneidet. Abbildung 3.3 verdeutlicht welche Änderungen durch den größeren Radius $r_{\max}$ entstehen. Der grün markierte Bereich in dem erreichbare Knoten ausgehend von einer Kachel liegen können umfasst nun nahezu die gesamte Fläche der erreichbaren 24 Kacheln. Unter Berücksichtigung der genannten Bedingungen hinsichtlich der Größe der Kacheln und der Reichweite eines Knotens, lässt sich das in Kapitel 3.1 beschrieben Verfahren ohne weitere Änderungen auf Quasi-Unit-Disk Graphen ausführen.

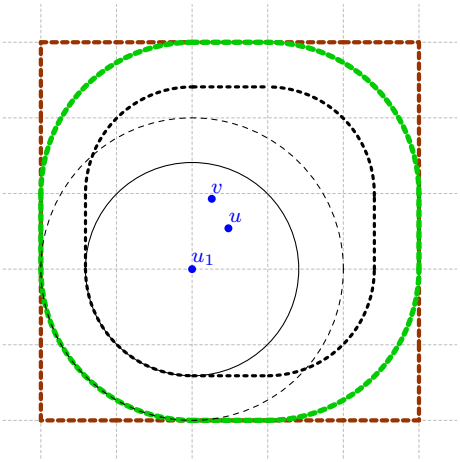


Abbildung 3.3: Maximale Reichweite der Knoten einer Kachel im Quasi-Unit-Disk Graph. Im Gegensatz zum Unit-Disk Graph hat sich die Anzahl der erreichbaren Kacheln um vier erhöht. Der Bereich in dem alle erreichbaren Knoten liegen hat sich auf den äußeren gepunkteten Bereich vergrößert.

3.3 Erweiterungen des Verfahrens

Im folgenden werden drei Ansätze vorgestellt, die den beschriebenen Beaconlosen Clustering Algorithmus hinsichtlich der benötigten Laufzeit, der Anzahl beteiligter Knoten und des Nachrichtenaufkommens effizienter gestalten.

3.3.1 Verkürzung der Laufzeit

Im Laufe der zweiten Phase des BCA für Unit-Disk Graphen aus Abschnitt 3.1 versendet ein ausführender Knoten u für jede offene Kachel einen SecondRequest. Je nach abgeschlossenen Kacheln der ersten Phase variiert die Anzahl der noch offenen Kacheln. Wurde in Phase 1 keine einzige Kachel abgeschlossen, bleiben 20 offene Kacheln übrig. Wurde hingegen für jede der 13 erreichbaren Kacheln ein Knoten gefunden, bleiben 7 offene Kacheln übrig. Aus dieser minimalen und maximalen Anzahl offener Kacheln zu Beginn von Phase 2 folgt, dass die Summe der versendeten SecondRequest-Nachrichten von u zwischen 7 und 20 liegen muss. Um diese Nachrichten zu reduzieren kann das Verfahren auch mit einem einzigen SecondRequest umgesetzt werden. Anstatt nach Abschluss einer offenen Kachel den SeconRequest für die nächste zu versenden, wird bereits im

ersten SecondRequest eine Liste aller offenen Kacheln mitgeschickt, sodass die übrigen Nachrichten des ausführenden Knotens eingespart werden können. Die angesprochenen Knoten beginnen daraufhin mit der Berechnung ihrer Timer für die erste Kachel der Liste und handeln diese gemäß des Verfahrens ab. Sobald ein SeconResponse empfangen wird oder dieser über den Zeitraum $t_{\max}$ ausbleibt, ist allen Knoten der ausführenden Kachel bekannt, dass die aktive Kachel abgeschlossen ist und fahren mit nächsten Kachel fort. Des Weiteren lassen sich Nachrichten einsparen, wenn ein Knoten der in Phase 2 einen FirstRequest aussendet, gleichzeitig alle erreichbaren offene Kacheln adressiert. Angenommen w sei ein Knoten dessen Timer t für die erste Kachel der empfangenen Liste offener Kacheln abläuft, sendet er gemäß des bisherigen Verfahrens einen FirstRequest für die aktuelle aktive Kachel aus. Anstatt nur eine Kachel auf einen erreichbaren Knoten x zu prüfen, werden alle Kacheln adressiert, die von w erreichbar und in der Liste enthalten sind. Dadurch werden keine zusätzlichen Nachrichten versandt, sondern im besten Fall mehrere Nachrichten eingespart, wenn w erreichbare Knoten in mehreren Kacheln hat. Wenn die übrigen Knoten, deren Timer weiterlaufen, eine oder mehrere SecondResponse-Nachrichten von Knoten w empfangen, brechen diese ihre Timer lediglich ab, wenn alle von ihnen erreichbaren Kacheln abgeschlossen sind.

Diese Erweiterung von Phase 2 lässt sich ohne Weiteres auf das Verfahren für Quasi-Unit-Disk Graphen übertragen.

3.3.2 Ausschluss von Knoten

Bei genauerer Untersuchung der zweiten Phase des Verfahrens ist festzustellen, dass ein und dieselbe Kachel für mehrere Knoten erreichbar ist. Durch die Timerberechnung in Abhängigkeit der Entfernung zur aktiven Kachel wird eine feste Rangfolge festgelegt, in der die Knoten die Erreichbarkeit eines unbekannten Knotens in der aktiven Kachel prüfen. Dabei kann der Fall eintreten, dass die Region der aktiven Kachel, die durch den Unit-Disk Radius eines Knotens erreicht wird bereits im Vorfeld von einem anderen Knoten geprüft wurde. Um diese mehrfachen Prüfungen einer Region zu unterbinden und den dadurch entstehenden Nachrichtenoverhead zu reduzieren, sollen nur die Knoten eine Kachel prüfen, deren Radius eine noch nicht geprüfte Region der Kachel beinhaltet. Alle übrigen Knoten werden vom weiteren Verfahren ausgeschlossen. Dazu vermerkt

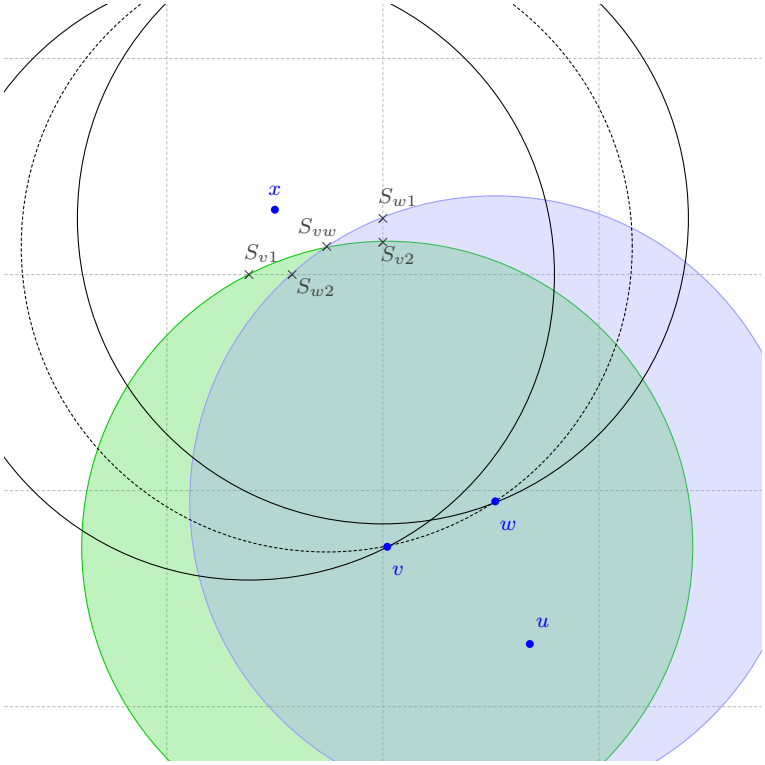


Abbildung 3.4: Einschränkung der Knoten aus der Kachel von u für die aktive Kachel von x in Phase 2 des Verfahrens. Die Region, die zwischen v und w liegt und durch die Radien um S_{v1}, S_{w1} und S_{vw} begrenzt wird entspricht dem Bereich, in dem noch relevante Knoten liegen können.

jeder Knoten die Position seiner Vorgänger und berechnet mit dieser Information welche Regionen einer Kachel bereits geprüft wurden und schließt daraus, ob seine Teilnahme am weiteren Verfahren für die aktive Kachel notwendig ist. Je nach relativer Position der aktiven Kachel zu der Kachel, aus der das Verfahren gestartet wird, ändert sich die im folgenden beschriebene Erweiterung an manchen Stellen. Die Grundidee lässt sich allerdings auf die übrigen Kacheln übertragen. Der Übersicht halber wird nur eine Kachel in Detail veranschaulicht. Die zweite Phase des Verfahrens wird in drei Teilschritte unterteilt und die Berechnung der Timer für eine aktive Kachel wie folgt abgeändert.

1. Alle Knoten deren Unit-Disk Radien die aktive Kachel schneiden berechnen ihren Timer t nicht mehr anhand ihrer euklidischen Distanz zur Kachel, sondern in Abhängigkeit zu den Schnittpunkten ihres Unit-Disk Radius mit den Seiten der aktiven Kachel. Die geometrische Betrachtung der Schnittpunk-

te einer Kachel und dem Unit-Disk Radius eines Knotens v zeigt, dass je nach Position von v bis zu vier Schnittpunkte entstehen können. Im ersten Fall tangiert der Radius die Kachel in einem Schnittpunkt und deckt keine Region der Kachel ab, weshalb ein Knoten für den dieser Fall eintritt kein Interesse für das Verfahren besitzt. Wenn zwei, drei oder vier Schnittpunkte entstehen, bestimmt der Knoten v den Schnittpunkt S_1, der den Abstand zum euklidisch nächsten Randpunkt seiner eigenen Kachel maximiert. Dieser Abstandswert wird zur Berechnung des Timers verwendet, sodass der Knoten mit dem größten Abstandswert den kürzesten Timer besitzt. Abbildung 3.4 zeigt hierzu ein Beispiel, in dem Knoten u das Verfahren startet und Knoten v und w in Phase 2 einen erreichbaren Knoten in der Kachel von Knoten x suchen. Der Unit-Disk Radius von v schneidet die Kachel von x in den zwei Punkten S_{v1} und S_{v2}, sodass S_{v1} der Schnittpunkt mit dem größeren euklidischen Abstand zur Kachel von u ist. Der entsprechende Schnittpunkt für Knoten w ist S_{w1}. Angenommen die Distanz der Kachel von u zu S_{v1} sei größer als die zu S_{w1}, so läuft der Timer von v als erstes ab und sendet einen FirstRequest aus. Die übrigen Knoten erhalten dadurch die Positionsinformation von v und können dessen Schnittpunkte berechnen. Hat einer der übrigen Knoten einen Schnittpunkt mit der gleichen Kachelseite wie S_{v2} und einen größeren Abstand zur Kachel von u als S_{v2}, ist eine noch nicht geprüfte Region der aktiven Kachel für diesen Knoten erreichbar. Alle Knoten, die diese Bedingung nicht erfüllen beenden die Teilnahme am weiteren Verfahren für die aktive Kachel.

2. Wenn der Knoten v nach Prüfung der aktiven Kachel keinen erreichbaren Knoten x feststellen konnte und folglich innerhalb von $t_{\max}$ keinen Second-Response versendet, treten alle übrig gebliebenen Knoten in einen neuen Wettstreit, den der Knoten gewinnt, dessen Schnittpunkt einen größeren Abstand zur Kachel von u aufweist, als S_{v2}. Das Beispiel in Abbildung 3.4 zeigt Knoten w der als erstes aus dem Wettstreit hervortritt, weil sein Schnittpunkt S_{w2} den größten Abstand zur Kachel von u hat. Knoten w sendet als nächstes einen FirstRequest an die aktive Kachel aus und wartet auf einen FirstResponse. Alle übrigen Knoten aus der Kachel von u kennen nun die Position von v und w.
3. Wenn Knoten w keine Antwort aus der aktiven Kachel erhält, kann die Menge der übrigen Knoten weiter eingeschränkt werden. Unter der Bedingung,

dass die Schnittpunkte S_{v1} und S_{w1} zweier Knoten v und w den größten Abstand zur Kachel von v und w haben, kann geometrisch gezeigt werden, dass Knoten, die noch eine ungeprüfte Region der aktiven Kachel erreichen lediglich in einem Bereich zwischen v und w liegen können. Dieser Bereich ergibt sich durch eine Kreiskonstruktion um die Schnittpunkte S_{v1}, S_{w1} und S_{vw} mit dem Unit-Disk Radius. Abbildung 3.4 zeigt den Bereich der sich von Knoten v bis zu Knoten w erstreckt, innerhalb des Kreises um S_{vw} liegt und außerhalb der Kreise um S_{v1} und S_{w1}. Knoten die in diesem Bereich liegen treten nun in den nächsten Wettstreit zur Prüfung eines erreichbaren Knotens. Wird der nächste Knoten wiederum in der aktiven Kachel nicht fündig, kann anhand seiner Position und der Schnittpunkte seines Radius mit den Unit-Disk Radien von v und w der Bereich ähnlich des hier beschriebenen Vorgehens weiter eingeschränkt werden.

Mit der Anwendung dieser Erweiterung für Unit-Disk Graphen lassen sich in der Regel überflüssige Nachrichten einsparen und die Laufzeit mit dem Wegfall der abzuwartenden Zeiträume $t_{\max}$ verkürzen. An dieser Stelle sei allerdings erwähnt, dass ein pathologischer Fall existiert, für den diese Erweiterung keine Verbesserung ermöglicht. Für den in Abbildung 3.4 dargestellten Fall lässt sich in der Region zwischen v und w eine Anordnung von beliebig vielen Knoten konstruieren, sodass jeder dieser Knoten eine Nachricht zur Prüfung der Erreichbarkeit eines Knotens aus der Kachel von x versenden muss, um den Zusammenhang des angestrebten ausgedünnten Graphen nicht zu gefährden.

In Quasi-Unit-Disk Graphen funktioniert die beschriebene Erweiterung leider nicht ohne Weiteres. Grund dafür ist die unsichere Erreichbarkeit eines Knotens innerhalb des äußeren Radius $r_{\max}$. So lässt sich keine präzise Aussage darüber machen, wann ein Knoten eine Region der aktiven Kachel erreicht, die im Vorfeld noch nicht geprüft wurde.

3.3.3 Hexagonales Grid

Das vorgestellte Verfahren basiert auf der Idee des geografischen Clusterings, die unter anderem von Frey und Görgen in [6] aufgegriffen wird. Anstatt eine Partitionierung der Ebene durch ein gleichmäßiges Netz aus Hexagonen zur Partitionierung zu nutzen, worauf sich Frey und Görgen in ihrer Arbeit konzentrieren, wurde hinsichtlich der Implementierung und der simpleren geometrischen Be-

rechnungen ein Grid aus quadratischen Kacheln verwendet. Im Nachfolgenden wird gezeigt, welche Änderungen die Nutzung eines Hexagonalen Grids hervorruft.

Abbildung 3.5 zeigt die Partitionierung der Nachbarschaft eines Knotens u mittels nummerierten quadratischen Kacheln, deren Diagonale d dem eingezeichneten Unit-Disk Radius r, beziehungsweise dem Quasi-Unit-Disk Radius $r_{\min}$ entspricht. Aufgrund der Beziehung zwischen der Kacheldiagonalen d und dem Unit-Disk Radius r können im Unit-Disk Graph lediglich Kanten zwischen einem Knoten aus Kachel 1 und Knoten der Kacheln 1 bis 21 existieren. Bei der Betrachtung der Quasi-Unit-Disk Graphen mit $r_{\min} = d$ und $r_{\max} \leq \sqrt{(2)} * r_{\min}$ können aufgrund des Radius $r_{\max}$ zusätzlich noch Kanten mit Knoten aus den vier weiteren Kacheln 22 bis 25 existieren. Wird die Partitionierung wie in Abbildung 3.6 mittels Hexagonen vorgenommen, deren Diagonale dem Unit-Disk Radius r, beziehungsweise Quasi-Unit-Disk Radius $r_{\min}$, entspricht, können Knoten die eine gemeinsame Kante mit Knoten u aus dem Hexagon 1 besitzen lediglich aus den Kacheln 1 bis 19 stammen. Im Quasi-Unit-Disk Graph ermöglicht der Radius $r_{\max} \leq \sqrt{(2)} * r_{\min}$ zusätzlich Kanten zu Knoten die in den Hexagonen 20 bis 31 liegen.

Gegenüber der hexagonalen Partitionierung im Unit-Disk Graph erfordert die Variante mit quadratischen Kacheln die Berücksichtigung zweier zusätzlicher Kacheln. Im Quasi-Unit-Disk Modell hingegen generiert die hexagonale Partitionierung 6 Kacheln mehr als die Kachelvariante. Um im Verfahren für Quasi-Unit-Disk Graphen möglichst wenige Nachrichten zu generieren scheint die Verwendung der Partitionierung durch quadratische Kacheln aufgrund der Anzahl zu betrachtender Kacheln intuitiv effizienter zu sein. Aus diesem Grund und zur Vereinfachung der Implementierung wird dem Verfahren die Partitionierung durch quadratische Kacheln zugrunde gelegt und bewusst auf die hexagonale Variante verzichtet.

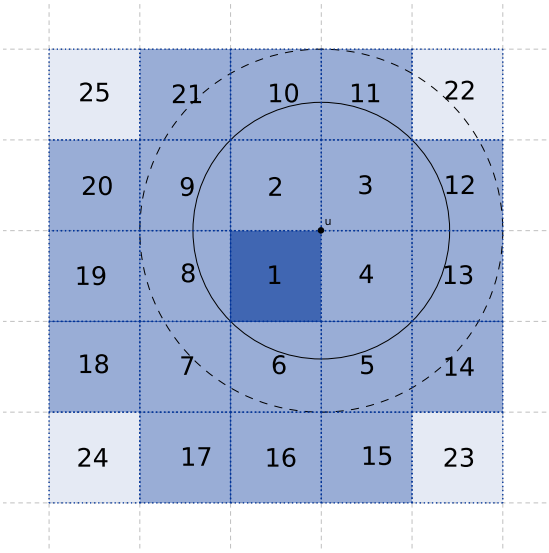


Abbildung 3.5: Partitionierung der zweidimensionalen Ebene durch Quadrate, deren Diagonale dem Unit-Disk Radius, beziehungsweise dem Quasi-Unit-Disk Radius $r_{\min}$ entspricht. Die Erweiterung des entwickelten Verfahrens für Quasi-Unit-Disk Graphen macht die Betrachtung der vier hinzukommenden Kacheln 22 bis 25 notwendig.

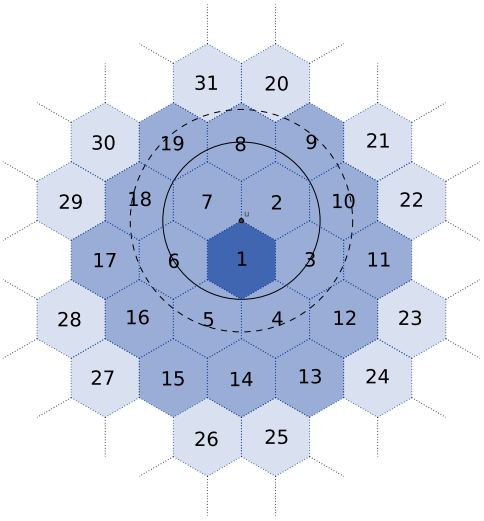


Abbildung 3.6: Bei der Verwendung eines hexagonalen Grids zur Partitionierung müssen im Verfahren für Unit-Disk Graphen nur 19 Hexagone betrachtet werden. Die Erweiterung für Quasi-Unit-Disk Graphen bringt allerdings 12 zusätzliche Kacheln mit sich.

Kapitel 4

Analyse des Algorithmus

In diesem Kapitel wird die Korrektheit des in Kapitel 3 beschriebenen Beaconlosen Clustering Algorithmus gezeigt. Dazu untersucht der folgende Abschnitt 4.1 zunächst das Verfahren für Unit-Disk Graphen. Im Anschluss wird in Abschnitt 4.2 gezeigt, welche Änderungen sich bei der Erweiterung des Verfahrens für Quasi-Unit-Disk Graphen ergeben.

Zur vereinfachenden Beweisführung wird die Annahme vorausgesetzt, dass keine zwei Knoten den gleichen euklidischen Abstand zum Mittelpunkt ihrer Kachel besitzen. Auf eine detaillierte Begründung der Annahme wird im Rahmen dieser Arbeit bewusst verzichtet. Die folgenden Definitionen des Korrektheitsbeweis in Abschnitt 4.1 basieren zum Teil auf [5] und der Arbeit [6] von Frey und Görgen.

4.1 Korrektheit im Unit-Disk Graph

Gegeben sei ein Unit-Disk Graph $G = (V, E)$ mit der Knotenmenge $V \in \mathbb{R}^2$, der Kantenmenge E und dem Unit-Disk Radius $r = 1$. Die zweidimensionale Ebene in der sich G befindet wird mit einem Grid aus quadratischen Kacheln mit der Diagonalen $d = r$ partitioniert, sodass die Ebene flächendeckend von einer fixierten Anordnung an Kacheln abgedeckt wird. Aus $d = r = 1$ folgt die Kachelbreite b mit $b = \left(\frac{1}{\sqrt{2}}\right)$. Jede Kachel wird durch deren oberen linken Eckpunkt p mit $C(p)$ eindeutig identifiziert. Zur Ausrichtung des Grids in der zweidimensionalen Ebene wird eine initiale Kachel $C(p_0)$ so positioniert, dass die linke obere Ecke dem Punkt $p_0 = (0, 0)$ entspricht und die rechte obere Ecke im Punkt $p_1 = (b, 0)$.

Definition 1 (Kachel)**.** *Eine Kachel $C(p)$ sei ein Quadrat der zweidimensionalen Ebene, dessen linke obere Ecke auf dem Punkt p liegt. Die Diagonale d einer Kachel sei $d = r$ mit $r = Unit - DiskRadius$.*

Zwei aneinander grenzende Kacheln $C(p)$ und $C(q)$ berühren sich mit einer Seitenkante $E(p, q)$. Der Punkt in dem vier Kacheln $C(p)$, $C(q)$, $C(r)$ und $C(s)$ aufeinander treffen wird mit $F(p, q, r, s)$ benannt. Um jeden Knoten der Menge V anhand seiner geografischen Position eindeutig einer Kachel zuordnen zu können, wird die folgende Ordnung definiert:

Definition 2 (Punkteordnung)**.** *Für zwei Punkte (x_1, y_1) und $(x_2, y_2) \in \mathbb{R}^2$ gilt folgende Ordnung:*

$$\begin{aligned} (x_1, y_1) < (x_2, y_2) \quad & \textit{if} \quad x_1 < x_2 \\ (x_1, y_1) < (x_2, y_2) \quad & \textit{if} \quad x_1 = x_2 \quad \wedge \quad y_1 < y_2 \end{aligned}$$

Mit Definition 2 kann ein Knoten v durch die Kachelfunktion $H(v)$ eindeutig einer Kachel zugeordnet werden.

Definition 3 (Kachelfunktion)**.** *Es sei $V \subset \mathbb{R}^2$ eine endliche Punktmenge und $v \in V$. Die Funktion $H(v)$ sei definiert durch*

$$H(v) = \begin{cases} C(p), & \textit{if } v \in C(p) \\ C(max(p, q)), & \textit{if } v \in E(p, q) \\ C(max(p, q, r, s)), & \textit{if } v \in F(p, q, r, s). \end{cases}$$

Lemma 1. *Wenn zwei Knoten u und v eines Unit-Disk Graphen in ein und derselben Kachel mit der Diagonalen d liegen, muss $\|uv\| \leq d$ sein.*

Beweis. Für zwei Knoten u und v die in der gleichen Kachel liegen gilt $c = H(u) = H(v)$. Der größtmögliche Abstand zweier Punkte in einer Kachel c ist deren Diagonale d. Folglich muss die Strecke $uv \leq d$ sein. □

Korollar 1. *Aus Lemma 1 folgt, dass Knoten aus ein und derselben Kachel garantiert alle Nachrichten mithören können, die ein Knoten aus dieser Kachel versendet, da der Unit-Disk Radius stets größer oder gleich der euklidischen Distanz beider Knoten ist.*

Bei der Betrachtung der Unit-Disk Radien einer Knotenmenge aus einer Kachel c und der Anzahl der umliegenden Kacheln, die von den genannten Radien

geschnitten werden, lassen sich zwei Beobachtungen feststellen, die im folgenden benötigt werden.

Beobachtung 1. *Der Unit-Disk Radius eines Knotens u schneidet maximal dreizehn unterschiedliche Kacheln, $H(u)$ ausgenommen.*

Beobachtung 2. *Die Gesamtheit der Unit-Disk Radien aller Knoten aus einer Kachel c schneidet maximal zwanzig unterschiedliche Kacheln, $H(u)$ ausgenommen.*

Definition 4 (Kachelgraph)**.** *Es sei $V \subset \mathbb{R}^2$ eine endliche Knotenmenge und $G = (V, E)$ der zugehörige Unit-Disk Graph über V. $K(G)$ sei der Kachelgraph $K = (V_K, E_K)$ mit der Knotenmenge V_K und der Kantenmenge E_K.*

$$\begin{aligned} K(G) &= (V_K, E_K) \\ V_K &= \big\{C(s) \mid s \in \mathbb{R}^2 \wedge \exists v \in V \colon H(v) = C(s)\big\} \\ E_K &= \{c_1c_2 \mid c_1, c_2 \in V_K \wedge \exists uv \in E \colon c_1 = H(v), c_2 = H(u)\} \end{aligned}$$

Wenn mindestens ein Knoten $u \in V$ in einer Kachel $C(p)$ liegt, ist $C(p)$ ein Knoten im Kachelgraph K. Hat zusätzlich mindestens ein Knoten v aus Kachel $C(q)$ eine gemeinsame Kante mit einem Knoten aus $C(p)$, so existiert eine Kante zwischen $C(p)$ und $C(q)$ in K.

Lemma 2. *Ein Knoten $c \in V_K$ im Kachelgraph $K = (V_K, E_K)$ eines beliebigen Unit-Disk Graphen hat einen maximalen Knotengrad von 20.*

Beweis. Laut Definition 4 existiert eine Kante c_0c_1 im Kachelgraph $K(G)$ genau dann, wenn im Unit-Disk Graph G eine Kante uv existiert, für die $c_0 = H(u)$ und $c_1 = H(v)$ gilt. Mit Beobachtung 2 und der Annahme, dass ausgehend von c_0 für alle Kacheln $c_1, \ldots, c_{20}$ die von einem Unit-Disk Radius geschnitten werden ein Knotenpaar uv in G existiert, folgt ein maximaler Knotengrad von 20 für den Knoten c_0 im Kachelgraph K. □

Beobachtung 3. *Bei einer Kacheldiagonalen von $d = 1$ können die Kanten eines Kachelgraphen $K = (V_K, E_K)$ vier unterschiedliche Längen haben. Es seien zwei Kacheln $c_1 = C(p)$ und $c_2 = C(q)$ aus V_K gegeben, deren Kante $c_1c_2 \in E_K$ sei. Wenn c_1 und c_2 eine gemeinsame Seite $E(p, q)$ haben, so entspricht $\|c_1c_2\| = \frac{1}{\sqrt{2}}$. Besitzen die Kacheln c_1 und c_2 einen gemeinsamen Punkt $F(p, q, r, s)$ beträgt die Länge der Kante $\|c_1c_2\| = 1$. Besitzen zwei Kacheln weder eine gemeinsame Seite $E(p, q)$ noch einen gemeinsamen*

Punkt $F(p,q,r,s)$ so entspricht die Länge der Kante $\|c_1c_2\| = 2 * \frac{1}{\sqrt{2}}$ wenn die Kacheln parallel zueinander liegen oder $\|c_1c_2\| = \sqrt{\frac{5}{2}}$ wenn die Kacheln nicht parallel zueinander liegen.

Definition 5 (Kachelnachbarschaft)**.** *Knoten u sei ein beliebiger Knoten aus V in der Kachel $C(p)$. $\mathcal{N}_0(u)$ definiert die Menge der 1-hop Nachbarn von u, die ebenfalls in $C(p)$ liegen. Die Menge $\mathcal{N}_1(u)$ umfasst alle 1-hop Nachbarn von u, die nicht in der Kachel $C(p)$ liegen. $\mathcal{N}_2(u)$ umfasst alle 2-hop Nachbarn von u, die über einen 1-hop Nachbarn aus $\mathcal{N}_0(u)$ erreichbar sind und kein Element von $\mathcal{N}_0(u)$ oder $\mathcal{N}_1(u)$ sind.*

$$\begin{aligned}
\mathcal{N}_0(u) &= \{v \in V \colon H(v) = H(u)\} \\
\mathcal{N}_1(u) &= \{v \in V \colon v \notin \mathcal{N}_0(u) \wedge uv \in E\} \\
\mathcal{N}_2(u) &= \{w \in V \setminus \{\mathcal{N}_0(u) \cup \mathcal{N}_1(u)\} : \exists v \in \mathcal{N}_0(u) \colon w \in \mathcal{N}_1(v)\}
\end{aligned}$$

Definition 6 (Knoten des Kachelgraphen)**.** *Wenn $H(u)$ ein Knoten im Kachelgraph $K = (V_K, E_K)$ ist, dann sei $V_K(u)$ die Menge aller 1-hop-Nachbarn von $H(u)$. Ferner lässt sich die Menge $V_K(u)$ in die zwei disjunkten Teilmengen V_{K_1} und V_{K_2} zerlegen.*

$$\begin{aligned}
V_K(u) &= \{c_1, c_2, c_3, \ldots, c_k\} \text{ mit } k \leq 20 \\
&= V_{K_1}(u) \cup V_{K_2}(u) \\
V_{K_1}(u) &= \{c \in V_K(u) \mid \exists v \in \mathcal{N}_1(u) \colon c = H(v)\} \\
V_{K_2}(u) &= \{c \in V_K(u) \mid \exists v \in \mathcal{N}_2(u) \colon c = H(v)\}
\end{aligned}$$

Definition 7 (Kanten des Kachelgraphen)**.** *$E_K(u)$ bezeichnet die Menge aller ausgehenden Kanten von $H(u)$ in K, die sich in zwei disjunkte Teilmengen E_{K_1} und E_{K_2} zerlegen lässt.*

$$\begin{aligned}
E_K(u) &= \{e_1, e_2, e_3, \ldots, e_k\} \text{ mit } k \leq 20 \\
&= E_{K_1}(u) \cup E_{K_2}(u) \\
E_{K_1}(u) &= \{c_1c_2 \in E_K(u) \mid \exists v \in \mathcal{N}_1(u) \colon c_1 = H(v) \wedge c_2 = H(u)\} \\
E_{K_2}(u) &= \{c_1c_2 \in E_K(u) \mid \exists v \in \mathcal{N}_2(u) \colon c_1 = H(v) \wedge c_2 = H(u)\}
\end{aligned}$$

Im Folgenden wird zunächst die Richtigkeit von Phase 1 und Phase 2 des Verfahrens für Unit-Disk Graphen aus Kapitel 3.1 einzeln untersucht. Anschließend wird die Korrektheit des gesamten Verfahrens gezeigt.

Lemma 3 (Phase 1)**.** *Es sei u ein beliebiger Knoten der Menge V im Unit-Disk Graphen $G = (V, E)$ und $K = (V_K, E_K)$ der zugehörige Kachelgraph. Nach Ausführung der ersten Phase des Beaconlosen Clustering Algorithmus kennt Knoten u genau einen Knoten aus jeder Kachel in V_{K1}.*

Beweis. Knoten u versendet einen Request, den alle Knoten $v \in \mathcal{N}_1(u) \cup \mathcal{N}_0(u)$ erhalten. Der Request enthält die Position von u und die Information, dass sich dieser Request nur an Knoten der Menge $\mathcal{N}_1(u)$ richtet. Ein Knoten v der den Request von u empfängt prüft nun, ob er sich in der selben Kachel wie Knoten u befindet. Dabei können zwei Fälle eintreten. In Fall 1 ($v \in \mathcal{N}_0(u)$) liegt v in der Kachel $H(u)$ und ignoriert den eingehenden Request. Im zweiten Fall ($v \in \mathcal{N}_1(u)$) liegt v in einer Kachel $H(v) \neq H(u)$ und der Knoten v weiß, dass sich die Nachricht an ihn richtet. Die Menge $\mathcal{N}_1(u)$ wird durch die Kachelfunktion H und Beobachtung 1 in die Teilmengen $n_1, n_2, \ldots, n_{13}$ zerlegt.

$$\begin{aligned} \mathcal{N}_1(u) &= n_1 \cup n_2 \cup \cdots \cup n_i \text{ mit } i \leq 13 \\ \forall v, w \in n_j &: H(v) = H(w) \text{ mit } n_j \subseteq \mathcal{N}_1(u) \text{ und } 1 \leq j \leq i \end{aligned}$$

Unter der Annahme, dass keine zwei Knoten in einer Kachel den gleichen euklidischen Abstand zum Mittelpunkt ihrer Kachel haben, gibt es in einer Teilmenge $n_i \subseteq \mathcal{N}_1(u)$ genau einen Knoten v der den euklidischen Abstand $\|mv\|$ zum Mittelpunkt m der Kachel $H(v)$ minimiert und gemäß des Verfahrens der einzige Knoten seiner Kachel $H(v)$ ist, der einen Response aussendet. Knoten u empfängt den Response von v und hat somit einen Knoten für eine Kante aus $E_{K_1}(u)$ kennengelernt. Im Laufe von Phase 1 erhält u für jede Teilmenge $n_i \subseteq \mathcal{N}_1(u)$ in der mindestens ein Knoten existiert, eine Nachricht von einem Knoten und kennt schließlich die gesamte Menge $E_{K_1}(u)$. Gäbe es noch eine für u unbekannte Kante $st \in E_{K_1}(u)\colon s = H(u) \wedge t = H(v)$ müsste ein Knoten $v \in \mathcal{N}_1(u)$ existieren, der den Request von u nicht empfängt oder auf diesen nicht antwortet. Für den Fall, dass Knoten v keinen Request empfängt, müsste $\|uv\| > r$ gelten und die Kante uv würde nicht existieren. Für den Fall, dass v nie einen Response aussendet muss ein Knoten $w \in H(v)$ existieren, der bereits einen Response ausgesendet hat und für u bekannt ist. Folgerichtig kennt u nach Phase 1 die Mengen $V_{K_1}(u)$ und $E_{K_1}(u)$. □

Lemma 4 (Phase 2)**.** *Es sei u ein beliebiger Knoten der Menge V im Unit-Disk Graphen $G = (V, E)$ und $K = (V_K, E_K)$ der zugehörige Kachelgraph. Nach Ausführung*

der zweiten Phase des Beaconlosen Clustering Algorithmus kennt Knoten u genau einen Knoten aus jeder Kachel in V_{K2}.

Beweis. Aus der bereits bekannten Menge $V_{K_1}(u)$ bestimmt Knoten u welche Kacheln noch in $V_{K_2}(u)$ enthalten sein könnten. Für jede dieser Kacheln $C \in V_{K_2}(u)$ sendet u einen Request aus, der sich nur an Knoten richtet, die ein Element aus $\mathcal{N}_0(u)$ sind und deren Unit-Disk Radien mindestens einen Schnittpunkt mit einer Kachel c besitzen. Ein Schnittpunkt des Unit-Disk Radius von Knoten v mit einer Kachel sei benannt durch $S(v, C)$. Für einen Knoten $v \in \mathcal{N}_0(u)$ der einen Request von u erhält können nun zwei Fälle eintreten. Im Fall 1 kann v keinen Schnittpunkt seines Unit-Disk Radius mit einer Kachel c aufweisen. Im Fall 2 besitzt v einen Unit-Disk Radius, der mindestens einen Schnittpunkt mit mindestens einer Kachel C aufweist. Mit Beobachtung 2 kann die Menge $\mathcal{N}_0(u)$ nun so abgebildet werden, dass die Unit-Disk Radien der Knoten einer Teilmengen $n_1, \ldots, n_{20}$ die gleiche Kachel schneiden.

$$\begin{aligned} \mathcal{N}_0(u) &= n_1 \cup n_2 \cup \cdots \cup n_i \quad \text{mit } i \leq 20, \text{sodass gilt} \\ \forall v \in n_j &: \exists p \in V_{K_2}(u) \colon \exists S(v, C(p)) \quad \text{mit } n_j \subseteq \mathcal{N}_0(u), 0 \leq j \leq i \end{aligned}$$

Das heißt, dass die Unit-Disk Radien aller Knoten $v \in n_j$ mit $0 \leq j \leq 20$ einen Schnittpunkt mit der selben Kachel $C(p)$ haben. Es sei angemerkt, dass ein Knoten in mehreren Teilmengen enthalten sein kann, wenn dessen Unit-Disk Radius mehrere Kacheln schneidet, die Element von V_{K_2} sein könnten. Die Knoten einer Teilmenge $n_j \in \mathcal{N}_0(u)$ lassen sich mit der Annahme, dass keine zwei Knoten den gleichen euklidischen Abstand zu einem fixen Punkt q besitzen in eine eindeutige Reihenfolge $v_1, v_2, \ldots, v_n \in n_j$ bringen. Knoten v_1 weist den kleinsten euklidischen Abstand zu q auf und sendet gemäß des Verfahrens den ersten Request an alle erreichbaren Knoten der Kachel $C(p)$ aus. Die Knoten $v_2, \ldots, v_n$ hören laut Korollar 1 diese Nachricht mit und unterbrechen vorerst ihr weiteres Vorgehen. Empfängt mindestens ein Knoten in $C(p)$ den Request von v_1, erhält Knoten v_1 genau eine Antwort von einem Knoten $w \in C(p)$. Die Korrektheit dieses Satzes ist äquivalent zum Beweis von Lemma 3 und wird an dieser Stelle nicht näher erläutert. Knoten v_1 wiederholt die Antwort aus Kachel $C(p)$ für die Knoten u und $v_2, \ldots, v_n$, die möglicherweise außerhalb des Senderadius von w liegen. Daraufhin brechen die Knoten $v_2, \ldots, v_n$ ihr Vorhaben, ebenfalls einen Request an die Kachel $H(w)$ auszusenden, endgültig ab und u erfährt von der Existenz des

Knotens $H(w) \in V_{K_2}(u)$ und der Kante $(H(u)H(w)) \in E_{K_2}(u)$. Erhält ein Knoten v_i mit $i \leq n$ nach Aussendung eines Requests innerhalb eines bestimmten Zeitraums keine Antwort von einem Knoten w und leitet somit keine Nachricht an u und $v_{i+1}, v_{i+2}, \ldots, v_n$ weiter, sendet der Knoten v_{i+1} als nächstes einen Request für die Kachel $H(w)$ aus, woraufhin die Knoten $v_{i+2}, v_{i+3}, \ldots, v_n$ ihr weiteres Vorgehen wiederholt unterbrechen. Dies wird fortgeführt bis ein Knoten w auf einen Request von $v_1, v_2, \ldots, v_n$ antwortet, oder selbst der letzte Knoten v_n keine Antwort erhält. Für die weitere Betrachtung wird die Menge $\mathcal{N}_2(u)$ so zerlegt, dass Knoten die in ein und derselben Kachel liegen in einzelnen Teilmengen $m_1, \ldots, m_{20}$ zusammengefasst werden. Eine Teilmenge $m_l \subseteq \mathcal{N}_2(u)$ mit $1 \leq l \leq 20$ besteht aus Knoten die in einer gemeinsamen Kachel liegen. Aus Definition 5 folgt, dass jeder Knoten $w_1, w_2, \ldots, w_n \in m_l$ eine Kante mit mindestens einem Knoten v aus einer Menge $n_j \subseteq \mathcal{N}_0(u)$ mit $0 \leq j \leq 20$ besitzt. Wenn eine Kante $(H(u)H(w)) \in E_{K_2}(u)$ existiert, muss folgendes gelten:

$$\exists v \in n_j, w \in m_l \quad : \quad vw \in E \quad \text{mit } n_j \subseteq \mathcal{N}_0(u) \text{ und } m_l \subseteq \mathcal{N}_2(u)$$

Da das Verfahren sukzessive für jeden Knoten v der Mengen $n_1, \ldots, n_{20}$ prüft, ob v eine gemeinsame Kante mit einem Knoten w aus einer der Mengen m_l besitzt, ist nach Ausführung des Verfahrens für jede Kachel in $V_{K_2}(u)$ genau ein Knotenpaar vw bekannt für das $(H(u)H(w)) \in E_{K_2}(u)$ gilt. □

Satz. *Sei $G = (V, E)$ ein Unit-Disk Graph, $K = (V_K, E_K)$ der zugehörige Kachelgraph und $u \in V$ ein beliebiger Knoten. Nach Ausführung des Beaconlosen Clustering Algorithmus kennt Knoten u alle ausgehenden Kanten E_K von $H(u)$ in K.*

Beweis. Wenn der Knoten u sowohl Phase 1 als auch Phase 2 des Verfahrens ausgeführt hat, folgt aus Lemma 3 und 4, dass u die vollständige Menge V_K des Kachelgraphen $K = (V_K, E_K)$ und somit auch die Menge aller Kanten E_K kennt. □

4.2 Korrektheit im Quasi-Unit-Disk Graph

Um die Beweisführung aus Abschnitt 4.1 auf das Verfahren für Quasi-Unit-Disk Graphen aus Abschnitt 4.2 zu übertragen sind nur wenige Änderungen hinsichtlich der Quasi-Unit-Disk Radien $r_{\min}$ und $r_{\max}$ nötig. Im Folgenden wird für die

vereinfachte Darstellung davon ausgegangen, dass die Größe des Unit-Disk Radius r dem Quasi-Unit-Disk Radius $r_{\min}$ entspricht und $r_{\max} = \sqrt{2} * r_{\min}$ gelte. Für die Definition der Diagonalen einer Kachel wird der Radius $r_{\min}$ verwendet, damit Korollar 1 weiterhin gültig ist. Würde die Diagonale größer als $r_{\min}$ definiert, könnte der unerwünschte Effekt eintreten, dass ein Knoten eine Nachricht aus seiner eigenen Kachel nicht empfängt, weil er außerhalb des Senderadius der Nachricht liegt. Durch die Betrachtung des Radius $r_{\max}$ erweitert sich die Beobachtung 1 auf 22 Kacheln, die von einem Quasi-Unit-Disk Radius $r_{\max}$ geschnitten werden können. Beobachtung 2 erweitert sich äquivalent aufgrund des Radius $r_{\max}$ auf insgesamt 24 erreichbare Kacheln. Das hat zur Folge, dass im Kachelgraph eine weitere Kantenlänge möglich ist. Ausgehend von einem Knoten u, der das Verfahren ausführt, und seiner Kachel $H(u)$, liegen die zusätzlich erreichbaren Kacheln c_1, c_2, c_3 und c_4 diagonal zu $H(u)$. Die Länge der Kante von $H(u)$ zu einer der Kacheln $c_1, \ldots, c_4$ entspricht der zweifachen Kacheldiagonalen $2 * d = 2 * r_{\min}$. Gleichzeitig ergibt sich durch die Erweiterung der Beobachtung 2 für Lemma 2 folgende Änderung:

Lemma 5. *Ein Knoten $c \in V_K$ im Kachelgraph $K = (V_K, E_K)$ eines beliebigen Quasi-Unit-Disk Graphen hat einen maximalen Knotengrad von 24.*

Beweis. Aus der Erweiterung der Beobachtung 2 für Quasi-Unit-Disk Graphen und dem Beweis zu Lemma 2 folgt für eine Kachel im Kachelgraph $K(G)$ mit einem Quasi-Unit-Disk Graph G ein maximaler Knotengrad von 24. □

Unter der Berücksichtigung der genannten Änderungen kann die Korrektheit des Verfahrens für Quasi-Unit-Disk Graphen mit der gleichen Vorgehensweise aus Abschnitt 4.1 für Unit-Disk Graphen gezeigt werden.

4.3 Effizienzbetrachtung

In diesem Abschnitt wird die Anzahl der am Verfahren beteiligten Knoten und der insgesamt versendeten Nachrichten betrachtet. Dazu werden in den folgenden Abschnitten die pathologischen Fälle des Best-Case, als auch des Worst-Case untersucht.

Best-Case Konstruktion Für den Best-Case wird angenommen, dass u der ausführende Knoten sei und die Kachel $H(u)$ im Kachelgraph den maximalen

Knotengrad von 20 für Unit-Disk Graphen und 24 für Quasi-Unit-Disk Graphen habe. Außerdem seien für jeden Knoten aus $H(u)$ die maximale Anzahl der umliegenden Kacheln erreichbar, sodass in jeder erreichbaren Region einer Kachel mindestens ein Knoten liegt.

Worst-Case Konstruktion Für den Worst-Case wird angenommen, dass Knoten u keine direkt erreichbaren Knoten außerhalb seiner Kachel aufweisen kann und alle Knoten der Kachel von u am Verfahren beteiligt werden. Für jede der umliegenden Kacheln existiert eine Reihe von Knoten $v_1, v_2, \ldots, v_n$ aus der Kachel $H(u)$ die in Phase 2 des Verfahrens der Reihe nach eine offene Kachel auf einen erreichbaren Knoten prüfen, sodass der letzte Knoten v_n fündig wird.

Abbildung 4.1 zeigt die Konstruktion eines Worst-Case für eine Kachel C. Der BCA wird auf Knoten u ausgeführt und v_1, v_2, v_3, v_n sind potentielle Brückenknoten für C. Im schlechtesten Fall befinden sich beliebig viele Knoten auf einer Linie zwischen v_1 und v_n, die durch einen Nachrichtenversand am Verfahren beteiligt sind, obwohl lediglich der letzte Knoten v_n die Eigenschaft eines Brückenknotens erfüllt. An dieser Stelle sei anzumerken, dass im Rahmen dieser Arbeit keine Möglichkeit gefunden wurde, den in diesem Fall entstehenden Nachrichtenoverhead zu reduzieren.

4.3.1 Anzahl aktiver Knoten

Alle Knoten, die das Verfahren starten oder während dessen einen FirstResponse oder einen SecondRequest versenden, werden als *aktive Knoten* bezeichnet und lassen sich durch ihre Zugehörigkeit einer Kachel unterteilen. Von besonderem Interesse ist die Anzahl der aktiven Knoten in der Kachel von $H(u)$. In allen anderen Kacheln existiert gemäß des Verfahrens maximal ein aktiver Knoten. Tabelle

	Best-Case	Worst-Case
UDG	3	20
QUDG	2	24

Tabelle 4.1: Anzahl aktiver Knoten in der ausführenden Kachel nach Verfahrensablauf im Best-Case und im Worst-Case.

4.1 zeigt die Anzahl aktiver Knoten nach Ausführung des BCA für Unit-Disk

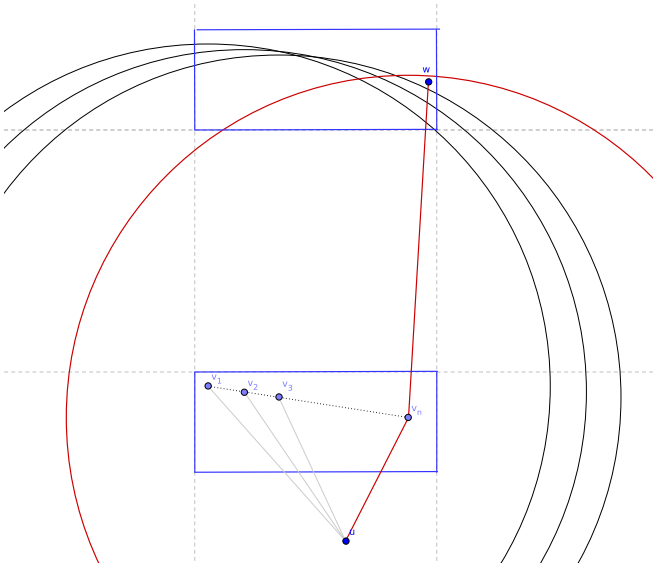


Abbildung 4.1: Konstruktion eines Worst-Case. Wird der BCA auf dem dargestellten Graphen ausgeführt, versendet der Reihe nach jeder Knoten $v_1, \ldots, v_n$ eine Nachricht bis v_n letztendlich Knoten w kennen lernt.

und Quasi-Unit-Disk Graphen im besten und im schlechtesten Fall. Im Unit-Disk Graph reichen im besten Fall bereits drei aktive Knoten in $H(u)$ aus, um im Kachelgraph einen Knotengrad von 20 zu erzielen. Aufgrund der größeren Reichweite eines Knotens im Quasi-Unit-Disk Graph reichen hier im Best-Case zwei aktive Knoten aus. Im Verfahren für Unit-Disk Graphen liegt die maximale Anzahl an aktiven Knoten einer Kachel $H(u)$ bei 20. Dieser Fall tritt ein, wenn alle Knoten aus der Kachel $H(u)$ lediglich eine Kante mit einem Knoten der umliegenden Kacheln besitzt, den ansonsten kein anderer Knoten aus $H(u)$ erreicht. Daraus und aufgrund der vier zusätzlichen Kacheln resultiert für den Worst-Case im Verfahren für Quasi-Unit-Disk Graphen eine maximale Anzahl aktiver Knoten pro Kachel von 24. Zusammenfassend lässt sich sagen, dass die Anzahl aktiver Knoten im Verfahren für Unit-Disk Graphen und Quasi-Unit-Disk Graphen sowohl im Best-Case als auch im Worst-Case durch eine konstante Größe beschränkt ist.

4.3.2 Anzahl versendeter Nachrichten

Im Verfahren für Unit-Disk Graphen ohne Erweiterungen lässt sich die Anzahl der insgesamt versendeten Nachrichten für den Best-Case und den Worst-Case wie in Tabelle 4.2 dargestellt berechnen. Im besten Fall kommt das Verfahren mit 42 Nachrichten aus, um aus jeder der 20 umliegenden Kachel einen Knoten kennenzulernen. Aus Abschnitt 4.3.1 ist bekannt, dass im Best-Case drei aktive Knoten in $H(u)$ existieren, sodass es insgesamt 23 aktive Knoten gibt, die im Laufe des Verfahrens eine Nachricht verschicken. Damit der ausführende Knoten u in Phase 1 des Verfahrens 13 Knoten kennenlernt muss er einen FirstRequest versenden, der die 13 erreichbaren Knoten veranlasst jeweils mit einem FirstResponse zu antworten. Für Phase 1 ergeben sich so 14 Nachrichten. In Phase 2 versendet Knoten u für die sieben offenen Kacheln jeweils einen SecondRequest. Pro Kachel sendet jeweils ein Knoten v aus $H(u)$ einen FirstRequest, auf den pro Kachel ein Knoten mit einem FirstResponse antwortet, der von v in Form eines SecondResponses weitergeleitet wird. So werden in Phase 2 insgesamt 28 Nachrichten versendet. Die Anzahl der versendeten Nachrichten im Worst-Case ist abhängig von der Menge der Knoten, die in Phase 2 einen FirstRequest an eine Kachel aussendet, ohne eine Antwort zu erhalten. Im schlechtesten Fall sind das für jede der 20 Kacheln beliebig viele Knoten. Für den definierten Worst-Case ergeben sich so $61 + 20 * n$ Nachrichten.

	Nachrichtentyp	Best-Case	Worst-Case
Phase 1	FirstRequests	1	1
	FirstResponses	13	0
Phase 2	SecondRequests	7	20
	FirstRequests	7	20*n
	FirstResponses	7	20
	SecondResponses	7	20
	Summe	42	61+20*n

Tabelle 4.2: Anzahl versendeter Nachrichten im Verfahren für Unit-Disk Graphen ohne Erweiterungen im Best-Case und im Worst-Case.

Die Tabelle 4.3 zeigt die Anzahl der versendeten Nachrichten bei Ergänzung des Verfahrens durch die in Abschnitt 3.3.1 beschriebene Verkürzung der Laufzeit. Grundsätzlich werden dadurch die versendeten SecondRequests aus Phase

2 zu einer Nachricht zusammengefasst und pro Knoten nur noch maximal ein FirstRequest ausgesendet, wodurch im Best-Case 11 Nachrichten eingespart werden. Durch die Erweiterung nimmt die Anzahl der versendeten FirstRequests um den Faktor 20 ab, sodass insgesamt maximal $42+n$ Nachrichten versandt werden.

	Nachrichtentyp	Best-Case	Worst-Case
Phase 1	FirstRequests	1	1
	FirstResponses	13	0
Phase 2	SecondRequests	1	1
	FirstRequests	2	n
	FirstResponses	7	20
	SecondResponses	7	20
	Summe	31	42+n

Tabelle 4.3: Anzahl versendeter Nachrichten im Verfahren für Unit-Disk Graphen mit verkürzter Laufzeit im Best-Case und im Worst-Case.

Wird das Verfahren zusätzlich um die Einschränkung der Knoten wie in Abschnitt 3.3.2 beschrieben erweitert, lassen sich im Regelfall weitere FirstRequests in Phase 2 einsparen. Die Anzahl der Nachrichten im Worst-Case beträgt allerdings nach wie vor $42 + n$ Nachrichten.

Kapitel 5

Implementierung

Dieses Kapitel beschäftigt sich mit der Implementierung des Beaconlosen Clustering Algorithmus, ergänzt um die verkürzte Laufzeit aus Abschnitt 3.3.1. Auf die Erweiterung aus Abschnitt 3.3.2 wird bewusst verzichtet. Da sich die Funktion für Unit-Disk Graphen und Quasi-Unit-Disk Graphen kaum unterscheidet, wird die Implementierung so gestaltet, dass sie für beide Modelle anwendbar ist. Die Beschreibung in diesem Kapitel beschränkt sich im Wesentlichen auf die Zustände, die ein Knoten durchläuft. Dafür werden die drei Rollen *ausführender Knoten*, *Knoten in Phase 1* und *Knoten in Phase 2* betrachtet. Der vollständige Programmcode des Verfahrens ist dem dieser Arbeit beiliegenden Datenträger zu entnehmen.

Um den Algorithmus zu testen und später ebenfalls zu simulieren wird für die Implementierung das Java-Simulationsframework *Sinalgo* [1] verwendet. Sinalgo enthält unter anderem vordefinierte Klassen für Knoten, Kanten, Nachrichten und die Implementierung der benötigten Modelle Unit-Disk und Quasi-Unit-Disk Graph, wodurch ein guter Ausgangspunkt für die Umsetzung des Algorithmus gegeben ist. Um die Implementierung der Knoten genauer betrachten zu können, wird vorab die Umsetzung der Kachelfunktion und der unterschiedlichen Nachrichtentypen erklärt.

[1] Distributed Computing Group(2007): Sinalgo. http://www.disco.ethz.ch/projects/sinalgo/ (abgerufen am 12.05.2014)

5.1 Die Kachelfunktion

Es wird für die Implementierung angenommen, dass die zweidimensionale Ebene in der sich die Knoten befinden ihren Ursprung im Punkt $(0, 0)$ hat und sich auf die positiven Bereiche beschränkt. Das global bekannte Gitter aus Kacheln wird am Ursprung ausgerichtet, sodass der Eckpunkt einer initialen Kachel an der alle übrigen ausgerichtet werden im Punkt $(0, 0)$ liegt. Identifiziert werden die Kacheln über die Position ihrer zum Ursprung nächsten Ecke. Die im Framework vorhandene Klasse *Node.java*, die zur Implementierung der Knoten verwendet wird, enthält bereits die Position eines Knotens, sodass mithilfe der bekannten Ausrichtung und Breite der Kacheln jeder Knoten berechnen kann, in welcher Kachel er sich befindet. Algorithmus 1 zeigt die Implementierung der Kachel-

Algorithmus 1 Die Berechnung der Position (c_x, c_y) einer Kachel c für einen Knoten u aus c mit der Position (x, y).

Eingabe: (x, y)
Ausgabe: (c_x, c_y)
$b \leftarrow Kachelbreite$
$c_x \leftarrow x - (x \mod b)$
$c_y \leftarrow y - (y \mod b)$

funktion, die der Zuordnung eines Knotens zu einer Kachel dient. Anhand der Position des Knotens wird die Position seiner Kachel berechnet. Um im späteren Verlauf ausgewählte Kacheln adressieren zu können, wird eine Vorschrift zur Positionsbestimmung der umliegenden Kacheln benötigt. Dazu zeigt Abbildung 5.1 ausgehend von einer beliebigen Kachel c_{12} mit den Koordinaten (x, y) die relative Position jeder erreichbaren Kachel. Zur Berechnung der absoluten Position einer Kachel nutzt ein Knoten die Koordinaten (x, y) der eigenen Kachel, die Kachelbreite b und die in Abbildung 5.1 aufgeführten relativen Positionsangaben. Die Kacheln c_0, c_4, c_{20} und c_{24} sind lediglich im Quasi-Unit-Disk Graph von Bedeutung und sind im Unit-Disk Graph für einen Knoten aus c_{12} nicht erreichbar.

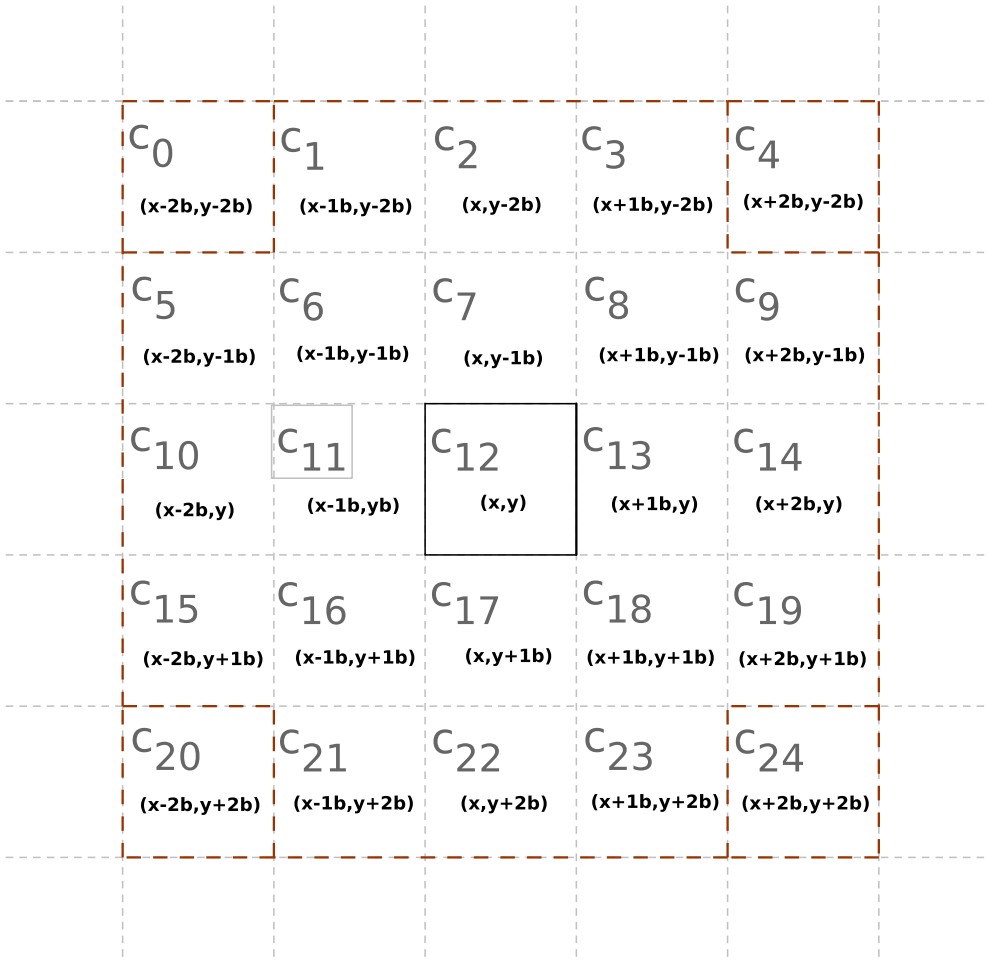


Abbildung 5.1: Nummerierung der erreichbaren Kacheln ausgehend von c_{12}. Mit den Koordinaten x und y aus c_{12} und der Kachelbreite b lassen sich die Positionen aller Kacheln $c_0, \ldots, c_{24}$ wie abgebildet berechnen. c_0, c_4, c_{20} und c_{24} sind nur im Quasi-Unit-Disk Graphen erreichbar.

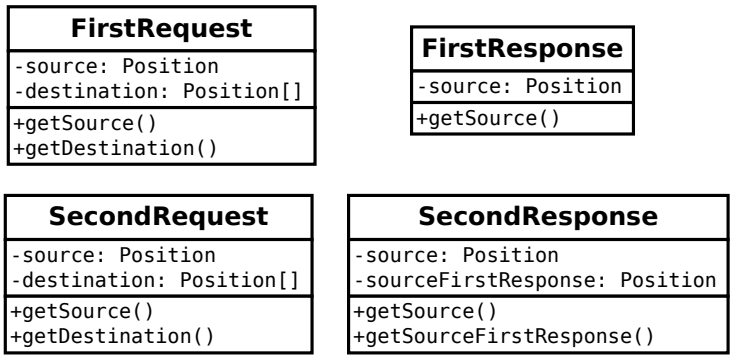


Abbildung 5.2: Objektklassen der vier benötigten Nachrichtentypen

5.2 Nachrichtenklassen

Für die Kommunikation der Knoten untereinander werden die vier Nachrichtentypen FirstRequest, FirstResponse, SecondRequest und SecondResponse genutzt. Für jeden dieser Nachrichtentypen existiert wie Abbildung 5.2 darstellt eine Klasse. Jede dieser Klassen besitzt ein Attribut *source* in dem die geographischen Koordinaten des Senders hinterlegt werden. Somit kann die Nachrichtenherkunft von jedem Empfänger eindeutig einer Kachel zugeordnet werden. Für die Adressierung eines First- oder SecondRequests besitzen diese ein *Array*, in dem die Positionskoordinaten der angestrebten Kacheln hinterlegt werden. Während ein FirstResponse lediglich die Position des Senders enthält, verfügt ein SecondResponse über ein weiteres Attribut für die Koordinaten eines zweiten Knotens.

5.3 Der ausführende Knoten

Wird auf einem Knoten u das Verfahren gestartet, durchläuft dieser das in Abbildung 5.3 dargestellte Ablaufdiagramm. Der erste Schritt besteht aus dem Erstellen und Versenden eines FirstRequest-Objekts. Dazu müssen die Positionen der Kacheln c_0 bis c_{24} berechnet werden. Dafür wird mit der Position der Kachel c_{12} und der Abbildung 5.1 ein Array Namens *direkteKacheln* der Länge 25 erstellt, in das die Positionen aller Kacheln geschrieben werden. Dieses Array wird zusammen mit der Position des Knotens u in einem FirstRequest versendet. In der darauf folgenden Zeit $t_{\max}$ wird pro Kachel, deren Knoten für u unmittelbar erreichbar sind, eine FirstResponse-Nachricht empfangen. Die darin enthaltenen

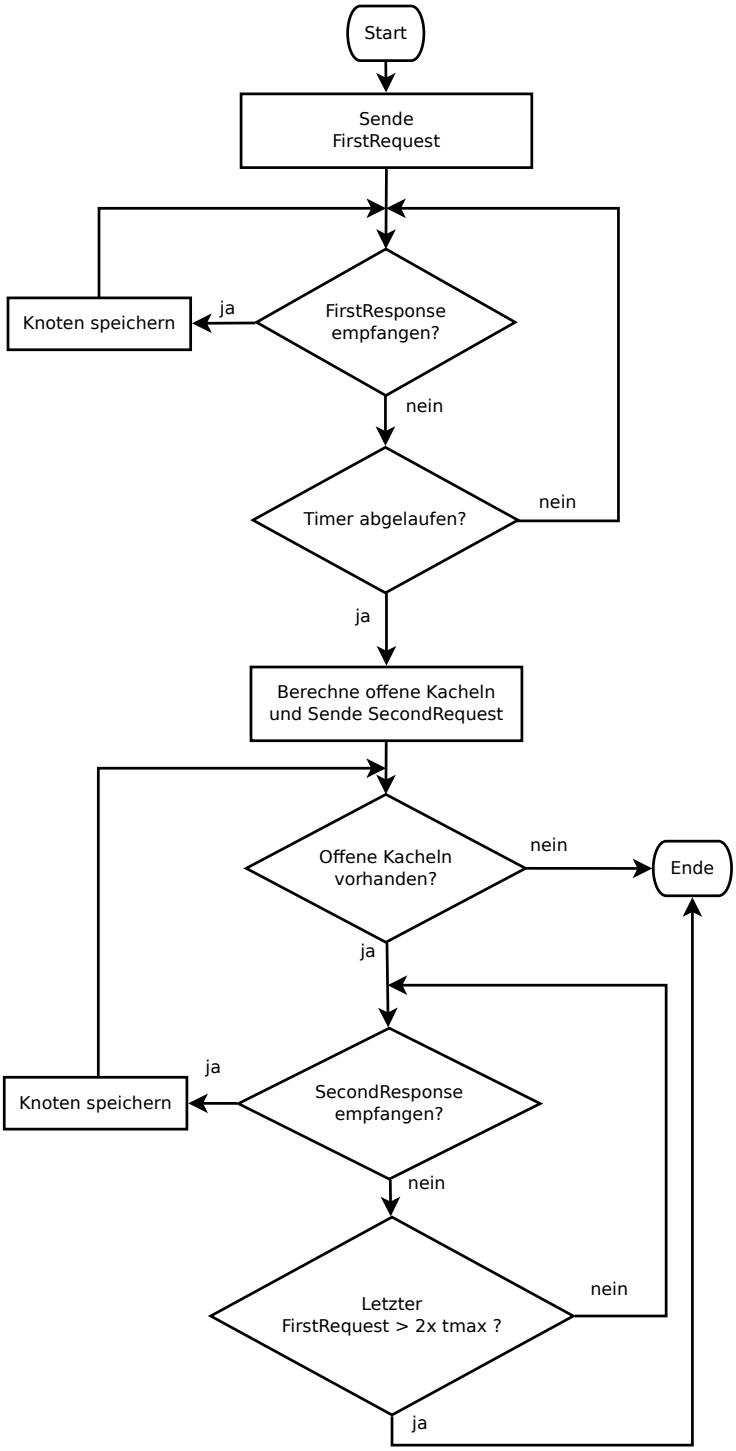


Abbildung 5.3: Ablaufplan für den Knoten, der das Verfahren startet.

Knotenpositionen werden in ein weiteres Array *erreichbareKacheln* mit 25 Feldern gespeichert. Die Felder dieses Arrays stehen repräsentativ für die einzelnen Kacheln c_0 bis c_{24} und enthalten die Position eines Knotens, falls für diese Kachel bereits ein Knoten bekannt ist. Wenn u kein Knoten im Quasi-Unit-Disk Graph ist, wird für die im Unit-Disk Graph nicht erreichbaren Kacheln c_0, c_4, c_{20} und c_{24} eine Pseudoposition $(-1, -1)$ hinterlegt, damit diese im weiteren Verlauf als abgeschlossene Kacheln gelten und nicht weiter berücksichtigt werden. Für jede eintreffende FirstResponse-Nachricht berechnet u anhand der enthaltenen Position und Algorithmus 1 aus welcher Kachel die Nachricht stammt und schreibt die enthaltenen Position in das entsprechende Feld des Arrays *erreichbareKacheln*. Nach Ablauf des Zeitfensters $t_{\max}$ überprüft Knoten u mithilfe des Arrays für welche Kacheln noch keine Knotenposition bekannt ist. Die Positionen dieser Kacheln werden in ein weiteres Array *indirekteKacheln* geschrieben, das zusammen mit der Position von u in einem SecondRequest versendet wird. Daraufhin beginnt Phase 2 des Verfahrens und Knoten u wartet so lange auf eintreffende SecondResponse-Nachrichten, bis entweder für alle Kacheln aus *indirekteKacheln* ein SecondResponse empfangen wurde oder keine weiteren Nachrichten mehr eintreffen. Wenn ein SecondResponse eintrifft, wird zur Position *sourceFirstResponse* die entsprechende Kachel c_i berechnet und im Array *erreichbareKacheln* in das Feld, welches die Kachel c_i repräsentiert geschrieben. Solange kein SecondResponse eintrifft, wird geprüft, welcher Zeitraum seit dem letzten mitgehörte FirstRequest vergangen ist. Ist dieser Zeitraum kleiner als $2 * t_{\max}$, könnten weitere SecondResponses eintreffen. Ist der letzte mitgehörte FirstRequest länger als $2 * tmax$ vergangen, folgen keine weiteren Nachrichten und Knoten u beendet das Verfahren.

5.4 Empfang eines FirstRequests

In Phase 1 des Verfahrens versendet Knoten u einen FirstRequest, um aus jeder Kachel, deren Knoten unmittelbar in seinem Senderadius liegen, exakt eine Antwort zu erhalten. Empfängt ein Knoten v einen FirstRequest durchläuft er die in Abbildung 5.4 dargestellten Zustände. Nach Erhalt der Nachricht wird geprüft, ob die Nachricht für Knoten v relevant ist. Dazu wird anhand der Position *source* des FirstRequests die Kachel c_{12} des Senders berechnet und mit der eigenen Kachel abgeglichen. Entspricht die Kachel c_{12} der eigenen, liegen Sen-

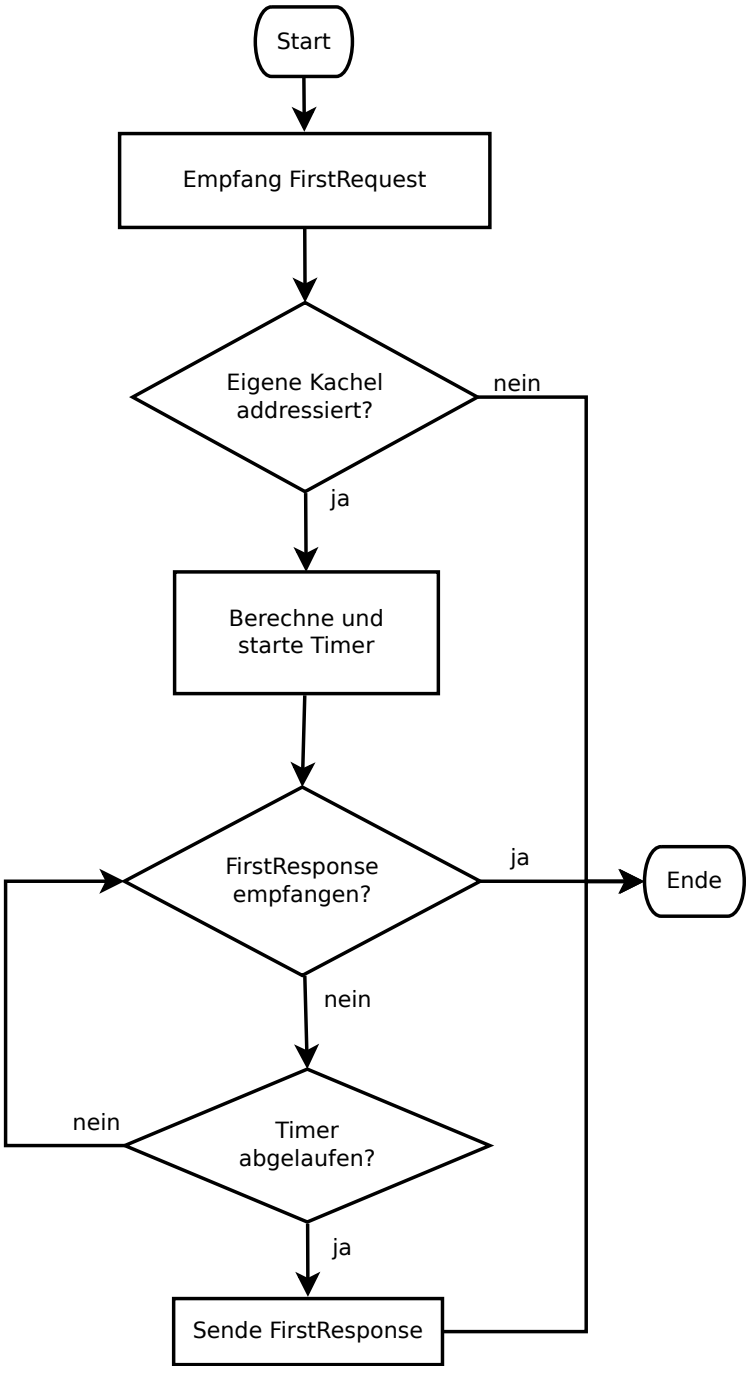


Abbildung 5.4: Ablaufplan für Knoten die eine FirstRequest-Nachricht empfangen.

der und Empfänger in der gleichen Kachel, sodass Knoten v seine Teilnahme an Phase 1 des Verfahrens beendet. Andernfalls berechnet Knoten v die euklidische Distanz d, die sich aus dem euklidischen Abstand der Position des Knotens zum Mittelpunkt seiner Kachel ergibt. Die Distanz d wird benötigt um wiederum den Timer t bestimmen zu können. Damit $t \leq t_{\max}$ gilt, wird der Timer durch $t(d, d_{\max}) = \frac{d}{d_{\max}} * t_{\max}$ berechnet, wobei $d_{\max}$ der maximal möglichen Distanz d innerhalb der Kachel entspricht. Sobald der Timer gestartet wird, werden alle mitgehörten FirstResponse-Nachrichten auf ihre Herkunft untersucht. Sobald ein FirstResponse empfangen wird, dessen Position *source* aus der Kachel von v stammt, bricht v seinen Timer ab und das Verfahren ist für v beendet. Wenn dieser Fall ausbleibt und der Timer t abläuft, versendet v einen FirstResponse mit seiner Position und beendet seine Teilnahme am Verfahren.

5.5 Empfang eines SecondRequests

Mit dem Beginn von Phase 2 versendet der ausführende Knoten u aus c_{12} einen SecondRequest. Empfängt ein Knoten w aus einer Kachel c_i mit $0 \leq i \leq 24$ diese Nachricht, durchläuft er die in Abbildung 5.5 dargestellten Schritte. Zuerst ist zu prüfen, für welche Kachel die Nachricht bestimmt ist. Dazu berechnet w die Kachel des Senders über die in der Nachricht enthaltene Position *source* und gleicht diese mit der eigenen Kachel c_i ab. Wenn $c_i \neq c_{12}$ ist, besitzt die Nachricht keine Relevanz und w beendet seine Teilnahme am Verfahren. Entspricht $c_1 = c_{12}$ beginnt Knoten w mit einer Timerberechnung. Dazu prüft w die Erreichbarkeit jeder Kachel c_j mit $0 \leq j \leq 24$ aus dem Array *destination* des SecondRequests. Dabei bezieht sich die Erreichbarkeit eines Knotens w im Unit-Disk Graph auf die Reichweite des Unit-Disk Radius r. Wird das Verfahren im Quasi-Unit-Disk Graph ausgeführt, entspricht die maximale Reichweite dem Radius $r_{\max}$ des Quasi-Unit-Disk Graphen. Für alle erreichbaren c_j bestimmt Knoten w einen Wert d_j, der wie folgt berechnet wird. Die euklidische Distanz von w aus c_{12} zum Mittelpunkt m einer Kachel c_j wird in einem Punkt s vom Rand der Kachel c_{12} geschnitten, sodass die Strecke $\|ms\|$ den Wert d_j definiert. Gleichzeitig wird für jeden Strecke d_j der Wert $d_{j\,\max}$ bestimmt, der den maximal möglichen Wert d_j eines Knotens aus c_{12} definiert. Mithilfe der berechneten Werte wird für jede erreichbare Kachel c_j aus *source* ein Timer mit $t_j = \frac{d_j}{d_{j\,\max}} * t_{\max}$ berechnet. Unter allen Timern t_j wählt Knoten w den kleinsten aus und startet diesen. Sobald nun ein FirstRequest emp-

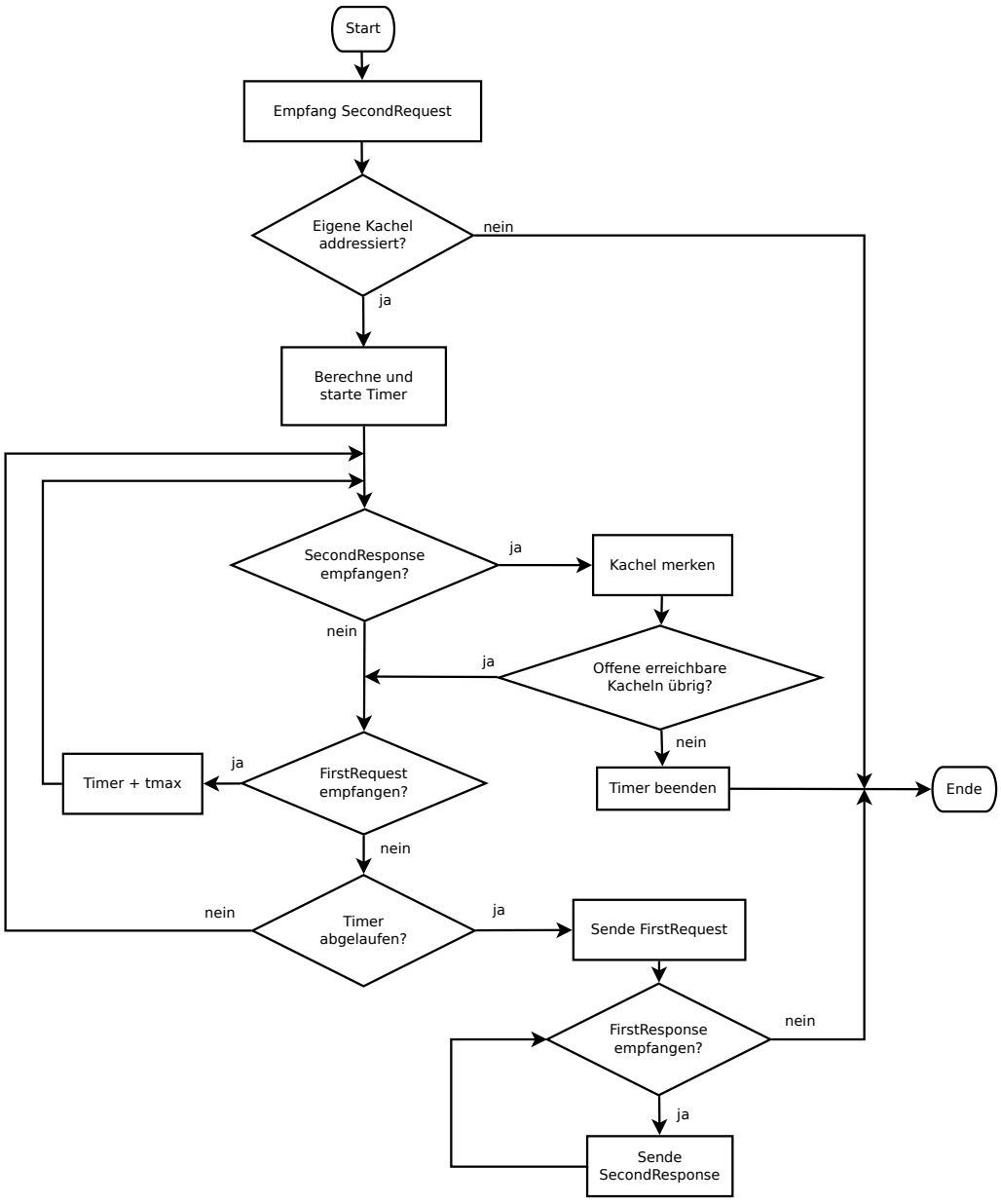


Abbildung 5.5: Ablaufplan für Knoten die eine SecondRequest-Nachricht empfangen.

fangen wird, erhöht Knoten w seinen laufenden Timer um die Dauer $t_{\max}$, denn dies ist der Zeitraum, in dem ein weiterer Knoten eine Antwort auf seinen versendeten FirstRequest erhalten kann. Wenn in der darauf folgenden Zeit $t_{\max}$ ein SecondResponse empfangen wird, vermerkt Knoten w welche Kachelinformation in dieser Nachricht steckt und prüft, ob für ihn noch offene Kacheln erreichbar sind. Ist das nicht der Fall, beendet w seinen Timer und schließt das Verfahren ab. Bleiben weiterhin erreichbare Kacheln offen, wird wiederum auf eintreffende FirstRequest-Nachrichten geachtet. Sobald der Timer von Knoten w abläuft, sendet er einen FirstRequest mit allen noch offenen erreichbaren Kacheln aus. Für jede FirstResponse-Nachricht, die im folgenden Zeitraum $t_{\max}$ empfangen wird, versendet w einen SecondResponse, um den Knoten seiner Kachel die empfangene Position *source* der FirstResponse-Nachricht mitzuteilen. Nach Ablauf der Zeit $t_{\max}$ endet das Verfahren für Knoten w.

Kapitel 6

Simulation

In diesem Kapitel wird der Beaconlose Clustering Algorithmus in einer Simulationsumgebung untersucht und mit einem in der Literatur gängigen Verfahren für Quasi-Unit-Disk Graphen verglichen. Dazu werden die im vorherigen Kapitel vorgestellte Implementierung und das Simulationsframework Sinalgo verwendet. Zur Erhebung geeigneter Messwerte wird der Simulationsvorgang mithilfe eines Perl-Skripts automatisiert.

6.1 Setup der Simulation

Zur erleichterten Messdatenerhebung wird das Perl-Skript aus Anhang A verwendet, das der Reihe nach einzelne Simulationsdurchläufe startet, Messwerte ausliest und für die Auswertung in Abschnitt 6.2 aufbereitet.

Vor der Ausführung lässt sich das Skript für die gewünschte Simulation mit den folgenden Parametern konfigurieren:

conModel Gibt an welches Modell verwendet wird, *UDG* für Unit-Disk oder *StaticQUDG* für Quasi-Unit-Disk.

nodeDenseMin Die minimale Knotendichte, für die Simulationen durchgeführt werden.

nodeDenseMax Die maximale Knotendichte, für die Simulationen durchgeführt werden.

rounds Anzahl durchzuführender Simulationsdurchläufe pro Knotendichte.

dimX Breite der Simulationsebene.

dimY Höhe der Simulationsebene.

radiusUnitDisk Gibt im Unit-Disk Modell den Unit-Disk Radius an.

radiusMin Gibt im Quasi-Unit-Disk Modell den Radius $r_{\min}$ an.

radiusMax Gibt im Quasi-Unit-Disk Modell den Radius $r_{\max}$ an.

conProbability Die Wahrscheinlichkeit mit der zwei Knoten im Quasi-Unit-Disk Graph verbunden sind, wenn $r_{\min} < d \leq r_{\max}$ für deren Abstand d gilt.

tMax Maximale Zeit $t_{\max}$, die zur Berechnung der Timerfunktionen verwendet wird.

Die *Knotendichte* eines Graphen gibt die zu erwartende durchschnittliche Anzahl direkt erreichbarer Nachbarn pro Knoten an. Für jede ganzzahlige Knotendichte $nodeDense$ für die gilt, dass $nodeDenseMin \leq nodeDense \leq nodeDenseMax$ ist, führt das Skript die folgenden Schritte aus. Zuerst wird anhand der Größe der Simulationsebene und der Knotendichte die absolute Anzahl zu generierender Knoten $numNodes$ für die entsprechende Knotendichte eines Simulationsdurchlaufs berechnet. Anschließend wird eine Datei mit den Konfigurationsparametern und der Anzahl zu generierender Knoten erstellt, in der im nächsten Schritt die Messwerte aller Simulationsdurchläufe mit der selben Knotendichte geschrieben werden. Dazu werden für die Anzahl $rounds$ aufeinander folgend Simulationen gestartet, denen per Kommandozeilen-Parameter die Simulationskonfiguration und der Dateiname für die Messwerte übergeben wird. Innerhalb einer Simulation wird anhand der Anzahl $numNodes$ eine zufällig positionierte und uniform gleichverteilte Knotenmenge generiert. Aus der mittleren Kachel der Simulationsebene wird ein zufälliger Knoten ausgewählt, auf dem das Verfahren ausgeführt wird. Für den Fall, dass die mittlere Kachel keinen Knoten beinhaltet, wird eine neue Knotenmenge der Größe $numNodes$ generiert. Wenn das Verfahren vom ausführenden Knoten beendet wird, werden die Messwerte dieses Simulationsdurchlaufs an die bestehende Datei angehängt und der nächste Durchlauf gestartet. Sobald alle Simulationen ausgeführt wurden, wird ausgehend von der Menge der gewonnenen Daten eine Datei zur Berechnung von Mittelwerten, zugehörigen Standardabweichungen und Extremwerten erstellt. Basierend auf dieser Zusammenfassung der Messergebnisse wird in Abschnitt 6.2 die Simulation ausgewertet.

Für die Auswertung in Abschnitt 6.2 wurde das Skript zur Simulationsdurchführung jeweils einmal für Unit-Disk Graphen und Quasi-Unit-Disk Graphen

ausgeführt. Die Konfiguration des Skripts vor der jeweiligen Ausführung ist in Listing 6.1 und 6.2 zu sehen.

```
$conModel = UDG;
$nodeDensMin = 4;
$nodeDensMax = 20;
$rounds = 500;
$dimX = 354;
$dimY = 354;
$udr = 100;
$qudrMin = $udr;
$qudrMax = 141;
$conProbability = 0.6;
$tmax = 100;
```

Listing 6.1: Konfiguration für die Simulation von UDG

```
$conModel = StaticQUDG;
$nodeDensMin = 4;
$nodeDensMax = 20;
$rounds = 500;
$dimX = 354;
$dimY = 354;
$udr = 100;
$qudrMin = $udr;
$qudrMax = 141;
$conProbability = 0.6;
$tmax = 100;
```

Listing 6.2: Konfiguration für die Simulation von QUDG

Um den BCA hinsichtlich benötigter Nachrichten und anderer Eigenschaften mit dem in der Literatur gängigen Ansatz vergleichen zu können, wird das folgende Verfahren *2-hop protocol* zur Einholung vollständiger 2-hop Nachbarschaftsinformation verwendet.

1. Ein beliebiger Knoten u auf dem das Verfahren gestartet wird sendet eine Nachricht aus, um seinen 1-hop Nachbarn $v_1, v_2, ..., v_n$ mitzuteilen, dass u seine 2-hop Nachbarschaft in Erfahrung bringen möchte.
2. Jeder Knoten $v_1, v_2, ..., v_n$ sendet wiederum eine Nachricht an seine 1-hop Nachbarn $w_1, w_2, ..., w_m$ und fordert diese zu einer Antwort auf.
3. Jeder Knoten $w_1, w_2, ..., w_m$ der ein 2-hop Nachbar von u ist sendet eine Antwort aus.
4. Die Knoten $v_1, v_2, ..., v_n$ erhalten die Antwort von $w_1, w_2, ..., w_m$ und informieren Knoten u über deren und die eigene Existenz.
5. Knoten u berechnet mit den empfangenen Informationen die 2-lokale Nachbarschaft.

Wenn n die Anzahl der 1-hop Nachbarn und m die Anzahl der 2-hop Nachbarn des Knotens u bestimmt, beträgt die mindestens benötigte Gesamtanzahl versendeter Nachrichten bei diesem Verfahren $1 + 2n + m$ Nachrichten. Folglich beträgt die Anzahl der Knoten, die während des Verfahrens eine Nachricht versendet haben $1 + n + m$.

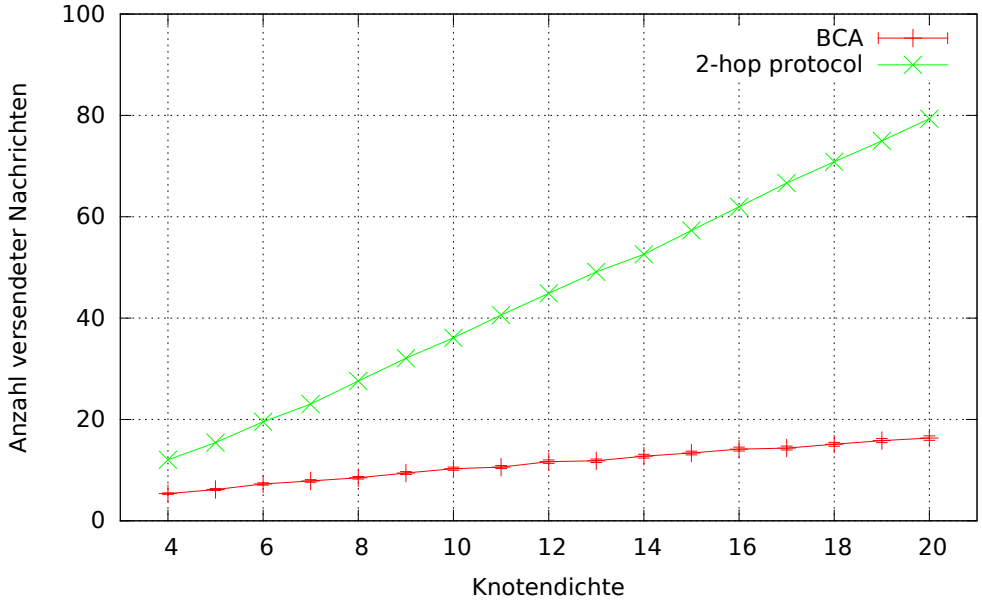
Anhand der auf diese Weise berechneten Anzahl beteiligter Knoten und versendeter Nachrichten im *2-hop protocol* wird im folgenden Abschnitt das Simulationsergebnis des BCA mit dem *2-hop protocol* verglichen.

6.2 Auswertung

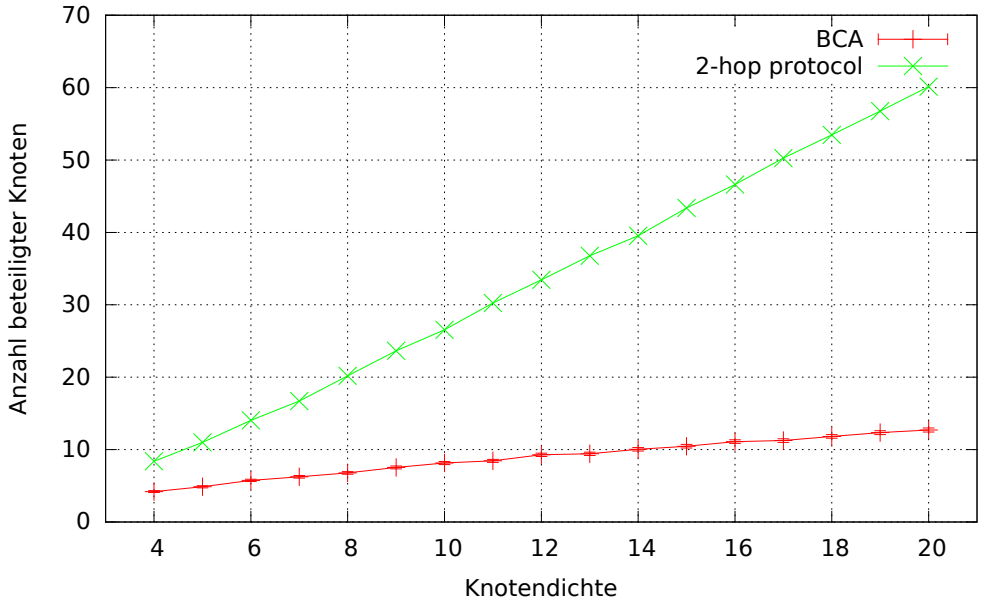
In diesem Abschnitt werden die Ergebnisse der Simulation präsentiert. Dazu wird der Beaconlose Clustering Algorithmus sowohl für Unit-Disk als auch für Quasi-Unit-Disk Graphen mit dem vorgestellten *2-hop protocol* zur Erkundung der vollen 2-hop Nachbarschaft verglichen. Alle im Folgenden beschriebenen Diagramme basieren auf den Daten der Simulationsergebnisse, die in Anhang B aufgeführt sind.

Abbildung 6.1 zeigt zwei Diagramme zur durchschnittlich benötigten Anzahl versendeter Nachrichten und beteiligter Knoten bei wachsender Knotendichte im BCA. Für 500 zufällig generierte Unit-Disk Graphen pro Knotendichte von 4 bis 20, ist in Diagramm 6.1(a) die durchschnittliche Anzahl versendeter Nachrichten mit einem Konfidenzintervall von 95 Prozent in der roten Kurve dargestellt. Die grüne Kurve des Diagramms zeigt die durchschnittliche Anzahl versendeter Nachrichten für das *2-hop protocol*, basierend auf der mittleren Anzahl von 1-hop und 2-hop Nachbarn der Simulationen pro Knotendichte. Diagramm 6.1(b) zeigt für beide Verfahren die entsprechende Anzahl beteiligter Knoten. Hinsichtlich der Anzahl versendeter Nachrichten ist dem oberen Diagramm in Abbildung 6.1 zu entnehmen, dass die Kurve des *2-hop protocols* kontinuierlich oberhalb der des BCA liegt und eine doppelt so große Steigung aufweist. Daraus resultiert, dass das *2-hop protocol* bei einer Knotendichte von vier in etwa die doppelte Menge an Nachrichten des entwickelten Verfahrens aufweist, bei einer Knotendichte von 12 bereits das Vierfache und bei einer Knotendichte von 20 das Fünffache. Bei der Betrachtung der Anzahl am Verfahren beteiligter Knoten in Diagramm 6.1(b) ist ebenfalls festzustellen, dass die Kurve des *2-hop protocols* durchgehend oberhalb der des BCA verläuft und eine doppelt so große Steigung besitzt. Beim *2-hop protocol* wird im Gegensatz zum BCA bei einer Knotendichte von vier im Schnitt die doppelte Menge an Knoten beteiligt, bei einer Knotendichte von 12 die drei- bis vierfache Menge und bei einer Knotendichte von 20 die vier- bis fünffache Menge. Beide Diagramme aus Abbildung 6.1 geben Aufschluss darüber, dass die Verwendung des entwickelten Verfahrens im Unit-Disk Graph eine eindeutige Ersparnis von versendeten Nachrichten und am Verfahren beteiligter Knoten gegenüber dem *2-hop protocol* mit sich bringt. Abbildung 6.2 soll nun zeigen, ob das selbige Ergebnis für Quasi-Unit-Disk Graphen eintritt.

Die zwei Diagramme in Abbildung 6.2 visualisieren die Anzahl versendeter

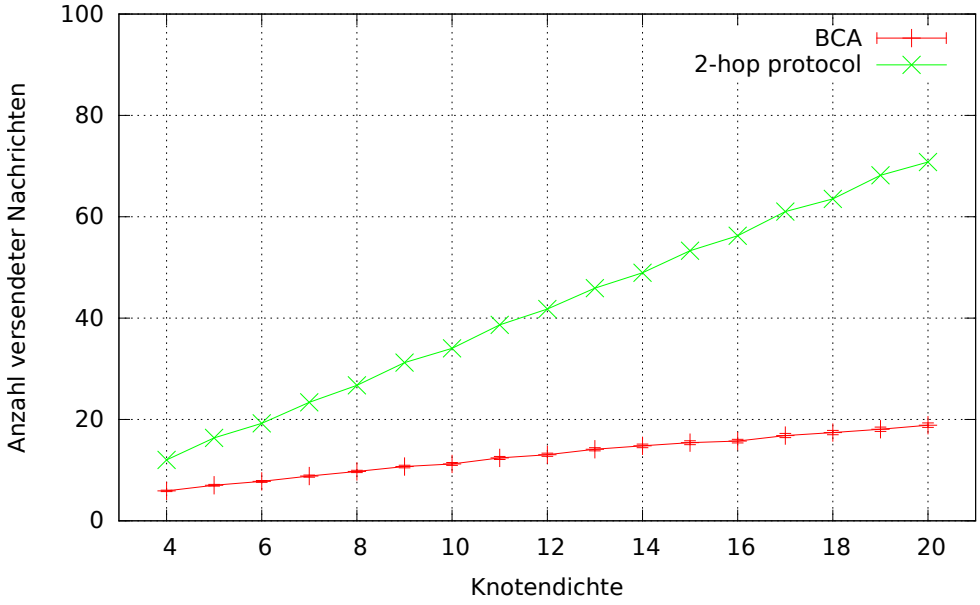


(a) Durchschnittliche Anzahl der versendeten Nachrichten bei zunehmender Knotendichte.

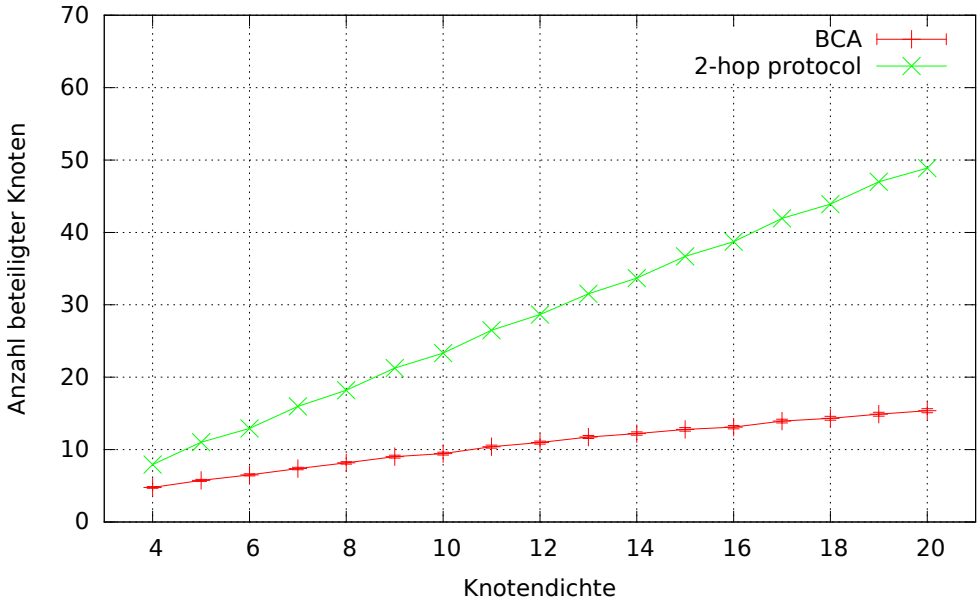


(b) Durchschnittliche Anzahl beteiligter Knoten bei zunehemender Knotendichte.

Abbildung 6.1: Auswertung der Simulation des BCA im Unit-Disk Graphen.



(a) Durchschnittliche Anzahl der versendeten Nachrichten bei zunehmender Knotendichte.



(b) Durchschnittliche Anzahl beteiligter Knoten bei zunehemender Knotendichte.

Abbildung 6.2: Auswertung der Simulation des BCA im Quasi-Unit-Disk Graphen.

Nachrichten und beteiligter Knoten im Quasi-Unit-Disk Graphen. Die roten Kurven zeigen die Ergebnisse des entwickelten BCA und basieren auf den Mittelwerten der Simulationsergebnisse und Konfidenzintervallen mit 95-prozentigem Konfidenzniveau. Das *2-hop protocol* wird durch die grünen Kurven repräsentiert, deren Werte auf den gemessenen Mittelwerten der Anzahl an 1-hop und 2-hop-Nachbarn und der in Abschnitt 6.1 vorgestellten Berechnung basieren. Sowohl die Betrachtung der versendeten Nachrichten in Diagramm 6.2(a) als auch die Untersuchung der Anzahl beteiligter Knoten in Diagramm 6.2(b) zeigen, dass der Vergleich der Anwendung des *2-hop protocols* und des entwickelten Verfahrens im Quasi-Unit-Disk Graphen die gleichen Erkenntnisse liefern, wie die Untersuchung im Unit-Disk Graph. Die Anzahl versendeter Nachrichten und benötigter Knoten des *2-hop protocols* sind kontinuierlich größer als die des BCA. Des Weiteren weisen die Kurven des *2-hop protocols* eine deutlich größere Steigung auf. Abbildung 6.2 bestätigt, dass die Verwendung des BCA gegenüber dem *2-hop protocol* auch im Quasi-Unit-Disk Graph eine Ersparnis mit sich bringt. Im Durchschnitt wird mindestens die Hälfte der versendeten Nachrichten und der beteiligten Knoten eingespart. Mit zunehmender Knotendichte nimmt die Anzahl ersparter Nachrichten und beteiligter Knoten stetig zu. Es ist zu vermuten, dass alle hier betrachteten Kurven des *2-hop protocols* mit einer wachsender Knotendichte > 20 unbeschränkt ansteigen, während sich die Kurven des entwickelten Verfahrens aufgrund dessen Konzeption einem Grenzwert annähern. Da Graphen mit einer Knotendichte größer als 20 aus dem Rahmen der durchgeführten Simulation fallen, kann diesbezüglich leider keine sichere Aussage getroffen werden.

Kapitel 7

Fazit und Ausblick

Im Rahmen dieser Arbeit wurde ein Beaconloser Clustering Algorithmus entwickelt, der sowohl im Unit-Disk Graph, als auch im Quasi-Unit-Disk Graph beaconlos eine lokale Sicht auf einen ausgedünnten Graphen bestehend aus konstant vielen Knoten pro Flächeneinheit bestimmt. Nach dem Beweis der Korrektheit des Algorithmus wurde der BCA analytisch untersucht, implementiert und in einer Simulationsanwendung beobachtet.

Ziel dieser Arbeit war es, einen Beitrag zur Entwicklung bisher unbekannter beaconloser Verfahren für geografisches Routing im Quasi-Unit-Disk Graphen zu leisten. Gelungen ist dies durch die Entwicklung des BCA, der die Möglichkeit bietet das bestehende nicht-beaconlose Routingverfahren aus [11] von Lillis et al. teilweise um die beaconlose Strategie zu erweitern und sogar hinsichtlich des verursachten Nachrichtenaufkommens und der geforderten Beteiligung im Netz deutlich zu verbessern. Simulativ wurde gezeigt, dass sich durch die Verwendung des entwickelten BCA im Durchschnitt mindestens die Hälfte der vom bestehenden Verfahren verursachten Nachrichten und beteiligten Knoten einsparen lässt, mit zunehmender Knotendichte sogar weitaus mehr. Die analytische Untersuchung deckte auf, dass sich pathologische Fälle konstruieren lassen für die der BCA zwar keine Verbesserung, allerdings auch keine Verschlechterung gegenüber dem bestehenden Verfahren aufweist.

Für zukünftige Arbeiten bietet das entwickelte Verfahren ein weites Spektrum offener Aufgaben. Während der BCA im Durchschnitt sehr gute Ergebnisse liefert, geben die aufgezeigten Sonderfälle einen Anhaltspunkt zur Weiterentwicklung des Algorithmus. Naheliegend ist beispielsweise die Implementierung der

vorgestellten Erweiterung zum Ausschluss von Knoten im Unit-Disk Graphen oder die Findung einer Möglichkeit weitere Knoten im Quasi-Unit-Disk Graphen auszuschließen. Auch für die Problematik des Nachrichtenoverheads im Worst-Case konnte in dieser Arbeit leider keine Lösung gefunden werden und bietet somit Potential zur Weiterentwicklung. Des Weiteren bleibt zu klären, ob sich Grenzwerte für die Anzahl beteiligter Knoten und versendeter Nachrichten offenbaren. Diesbezüglich konnten keine sicheren Aussagen getroffen werden, da die durchgeführte Simulation auf eine maximale Knotendichte von 20 beschränkt wurde. Eine weitere Interessante simulative Untersuchung stellt die Verwendung des hexagonalen Clusterings und der einhergehenden Vor- und Nachteile gegenüber dem gewählten Clustering mittels quadratischer Kacheln dar. Der BCA wurde in dieser Arbeit nur für statische Graphen betrachtet, obwohl die beaconlose Strategie grundsätzlich auch zur Verwendung in dynamischen Netzen mit mobilen Knoten konzipiert wurde. Es bleibt daher zu prüfen, ob der BCA ohne Weiteres für mobile Netze anwendbar ist.

Die wohl bedeutendste Aufgabe folgender Arbeiten liegt in der Umsetzung von geografischen Routingverfahren für Quasi-Unit-Disk Graphen, die nicht nur partiell, sondern gänzlich auf der beaconlosen Strategie basieren. Die Verwendung des BCA zur teils beaconlosen Erweiterung des Verfahrens von Lillis et al. bildet dabei eine Grundlage für dessen vollständige Umsetzung mittels beaconloser Strategie. Ein daraus resultierendes vollständig beaconloses Routingverfahren für Quasi-Unit-Disk Graphen mit minimalem Nachrichtenoverhead ist aufgrund der einhergehenden Energieeffizienz besonders erstrebenswert.

Literaturverzeichnis

[1] BARRIÈRE, L ; FRAIGNIAUD, P ; NARAYANAN, L ; OPATRNY, J: Robust Position-Based Routing in Wireless Ad Hoc Networks with Irregular Transmission Ranges. In: *Wireless Communications and Mobile Computing* 3 (2003), S. 141–153

[2] BENTER, M ; FREY, H ; NEUMANN, F: Reactive Planar Spanner Construction in Wireless Ad Hoc and Sensor Networks. In: *INFOCOM, Proceedings IEEE* (2013), S. 2193–2201

[3] BOSE, P ; MORIN, P ; STOJMENOVIĆ, I ; URRUTIA, J: Routing with Guaranteed Delivery in ad hoc Wireless Networks. In: *Wireless Networks* 7 (2001), Nr. 6, S. 609–616

[4] CHÁVEZ, E ; DOBREV, S ; KRANAKIS, E ; OPATRNY, J ; STACHO, L ; URRUTIA, J: Local Construction of Planar Spanners in Unit Disk Graphs with Irregular Transmission Ranges. In: *Proceedings of the 7th Latin American Conference on Theoretical Informatics* Bd. 3887. Berlin, Heidelberg : Springer-Verlag, 2006 (LATIN'06), S. 286–297

[5] FREY, H: *Geographisches Routing - Grundlagen und Basisalgorithmen*. Aachen : Shaker Verlag, 2005

[6] FREY, H ; GÖRGEN, D: Planar Graph Routing on Geographical Clusters. In: *Ad Hoc Networks* 3 (2005), Nr. 5, S. 560–574

[7] FREY, H ; GÖRGEN, D: Geographical Cluster-Based Routing in Sensing-Covered Networks. In: *IEEE Transactions On Parallel Distributed Systems* 17 (2006), Nr. 9, S. 899–911

[8] GUAN, X: Better Face Routing Protocols. In: DOLEV, S (Hrsg.): *Algorithmic Aspects of Wireless Sensor Networks* Bd. 5804. Berlin, Heidelberg : Springer-Verlag, 2009, S. 167–178

[9] KUHN, F ; WATTENHOFER, R ; ZHANG, Y ; ZOLLINGER, A: Geometric Ad-hoc Routing: Of Theory and Practice. In: *Proceedings of the Twenty-second Annual Symposium on Principles of Distributed Computing*. New York, NY, USA : ACM, 2003 (PODC '03), S. 63–72

[10] KUHN, F ; WATTENHOFER, R ; ZOLLINGER, A: Ad Hoc Networks Beyond Unit Disk Graphs. In: *Wireless Networks* 14 (2007), Nr. 5, S. 715–729

[11] LILLIS, K ; PEMMARAJU, S ; PIRWANI, I: Topology Control and Geographic Routing in Realistic Wireless Networks. In: KRANAKIS, E (Hrsg.) ; OPATRNY, J (Hrsg.): *Ad-Hoc & and Sensor Wireless Networks* Bd. 4686. Berlin, Heidelberg : Springer-Verlag, 2007, S. 15–31

[12] RÜHRUP, S ; KALOSHA, H ; NAYAK, A ; STOJMENOVIĆ, I: Message-Efficient Beaconless Georouting With Guaranteed Delivery in Wireless Sensor, Ad Hoc, and Actuator Networks. In: *IEEE/ACM Transactions on Networking* 18 (2010), Nr. 1, S. 95–108

[13] WANG, Y ; LI, X: Localized Construction of Bounded Degree and Planar Spanner for Wireless Ad Hoc Networks. In: *Mobile Networks and Applications* 11 (2006), Nr. 2, S. 161–175

Anhang A

Perl-Skript

Quellcode A.1 zeigt ein Perl-Skript, das zur Automatisierung der Simulation von Quasi-Unit-Disk Graphen erstellt wurde. Das Skript startet alle Simulationsdurchläufe und berechnet anschließend eine zusammenfassende Auswertung. Das tabellarisch dargestellte Ergebnis einer Ausführung des Skripts ist im Anhang B in Abbildung B.2 aufgeführt.

```
1  #!/usr/bin/perl
2  use Math::Trig;
3  use POSIX;
4
5  #$conModel = UDG;        #ConnectivityModel (UDG oder StaticQUDG)
6  $conModel = StaticQUDG; #ConnectivityModel (UDG oder StaticQUDG)
7  $nodeDensMin = 4;        #Beginne der Knotendichte
8  $nodeDensMax = 20;       #Ende der Knotendichte
9  $rounds = 500;           #Anzahl der auszuführenden Simulationen
10
11 $dimX = 354;             #Breite der Spielfläche
12 $dimY = 354;             #Höhe der Spielfläche
13 $udr = 100;              #Unit-Disk Radius bzw. Quasi-Unit-Disk Radius rMin
14 $qudrMin = $udr;         #Quasi Unit Disk Radius rMin
15 $qudrMax = 141;          #Quasi Unit Disk Radius rMax
16 $conProbability = 0.6;  #connectionProbability des QUDG-Modells
17 $tmax = 100;             #Größe des Timers tMax
18
19
20 #Schleife zur Ausführung der Simulationen pro Knotendichte
21 for ($nodeDens = $nodeDensMin; $nodeDens<=$nodeDensMax; $nodeDens+=1) {
22
23         #Berechnung Anzahl Knoten bei gegebener Knotendichte für UDG
24         #$numNodes=ceil(($nodeDens*$dimX*$dimY)/(pi*$udr**2));
25
26         #Berechnung Anzahl Knoten bei gegebener Knotendichte für QUDG
27         $numNodes = ceil( $nodeDens * ($dimX*$dimY) / (pi*$qudrMin**2 + ($conProbability*(pi*$qudrMax**2-
       pi*$qudrMin**2))) );
28
29         #Datei zum Speichern der Messwerte
30         $file="messwerte/$conModel\_$nodeDens.dat";
31
32         #Datei anlegen (wenn nicht vorhanden) und zum schreiben öffnen
33         open(FILE, ">$file");
```

```
34          print FILE
35                  "#file = $file\n",
36                  "\n",
37                  "#dimX = $dimX\n",
38                  "#dimY = $dimY\n",
39                  "#nodeDens = $nodeDens\n",
40                  "#numNodes = $numNodes\n",
41                  "#conModel = $conModel\n",
42                  "#udr = $udr\n",
43                  "#qudrMin = $qudrMin\n",
44                  "#qudrMax = $qudrMax\n",
45                  "#conProbability = $conProbability\n",
46                  "#tmax = $tmax\n",
47                  "#rounds = $rounds\n",
48                  "\n",
49                  "#time\t",
50                  "messages\t",
51                  "firstRequests\t",
52                  "firstResponses\t",
53                  "secondRequests\t",
54                  "secondResponses\t",
55                  "oneHopNeighbours\t",
56                  "twoHopNeighbours\t",
57                  "twoHopConnections\t",
58                  "clusterNeighbours\t",
59                  "clusterNeighboursConnections\t",
60                  "clusterNeighbourNeighbours\t",
61                  "activeNodes\t",
62                  "activeNodesDirect\t",
63                  "activeNodesIndirect\t",
64                  "bridgeNodes\t",
65                  "clusters\t",
66                  "clustersDirect\t",
67                  "clustersIndirect\t",
68                  "nodesDisabled\t",
69                  "nodesUseless\n";
70          #Datei schließen
71          close(FILE);
72
73          # Aufruf der Sinalgo-Ausführungen pro Knotendichte
74          for($i = 0; $i < $rounds; $i += 1) {
75                  system("java -cp binaries/bin sinalgo.Run " .
76                  "-project bca " .
77                  "-gen $numNodes bca:CNode Random C=$conModel " .
78                  "-batch " .
79                  "-overwrite " .
80                  "dimX=$dimX " .
81                  "dimY=$dimY " .
82                  "UDG/rMax=$udr " .
83                  "QUDG/rMin=$qudrMin " .
84                  "QUDG/rMax=$qudrMax " .
85                  "QUDG/connectionProbability=$conProbability " .
86                  "Timer/tMax=$tmax " .
87                  "Logging/file=$file"
88                  );
89          }
90
91          #Variablen zum Speichern der Mittelwertberechnung
92          my %sum = ();   #Summe
93          my %avg = ();   #Durchschnitt
94          my %var1 = ();  #Hilfsvariable Varianz
95          my %var2 = ();  #Hilfsvariable Varianz (s^2)
96          my %sd = ();    #Standardabweichung
97          my %kmin = ();  #Konfidenzintervall min
98          my %kmax = ();  #Konfidenzintervall max
99
100         # Einlesen der von Sinalgo beschriebenen Datei
```

```
101         open(DATA, "<$file");
102                 <DATA> for 1..15;
103                 while (my $zeile = <DATA>) {
104                         chomp($zeile);
105                         my @spalten = split (/\t/,$zeile);
106                         $sum{$_+1} = $sum{$_+1} + $spalten[$_] for 0..$#spalten;
107                 }
108         close(DATA);
109
110         # Berechnung des Mittelwerts
111         $avg{$_} = $sum{$_}/$rounds for 1..21;
112
113         #Berechnung der Varianz
114         open(DATA, "<$file");
115                 <DATA> for 1..15;
116                 while (my $zeile = <DATA>) {
117                         chomp($zeile);
118                         my @spalten = split (/\t/,$zeile);
119                         $var1{$_+1} += ($spalten[$_]-$avg{$_+1})**2 for 0..$#spalten;
120                 }
121         close(DATA);
122
123         #Berechnung der Konfidenzintervalle
124         $var2{$_} = $var1{$_}/($rounds-1) for 1..21;
125         $sd{$_} = 1.965*sqrt($var2{$_})/sqrt($rounds) for 1..21; #t=1,965 für 500 Runden
126         $kmin{$_} = $avg{$_}-$sd{$_} for 1..21; #t=1,965 für 500 Runden
127         $kmax{$_} = $avg{$_}+$sd{$_} for 1..21; #t=1,965 für 500 Runden
128
129         # Schreiben von Mittelwert und Konfidenzintervallen
130         open(DATA, ">>$file");
131                 print DATA "-\t" for 1..21;
132                 print DATA "\n";
133                 print DATA "$avg{$_}\t" for 1..21;
134                 print DATA "\t Average\n";
135                 print DATA "$sd{$_}\t" for 1..21;
136                 print DATA "\t Mittelwertabweichung (t*s/sqrt(n))\n";
137         close(DATA);
138
139
140         # Anlegen einer Datei für die Zusammenfassung der Messwerte
141         $fileSum = "messwerte/$conModel\_from_$nodeDensMin\_to_$nodeDensMax.dat";
142         unless (-e $fileSum) {
143                 open(DATA, ">$fileSum");
144                 print DATA
145                         "#file = $fileSum\n",
146                         "\n",
147                         "#dimX = $dimX\n",
148                         "#dimY = $dimY\n",
149                         "#nodeDensMin = $nodeDensMin\n",
150                         "#nodeDensMax = $nodeDensMax\n",
151                         "#conModel = $conModel\n",
152                         "#udr = $udr\n",
153                         "#qudrMin = $qudrMin\n",
154                         "#qudrMax = $qudrMax\n",
155                         "#conProbability = $conProbability\n",
156                         "#tmax = $tmax\n",
157                         "#rounds per nodeDens = $rounds\n",
158                         "\n",
159                         "#nodeDens\t",
160                         "numNodes\t",
161                         "avg(time)\t",                          "sd(time)\t",
162                         "avg(messages)\t",                      "sd(messages)\t",
163                         "avg(firstRequests)\t",                 "sd(firstRequests)\t",
164                         "avg(firstResponses)\t",                "sd(firstResponses)\t",
165                         "avg(secondRequests)\t",                "sd(secondRequests)\t",
166                         "avg(secondResponses)\t",               "sd(secondResponses)\t",
167                         "avg(oneHopNeighbours)\t",              "sd(oneHopNeighbours)\t",
```

```
168                         "avg(twoHopNeighbours)\t",              "sd(twoHopNeighbours)\t",
169                         "avg(twoHopConnections)\t",             "sd(twoHopConnections)\t",
170                         "avg(clusterNeighbours)\t",             "sd(clusterNeighbours)\t",
171                         "avg(clusterNeighboursConnections)\t",  "sd(clusterNeighboursConnections)\t",
172                         "avg(clusterNeighbourNeighbours)\t",    "sd(clusterNeighbourNeighbours)\t",
173                         "avg(activeNodes)\t",                   "sd(activeNodes)\t",
174                         "avg(activeNodesDirect)\t",             "sd(activeNodesDirect)\t",
175                         "avg(activeNodesIndirect)\t",           "sd(activeNodesIndirect)\t",
176                         "avg(bridgeNodes)\t",                   "sd(bridgeNodes)\t",
177                         "avg(clusters)\t",                      "sd(clusters)\t",
178                         "avg(clustersDirect)\t",                "sd(clustersDirect)\t",
179                         "avg(clustersIndirect)\t",              "sd(clustersIndirect)\t",
180                         "avg(nodesDisabled)\t",                 "sd(nodesDisabled)\t",
181                         "avg(nodesUseless)\t",                  "sd(nodesUseless)\n",
182                         "\n";
183                 close(DATA);
184         }
185
186         # Schreiben der berechneten Werte
187         open(DATA, ">>$fileSum");
188                 print DATA "$nodeDens\t$numNodes\t";
189                 print DATA "$avg{$_}\t$sd{$_}\t" for 1..21;
190                 print DATA "\n";
191         close(DATA);
192 }
```

Quellcode A.1: Perl-Skript zur Simulation von Quasi-Unit-Disk Graphen.

Anhang B

Simulationsergebnisse

In den zwei Abbildungen B.1 und B.2 sind zusammengefasste Messergebnisse der Simulationsdurchführung tabellarisch aufgeführt. Die Rohdaten der durchgeführten Simulationen sind auf dem dieser Arbeit beiliegenden Datenträger zu finden. Die Messwerte wurden durch die Ausführung des in Anhang 6.1 gezeigten Skripts erhoben. Dazu wurde das Skript jeweils einmal für Unit-Disk Graphen und Quasi-Unit-Disk Graphen ausgeführt. Neben Mittelwerten ($\varnothing$), zugehörigen Konfidenzintervallen (+/-) und Extremwerten (*min*,*max*) der Merkmale *c* bis *t* beinhalten die dargestellten Simulationsmessdaten die Skriptkonfiguration und eine kurze Erläuterung der Merkmale *a* bis *v*. Es ist anzumerken, dass es sich bei den Merkmalen *u* und *v* um berechnete Größen eines Referenzverfahrens aus Abschnitt 6.2 handelt. Alle übrigen Messwerte stammen aus der Simulation.

#file = messwerte/UDG_from_4_to_20.dat;

	#dimX = 354;	a	Knotendichte	l	Minimum aktive Knoten aller Simulationen															
	#dimY = 354;	b	Gesamtzahl Knoten	m	Maximum aktive Knoten aller Simulationen															
	#nodeDensMin = 4;	c	Zeit bzw. Runden aus Sinalgo	n	aktive Knoten aus der 1-hop Nachbarschaft															
	#nodeDensMax = 20;	d	versendete Nachrichten gesamt	o	aktive Knoten aus der 2-hop Nachbarschaft															
	#conModel = UDG;	e	versendete FirstRequestMessages	p	Brückenknoten															
	#udr = 100;	f	versendete FirstResponseMessages	q	Kacheln mit aktivem Knoten															
	#qudrMin = 100;	g	versendete SecondRequestMessages	r	1-hop-Nachbarn aus der Startkachel, die nie eine Nachricht versendet haben															
	#qudrMax = 141;	h	versendete SecondResponseMessages	s	1-hop Nachbarn, die keine Antwort auf ihren FirstRequest erhalten															
	#conProbability = 0.6;	i	1-hop-Nachbarn des initialen Knotens	t	Knoten die während des Verfahrens mindestens eine Nachricht versendet haben															
	#tmax = 100;	j	2-hop-Nachbarn des initialen Knotens	u	Nachrichten für volle 2-hop Nachbarschaftserkundung															
	#rounds per nodeDens = 500;	k	aktive Knoten	v	beteiligte Knoten zur 2-hop Nachbarschaftserkundung															
a	nodeDens	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		
b	numNodes	16	20	24	28	32	36	40	44	48	52	56	60	64	68	72	76	80		
c	ø time	322,68	329,02	335,27	345,46	345,03	358,65	368,58	366,75	390,43	387,93	405,22	408,17	431,34	426,80	447,09	464,85	478,59		
	+/- time	3,10	4,34	4,64	5,50	5,47	6,87	7,75	7,34	9,80	8,91	9,73	10,12	11,14	10,85	11,40	avg(	13,78		
d	ø messages	5,37	6,17	7,28	7,88	8,50	9,44	10,27	10,61	11,68	11,84	12,75	13,38	14,16	14,36	15,10	15,84	16,33		
	+/- messages	0,15	0,18	0,22	0,25	0,26	0,28	0,31	0,31	0,34	0,34	0,37	0,38	0,39	0,38	0,38	0,42	0,44		
e	ø firstRequests	1,33	1,44	1,56	1,69	1,68	1,89	2,00	2,00	2,28	2,27	2,45	2,48	2,78	2,70	2,92	3,18	3,29		
	+/- firstRequests	0,05	0,06	0,07	0,08	0,08	0,09	0,10	0,09	0,12	0,11	0,11	0,11	0,13	0,12	0,12	0,14	0,15		
f	ø firstResponses	2,89	3,46	4,23	4,61	5,19	5,72	6,28	6,60	7,16	7,33	7,80	8,22	8,56	8,83	9,16	9,43	9,68		
	+/- firstResponses	0,11	0,13	0,13	0,14	0,14	0,14	0,15	0,15	0,16	0,16	0,17	0,17	0,17	0,17	0,17	0,18	0,18		
g	ø secondRequests	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		
	+/- secondRequests	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
h	ø secondResponses	0,15	0,27	0,49	0,58	0,64	0,83	0,99	1,01	1,24	1,24	1,50	1,68	1,82	1,83	2,02	2,23	2,36		
	+/- secondResponses	0,04	0,05	0,08	0,09	0,09	0,11	0,12	0,12	0,13	0,13	0,14	0,14	0,15	0,15	0,15	0,16	0,17		
i	ø oneHopNeighbours	3,64	4,43	5,51	6,35	7,39	8,41	9,57	10,37	11,46	12,36	13,05	13,95	15,34	16,39	17,37	18,21	19,22		
	+/- oneHopNeighbours	0,15	0,17	0,18	0,20	0,22	0,22	0,25	0,24	0,27	0,28	0,27	0,27	0,30	0,31	0,31	0,33	0,33		
j	ø twoHopNeighbours	3,75	5,56	7,53	9,36	11,80	14,24	15,96	18,87	20,99	23,41	25,49	28,42	30,27	32,88	35,10	37,55	39,92		
	+/- twoHopNeighbours	0,16	0,20	0,22	0,27	0,27	0,30	0,30	0,33	0,33	0,34	0,38	0,37	0,36	0,37	0,40	0,41	0,44		
k	ø activeNodes	3,99	4,63	5,51	5,93	6,51	7,15	7,73	8,02	8,73	8,86	9,41	9,88	10,28	10,53	10,99	11,32	11,63		
	+/- activeNodes	0,12	0,14	0,15	0,17	0,17	0,18	0,19	0,19	0,20	0,20	0,21	0,21	0,22	0,21	0,21	0,23	0,23		
l	min activeNodes	1	1	1	1	3	1	3	3	4	3	4	4	5	5	5	6	5		
m	max activeNodes	8	9	12	11	14	15	16	16	17	18	16	19	17	17	19	19	19		
n	ø activeNodesOneHop	2,71	3,16	3,71	3,98	4,48	4,83	5,19	5,46	5,78	5,94	6,11	6,34	6,54	6,73	6,92	7,00	7,11		
	+/- activeNodesDirect	0,11	0,12	0,12	0,12	0,12	0,11	0,12	0,12	0,11	0,11	0,12	0,11	0,11	0,11	0,11	0,11	0,11		
o	ø activeNodesTwoHop	0,15	0,27	0,48	0,57	0,62	0,82	0,96	0,99	1,22	1,21	1,48	1,66	1,78	1,82	1,98	2,16	2,31		
	+/- activeNodesIndirect	0,04	0,05	0,07	0,08	0,09	0,10	0,11	0,12	0,13	0,12	0,14	0,14	0,15	0,15	0,14	0,15	0,16		
p	ø bridgeNodes	0,12	0,20	0,32	0,38	0,40	0,51	0,59	0,57	0,72	0,72	0,83	0,89	0,96	0,98	1,09	1,16	1,22		
	+/- bridgeNodes	0,03	0,04	0,05	0,05	0,05	0,06	0,06	0,06	0,07	0,07	0,07	0,07	0,07	0,07	0,07	0,08	0,08		
q	ø clusters	3,86	4,43	5,19	5,55	6,11	6,64	7,14	7,45	8,00	8,15	8,59	8,99	9,33	9,55	9,90	10,16	10,42		
	+/- clusters	0,11	0,12	0,13	0,14	0,13	0,14	0,15	0,15	0,15	0,15	0,16	0,16	0,16	0,16	0,16	0,17	0,17		
r	ø nodesDisabled	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,01	0,01	0,01	0,01	0,03	0,03		
	+/- nodesDisabled	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,01	0,01	0,01	0,01	0,02	0,02	0,02		
s	ø nodesUseless	0,21	0,24	0,24	0,31	0,28	0,38	0,41	0,43	0,55	0,55	0,62	0,59	0,82	0,72	0,83	1,01	1,07		
	+/- nodesUseless	0,04	0,05	0,05	0,05	0,06	0,06	0,06	0,06	0,08	0,08	0,07	0,08	0,09	0,08	0,09	0,11	0,11		
t	ø usedNodes	4,20	4,87	5,75	6,24	6,79	7,53	8,15	8,45	9,28	9,42	10,04	10,47	11,10	11,25	11,82	12,34	12,70		
	+/- usedNodes	0,13	0,15	0,16	0,18	0,18	0,20	0,21	0,21	0,23	0,23	0,24	0,24	0,25	0,25	0,25	0,27	0,28		
u	2-hop protocol – messages	12,04	15,42	19,55	23,05	27,59	32,06	36,11	40,61	44,90	49,12	52,58	57,31	61,95	66,66	70,84	74,96	79,36		
v	2-hop protocol – usedNodes	8,39	10,99	14,04	16,70	20,19	23,65	26,54	30,24	33,44	36,76	39,54	43,37	46,61	50,27	53,47	56,75	60,14		

Abbildung B.1: Messdaten der Simulation von Unit-Disk Graphen.

#file = messwerte/StaticQUDG_from_4_to_20.dat;

#dimX = 354;	a	Knotendichte	l	Minimum aktive Knoten aller Simulationen																
#dimY = 354;	b	Gesamtzahl Knoten	m	Maximum aktive Knoten aller Simulationen																
#nodeDensMin = 4;	c	Zeit bzw. Runden aus Sinalgo	n	aktive Knoten aus der 1-hop Nachbarschaft																
#nodeDensMax = 20;	d	versendete Nachrichten gesamt	o	aktive Knoten aus der 2-hop Nachbarschaft																
#conModel = StaticQUDG;	e	versendete FirstRequestMessages	p	Brückenknoten																
#udr = 100;	f	versendete FirstResponseMessages	q	Kacheln mit aktivem Knoten																
#qudrMin = 100;	g	versendete SecondRequestMessages	r	1-hop-Nachbarn aus der Starkachel, die nie eine Nachricht versendet haben																
#qudrMax = 141;	h	versendete SecondResponseMessages	s	1-hop Nachbarn, die keine Antwort auf ihren FirstRequest erhalten																
#conProbability = 0.6;	i	1-hop-Nachbarn des initialen Knotens	t	Knoten die während des Verfahrens mindestens eine Nachricht versendet haben																
#tmax = 100;	j	2-hop-Nachbarn des initialen Knotens	u	Nachrichten für volle 2-hop Nachbarschaftserkundung																
#rounds per nodeDens = 500;	k	aktive Knoten	v	beteiligte Knoten zur 2-hop Nachbarschaftserkundung																

	nodeDens	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
b	numNodes	10	13	15	18	20	23	25	28	30	33	35	38	40	43	45	48	50
c	ø time	314,38	316,24	320,91	322,75	327,37	327,97	331,52	336,88	341,59	345,25	350,73	355,31	361,29	365,73	378,42	384,66	386,97
	+/- time	1,72	1,86	2,87	2,96	4,10	3,55	4,29	4,67	5,30	5,44	5,93	6,41	7,17	7,25	8,54	8,74	8,61
d	ø messages	5,91	7,01	7,78	8,83	9,74	10,71	11,22	12,40	13,04	14,12	14,79	15,42	15,73	16,83	17,41	18,07	18,87
	+/- messages	0,14	0,17	0,19	0,22	0,23	0,26	0,28	0,30	0,31	0,32	0,36	0,39	0,36	0,41	0,42	0,43	0,45
e	ø firstRequests	1,18	1,23	1,31	1,38	1,43	1,46	1,51	1,63	1,68	1,78	1,83	1,86	1,94	2,01	2,16	2,22	2,30
	+/- firstRequests	0,04	0,04	0,05	0,05	0,06	0,06	0,07	0,07	0,08	0,08	0,08	0,09	0,09	0,09	0,11	0,10	0,11
f	ø firstResponses	3,61	4,57	5,24	6,05	6,83	7,68	8,08	8,89	9,43	10,15	10,59	11,16	11,40	12,20	12,44	12,99	13,43
	+/- firstResponses	0,11	0,13	0,14	0,15	0,14	0,16	0,17	0,17	0,18	0,17	0,19	0,20	0,18	0,20	0,19	0,21	0,21
g	ø secondRequests	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	+/- secondRequests	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
h	ø secondResponses	0,12	0,21	0,23	0,40	0,48	0,57	0,63	0,88	0,92	1,19	1,37	1,40	1,39	1,62	1,81	1,86	2,14
	+/- secondResponses	0,04	0,05	0,06	0,07	0,08	0,09	0,10	0,11	0,12	0,13	0,15	0,16	0,15	0,16	0,18	0,18	0,19
i	ø oneHopNeighbours	4,08	5,36	6,32	7,43	8,52	9,98	10,69	12,18	13,14	14,40	15,28	16,61	17,55	19,08	19,61	21,21	21,93
	+/- oneHopNeighbours	0,13	0,15	0,17	0,18	0,18	0,21	0,22	0,22	0,22	0,24	0,26	0,27	0,27	0,28	0,26	0,32	0,32
j	ø twoHopNeighbours	2,85	4,66	5,61	7,54	8,68	10,26	11,65	13,31	14,52	16,11	17,39	19,08	20,16	21,86	23,31	24,78	25,95
	+/- twoHopNeighbours	0,11	0,13	0,15	0,17	0,17	0,20	0,19	0,22	0,20	0,23	0,23	0,25	0,26	0,27	0,26	0,30	0,31
k	ø activeNodes	4,69	5,66	6,35	7,26	8,01	8,89	9,26	10,22	10,77	11,53	12,02	12,57	12,83	13,66	13,96	14,46	15,01
	+/- activeNodes	0,12	0,14	0,15	0,16	0,16	0,19	0,19	0,19	0,20	0,20	0,22	0,24	0,22	0,24	0,24	0,25	0,25
l	min activeNodes	1	2	2	2	3	3	4	5	5	6	6	6	7	8	8	8	8
m	max activeNodes	8	11	11	13	13	16	16	16	19	18	18	21	22	22	23	22	24
n	ø activeNodesOneHop	3,48	4,32	4,97	5,60	6,27	6,99	7,32	7,90	8,38	8,78	9,04	9,56	9,78	10,34	10,39	10,85	10,99
	+/- activeNodesOneHop	0,11	0,12	0,13	0,14	0,13	0,14	0,15	0,14	0,15	0,14	0,15	0,15	0,14	0,14	0,14	0,15	0,14
o	ø activeNodesDirect	0,12	0,20	0,23	0,40	0,47	0,57	0,62	0,87	0,91	1,17	1,35	1,37	1,38	1,59	1,78	1,83	2,08
	+/- activeNodesDirect	0,04	0,05	0,06	0,07	0,08	0,09	0,10	0,11	0,12	0,13	0,15	0,15	0,15	0,16	0,17	0,18	0,18
p	ø activeNodesIndirect	0,09	0,14	0,15	0,26	0,27	0,33	0,32	0,45	0,49	0,58	0,63	0,64	0,67	0,73	0,79	0,79	0,94
	+/- activeNodesIndirect	0,03	0,03	0,03	0,04	0,04	0,05	0,05	0,05	0,06	0,06	0,06	0,07	0,07	0,07	0,07	0,07	0,07
q	ø bridgeNodes	4,60	5,52	6,20	7,00	7,74	8,56	8,94	9,77	10,29	10,95	11,39	11,93	12,16	12,93	13,17	13,68	14,08
	+/- bridgeNodes	0,11	0,12	0,14	0,14	0,14	0,16	0,17	0,16	0,17	0,17	0,18	0,19	0,17	0,20	0,19	0,20	0,20
r	ø clusters	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	+/- clusters	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
s	ø nodesDisabled	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
	+/- nodesDisabled	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
t	ø nodesUseless	0,09	0,10	0,16	0,12	0,16	0,14	0,19	0,19	0,20	0,21	0,20	0,22	0,27	0,28	0,36	0,43	0,36
	+/- nodesUseless	0,03	0,03	0,04	0,03	0,04	0,03	0,04	0,04	0,04	0,04	0,04	0,05	0,05	0,05	0,06	0,06	0,06
	ø usedNodes	4,78	5,75	6,51	7,38	8,17	9,02	9,45	10,41	10,97	11,74	12,22	12,79	13,10	13,94	14,32	14,89	15,37
	+/- usedNodes	0,12	0,14	0,15	0,17	0,17	0,20	0,20	0,20	0,21	0,21	0,23	0,25	0,23	0,26	0,27	0,27	0,28
u	2-hop protocol – messages	12,02	16,37	19,24	23,39	26,72	31,23	34,02	38,67	41,80	45,91	48,96	53,30	56,26	61,02	63,52	68,19	70,80
v	2-hop protocol – usedNodes	7,93	11,01	12,93	15,97	18,20	21,25	23,33	26,49	28,66	31,51	33,68	36,69	38,71	41,94	43,91	46,99	48,88

Abbildung B.2: Messdaten der Simulation von Quasi-Unit-Disk Graphen.