Masterarbeit

Algorithm-Engineering für ein geometrisches Set-Cover-Problem

Dominique Bau

Abgabedatum: 1. März 2021
Betreuer: Prof. Dr. Alexander Wolff
Johannes Zink

Julius-Maximilians-Universität Würzburg
Lehrstuhl für Informatik I
Algorithmen und Komplexität

Zusammenfassung

In dieser Arbeit haben wir einen Local-Search-Algorithmus, welcher das geometrische Set-Cover-Problem für Kreise approximiert, dahingehend verändert, dass ein allgemeineres neu definiertes Problem mit dem Namen *Voronoi-Cover-Problem* approximiert werden kann. Das Voronoi-Cover-Problem unterscheidet sich nur geringfügig vom geometrischen Set-Cover-Problem für Kreise. Für das geometrische Set-Cover-Problem mit Kreisen sei $\mathcal{S}$ die Menge der Kreise und U eine Menge aus Punkten in der Ebene. Ziel ist es, eine Menge $T \subseteq \mathcal{S}$ mit minimaler Kardinalität zu finden, sodass die Vereinigung der Kreise aus T das gesamte Universum U beinhaltet. Für das Voronoi-Cover-Problem wird zusätzlich benötigt, dass die Voronoi-Zellen eines Voronoi-Diagrams über den einzelnen Kreismittelpunkten aus T nur eine bestimmte Anzahl an Elementen enthalten dürfen. Die Voronoi-Zelle eines Kreismittelpunktes enthält die Punkte, welche näher an diesem Kreismittelpunkt als an allen anderen Kreismittelpunkten liegen. Dieses Problem taucht oft in der Netzwerktechnik auf, bei der Gateways nur eine bestimmte Anzahl an Sensoren versorgen können und Sensoren möglichst nur bis zu dem nächsten Gateway senden sollen, um Störsignale bei anderen Gateways zu verhindern. Für den von uns entwickelten *Voronoi-Local-Search-Algorithmus* wurden zudem Beschleunigungstechniken ausgearbeitet und anhand eines praxisbezogenen Beispiels (jedes Gebäude in Würzburg wird als ein Sensor modelliert) getestet. Außerdem wird der Voronoi-Local-Search-Algorithmus mit Algorithmen für das geometrische Set-Cover-Problem und einem ganzzahligen linearen Programm für das Voronoi-Cover-Problem in Laufzeit und Lösungsgüte für verschiedene Eingabegrößen und Sendereichweiten der Gateways/Sensoren verglichen.

Inhaltsverzeichnis

1 Einleitung

Um sowohl in Städten als auch in ländlichen Gebieten kleine Datenpakete von wenigen Bytes über eine große Strecke mit wenig Energie senden zu können, bietet sich die *LoRaWAN*-Technologie (Long Range Wide Area Network)[1] an. Mithilfe dieser Technologie können kleinste Geräte mit sehr geringem Energieaufwand über weite Strecken senden. Meist gibt es dabei Sender, welche Daten an Knotenpunkte senden. Die Knotenpunkte können diese Daten dann über das Internet an die gewünschte Stelle zur Verarbeitung übermitteln. Sobald ein solches Netzwerk besteht, können beispielsweise der Wasserstand eines Flusses, der Funktionsstatus einer Straßenlaterne oder viele andere kleine Daten täglich oder auch stündlich abgefragt werden. Somit wäre es für eine Stadt möglich, automatisch Hochwassermaßnahmen einzuleiten oder auch zeitnah zu erkennen, dass eine Straßenlaterne repariert werden muss. Um ein Netzwerk dieser Art aufzubauen, muss jedoch zunächst über die Platzierung der Geräte beraten werden. Trotz einer Reichweite von bis zu 2km in Stadtgebieten reicht es nicht, nur ein solches Gerät aufzustellen. Gleichzeitig ist es wünschenswert, so wenige Geräte wie möglich zu postieren. Diese Platzierung der Geräte ist in der Theorie – wie sich später in dieser Arbeit herausstellt – ein NP-vollständiges Problem. Zunächst werden das Problem und weitere Feinheiten dessen genauer beschrieben.

1.1 Problemstellung aus der Praxis

Die vorliegende Problemstellung kommt, wie bereits angedeutet, aus der Netzwerktechnik. Dabei gibt es *Gateways*, welche Zugang zu einem Netzwerk bieten, und *Sensoren*, welche Daten über das Netzwerk übertragen sollen. Befinden sich ein Gateway und ein Sensor nah genug beieinander, können sie miteinander kommunizieren, wodurch der Sensor seine Daten in das Netzwerk speisen kann, ohne direkt mit dem Netzwerk verbunden zu sein. Die Platzierung der Sensoren wird dabei als gegeben angenommen. Zusätzlich ist die maximale Sendereichweite der Sensoren und Gateways gegeben. Da für diese Arbeit zunächst die Bebauung der Fläche und die geographischen Gegebenheiten vernachlässigt werden, können alle Sensoren und Gateways in alle Richtungen gleich weit senden. Daher haben alle Sensoren und Gateways dieselbe maximale Sendereichweite.

Ziel des Algorithmus ist, die Anzahl der Gateways zu minimieren und die Platzierung dieser Gateways zu bestimmen, sodass alle Sensoren in Sendereichweite zu mindestens einem Gateway liegen und somit Anschluss an das Netzwerk haben.

Ein solches Gateway kann nicht unendlich viele Sensoren versorgen, weswegen es wünschenswert ist, die Anzahl der Sensoren für jedes Gateway beschränken zu können.

[1] https://www.lora-wan.de

Sobald ein Sensor und ein Gateway in Reichweite zueinander liegen, kann der Sensor durch Anpassung einiger Variablen seine Sendereichweite bis zu einem gewissen Grad einschränken. Das kann sinnvoll sein, denn wenn zwei Sensoren auf derselben Frequenz zu einem Gateway senden, kann das Gateway möglicherweise keine der beiden Nachrichten entschlüsseln. Schränkt man demnach die Sendeleistung eines Sensors soweit ein, dass eine Nachricht nur das nächste Gateway und nicht noch weitere Gateways erreicht, können einige Störsignale verhindert werden. Im weiteren Verlauf dieser Arbeit wird auch von einer Zuweisung beziehungsweise Zuordnung des Sensors zu einem Gateway gesprochen, wenn der Sensor seine Sendereichweite auf dieses Gateway beschränkt. Damit eine solche Zuordnung die Störsignale möglichst minimiert, kann ein Sensor nur dem nächsten Gateway zugeordnet werden.

Wenn ein Gateway beliebig viele Sensoren versorgen könnte, könnte jeder Sensor ohne Probleme dem jeweils nächsten Gateway zugeordnet werden. Zusammen mit der Einschränkung der maximal zugeordneten Sensoren pro Gateway bedarf es jedoch möglicherweise einer anderen Platzierung der Gateways oder sogar weiteren Gateways, sodass jeder Sensor dem nächsten Gateway zugeordnet werden kann.

Um keine unendlich große Menge für mögliche Gateway-Standorte verarbeiten zu müssen, und da Gateways auch in der Realität nicht überall aufgestellt werden können, wird als Eingabe zusätzlichen zu den Sensoren eine endliche Menge an möglichen Gateway-Standorten erwartet.

1.2 Literaturüberblick

Zur Platzierung von Geräten innerhalb eines Funk-Netzwerks gibt es einige bestehende Arbeiten. Eine Herangehensweise an das Problem ist, die zu optimierenden Werte in Funktionen zu modellieren und diese mithilfe verschiedener Heuristiken zu minimieren (beziehungsweise zu maximieren) [LGS+15, LGZ+15]. Eine Funktion kann dabei beispielsweise den Abstand der Sensoren zu Gateways oder die Anzahl der Sensoren, die ein Gateway abdeckt, modellieren. Wird dabei angenommen, dass die Anzahl der Gateways vorher feststeht, kann ein k-Means oder k-Medoids Ansatz zur Platzierung der Gateways gewählt werden [LGS+15].

Eine andere Herangehensweise, welche auch in dieser Arbeit gewählt wurde, ist die Modellierung des Problems als Set-Cover-Problem (oder auch Hitting-Set-Problem)[Vaz13]. Für die Lösung einer Instanz des Set-Cover-Problems bestehen einige Ansätze in der Literatur. Für eine optimale Lösung kann das Problem als ganzzahliges lineares Programm formuliert werden. Da das Set-Cover-Problem jedoch bekannterweise NP-vollständig ist [Kar72], wurden für größere Instanzen Approximationsalgorithmen entwickelt. Dabei gibt es unter anderem ε-Netz-Algorithmen [AP20, BMR18], Greedy-Ansätze [JMS02, JMM+03] und auch Local-Search-Algorithmen [MR10]. Die ε-Netz-Algorithmen und der Local-Search-Algorithmus werden im Folgenden noch genauer behandelt. Ein Greedy-Ansatz würde immer die Menge in das Set-Cover aufnehmen, welche am meisten nicht-abgedeckte Elemente des Universums abdeckt. Da dieser Ansatz von Grund auf schon gegen die Limitierung der maximalen Sensoren-Anzahl pro Gateway vorgeht, wird

dieser im Folgenden nicht weiter behandelt.

1.3 Gliederung der Arbeit

Um die beschriebenen Limitierungen für Gateways und Sensoren einzuhalten, wird in Kapitel 3.4 ein bestehender Local-Search-Algorithmus [MR10] dahingehend verändert. In Kapitel 2 werden die benötigten Grundlagen definiert und beschrieben. Dazu gehört das Set-Cover-Problem und eine geometrische Variante dessen. Außerdem werden ε-Netz-Algorithmen, der darauffolgend veränderte Local-Search-Algorithmus und das ILP des Set-Cover-Problems beschrieben. In Kapitel 3 wird das geometrische Set-Cover-Problem und der Local-Search-Algorithmus schrittweise an die Problemstellung aus Kapitel 1.1 angepasst. Dadurch entsteht die neue Version des Problems namens *Voronoi-Cover-Problem* und auch eine neue Version des Algorithmus namens *Voronoi-Local-Search*. Zusätzlich wird das ILP des Set-Cover-Problems für die zusätzlichen Einschränkungen umformuliert. Kapitel 4 beinhaltet auf den Voronoi-Local-Search-Algorithmus angewandte Beschleunigungstechniken, um die Laufzeit des Algorithmus in der Praxis möglichst klein zu halten. In Kapitel 5 werden Tests für Vergleiche der Laufzeit und der Güte der Lösung des Voronoi-Local-Search-Algorithmus und anderer Algorithmen analysiert. Außerdem wird beschrieben, wie die Instanzen für diese Tests generiert wurden. Kapitel 6 fasst die erarbeiteten Erkenntnisse über den Voronoi-Local-Search-Algorithmus zusammen. Ein Ausblick auf mögliche Erweiterungen und weitere Tests bietet Kapitel 7.

2 Grundlagen

Damit geeignete Algorithmen aus der Theorie für das Problem benutzt werden können, wird zunächst die praktische Aufgabenstellung als abstraktes Problem modelliert. Um die Erreichbarkeit von Sensoren und Gateways darstellen zu können, werden Gateways als Kreisscheiben und Sensoren als Punkte dargestellt. Der Radius der Kreise ist dabei die maximale Sendereichweite zwischen Sensoren und Gateways. Dabei gilt: Liegt ein Sensor innerhalb des Kreises eines Gateways, können Gateway und Sensor miteinander kommunizieren.

Zunächst schauen wir uns jedoch die grundlegenden bekannten Probleme namens Set-Cover-Problem und Hitting-Set-Problem an, welche ohne geometrische Objekte auskommen.

2.1 Das Set-Cover- und Hitting-Set-Problem

Insbesondere aus der Literatur zur Algorithmik und Komplexität ist das *Set-Cover-Problem* beziehungsweise *Mengenüberdeckungs-Problem* bekannt.

Set-Cover-Problem	
Eingabe:	Eine Menge U und eine Menge $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$, die aus Teilmengen von U besteht. Dabei gilt $\bigcup_{S_i \in \mathcal{S}} S_i = U$.
Fragestellung:	Welche ist eine kleinste Menge $T \subseteq \mathcal{S}$, sodass $\bigcup_{S_i \in T} S_i = U$ gilt?

Dabei ist U die Menge des Universums. Die Teilmengen $S_i \in \mathcal{S}$ beinhalten jeweils einen Teil der Menge U. Befindet sich ein $u \in U$ in einer bestimmten Menge $S_i \in \mathcal{S}$, so wird davon gesprochen, dass S_i das Element u abdeckt. Ziel ist es, eine Menge $T \subseteq \mathcal{S}$ mit minimaler Kardinalität zu finden, deren Elemente vereinigt die Menge U ergeben. Eine Menge $M \subseteq \mathcal{S}$, deren Teilmengen vereinigt das gesamte Universum U ergeben, wird dabei *Set-Cover* genannt. Daher wird die Menge T ein *minimales* Set-Cover genannt.

Alternativ kann die Problemstellung auch als *Hitting-Set-Problem* formuliert werden.

Hitting-Set-Problem	
Eingabe:	Eine Menge V und eine Menge $\mathcal{R} = \{R_1, R_2, \ldots, R_m\}$, die aus Teilmengen von V besteht. Dabei gilt $\lvert V \cap R_i \rvert \geq 1$ für alle $R_i \in \mathcal{R}$.
Fragestellung:	Welche ist eine kleinste Menge $H \subseteq V$, sodass $\lvert H \cap R_i \rvert \geq 1$ für alle $R_i \in \mathcal{R}$ gilt?

Für das Hitting-Set-Problem existiert ebenso eine Menge V, welche das Universum bezeichnet und eine Menge $\mathcal{R}$, dessen Elemente R_i Teilmengen von V sind. Jedoch ist das Ziel des Hitting-Set-Problems eine kleinste Teilmenge H des Universums zu finden, die jede Menge aus der Menge $\mathcal{R}$ mindestens einmal trifft. Analog zum Set-Cover ist die Menge V ein gültiges *Hitting-Set*. Die Menge H wird somit als ein *minimales* Hitting-Set bezeichnet.

Eine Set-Cover-Instanz und eine Hitting-Set-Instanz können ohne Beschränkung in das jeweils andere Problem überführt werden. Dafür sei die vorliegende Instanz eine Instanz des Set-Cover-Problems, welche in eine Hitting-Set-Instanz überführt werden soll. Für die Überführung werden die Teilmengen $S_i \in \mathcal{S}$ zu Elementen des Universums V, und die Elemente des Universums U werden zu Teilmengen $R_i \in \mathcal{R}$. Dabei enthält eine Teilmenge R_i alle Elemente aus V, deren jeweilige Teilmenge S_i das zu R_i gehörige Element aus U enthielt. Sei T ein minimales Set-Cover und H eine Menge, welche die jeweiligen Elemente aus V enthält, die aus den Teilmengen $S_i \in T$ bei der Überführung entstanden sind, so ist H ein minimales Hitting-Set. Demnach gelten alle bekannten Schranken und Beweise zum Hitting-Set-Problem auch für das Set-Cover-Problem und umgekehrt genauso.

2.1.1 Ganzzahliges lineares Programm

Zusätzlich können die Probleme auch als *ganzzahliges lineares Programm* beschrieben werden. Durch ein ganzzahliges lineares Programm (kurz *ILP*, von engl. Integer Linear Program) können Optimierungsprobleme wie das Set-Cover-Problem beschrieben und mithilfe eines ILP-Lösungsverfahrens optimal gelöst werden. Im Folgenden wird das ILP für das Set-Cover-Problem aufgestellt. Für das ILP sei U erneut das Universum und $\mathcal{S}$ eine Menge aus Teilmengen von U. Die Menge $\mathcal{S}_u$ für ein Element $u \in U$ enthält alle Mengen aus $\mathcal{S}$, welche u enthalten. Für $S_i \in \mathcal{S}$ sei Variable $x_{S_i} \in \{0, 1\}$ mit der intendierten Bedeutung

$$x_{S_i} = 1 \Leftrightarrow S_i \in \mathcal{S} \text{ ist Teil des zu berechneten Set-Covers.}$$

Das gesamte ILP kann demnach folgendermaßen beschrieben werden.

Minimiere $\sum_{S_i \in \mathcal{S}} x_{S_i}$ unter den folgenden Nebenbedingungen.

Für jedes Element $u \in U$ ist mindestens eine Teilmenge aus $\mathcal{S}_u$ im minimalen Set-Cover:

$$\sum_{S_i \in \mathcal{S}_u} x_{S_i} \geq 1, \qquad \text{für } u \in U$$

Jede Teilmenge $S_i \in \mathcal{S}$ kann nur entweder im Set-Cover oder nicht im Set-Cover sein:

$$x_{S_i} \in \{0, 1\}, \qquad \text{für } S_i \in \mathcal{S}$$

Analog dazu kann auch das ILP für das Hitting-Set-Problem erstellt werden. Das Set-Cover-Problem gehört zu den 21 NP-vollständigen Problemen von Karp [Kar72]. Daher ist auch das Hitting-Set-Problem NP-vollständig. Durch die NP-Vollständigkeit können

mit gängigen ILP-Lösungsverfahren nur kleinere Instanzen des Problems in akzeptabler Zeit berechnet werden.

Ein bekannter Greedy-Algorithmus bietet für das Set-Cover-Problem eine $ln(d)$-Approximation, wobei d die Anzahl der Elemente der größten Teilmenge $S_i \in \mathcal{S}$ ist [Chv79]. Außerdem ist bekannt, dass das Set-Cover-Problem nicht besser approximiert werden kann, solange NP $\notin TIME(n^{O(log\,log\,n)})$ gilt [Fei98].

2.2 Geometrisches Set-Cover- und geometrisches Hitting-Set-Problem

Da sich die Punkte und Kreise in unserem Problem zusätzlich in einer Ebene befinden, handelt es sich im Besonderen um das *geometrische Set-Cover-Problem*. Das geometrische Set-Cover-Problem unterscheidet sich zum Set-Cover-Problem darin, dass die Menge $\mathcal{S}$ aus geometrischen Formen und die Menge U aus Punkten besteht. Dabei enthält eine Teilmenge $S_i \in \mathcal{S}$ ein Element $u \in U$, falls sich u in der Fläche (oder für höhere Dimensionen im Raum) der geometrischen Form von S_i befindet. Im Folgenden wird das Problem nur in einer Ebene und somit im Zweidimensionalen betrachtet.

Geometrisches Set-Cover-Problem	
Eingabe:	Eine Menge U aus Punkten und eine Menge $\mathcal{S}$ aus geometrischen Formen, wobei gilt, dass die vereinigte Fläche der Formen aus $\mathcal{S}$ alle Punkte aus U beinhaltet.
Fragestellung:	Welche ist eine kleinste Menge $T \subseteq \mathcal{S}$, sodass die Vereinigung der Flächen aus T alle Punkte aus U beinhaltet?

Auf unsere Problemstellung angewandt, entsprechen die Kreise in D den Teilmengen S_i, und D entspricht somit der Menge $\mathcal{S}$. Die Punkte in P entsprechen den Punkten in U. Zusätzlich muss dafür die Annahme getroffen werden, dass alle Punkte in P in mindestens einem Kreis aus D liegen, da sonst die Vereinigung der Kreise nicht die gesamte Punktmenge überdeckt. Liegt ein Punkt jedoch in keinem Kreis bedeutet das, dass diese Instanz auch keine gültige Lösung hat. Daher ist diese Annahme keine weitere Einschränkung des vorliegenden Problems. Somit ist die Menge D auch ein gültiges Set-Cover.

Analog dazu existiert auch eine geometrische Version des Hitting-Set-Problems.

Geometrisches Hitting-Set-Problem	
Eingabe:	Eine Menge V aus Punkten und eine Menge $\mathcal{R}$ aus geometrischen Formen wobei gilt, dass in der Fläche aller Elemente von $\mathcal{R}$ mindestens ein Element aus V liegt.
Fragestellung:	Welche ist eine kleinste Menge $H \subseteq V$, sodass jede Fläche aus $\mathcal{R}$ mindestens ein Element aus H enthält?

Auch die geometrischen Versionen der beiden Probleme können in eine Instanz des jeweils anderen Problems überführt werden.

Somit kann die folgende Problemdefinition für die Platzierung der Gateways erstellt werden. Es existiert eine Menge P von Punkten in der Ebene und eine Menge D von Kreisscheiben mit einem Radius von r. Ziel ist es, eine Menge $C \subseteq D$ mit minimaler Kardinalität zu finden, sodass sich jeder Punkt aus P in mindestens einer Kreisscheibe aus C befindet. Die Menge C repräsentiert dabei die Gateways, die aus den möglichen Gateway-Standorten (D) letztendlich aufgestellt werden. Nach dieser Definition ist die Menge C eine optimale Lösung für die Problemstellung ohne Limitierung zugeordneter Sensoren pro Gateway und auch die Lösung für das dabei entstandene geometrische Set-Cover-Problem.

Trotz der geometrischen Einschränkung ist bewiesen, dass das geometrische Hitting-Set-Problem bereits für einheitlich große Kreisscheiben NP-vollständig [FPT81] ist, wodurch dasselbe auch für das geometrische Set-Cover-Problem gilt. Dennoch kann die geometrische Version der Probleme besser approximiert werden als die allgemeine Variante.

2.2.1 Epsilon-Netz-Algorithmen

Es existiert ein Fast-Linearzeitalgorithmus für eine $\mathcal{O}(1)$-Approximierung des geometrischen Hitting-Set-Problems für eine Menge $\mathcal{R}$ mit endlicher VC-Dimension[1] [AP20]. Da dieser Algorithmus in der Praxis durch große konstante Faktoren sehr langsam wird und die Approximationsgüte über 40 steigt, wurde eine veränderte Version des Algorithmus namens `dnet` erstellt [BMR18]. Obwohl dessen fast-lineare Laufzeit nicht bewiesen wurde, lassen die Experimente jedoch vermuten, dass es sich um eine solche Laufzeit handelt. Der `dnet`-Algorithmus bietet eine $(13{,}4+\varepsilon)$-Approximierung für eine Menge $\mathcal{R}$ mit endlicher VC-Dimension. In der Praxis soll der `dnet`-Algorithmus hingegen eine durchschnittliche Approximationsgüte von 1,3 bieten. Beide genannten Algorithmen verwenden *ε-Netze*. Sei V eine Menge und $\mathcal{R}$ eine Menge aus Teilmengen von V. Außerdem sei L ebenfalls eine Teilmenge von V. Ein ε-Netz von L ist eine Teilmenge $E \subseteq L$ für die gilt, dass für jede Menge $R_i \in \mathcal{R}$ mit $|R_i \cap L| \geq \varepsilon |L|$ die Ungleichung $|R_i \cap E| \geq 1$ gilt. Das bedeutet, dass jede Teilmenge aus $\mathcal{R}$, welche mindestens $1/\varepsilon \cdot |L|$ Elemente enthält, auch einen Punkt aus E enthalten muss.

2.2.2 Der Local-Search-Algorithmus

Eine langsamere Variante, welche aber eine $(1+\varepsilon)$-Approximierung für verschiedene geometrische Formen (darunter auch Kreise) bietet, arbeitet mit Local-Search [MR10]. Da es sich um einen Hitting-Set-Algorithmus handelt, in dieser Arbeit aber die Set-Cover-Variante verwendet wird, wird der Algorithmus auch nur als Set-Cover-Algorithmus beschrieben. Der Local-Search-Algorithmus für das Set-Cover-Problem ist abhängig von einer ganzzahligen Variable k. Zu Beginn werden alle Kreise aus D als Set-Cover C verwendet. Im Laufe des Algorithmus werden so lange k Kreise aus dem momentanen Set-Cover C

[1] Eine endliche Menge aus Kreisen hat in jedem Fall eine endliche VC-Dimension [VC15].

durch $k-1$ Kreise aus D ersetzt, während C stets ein Set-Cover bleibt, bis kein solches Ersetzen mehr möglich ist. Dies bietet eine $(1+\varepsilon)$-Approximation, wobei $\varepsilon = b/\sqrt{k}$ für ein konstantes b gilt. Außerdem läuft der Algorithmus in polynomieller Laufzeit, wodurch man einen PTAS für das geometrische Set-Cover-Problem mit Kreisscheiben erhält. Sei $n = |D|$ und $m = |U|$, so liegt die genaue Laufzeit des naiven Algorithmus von Mustafa und Ray [MR10] bei $\mathcal{O}(mn^{2k+1})$. Da die Laufzeit länger ist als bei den ε-Netz-Algorithmen, gilt es weiterhin, Laufzeit und Approximationsgüte gegeneinander abzuwägen. In dieser Arbeit gibt es anfangs nur eine statische Eingabe, die es zu lösen gilt. Daher wird die Approximationsgüte gegenüber der Laufzeit priorisiert und somit der Local-Search-Algorithmus als Ausgang für den hier entwickelten Algorithmus gewählt. Zu Vergleichszwecken wird der neue Algorithmus auf Laufzeit und Approximationsgüte gegen den `dnet`-Algorithmus getestet.

Um die Güte der Lösung des veränderten Local-Search-Algorithmus weiterhin abschätzen zu können, wurde auch das ILP des Set-Cover-Problems implementiert. Das ILP des Set-Cover-Problems kann ohne weiteres für das geometrische Set-Cover-Problem übernommen werden, da sich die Verwendung der Mengen U und $\mathcal{S}$ nicht verändert. Somit konnten die Ergebnisse des veränderten Local-Search-Algorithmus mit den optimalen Ergebnissen des ILPs für kleine Instanzen verglichen werden. Die Ergebnisse dieser Experimente werden in Kapitel 5 analysiert.

3 Das geometrische Set-Cover-Problem mit zusätzlichen Einschränkungen

Ein reiner Set-Cover-Algorithmus bietet jedoch keine Einschränkungen, damit jedes Gateway eine obere Schranke für die Anzahl der versorgten Sensoren hat, während zusätzlich jeder Sensor nur dem nächsten Gateway im Set-Cover zugeordnet wird. Daher haben wir im Folgenden der Local-Search-Algorithmus aus Kapitel 2.2.2 dahingehend verändert. Insbesondere kann die Approximationsgüte nach der Veränderung nicht mehr garantiert werden, jedoch scheint der Algorithmus in der Praxis ähnlich gut zu funktionieren (siehe Kapitel 5).

3.1 Genaue Zuordnung eines Sensors zum nächsten Gateway

Die Einschränkung, dass jeder Sensor nur dem nächsten Gateway zugeordnet werden kann, kann bereits ohne Änderung des eigentlichen Local-Search-Algorithmus erfüllt werden. Innerhalb des Set-Covers C des Local-Search-Algorithmus liegt jeder Punkt aus P nach Definition in mindestens einem Kreis. Somit kann jeder Sensor (repräsentiert durch ein $p \in P$), mindestens einem Gateway (repräsentiert durch ein $c \in C$) zugeordnet werden. Kann ein Sensor mehreren Gateways zugeordnet werden, so wird das nächste Gateway verwendet.

3.2 Limitierung der Anzahl an Sensoren pro Gateway

Da Gateways in der Realität keine unendliche Menge an Sensoren versorgen können, wird eine Variable ℓ eingeführt. Diese Variable beschreibt die maximale Anzahl an Sensoren, die ein Gateway versorgen kann. Um diese Einschränkung erfüllen zu können, muss davon ausgegangen werden, dass die gegebene Set-Cover-Instanz mit einem solchen Limit lösbar ist. Ein einzelner Kreis, der $\ell + 1$ Punkte beinhaltet, ist zwar eine lösbare Set-Cover-Instanz, jedoch kann das Limit ℓ offensichtlich nicht erfüllt werden. Zusätzlich zu der Einschränkung, dass jeder Sensor nur dem nächsten Gateway zugeordnet werden kann, wäre eine erste Idee, für einen Kreis nur die nächsten ℓ Punkte als erreichbar zu definieren, welche sich innerhalb des maximalen Radius r um den Kreismittelpunkt befinden. Somit würde man den neuen Radius $r' < r$ des Kreises auf einen Wert setzen, sodass maximal ℓ und dennoch so viele Punkte wie möglich innerhalb des Kreises liegen. Das kann schon in der Vorverarbeitung geschehen und verändert den bestehenden Algorithmus daher nicht. Im Algorithmus 1 (Zeile 1 - 5) ist ein Beispiel zu finden, wie diese Vorverarbeitung erfolgen kann, sodass der eigentliche Local-Search-Algorithmus (Zeilen

6 - 16) unberührt bleibt.

Algorithmus 1 : Local-Search für Set-Cover mit ℓ

Eingabe : Mittelpunkte und maximaler Radius r der Kreise D,
Punktemenge P, Integer k, Integer ℓ
Ausgabe : Möglichst kleines Set-Cover C

```
1  for Kreis d ∈ D do
2      P_d ← {p|p ∈ P, p liegt in d}
3      sortiere P_d aufsteigend nach den Distanzen zu d
4      Behalte nur die ℓ ersten Elemente in P_d
5      Setze den Radius von d auf die Distanz des letzten Punkts aus P_d
6  C ← D
7  weitersuchen ← true
8  while weitersuchen do
9      weitersuchen ← false
10     foreach C_i ⊆ C mit |C_i| = k do
11         foreach D_j ⊆ D mit |D_j| = k − 1 do
12             if ((C\C_i) ∪ D_j) ist Set-Cover für P then
13                 C ← ((C\C_i) ∪ D_j)
14                 weitersuchen ← true
15                 unterbreche beide for-Schleifen
16 return C
```

Problem an dieser Änderung ist jedoch, dass Instanzen, welche vor der Vorverarbeitung lösbar waren, unlösbar werden können, da die Beschränkung der Kreise auf die nächsten ℓ Punkte eine zu starke Beschränkung ist. Eine Beispielinstanz kann in Abbildung 3.1 betrachtet werden. Innerhalb dieser liegen die Sensoren p_1 und p_2 näher an z_1 als an z_2 und die Sensoren p_3 und p_4 liegen näher an z_2 als an z_1. Zusätzlich liegen p_2 und p_3 jeweils näher an z_1 und z_2 als p_1 und p_4.

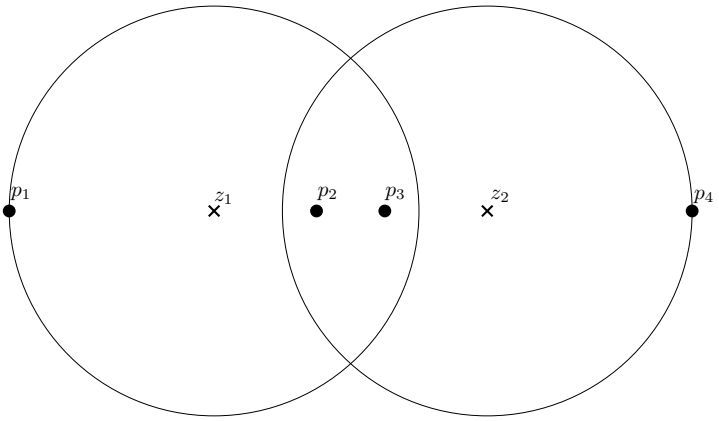


Abb. 3.1: Beispielinstanz mit zwei Kreisen um die Mittelpunkte z_1 und z_2 und vier Punkten p_1, p_2, p_3 und p_4.

Sei nun das Limit für die Anzahl der Gateways $\ell = 2$, so wäre $C = \{z_1, z_2\}$ eine Lösung für das geometrische Set-Cover-Problem mit den obigen Einschränkungen. Dabei werden dem Gateway z_1 die Sensoren p_2 und p_1 und dem Gateway z_2 die Sensoren p_3 und p_4 zugeordnet. Der Algorithmus 1 würde jedoch den Radius der Kreise um z_1 und z_2 soweit einschränken, dass beide Kreise nur noch die Punkte p_2 und p_3 beinhalten. Somit wären die Punkte p_1 und p_4 durch keinen Kreis abgedeckt, wodurch die Instanz nach der Definition in Kapitel 2.2 keine zulässige geometrische Set-Cover-Instanz mehr wäre.

3.3 Das Voronoi-Cover-Problem

Um die Problemstellung auf die zusätzlichen Einschränkungen anzupassen, haben wir ein neues Problem definiert. Für diese Problemdefinition wird das Konzept eines *Voronoi-Diagramms* benötigt. Das Voronoi-Diagramm für eine Menge M aus Punkten setzt sich aus einer *Voronoi-Zelle* um jeden Punkt zusammen. Dabei ist die Voronoi-Zelle um einen Punkt, die Fläche in der alles, was sich innerhalb dieser Voronoi-Zelle befindet, näher an diesem Punkt ist als an allen anderen Punkten der Menge M. Das geometrische Set-Cover-Problem mit den zusätzlichen Einschränkungen aus Kapitel 1.1 wird im Folgenden als *Voronoi-Cover-Problem* bezeichnet.

VORONOI-COVER-PROBLEM	
Eingabe:	Eine Menge U aus Punkten und eine Menge $\mathcal{S}$ aus Kreisen wobei gilt, dass die vereinigte Fläche der Kreise aus $\mathcal{S}$ alle Punkte aus U beinhaltet und eine natürliche Zahl ℓ.
Fragestellung:	Welche ist eine kleinste Menge $T \subseteq \mathcal{S}$, sodass die Vereinigung der Flächen aus T alle Punkte aus U beinhaltet und zusätzlich in jeder Voronoi-Zelle der Kreismittelpunkte der Elemente von T maximal ℓ Punkte liegen?

3.4 Der Voronoi-Local-Search-Algorithmus

Ein Algorithmus zur Lösung des Voronoi-Cover-Problems, der auf dem Local-Search-Algorithmus für geometrische Set-Cover basiert, wird im Folgenden als *Voronoi-Local-Search-Algorithmus* bezeichnet. Der Ablauf des Voronoi-Local-Search-Algorithmus kann wie folgt skizziert werden.

Erneut wird mit allen Kreisen der Gateways als Set-Cover begonnen. In jedem Schritt wird versucht, k Kreise aus dem Set-Cover C durch $k-1$ Kreise aus der Kreismenge D zu ersetzen, sodass C weiterhin ein Set-Cover bleibt und die Voronoi-Zellen der Kreismittelpunkte von C nicht mehr als ℓ Punkte aus P beinhalten. Sobald der Algorithmus keinen solchen Austausch mehr findet, terminiert er. Für den Algorithmus 2 sei V_c die Voronoi-Zelle eines Kreises $c \in C$ und $|V_c|$ die Anzahl der Punkte innerhalb der Voronoi-Zelle.

Algorithmus 2 : Voronoi-Local-Search-Algorithmus

Eingabe : Mittelpunkte und maximaler Radius r der Kreise D, Punktemenge P, Integer k, Integer ℓ

Ausgabe : Möglichst kleines Set-Cover C

```
1  C ← D
2  weitersuchen ← true
3  while weitersuchen do
4      weitersuchen ← false
5      foreach C_i ⊆ C mit |C_i| = k do
6          foreach D_i ⊆ D mit |D_i| = k − 1 do
7              if ((C\C_i) ∪ D_i) ist Set-Cover für P then
8                  if ∀c ∈ ((C\C_i) ∪ D_i) : |V_c| < ℓ then
9                      C ← ((C\C_i) ∪ D_i)
10                     weitersuchen ← true
11                     unterbreche beide for-Schleifen
12 return C
```

Es war uns bislang nicht möglich eine obere Schranke für die Lösungsgüte des Voronoi-Local-Search-Algorithmus zu beweisen, jedoch wurden Experimente mit dem ILP für das Voronoi-Cover-Problem durchgeführt, um ein Gefühl für die Lösungsgüte des Algorithmus zu bekommen. Die Experimente dazu werden in Kapitel 5.5 analysiert.

3.5 Angepasstes ganzzahliges lineares Programm

Um die Experimente durchführen zu können, wurde zunächst das ILP für das Set-Cover-Problem an das Voronoi-Cover-Problem angepasst. Für das angepasste ILP gilt erneut, dass die Menge D_p alle Kreise enthält, in welchen der Punkt p liegt. Zusätzlich sind die Kreise innerhalb der Menge D_p nach der Distanz ihrer Mittelpunkte zum Punkt p auf-

steigend sortiert. Der Index eines Kreises innerhalb dieser geordneten Menge wird mit $i_{p,d}$ beschrieben. Außerdem sei P_d die Menge aller Punkte in Reichweite eines Kreises d.

$x_d = 1 \Leftrightarrow d$ ist Teil des minimalen Set-Covers
$x_{p,d} = 1 \Leftrightarrow d$ ist der zu p nächste Kreis, der sich im Set-Cover befindet und p abdeckt

Minimiere $\sum_{d \in D} x_d$ unter den folgenden Nebenbedingungen.

Jeder Punkt muss mindestens einem Kreis zugeordnet sein:

$$\sum_{d \in D_p} x_{p,d} \geq 1 \qquad \text{für } p \in P$$

Jeder Kreis darf maximal ℓ Punkte enthalten:

$$\sum_{p \in P_d} x_{p,d} \leq \ell \qquad \text{für } d \in D$$

Ein Punkt kann nur dann einem Kreis zugeordnet werden, wenn dieser Kreis im Set-Cover enthalten ist:

$$x_{p,d} \leq x_d \qquad \text{für } p \in P, d \in D_p$$

Es darf kein Kreis im Set-Cover sein, der näher an einem Punkt ist als der Kreis, welchem der jeweilige Punkt zugeordnet ist:

$$\sum_{d' \in D_p : i_{p,d'} < i_{p,d}} x_{d'} \leq |D_p| - |D_p| x_{p,d} \qquad \text{für } p \in P, d \in D_p$$

Jeder Kreis kann nur entweder nicht im Set-Cover oder im Set-Cover sein:

$$x_d \in \{0, 1\} \qquad \text{für } d \in D$$

Jeder Punkt kann einem Kreis nur zugeordnet oder nicht zugeordnet sein:

$$x_{p,d} \in \{0, 1\} \qquad \text{für } p \in P, d \in D_p$$

4 Beschleunigungstechniken

Um unseren Voronoi-Local-Search-Algorithmus in der Praxis noch zu beschleunigen, wurden mehrere Beschleunigungstechniken auf den ursprünglichen Local-Search-Algorithmus angewandt und danach auf den Voronoi-Local-Search-Algorithmus übertragen. Im Besonderen wird die Beschleunigungstechnik in Kapitel 4.2 nur auf dem Local-Search-Algorithmus ohne die Einschränkungen aus Kapitel 3 beschrieben. Durch die Einschränkungen des Voronoi-Local-Search-Algorithmus ist für diese Beschleunigungstechnik nicht mehr gegeben, dass die Güte der Lösung des Voronoi-Local-Search-Algorithmus dieselbe ist wie ohne Beschleunigungstechnik. Trotzdem wurde diese Beschleunigungstechnik auch exakt so auf den Voronoi-Local-Search-Algorithmus angewandt. Experimente, ob sich dadurch die Lösungsgüte auch in der Praxis verändert, wurden in Kapitel 5 durchgeführt.

4.1 Inkrementierung der Anzahl der betrachteten Objekte

Im bisherigen Local-Search-Algorithmus werden stets k Kreise durch $k-1$ Kreise ersetzt. In der Praxis resultiert das jedoch, besonders zu Beginn des Algorithmus, öfters in einer trivialen Löschung eines Kreises statt in einer tatsächlichen Ersetzung der Kreise aus dem Set-Cover. Dabei werden beispielsweise (für $k = 3$) die Kreise $\{a, b, c\} \subseteq C$ durch die Kreise $\{a, b\} \subseteq D$ ausgetauscht. Somit wurde effektiv nur der Kreis c aus dem Set-Cover entfernt. Solche Löschungen einzelner Kreise würden auch passieren, wenn zu Beginn $k = 1$ gilt, da in diesem Fall nur Kreise, welche für das Set-Cover überflüssig sind, aus dem Set-Cover gelöscht werden. Da jedoch eine Überprüfung, ob ein einzelner Kreis aus dem Set-Cover gelöscht werden kann, in der Praxis schneller ist als zu überprüfen, ob drei Kreise durch zwei Kreise ersetzt werden können, ist es von Vorteil, diese einzelnen Löschungen zunächst mithilfe des Algorithmus für $k = 1$ auszuführen. Analog kann diese Argumentation auf jedes k bis zum gewünschten k angewandt werden. Daher wird der Algorithmus auf folgende Weise verändert. Sei beim Aufruf des Algorithmus $k = k_{max}$, so wird zunächst der Algorithmus für $k = 1$ ausgeführt. Sobald der Algorithmus für $k = 1$ keine Verbesserung des Set-Cover mehr finden kann, wird das k inkrementiert und der Algorithmus mit dem Set-Cover aus der Lösung des vorherigen Schritts erneut ausgeführt. Dies wird solange wiederholt, bis das gewünschte $k = k_{max}$ erreicht und der Algorithmus dafür ausgeführt wurde.

Diese Veränderung stellt dabei nicht sicher, dass im weiteren Verlauf nicht doch noch ein Fall eintreten kann, indem k Kreise durch $k - 1$ derselben Kreise getauscht werden. Jedoch werden einige langsame Löschungen der oben beschriebenen Form ($\{a, b, c\} \subseteq C$ werden durch $\{a, b\} \subseteq D$ getauscht) durch einfache Löschungen der einzelnen Kreise ersetzt. Eine genauere Analyse zu dieser Beschleunigungstechnik befindet sich in Kapitel 5.3.1.

4.2 Einführung eines Gitters

Eine weitere Beschleunigungstechnik ist die Einführung eines Gitters, um nicht jede Kombination aus bis zu k Kreisen für das Ersetzen durch $k-1$ Kreise in Betracht zu ziehen. Stattdessen kommen nur Kreisscheiben in Frage, welche nah genug beieinander liegen, sodass diese Kreise durch einen Kreis weniger ersetzt werden können. Dabei soll sich die Güte der Lösung des Algorithmus jedoch nicht verändern. Dafür wird ein Gitter über die Ebene gespannt, dessen Quadrate eine Seitenlänge von $2r$ haben, wobei r der Radius der Kreise (und somit die maximale Sendereichweite der Gateways) ist. Der Algorithmus soll für ein bestimmtes k dann bei jedem Quadrat des Gitters beginnend eine quadratische Fläche der Größe $((2k-1)\cdot 2r)^2$ betrachten. Diese Fläche setzt sich aus insgesamt $(2k-1)^2$ Quadraten des Gitters zusammen. Da das Gitter nicht unendlich in alle Richtungen reicht, ist es ausreichend, an allen Quadraten zu beginnen, sodass die Bereiche keine Außenkante des Gitters schneiden. Innerhalb dieser $((2k-1)\cdot 2r)^2$ großen Bereiche befinden sich alle möglichen Quadrate der Größe[1] $((2k-2)\cdot 2r)^2$, welche in der Zeichenfläche zu finden sind. Somit betrachtet der Algorithmus alle möglichen Quadrate der Größe $((2k-2)\cdot 2r)^2$ in unserer Zeichenfläche. Diese Erkenntnis ergibt zusammen mit später folgenden Lemma 4.1 den Grundstein dieser Beschleunigungstechnik.

In Abbildung 4.1 kann ein Beispiel des Gitters betrachtet werden. In dem Beispiel ist ein betrachteter Bereich des Algorithmus beginnend bei dem Quadrat an der Stelle $(1,1)$ für $k=3$ in Rot zu sehen. Dieser Bereich beinhaltet alle Quadrate der Größe $((2k-2)\cdot 2r)^2$, dessen obere linke Ecke innerhalb des Quadrates $(1,1)$ liegen. Ein Beispiel für ein solches Quadrat ist in Abbildung 4.1 in Grün eingezeichnet.

Wird der Algorithmus für ein bestimmtes k ausgeführt, so wird für den betrachteten Bereich ein neues k' gewählt, sodass k' das Minimum von k und der Anzahl der Kreismittelpunkte im betrachteten Bereich ist. Im weiteren Verlauf wird in diesem Bereich dann nach einem Tausch von k' Kreisen aus C durch $k'-1$ Kreisen aus D gesucht, sodass C weiterhin ein Set-Cover bleibt. Diese Feinheit ist für den Beweis des Lemmas 4.1 erforderlich. Wichtig dabei ist, dass die Wahl von k' die Größe des betrachteten Bereichs nicht beeinflusst, denn diese Größe ist weiterhin nur abhängig von k. Somit fehlt nur noch zu zeigen, dass Lemma 4.1 gilt.

Lemma 4.1. *Es ist ausreichend, Kreisscheiben nur dann für einen Austausch im Local-Search-Algorithmus mit Gitter zu betrachten, wenn sich die Mittelpunkte der Kreise alle in einem quadratischen Bereich mit einer Seitenlänge von $(2k-2)\cdot 2r$ befinden.*

Beweis. Dafür sei Algorithmus $Algo_{allg}$ der Algorithmus, welcher alle Kreise für einen Tausch in Betracht zieht und Algorithmus $Algo_{gitter}$ derjenige, der alle beschränkten quadratischen Bereiche mit einer Seitenlänge von $(2k-2)\cdot 2r$ in Betracht zieht. Es gilt zu zeigen, dass wenn der Algorithmus $Algo_{gitter}$ keinen gültigen Tausch von bis zu k Kreisen aus C durch einen Kreis weniger aus D mehr findet, der Algorithmus $Algo_{allg}$ auch keinen solchen Tausch mehr finden kann.

Dafür wird angenommen, dass der Algorithmus $Algo_{allg}$ noch einen weiteren Austausch findet, obwohl der Algorithmus $Algo_{gitter}$ bereits terminiert ist. Dabei sei $A \subseteq C$ die

[1] Man beachte hier den Unterschied zur vorherigen Formel. Aus -1 wurde hier -2.

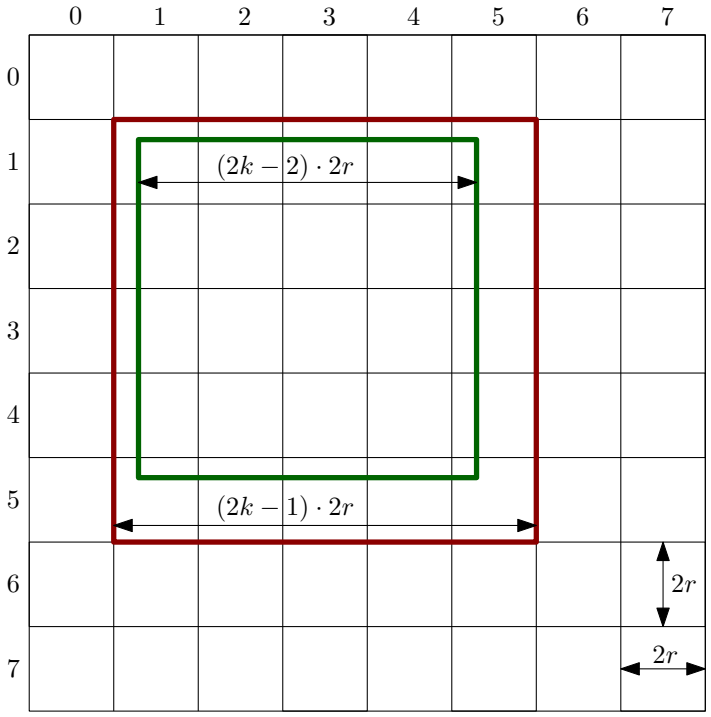


Abb. 4.1: Visualisierung des Gitters mit einem Beispiel eines betrachteten Bereichs des Algorithmus (rot) und einem Bereich der Größe $((2k-2)\cdot 2r)^2$ (grün).

Menge der k Kreise, welche aus dem Set-Cover genommen, und $B \subseteq D$ die Menge der $k-1$ Kreise, welche von dem Algorithmus $Algo_{allg}$ stattdessen in das Set-Cover aufgenommen werden sollen. Hierbei werden des Weiteren zwei Fälle unterschieden.

1. Fall: Mindestens ein Kreis aus A kann aus C gelöscht werden, sodass C immer noch ein Set-Cover ist. Sei $a \in A$ ein Kreis, der diese Eigenschaft erfüllt. Demnach ist es nicht von Bedeutung, welche $k-1$ Kreise sich zusätzlich zu a in A befinden. Eine gültige Wahl für B wäre in diesem Fall $B = A \backslash a$. Da der Algorithmus mit dem Gitter, das k' für jeden der betrachteten Bereiche, in dem sich a befindet, so wählt, dass sich zusätzlich zu a noch mindestens $k'-1$ Kreise in diesem Bereich befinden, wird auch der Algorithmus $Algo_{gitter}$ eine Menge A' mit $a \in A'$ und $|A'| = k'$ und eine Menge B' mit $|B'| = k'-1$ innerhalb des betrachteten Bereichs finden können. Demnach könnte der Algorithmus $Algo_{gitter}$ nicht terminiert sein, was ein Widerspruch zur Annahme ist.

2. Fall: Keiner der Kreise in A kann aus C gelöscht werden, ohne dass Punkte aus P nicht mehr durch C überdeckt wären. Falls dies gilt, muss jeder Kreis $a \in A$ mindestens von einem Kreis aus B zumindest berührt werden oder mit einem Kreis aus B überlappen, um ebendiese Punkte innerhalb a abzudecken, welche in $C \backslash a$ nicht mehr überdeckt wären. Im Folgenden wird weiter von einem Berühren gesprochen, auch wenn zwei Kreise sich möglicherweise überlappen oder identisch sind. Die Kreise aus $A \cup B$ formen durch Berührungen gewisse Zusammenhangskomponenten. Ein Kreis liegt in derselben Zusammenhangskomponente wie ein anderer Kreis, falls sich

zwischen den Kreisen abwechselnd ein Kreis aus A mit einem Kreis aus B berühren. Zwei Kreise aus B liegen beispielsweise in einer Zusammenhangskomponente, falls sie beide den gleichen Kreis aus A berühren. Dabei ist nicht entscheidend, ob sich die Kreise aus B gegenseitig berühren oder nicht. Da sich in B ein Kreis weniger befindet als in A, muss es eine Zusammenhangskomponente geben, in der sich weniger Kreise aus B befinden als aus A. Seien $A' \subseteq A$ alle Kreise aus A, und $B' \subseteq B$ alle Kreise aus B, welche sich in dieser Zusammenhangskomponente befinden. So ist $C' = (C \backslash A') \cup B'$ mit $|C| - 1 = |C'|$ ebenfalls ein Set-Cover. Für A' und B' gilt $|A'| \leq k$ und $|B'| \leq k - 1$ und $|B'| < |A'|$. Aufgrund der Wahl von A' und B' können die am weitesten von einander entfernten Kreismittelpunkte in $A' \cup B'$ maximal $(2k - 2) \cdot 2r$ weit voneinander entfernt sein, da sich die Kreise berühren müssen. Daher würde der Algorithmus mit Gitter auch diese Verkleinerung des Set-Covers finden und könnte nicht terminiert sein, was ebenfalls ein Widerspruch zur Annahme ist. □

Wie zuvor beschrieben, gilt dieser Beweis nur für den Local-Search-Algorithmus ohne die Einschränkungen aus Kapitel 3. Denn für den Voronoi-Local-Search-Algorithmus müssen sich die Kreise nicht zwangsläufig abwechselnd berühren, da durch das Einfügen eines Kreises Punkte, welche zuvor in der Voronoi-Zelle eines anderen Kreises lagen, nun in der neuen Voronoi-Zelle liegen können. Dadurch kann möglicherweise ein Kreis gelöscht werden, welcher zuvor zu viele Kreise an benachbarte Voronoi-Zellen abgegeben hätte. Jedoch überschreiten die benachbarten Voronoi-Zellen, durch die Abnahme von Punkten durch einen neuen Kreis auf einer gegenüberliegenden Seite, das Punktelimit nicht. Dennoch wurden Tests für den Voronoi-Local-Search-Algorithmus mit und ohne diesem Gitter durchgeführt. Dabei haben wir herausgefunden, dass das Gitter in der Praxis keine bemerkenswerte Verschlechterung der Lösung mit sich zieht. Diese Tests können in Kapitel 5.3.2 betrachtet werden. Deshalb haben wir dieses Gitter auch auf den Voronoi-Local-Search-Algorithmus angewandt.

Der Algorithmus 3 beschreibt, wie der Local-Search-Algorithmus innerhalb einer Gitterzelle vorgeht. Dabei sei i das momentane k, D' die Kreise, deren Kreismittelpunkte sich im momentan betrachteten Bereich aus D befinden, C' die Kreise dessen Kreismittelpunkte sich im momentan betrachteten Bereich aus C befinden und C das momentane Set-Cover. Außerdem kann dem Algorithmus 4 entnommen werden, wie die Suche in einer Gitterzelle aufgerufen wird.

Algorithmus 3 : sucheInGitterzelle

Eingabe : Integer i, Menge D', Menge C', Menge C
Ausgabe : Gibt *true* zurück falls i Kreise aus C' durch $i-1$ Kreise aus D' getauscht wurden, sodass C noch ein Set-Cover ist. Sonst $false$.

```
1  C'_i ← {S|S ⊆ C', |S| = i} // alle Teilmengen mit Kardinalität i aus C'
2  D'_{i-1} ← {S|S ⊆ D', |S| = i − 1}
3  foreach C'_i ∈ C'_i do
4      nichtAbgedeckt ← {p|p ∈ P, p nicht abgedeckt von C\C'_i}
5      foreach D'_{i-1} ∈ D'_{i-1} do
6          abgedeckt ← {p|p ∈ P, p abgedeckt von D'_{i-1}}
7          if nichtAbgedeckt\abgedeckt == ∅ then
8              C ← (C\C'_i) ∪ D'_{i-1}
9              return true
10 return false
```

Algorithmus 4 : LocalSearch für SetCover

Eingabe : Kreismenge D, Punktemenge P, Integer k
Ausgabe : Möglichst kleines Set-Cover C

```
1  d_x ← maximale Distanz in X-Richtung zwischen Punkten
2  d_y ← maximale Distanz in Y-Richtung zwischen Punkten
3  n_x ← ⌈d_x/(2r)⌉
4  n_y ← ⌈d_y/(2r)⌉
5  Teile D in Gitterzellen D[n_x][n_y] ein
6  C[x][y] ← D[x][y] // C mit D zu Beginn gleichsetzen
7  for i ← 1 to k do
8      X ← range(0, n_x − (2i − 1))
9      Y ← range(0, n_y − (2i − 1))
10     weitersuchen ← true
11     while weitersuchen do
12         weitersuchen ← false
13         foreach x ∈ X do
14             foreach y ∈ Y do
15                 D' ← ⋃_{a=0}^{2i-2} ⋃_{b=0}^{2i-2} D[x + a, y + b]
16                 C' ← ⋃_{a=0}^{2i-2} ⋃_{b=0}^{2i-2} C[x + a, y + b]
17                 i' ← min(|C'|, i)
18                 if sucheInGitterzelle(i', D', C', C) then
19                     weitersuchen ← true
20                     unterbreche beide for-Schleifen
21 return C
```

4.3 Implementationsdetails

Um zusätzlich unnötige Operationen zu vermeiden, können die Teilmengen in $\mathcal{C}'_i$ beziehungsweise $\mathcal{D}'_{i-1}$ aus Algorithmus 3 in einer bestimmten Reihenfolge überprüft werden. Dafür wird zu Beginn ein beliebiger Kreis c aus den Teilmengen der jeweiligen Menge ausgewählt und zunächst alle Teilmengen $C'_i \in \mathcal{C}'_i$ beziehungsweise $D'_{i-1} \in \mathcal{D}'_{i-1}$ verarbeitet, welche diesen Kreis beinhalten. Somit muss die Menge *nichtAbgedeckt* beziehungsweise *abgedeckt* für c so lange nicht erneut berechnet werden, bis alle Teilmengen, welche c beinhalten, abgearbeitet wurden.

Außerdem sollten die Mengen X und Y aus Algorithmus 4 vor jeder Verwendung (Zeile 14 und 15) zufällig angeordnet werden, damit nicht nach jedem Fund in denselben Bereichen das Suchen angefangen wird. Zusätzlich können Bereiche, in denen keine Verbesserung des Set-Covers gefunden werden konnte, markiert werden, sodass sie solange nicht mehr in Betracht gezogen werden, bis sich etwas in den jeweiligen Zellen durch andere Bereiche verändert. Sobald sich das momentane k verändert (in Algorithmus 4 durch die `for`-Schleife in Zeile 7) müssen diese Markierungen ebenfalls alle zurückgesetzt werden, da sich die Größe der zu betrachtenden Bereiche verändert.

Diese Implementationsdetails wurden in Kapitel 5 nicht gesondert getestet, da wir keinen Grund sehen, wieso diese Änderungen den Algorithmus in der Praxis merklich langsamer machen könnten.

5 Experimente

Im Folgenden werden die aufgeführten Tests sowohl für die Güte der Lösung als auch die Laufzeit des Voronoi-Local-Search-Algorithmus für verschiedene Instanz-Größen im Vergleich zu anderen Algorithmen analysiert. Der Voronoi-Local-Search-Algorithmus und die ILPs wurden dabei in Java implementiert. Für die ILPs wurde das ILP-Lösungsverfahren `CPLEX`[1] verwendet.

5.1 Testdatensatz

Um verschiedene Algorithmen und Problemversionen mit unterschiedlichen Eingabegrößen testen zu können, haben wir einen Datensatz am Beispiel von Würzburg erstellt. Der Datensatz ist mithilfe von OpenStreetMap[2] erstellt worden. Dabei wurde pro Gebäude in Würzburg ein Punkt erzeugt, welcher letztendlich einen Sensor abbilden soll. Der Datensatz bestehend aus den Gebäuden kann in Abbildung 5.1 betrachtet werden. Insgesamt besteht der Datensatz aus etwas mehr als 28.000 Gebäuden. Diese Gebäude sollen jeweils einen Sensor darstellen. Um eine Testinstanz mit weniger Sensoren erstellen zu können, wurde die gewünschte Menge an Sensoren zufällig aus den über 28.000 Sensoren ausgewählt. Für die Laufzeit- und Gütetests des Voronoi-Local-Search-Algorithmus wurden zusätzlich mögliche Gateway-Standorte erstellt. Die möglichen Standorte der Gateways wurden dabei mit einem Gitter über den Sensoren so verteilt, dass jeder Sensor in Sendereichweite zu mehreren Gateways liegt. Für das Gitter wurde erneut ein Netz aus Quadraten über die Ebene gelegt. Die Seitenlänge der Quadrate wurde dabei abhängig von der Sendereichweite r und einer Variablen c auf $c \cdot (r/\sqrt{2})$ gesetzt. Für $c = 1{,}0$ hieße das, dass das Gitter wie in Abbildung 5.2 aufgebaut ist. Ein möglicher Standort wird dabei an jeder Kreuzung einer horizontalen mit einer vertikalen Gitterlinie platziert. Somit läge für $c = 1{,}0$ jeder Sensor in Reichweite zu mindestens vier Gateways. Um das Gitter noch etwas feiner zu machen, wurde der Wert von c auf 0,9 gesetzt. Zusätzlich wurde ein bestimmter Prozentsatz der Sensoren als möglicher Standort für einen Gateway gewählt, um Bereiche mit besonders vielen Sensoren besser abzudecken. Sonst könnte es dazu kommen, dass das Sensoren-Limit der Gateways nicht eingehalten werden kann. Für die nachfolgenden Tests wurde jeder Sensor mit einer Wahrscheinlichkeit von 20% als möglicher Gateway-Standort ausgewählt. Würden an den vom Algorithmus zurückgegebenen Standorten Gateways aufgestellt werden, würden nach Definition alle Sensoren abgedeckt werden.

[1] https://www.ibm.com/de-de/analytics/cplex-optimizer
[2] https://www.openstreetmap.de/

Abb. 5.1: Gebäude in Würzburg. Pro Gebäude wurde ein schwarzer Punkt beziehungsweise ein schwarzes Polygon gezeichnet.

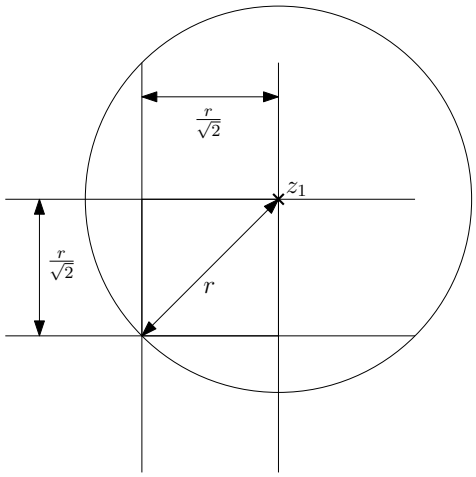


Abb. 5.2: Andeutung des Gitters für die möglichen Standorte der Gateways (mit $c = 1{,}0$). Der Kreis um z_1 repräsentiert dabei einen möglichen Standort für ein Gateway, welcher durch das Gitter erzeugt wird.

Für die Visualisierungen der Testergebnisse wird jeweils der Durchschnitt von zehn verschiedenen zufällig gewählten Testinstanzen für dieselbe Anzahl an Sensoren dargestellt. Bei den Graphen für die Laufzeiten ist zu beachten, dass es sich bei der Zeit-Achse um eine logarithmische Skala handelt.

Die einzelnen Algorithmen wurden auf dem High-Performance-Computing-Cluster der Julius-Maximilians-Universität Würzburg ausgeführt. Dafür wurden jeweils 2 Kerne und (vor allem für das angepasste ILP) bis zu 30 GB Speicher[3] reserviert. Zusätzlich gab es pro Testinstanz ein Zeitlimit von einer halben Stunde. Ist der Algorithmus binnen dieser halben Stunde noch nicht terminiert, wurde das bisher beste Ergebnis ausgegeben. Um die nachfolgenden Tests mit einer gesamten Laufzeit von über 22 Tagen durchzuführen, wurde jedes Instanz-Algorithmus-Paar als ein einzelner Arbeitsauftrag auf dem Cluster ausgeführt. Dadurch konnten die insgesamt 1470 Arbeitsaufträge des Clusters in kleinen Gruppen parallel durchgeführt werden. Dies ermöglichte diese breite Menge an Tests.

5.2 Teststrecken

Zur besseren Gruppierung der Tests gab es verschiedene Teststrecken. Die einzelnen Teststrecken unterscheiden sich in den Größen der Eingabe und der maximalen Sendereichweite zwischen Sensoren und Gateways. Insgesamt wurden vier Teststrecken durchgeführt. Dabei waren zwei Teststrecken für große Instanzen, wobei eine Teststrecke eine kleine und die andere eine große maximale Sendereichweite hatte. Die übrigen beiden Teststrecken beinhalten kleinere Instanzen für den besseren Vergleich mit den ILPs, wobei erneut eine Teststrecke eine kleine und die andere eine große maximale Sendereichweite hatte. Die Sendereichweiten betragen dabei 750 und 200 Meter. Da Würzburg als Stadt eher bergig

[3] Die 30 GB entsprechen nicht notwendigerweise dem benötigten Speicher, jedoch kam es bei dem ILP für weniger Speicher oft zu einem Out-Of-Memory-Error.

ist, ist nicht davon auszugehen, dass die in Kapitel 1 beschriebenen 2km erreicht werden können. Daher wurden zwei Werte unter 1km gewählt. Zusätzlich wird das Limit für die Anzahl an Sensoren pro Gateway nur dann gesetzt, wenn der Voronoi-Local-Search-Algorithmus gegen einen Algorithmus antritt, welcher dieses Limit ebenfalls einhalten kann. Ansonsten kann jedes Gateway beliebig viele Sensoren versorgen. Falls das Limit gesetzt wurde, wurde es stets auf einen Wert von 300 gesetzt. Der Voronoi-Local-Search-Algorithmus wurde dabei immer mit $k = 3$ ausgeführt. Aufgrund der Beschleunigungstechnik aus Kapitel 4.1 ergeben sich daher drei Graphen in den Visualisierungen. Der Laufzeitgraph mit der Beschreibung $k = 1$ beschreibt dabei, nach welcher Zeit der Algorithmus aufgehört hat Ersetzungen für $k = 1$ zu suchen und angefangen hat mit $k = 2$ zu suchen. Analog dazu zeigt der Graph für die Güte der Lösung mit der Beschreibung $k = 1$ wie groß die Lösungsmenge zu diesem Zeitpunkt war.

Teststrecke 1 Die erste Teststrecke soll einen groben Überblick für die verschiedenen Laufzeiten bieten. Daher werden Instanzen mit einer Größe zwischen 4.000 und 20.000 Sensoren getestet. Die jeweils nächstgrößere Instanz-Größe hat dabei immer 2.000 Sensoren mehr. Die maximale Sendereichweite liegt durchgehend bei 750 Metern. Da die Anzahl der möglichen Standorte für Gateways abhängig von der Sendereichweite ist, kann in Abbildung 5.3 die durchschnittliche Anzahl der möglichen Gateway-Standorte für bestimmte Instanz-Größen betrachtet werden. Der Voronoi-Local-Search-Algorithmus hat in dieser Teststrecke keine Verbesserungen während des Schritts für $k = 3$ innerhalb des Zeitlimits gefunden. Daher wird dieser Schritt in den Graphen für die Lösungsgüte zusammen mit $k = 2$ aufgeführt.

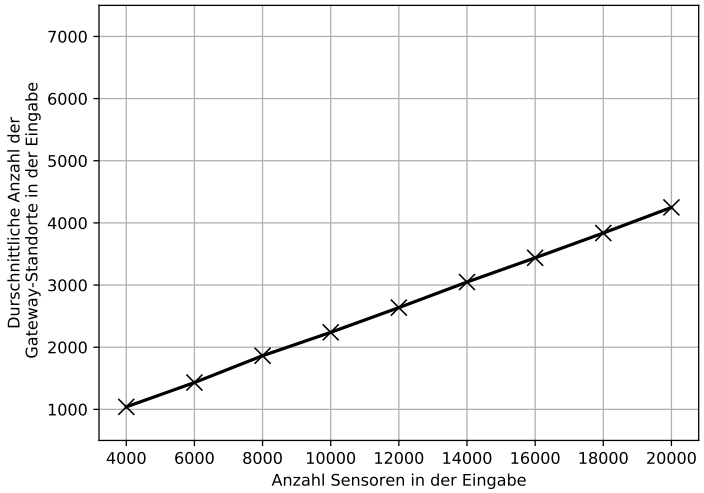


Abb. 5.3: Durchschnittliche Anzahl der Gateway-Standorte für bestimmte Instanz-Größen der Teststrecke 1.

Teststrecke 2 Für die zweite Teststrecke wurde lediglich die maximale Sendereichweite auf 200 Meter verkleinert. Dadurch wächst die Eingabegröße der möglichen Gateway-

Standorte im Gegensatz zu Teststrecke 1 (siehe Abbildung 5.4). Die Instanz-Größen sind weiterhin zwischen 4.000 und 20.000 mit einer Schrittgröße von 2.000 Sensoren. Es wurden erneut zufällig Sensoren aus dem gesamten Datensatz ausgewählt, bis die gewünschte Sensoren-Anzahl erreicht wurde. Auch in dieser Teststrecke hat der Voronoi-Local-Search-Algorithmus keine Verbesserungen für $k = 3$ innerhalb des Zeitlimits gefunden. Daher wurde auch hier die Kurve für $k = 3$ zur besseren Übersicht zusammen mit $k = 2$ in dem Graphen für die Lösungsgüte aufgeführt.

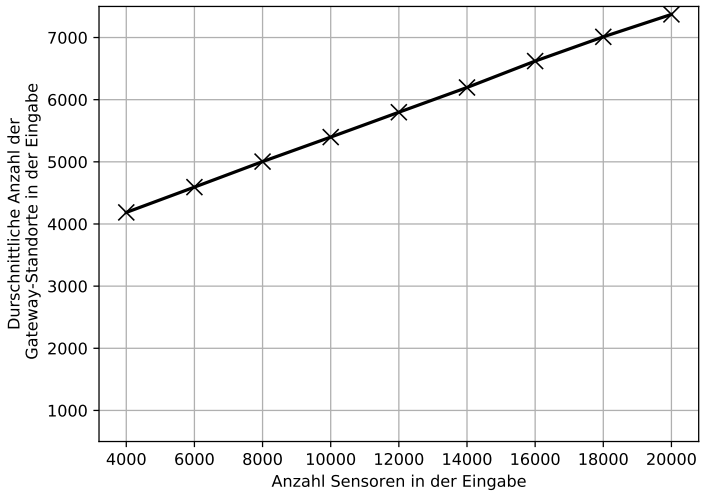


Abb. 5.4: Durchschnittliche Anzahl der möglichen Gateway-Standorte für bestimmte Instanz-Größen der Teststrecke 2.

Teststrecke 3 Die Teststrecke drei soll die Laufzeit und Güte des Voronoi-Local-Search-Algorithmus im Besonderen mit denen der ILPs vergleichen. Daher handelt es sich um kleinere Instanzen als bei den vorherigen Teststrecken. Dabei werden Eingaben mit einer Größen zwischen 500 und 4.000 Sensoren getestet. Die verschiedenen Instanz-Größen sind dabei 500 Sensoren voneinander entfernt. Die maximale Sendereichweite liegt auch hier wie schon in Teststrecke 1 bei 750 Metern. Die durchschnittliche Anzahl der möglichen Gateway-Standorte für eine bestimmte Anzahl an Sensoren in der Eingabe kann in Abbildung 5.5 betrachtet werden.

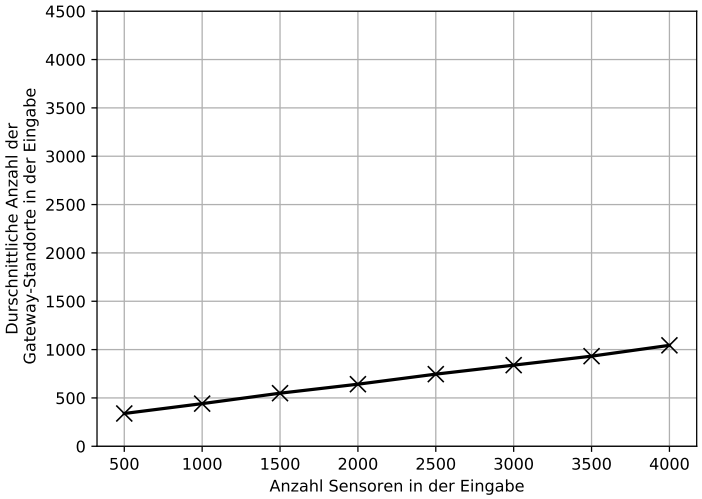


Abb. 5.5: Durchschnittliche Anzahl der möglichen Gateway-Standorte für bestimmte Instanz-Größen der Teststrecke 3.

Teststrecke 4 Die letzte Teststrecke hat dieselben Instanz-Größen wie Teststrecke 3. Lediglich die maximale Sendereichweite wurde erneut auf 200 Meter verringert. Abbildung 5.6 beschreibt, wie viele mögliche Gateway-Standorte durchschnittlich für die jeweilige Anzahl Sensoren in der Eingabe erstellt wurden.

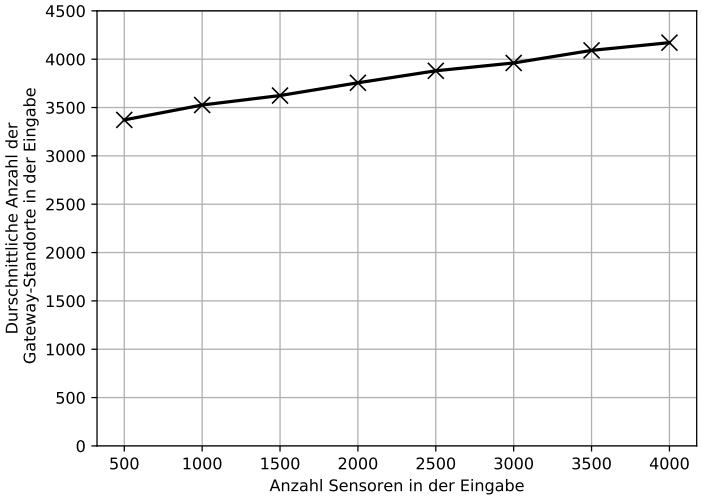


Abb. 5.6: Durchschnittliche Anzahl der möglichen Gateway-Standorte für bestimmte Instanz-Größen der Teststrecke 4.

5.3 Analyse der Beschleunigungstechniken

Im Folgenden werden die Beschleunigungstechniken getestet, indem der Voronoi-Local-Search-Algorithmus gegen den Voronoi-Local-Search-Algorithmus ohne diese eine Beschleunigungstechnik antritt. Somit kann jeder ausschlaggebende Unterschied in den Tests genau auf diese Beschleunigungstechnik zurückgeführt werden.

5.3.1 Inkrementierung der Anzahl der betrachteten Objekte

Für diese Beschleunigungstechnik (beschrieben in Kapitel 4.1) wurde Teststrecke 1 verwendet, da diese über ein großes Intervall verschiedener Instanz-Größen reicht. Zudem war in Teststrecke 2 kein nennenswerter Unterschied zu sehen.

In Abbildung 5.7 kann die Laufzeit des vollständigen Voronoi-Local-Search-Algorithmus für die verschiedenen Schritte der Variablen k und die Laufzeit des Voronoi-Local-Search-Algorithmus, welcher schon beim Start mit $k = 3$ beginnt, betrachtet werden.

Darin ist zu sehen, dass beide Algorithmen letztendlich das Zeitlimit von einer halben Stunde (1.800 Sekunden) erreichen. Somit kann allein aufgrund der Laufzeit nicht von einer Verbesserung ausgegangen werden. Betrachtet man jedoch zusätzlich noch Abbildung 5.8 können Rückschlüsse gezogen werden.

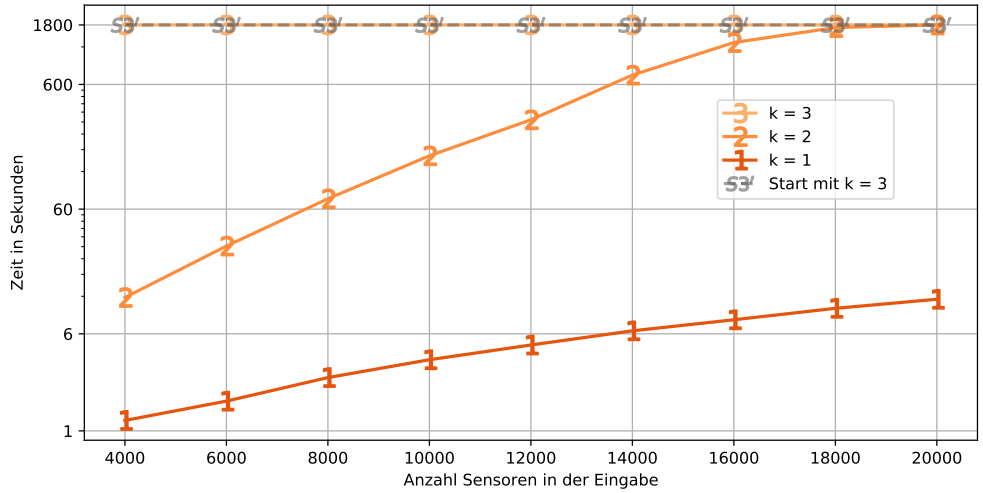


Abb. 5.7: Laufzeiten des Voronoi-Local-Search-Algorithmus für verschiedene k und des Voronoi-Local-Search-Algorithmus für ein zu Beginn festes k (Teststrecke 1). Beide Algorithmen erreichen bereits ab 4.000 Sensoren das Zeitlimit einer halben Stunde.

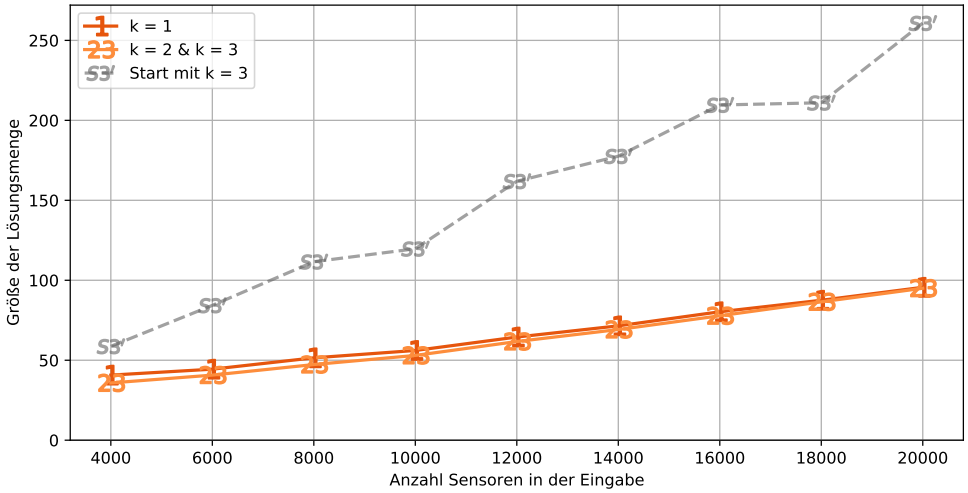


Abb. 5.8: Qualität der Lösung des Voronoi-Local-Search-Algorithmus für verschiedene k und des Voronoi-Local-Search-Algorithmus für ein ab Beginn festes k (Teststrecke 1).

Obwohl beide Algorithmen nach dem Zeitlimit abgebrochen wurden, hat der normale Voronoi-Local-Search-Algorithmus mit $k = 1$ bereits nach wenigen Sekunden bessere Lösungen als der Algorithmus ohne Beschleunigungstechnik gefunden. Des Weiteren kann davon ausgegangen werden, dass der Algorithmus ohne Beschleunigungstechnik nach dem Zeitlimit nicht schneller als der Voronoi-Local-Search-Algorithmus ist, da zu diesem Zeitpunkt die beiden Algorithmen mit $k = 3$ identisch arbeiten.

Daher kann daraus gefolgert werden, dass die Beschleunigungstechnik aus Kapitel 4.1 den Algorithmus in der Praxis wirklich beschleunigt.

5.3.2 Einführung des Gitters

Um zu testen, ob die Beschleunigungstechnik aus Kapitel 4.2 den Algorithmus auch wirklich beschleunigt und nicht etwa durch den Extraaufwand verlangsamt, tritt Voronoi-Local-Search-Algorithmus im Folgenden gegen eine Version des Voronoi-Local-Search-Algorithmus ohne Gitter an.

In Abbildung 5.9 können die Laufzeiten der beiden Algorithmen für die Teststrecke 1 betrachtet werden.

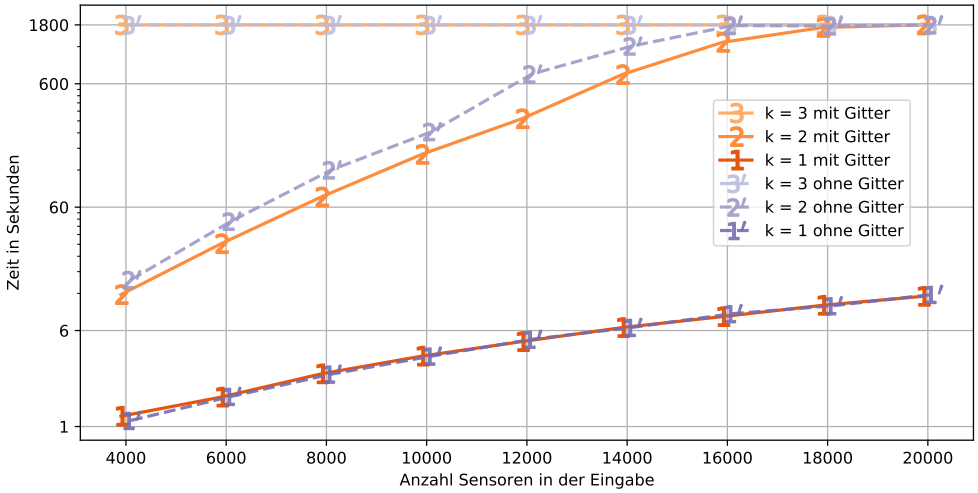


Abb. 5.9: Laufzeiten des Voronoi-Local-Search-Algorithmus und des Voronoi-Local-Search-Algorithmus ohne Gitter für verschiedene k (Teststrecke 1). Beide Algorithmen für $k = 3$ erreichen bereits ab 4.000 Sensoren das Zeitlimit einer halben Stunde.

Auf den ersten Blick ist zu erkennen, dass die Laufzeiten für die Schritte $k = 1$ und $k = 3$ identisch sind. Für $k = 1$ liegt dies daran, dass das Gitter für diesen Schritt, wie in Kapitel 4.2 beschrieben, keine Wirkung hat. Daraus kann jedoch geschlussfolgert werden, dass das Gitter die Laufzeit des Algorithmus durch zusätzliche Operationen nicht merklich verlangsamt. Für die Schritte mit $k = 3$ sind die Laufzeiten identisch, da beide Algorithmen das Laufzeitlimit erreichen und abgebrochen werden.

Die Laufzeit des Schritts für $k = 2$ zeigt jedoch, dass der Algorithmus durch die Beschleunigungstechnik schneller wurde. Das gleiche Ergebnis würde man für $k = 3$ erwarten, wenn der Algorithmus nicht aufgrund eines Zeitlimits terminiert wird. Des Weiteren ist, wie in Kapitel 4.2 beschrieben, zu prüfen, ob der Algorithmus durch das Gitter eine schlechtere Lösung generiert. Dafür kann in Abbildung 5.10 die Größe der jeweiligen Lösungsmenge betrachtet werden. Darin ist zu erkennen, dass die Größen der Lösungsmengen für beide Algorithmen sehr nah beieinander liegen. Kleine Abweichungen können dabei auf die Zufallskomponente des Algorithmus zurückgeführt werden.

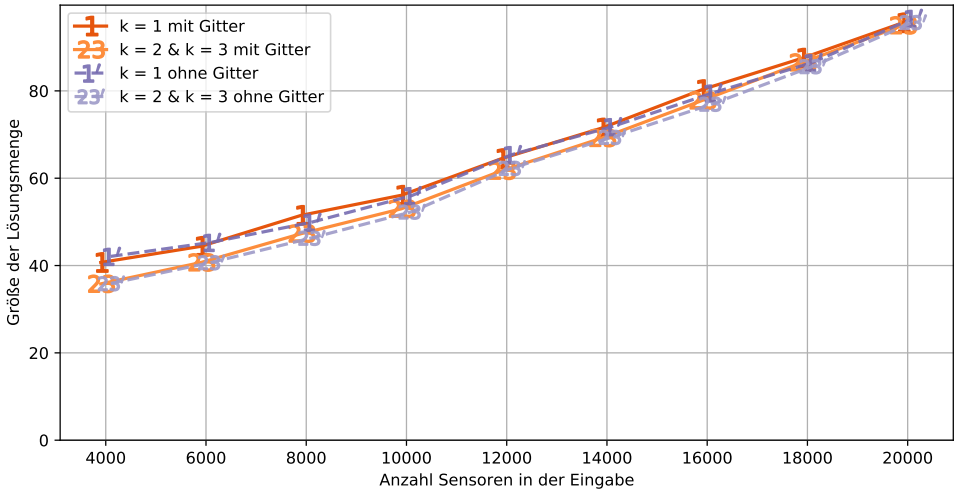


Abb. 5.10: Qualität der Lösung des Voronoi-Local-Search-Algorithmus für verschiedene k mit und ohne Gitter (Teststrecke 1).

Somit kann festgehalten werden, dass das Gitter den Algorithmus in Teststrecke 1 schneller aber nicht merklich schlechter macht. Zudem wurde Teststrecke 2 ebenfalls für den Voronoi-Local-Search-Algorithmus ohne Gitter ausgeführt, da eine Veränderung an der maximalen Sendereichweite auch das Gitter verändert. Dadurch, dass in Teststrecke 2 die maximale Sendereichweite verkleinert wurde, wurde das Gitter auch feiner. In Abbildung 5.11 können die Laufzeiten der Algorithmen und in Abbildung 5.12 die Größen der Lösungsmenge für Teststrecke 2 betrachtet werden.

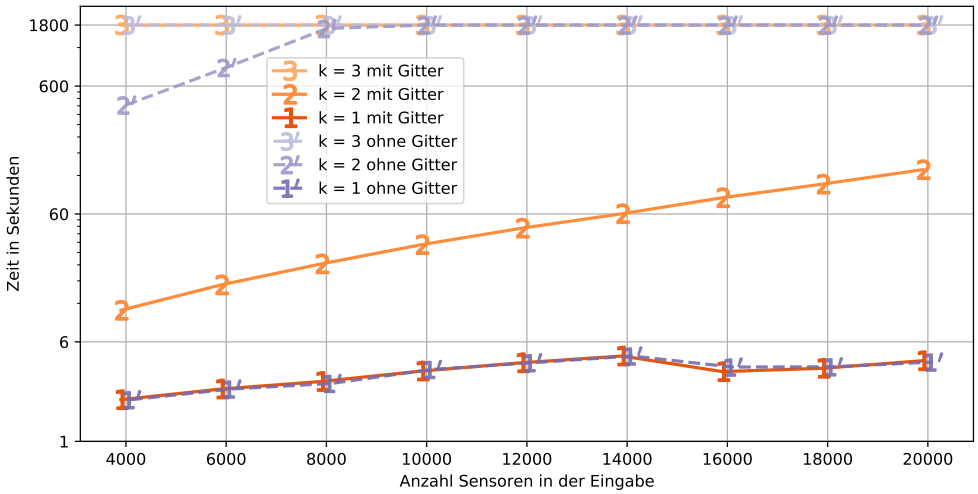


Abb. 5.11: Laufzeiten des Voronoi-Local-Search-Algorithmus und des Voronoi-Local-Search-Algorithmus ohne Gitter für verschiedene k (Teststrecke 2). Beide Algorithmen für $k = 3$ erreichen bereits ab 4.000 Sensoren das Zeitlimit einer halben Stunde.

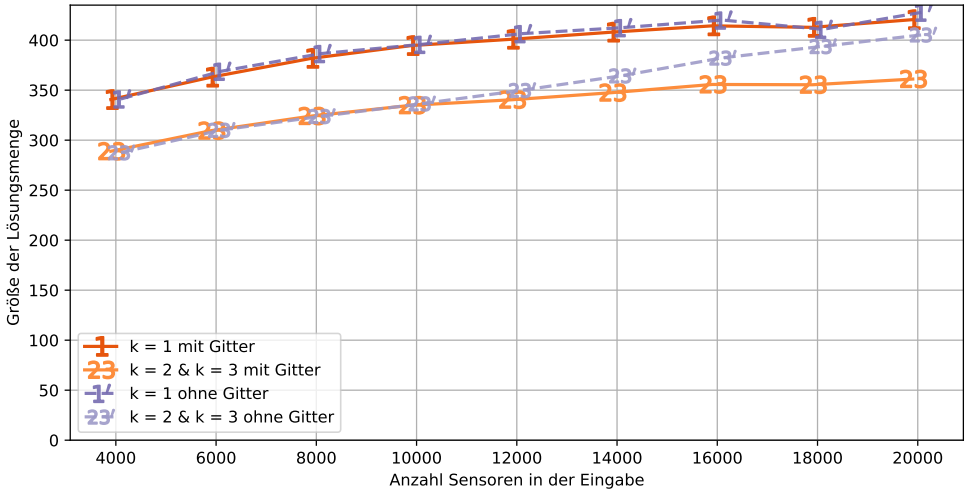


Abb. 5.12: Qualität der Lösung des Voronoi-Local-Search-Algorithmus für verschiedene k mit und ohne Gitter (Teststrecke 2).

Erneut sind die Laufzeiten der Algorithmen für $k = 1$ sinnvollerweise fast identisch. Für den Wert von $k = 2$ zeigt sich mit einem feineren Gitter auch eine bessere Beschleunigung des Algorithmus durch das Gitter.

Die Qualität der Lösung ist auch für das feinere Gitter fast gleich. Da jedoch der Algorithmus ohne Gitter schon bei 10.000 Sensoren in der Eingabe und im $k = 2$ Schritt das Zeitlimit erreicht, wird ab dieser Instanz-Größe die Lösung auch stetig schlechter als die Lösung des normalen Voronoi-Local-Search-Algorithmus.

Somit kann abschließend gesagt werden, dass, je feiner das Gitter durch die maximale Sendereichweite gewählt werden kann, desto schneller wird der Voronoi-Local-Search-Algorithmus durch das Gitter. Außerdem gibt der Voronoi-Local-Search-Algorithmus trotz des Gitters keine merklich größeren Lösungen aus.

5.4 Vergleich zwischen dem `dnet`-Algorithmus und dem Voronoi-Local-Search-Algorithmus

Um den Voronoi-Local-Search-Algorithmus mit dem `dnet`-Algorithmus zu vergleichen, wurden die Teststrecken 1 und 2 für beide Algorithmen durchgeführt. Da der `dnet`-Algorithmus nur das Set-Cover-Problem und nicht auch das Voronoi-Cover-Problem lösen kann, wurde für den Voronoi-Local-Search-Algorithmus das Limit für die Anzahl der Sensoren pro Gateway auf unendlich gesetzt.

In den Abbildungen 5.13 und 5.14 können die Ergebnisse der Teststrecke 1 betrachtet werden. Darin ist zu erkennen, dass der `dnet`-Algorithmus zwar um ein vielfaches schneller ist, jedoch ist die Lösungsmenge um einen Faktor von 1,58 bis zu 1,84 größer als die des Voronoi-Local-Search-Algorithmus für $k = 2$.

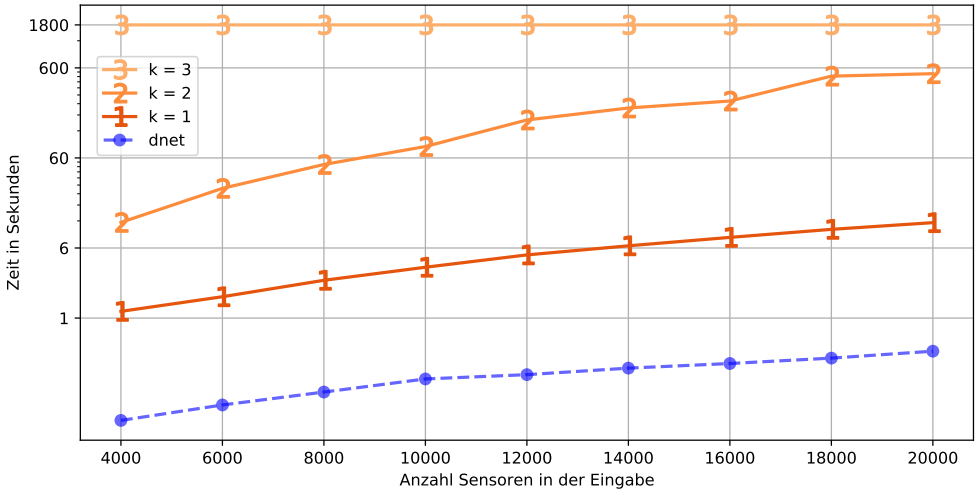


Abb. 5.13: Laufzeiten des Voronoi-Local-Search-Algorithmus für verschiedene k und des `dnet`-Algorithmus im Vergleich (Teststrecke 1).

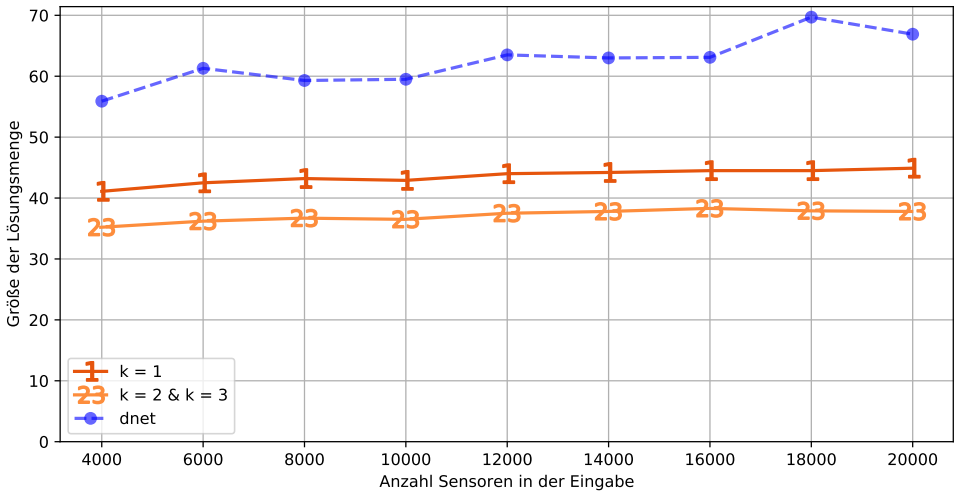


Abb. 5.14: Qualität der Lösung des Voronoi-Local-Search-Algorithmus für verschiedene k und des `dnet`-Algorithmus (Teststrecke 1).

Auch in Teststrecke 2 (siehe Abbildungen 5.15 und 5.16) zeichnet sich ein ähnliches Bild ab. Dabei ist der `dnet`-Algorithmus erneut sehr schnell, aber die Lösungsmenge ist ebenfalls etwa 1,7 Mal größer als die des Voronoi-Local-Search-Algorithmus. Dieses Ergebnis war, wie schon in Kapitel 2.2.2 angeschnitten, zu erwarten. Die Priorität des Voronoi-Local-Search-Algorithmus liegt stärker in der Qualität der Lösung als in einer kürzeren Laufzeit.

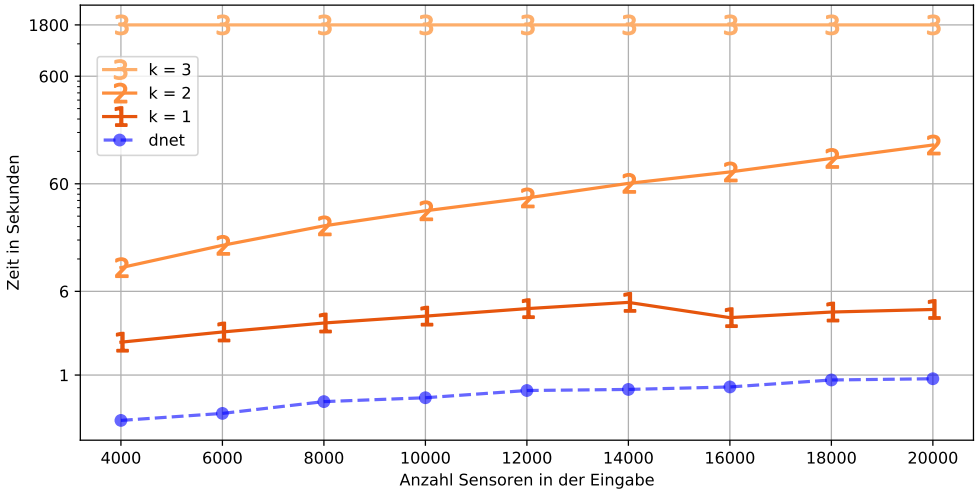


Abb. 5.15: Laufzeiten des Voronoi-Local-Search-Algorithmus für verschiedene k und des `dnet`-Algorithmus im Vergleich (Teststrecke 2).

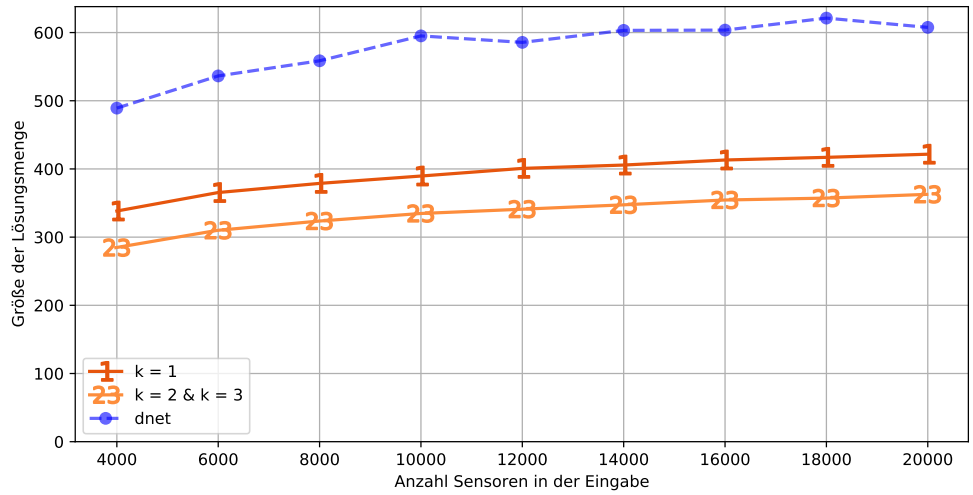


Abb. 5.16: Qualität der Lösung des Voronoi-Local-Search-Algorithmus für verschiedene k und des `dnet`-Algorithmus (Teststrecke 2).

5.5 Vergleich zwischen den ILPs und dem Voronoi-Local-Search-Algorithmus

Um die Lösungsgüte des Voronoi-Local-Search-Algorithmus abschätzen zu können, tritt der Voronoi-Local-Search-Algorithmus im Folgenden gegen die ILPs aus den Kapiteln 2.1.1 und 3.5 an. Das Besondere an den ILPs ist, dass falls ein ILP vor dem Zeitlimit terminiert, es die optimale Lösung für das Problem berechnet hat. Daher kann in diesen Fällen

gut abgeschätzt werden, wie viel schlechter die Lösung des Voronoi-Local-Search-Algorithmus im Vergleich zur optimalen Lösung ist. Für Instanzen, in denen ein ILP aufgrund des Zeitlimits abgebrochen wurde, konnte das ILP nicht unbedingt die optimale Lösung finden oder diese als optimale Lösung verifizieren. Somit kann für solche Instanzen abgeschätzt werden, ab welcher Instanz-Größe der Voronoi-Local-Search-Algorithmus besser als das ILP wird – vorausgesetzt, die Lösung wird vor Ablauf des Zeitlimits von einer halben Stunde erwartet. Soll eine Lösung schneller errechnet werden, wird der Voronoi-Local-Search-Algorithmus schon für kleinere Instanzen zur besseren Wahl. Da der Vergleich mit den ILPs der wahrscheinlich interessanteste Vergleich ist, wurde für die ILPs jede der vier Teststrecken durchgeführt.

5.5.1 Vergleich mit dem ILP für das Set-Cover-Problem

Zunächst tritt der Voronoi-Local-Search-Algorithmus gegen das ILP für das Set-Cover--Problem an. Somit existiert keine Limitierung für die Anzahl der Sensoren pro Gateway, wodurch es sich hierbei um das geometrische Set-Cover-Problem handelt. In Abbildung 5.17 können die Laufzeiten für den Voronoi-Local-Search-Algorithmus und das ILP für Teststrecke 1 betrachtet werden. Die dazugehörigen Lösungsgüten können Abbildung 5.18 entnommen werden. In diesen Abbildungen ist zu sehen, dass das ILP für das Set-Cover-Problem Instanzen mit bis zu 14.000 Sensoren in der Eingabe[4] innerhalb des Zeitlimits optimal lösen kann. Dabei liegt die optimale Lösung meist bei knapp unter 30 Gateways. Der Voronoi-Local-Search-Algorithmus (mit $k = 2$) kann innerhalb von etwa einem Zehntel der Zeit eine Lösung finden, welche durchschnittlich etwa 1,3 Mal schlechter ist. Jedoch produziert das ILP des Set-Cover-Problems für Instanzen mit 20.000 Sensoren in der Eingabe weiterhin bessere Lösungen – Zumindest innerhalb eines Zeitlimits von einer halben Stunde.

[4] Zusammen mit den möglichen Gateway-Standorten liegt die gesamte Eingabegröße daher bei etwa 17.000 Elementen.

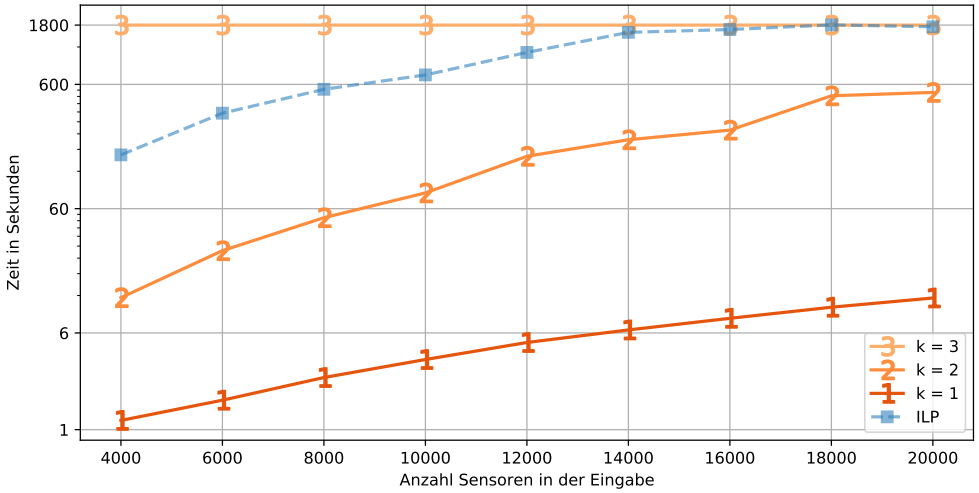


Abb. 5.17: Laufzeiten des Voronoi-Local-Search-Algorithmus für verschiedene k und des ILPs für das Set-Cover-Problem (Teststrecke 1).

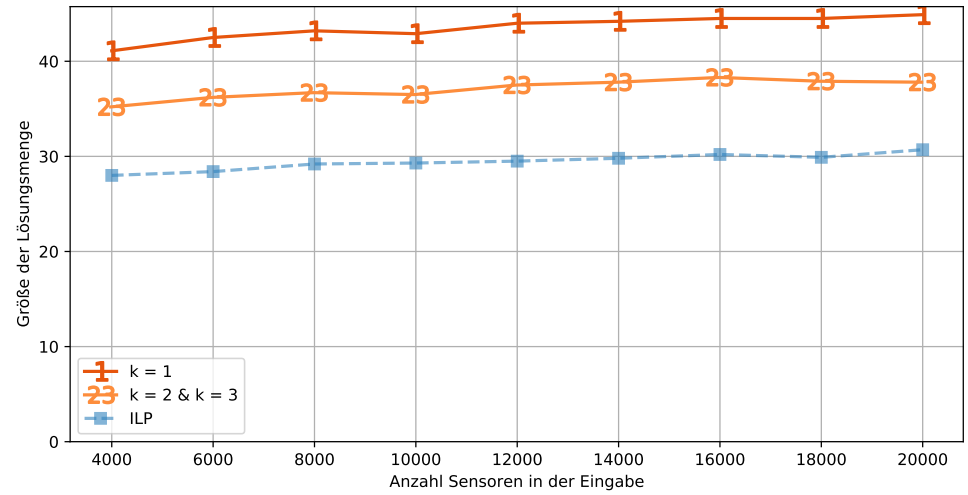


Abb. 5.18: Qualität der Lösung des Voronoi-Local-Search-Algorithmus für verschiedene k und des ILPs für das Set-Cover-Problem (Teststrecke 1).

Die Graphen für die Laufzeit und die Güte der Lösung der beiden Algorithmen in Teststrecke 2 können in Abbildung 5.19 beziehungsweise 5.20 gesehen werden. In dieser Teststrecke kann das ILP innerhalb des Zeitlimits kaum eine optimale Lösung finden beziehungsweise bestätigen. Das wird wahrscheinlich an der größeren Anzahl an möglichen Gateway-Standorten liegen. Dennoch findet das ILP auch hier bessere Lösungen als der Voronoi-Local-Search-Algorithmus. Für $k = 2$ sind die Lösungen des Voronoi-Local-Search-Algorithmus meist um einen Faktor von 1,17 schlechter. Jedoch konnten diese Lösungen auch in einem Bruchteil der Zeit errechnet werden.

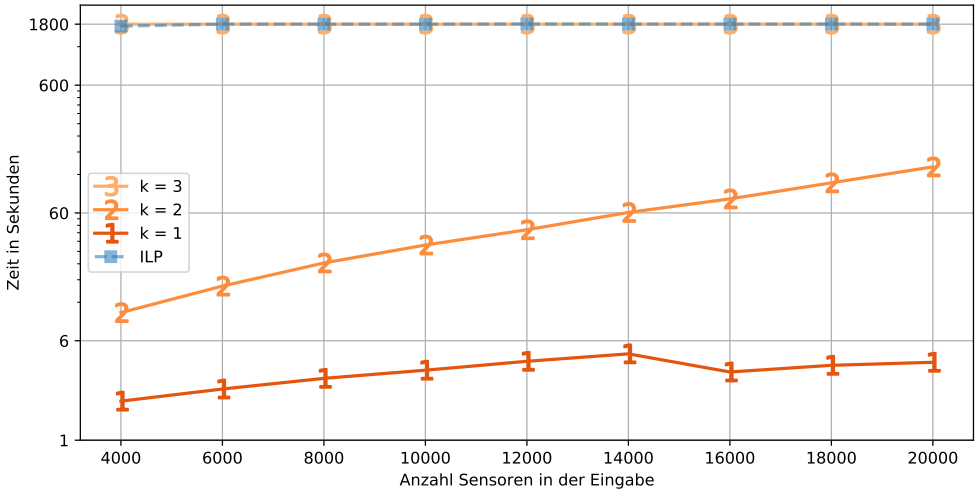


Abb. 5.19: Laufzeiten des Voronoi-Local-Search-Algorithmus für verschiedene k und des ILPs für das Set-Cover-Problem (Teststrecke 2). Beide Algorithmen erreichen bereits ab 4.000 Sensoren das Zeitlimit einer halben Stunde.

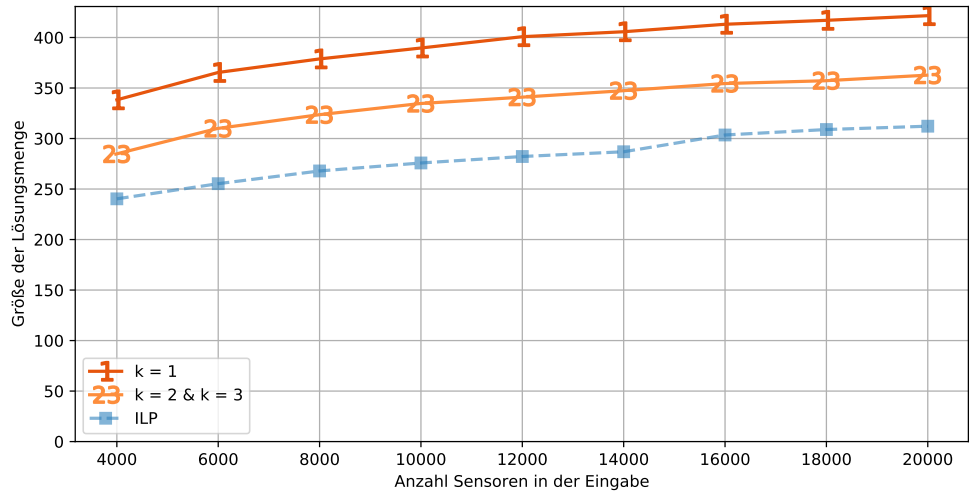


Abb. 5.20: Qualität der Lösung des Voronoi-Local-Search-Algorithmus für verschiedene k und des ILPs für das Set-Cover-Problem (Teststrecke 2).

In Teststrecke 3 (siehe Abbildung 5.21 und 5.22) hat das ILP einen ungewöhnlichen Laufzeitgraphen. Klar ersichtlich ist, dass das ILP bei einer Eingabe von 2.000 Sensoren schneller ist als mit 1.500 Sensoren und weniger. Möglicherweise werden für verschiedene Instanz-Größen auch verschiedene Heuristiken für das ILP verwendet und die gewählte Heuristik für Instanz-Größen über 2.000 Sensoren hat in diesem Fall einen Vorteil gegenüber dem Lösungsverfahren für die kleineren Instanzen. Da jedoch pro Sensoren-Anzahl zehn verschiedene Instanzen berechnet wurden, ist es unwahrscheinlich, dass es sich um

reinen Zufall handelt. Die Laufzeit des Voronoi-Local-Search-Algorithmus weist hingegen einen regelmäßigen Verlauf auf. Insgesamt ist der Voronoi-Local-Search-Algorithmus für $k = 2$ auch in dieser Teststrecke durchschnittlich schneller. Jedoch ist dafür erneut die Lösung des Voronoi-Local-Search-Algorithmus um einen Faktor von etwa 1,2 größer als die des ILPs.

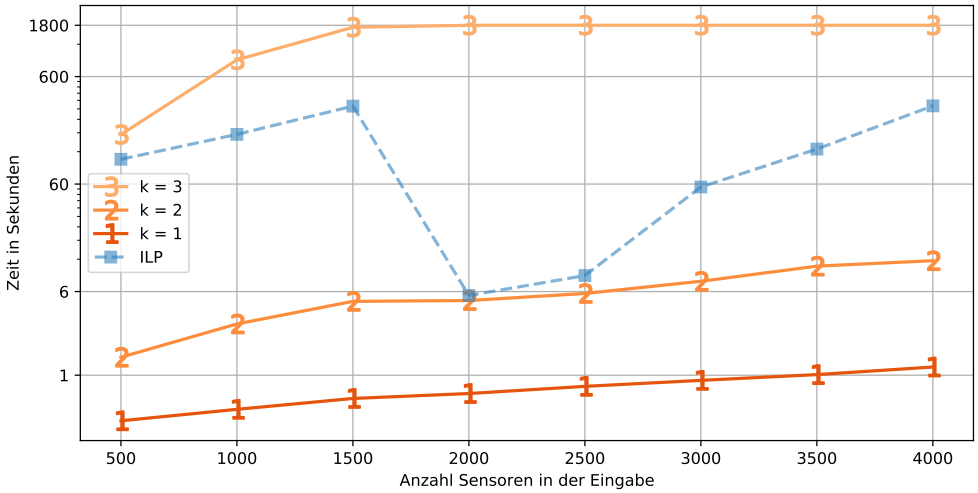


Abb. 5.21: Laufzeiten des Voronoi-Local-Search-Algorithmus für verschiedene k und des ILPs für das Set-Cover-Problem (Teststrecke 3).

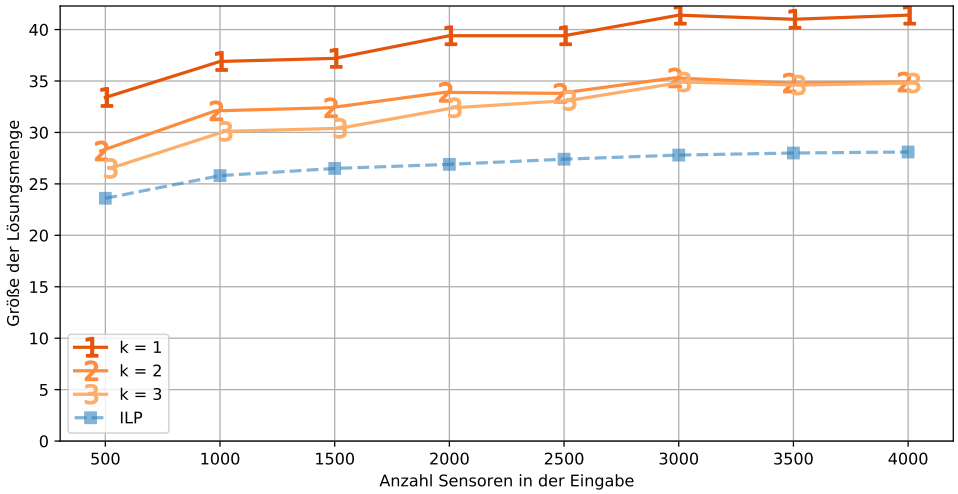


Abb. 5.22: Qualität der Lösung des Voronoi-Local-Search-Algorithmus für verschiedene k und des ILPs für das Set-Cover-Problem (Teststrecke 3).

In Teststrecke 4 (siehe Abbildung 5.23 und 5.24) schneidet der Voronoi-Local-Search-Algorithmus für $k = 2$, im Vergleich zu den anderen Teststrecken, in der Laufzeit

besonders gut ab. Der Voronoi-Local-Search-Algorithmus ist für $k = 2$ bis zu 180-mal schneller als das ILP. Dabei ist die Lösung des Voronoi-Local-Search-Algorithmus für $k = 2$ erneut um einen Faktor von etwa 1,2 größer als die des ILPs.

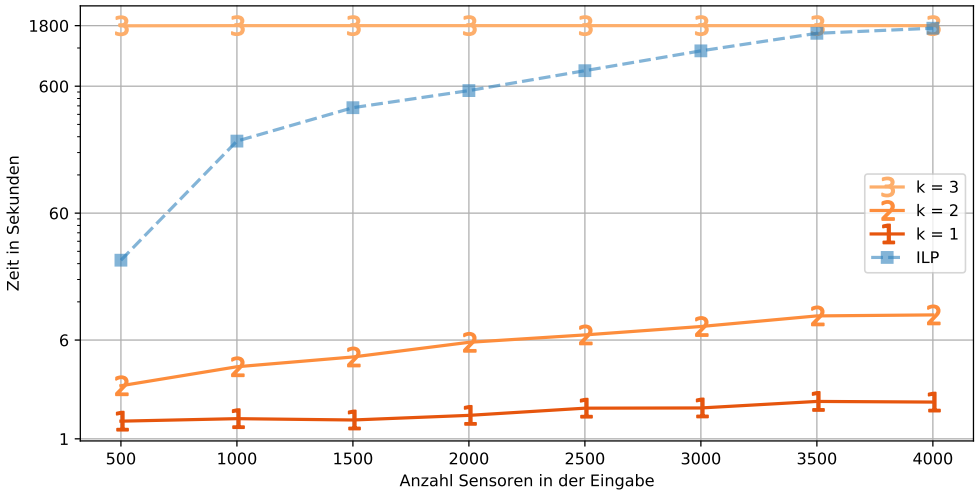


Abb. 5.23: Laufzeiten des Voronoi-Local-Search-Algorithmus für verschiedene k und des ILPs für das Set-Cover-Problem (Teststrecke 4).

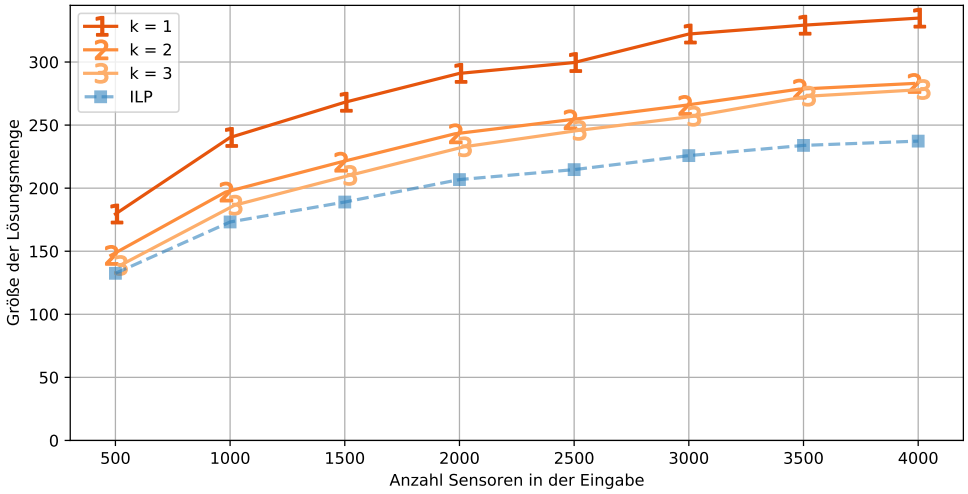


Abb. 5.24: Qualität der Lösung des Voronoi-Local-Search-Algorithmus für verschiedene k und des ILPs für das Set-Cover-Problem (Teststrecke 4).

Zusammenfassend kann gesagt werden, dass der Voronoi-Local-Search-Algorithmus mit $k = 2$ das Set-Cover-Problem für die vorliegenden Instanz-Größen mit einem durchschnittlichen Approximationsfaktor von etwa 1,25 löst.

5.5.2 Vergleich mit dem ILP für das Voronoi-Cover-Problem

Im Folgenden werden die vier Teststrecken auch für das ILP des Voronoi-Cover-Problems durchgeführt und mit dem Voronoi-Local-Search-Algorithmus verglichen.

In Abbildung 5.25 kann die Laufzeit und in Abbildung 5.26 die Größe der Lösungsmenge der beiden Algorithmen für Teststrecke 1 betrachtet werden. Der Graph der Lösungsgüte des ILPs hört dabei bei 14.000 Sensoren in der Eingabe auf, da das ILP für mehr Sensoren keine einzige Lösung mehr innerhalb des Zeitlimits von einer halben Stunde generieren konnte. Jedoch sind die Lösungen schon ab einer Sensor-Anzahl von 4.000 im Vergleich zu den Lösungen des Voronoi-Local-Search-Algorithmus aufgrund ihrer Größe unbrauchbar.

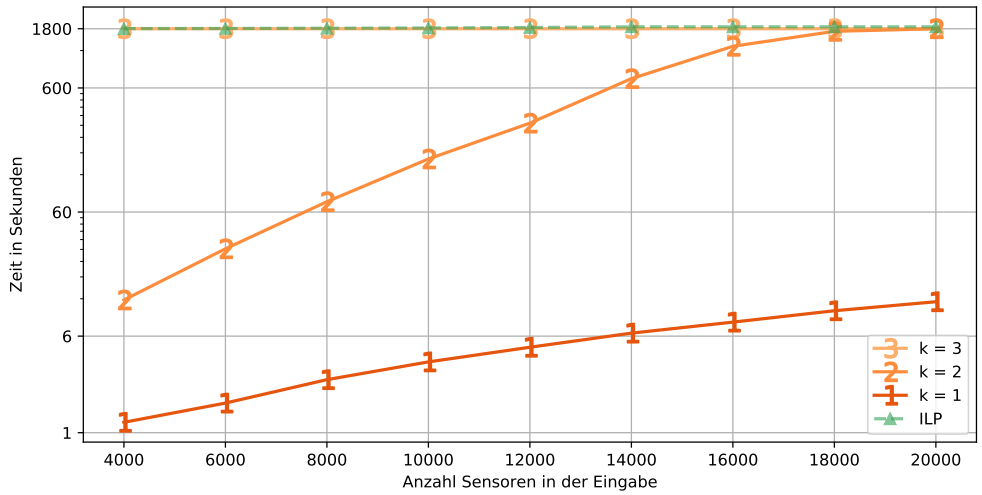


Abb. 5.25: Laufzeiten des Voronoi-Local-Search-Algorithmus für verschiedene k und des ILPs für das Voronoi-Cover-Problem (Teststrecke 1). Beide Algorithmen erreichen bereits ab 4.000 Sensoren das Zeitlimit einer halben Stunde.

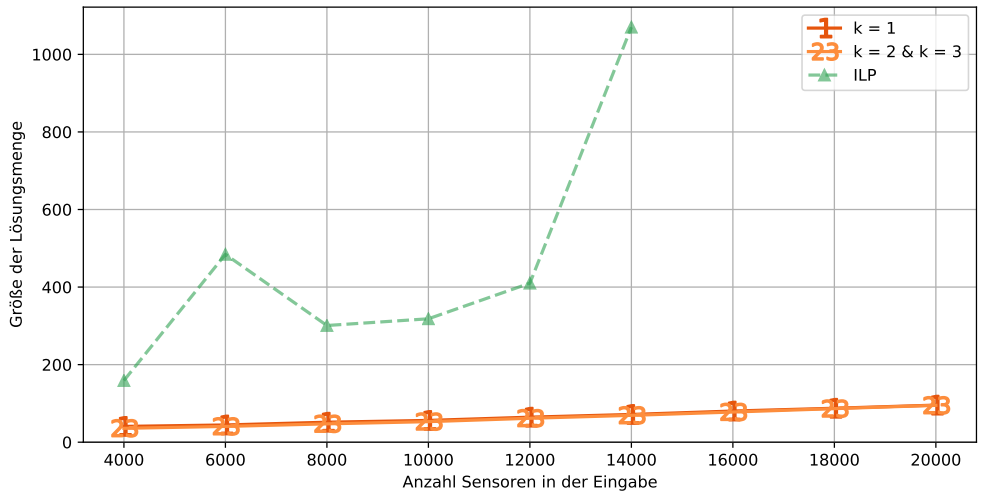


Abb. 5.26: Qualität der Lösung des Voronoi-Local-Search-Algorithmus für verschiedene k und des ILPs für das Voronoi-Cover-Problem (Teststrecke 1).

In Teststrecke 2 (siehe Abbildungen 5.27 und 5.28) zeichnet sich ein ähnliches Bild ab. Zwar konnte das ILP für Eingaben mit 4.000 Sensoren trotz Erreichen des Zeitlimits noch gute Lösungen generieren, jedoch wird das ILP ab 6.000 Sensoren schlechter als der Voronoi-Local-Search-Algorithmus und ab 8.000 Sensoren wird auch hier die Lösung des ILPs mehr als doppelt so groß wie die des Voronoi-Local-Search-Algorithmus.

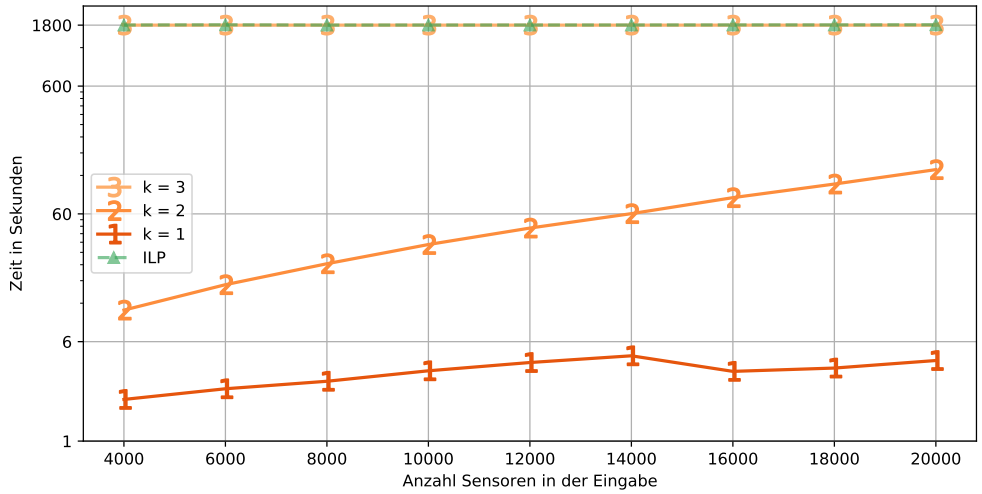


Abb. 5.27: Laufzeiten des Voronoi-Local-Search-Algorithmus für verschiedene k und des ILPs für das Voronoi-Cover-Problem (Teststrecke 2). Beide Algorithmen erreichen bereits ab 4.000 Sensoren das Zeitlimit einer halben Stunde.

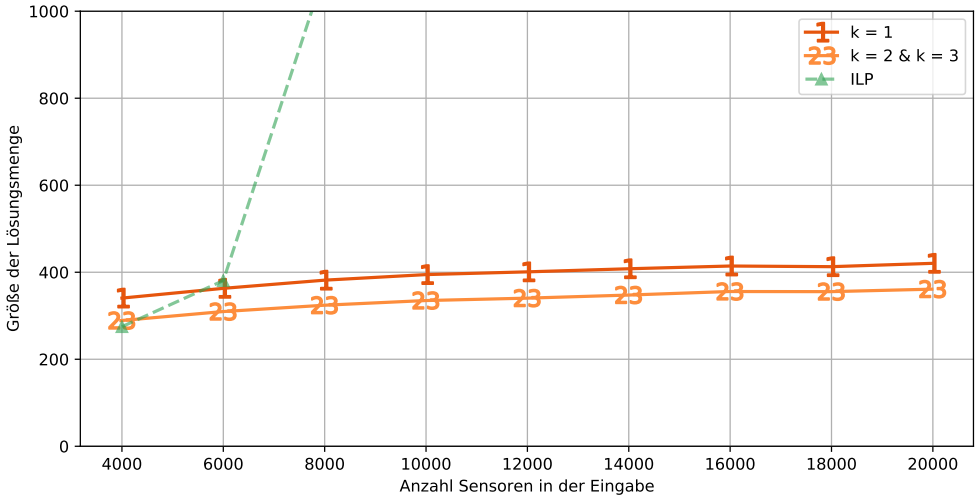


Abb. 5.28: Qualität der Lösung des Voronoi-Local-Search-Algorithmus für verschiedene k und des ILPs für das Voronoi-Cover-Problem (Teststrecke 2).

In den Abbildungen 5.29 und 5.30 können die Ergebnisse der Teststrecke 3 betrachtet werden. Bis zu einer Eingabe von 1.500 Sensoren konnte das ILP eine optimale Lösung für das Voronoi-Cover-Problem innerhalb des Zeitlimits berechnen. Jedoch war die durchschnittliche Lösung des Voronoi-Local-Search-Algorithmus für $k \geq 2$ und eine feste Sensoren-Anzahl in der Eingabe maximal um einen Faktor von 1,22 größer als die des ILPs. Ab einer Sensoren-Anzahl über 2.500 hat der Voronoi-Local-Search-Algorithmus bereits für $k = 2$ eine kleinere Lösungsmenge als das ILP.

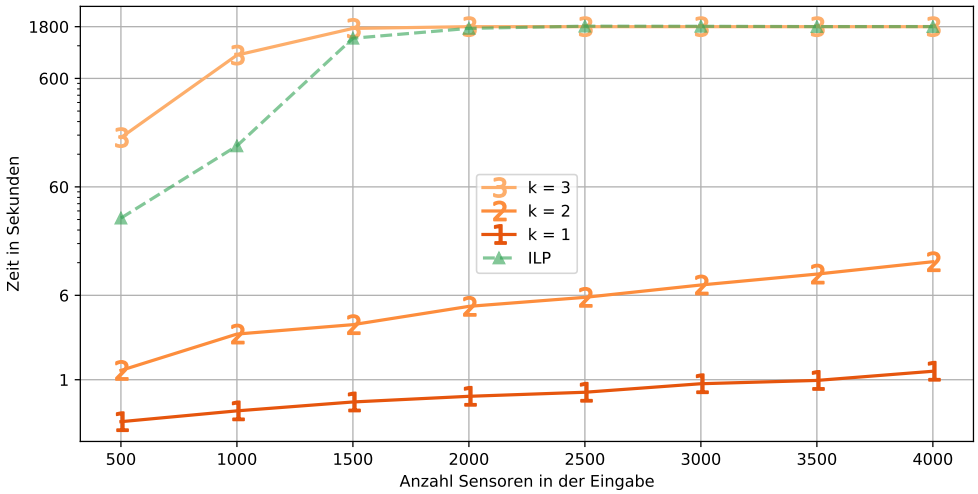


Abb. 5.29: Laufzeiten des Voronoi-Local-Search-Algorithmus für verschiedene k und des ILPs für das Voronoi-Cover-Problem (Teststrecke 3).

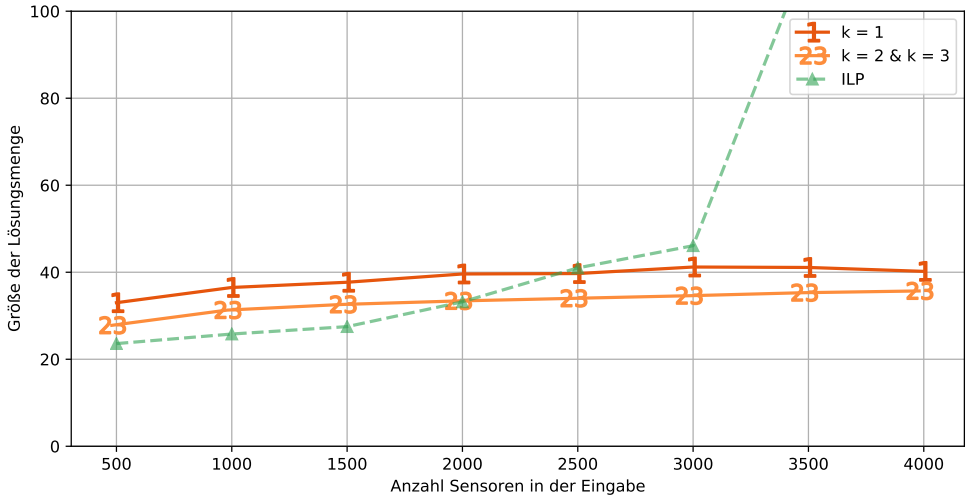


Abb. 5.30: Qualität der Lösung des Voronoi-Local-Search-Algorithmus für verschiedene k und des ILPs für das Voronoi-Cover-Problem (Teststrecke 3).

Lediglich in Teststrecke 4 (siehe Abbildungen 5.31 und 5.32) konnte das ILP für das Voronoi-Cover-Problem über die gesamte Teststrecke eine kleinere Lösung als der Voronoi-Local-Search-Algorithmus finden. Jedoch waren die Lösungen des Voronoi-Local-Search-Algorithmus mit $k = 2$ maximal um einen Faktor von 1,25 schlechter als die des ILPs. Dabei war der Voronoi-Local-Search-Algorithmus mit $k = 2$ auch hier bis zu 180-mal schneller.

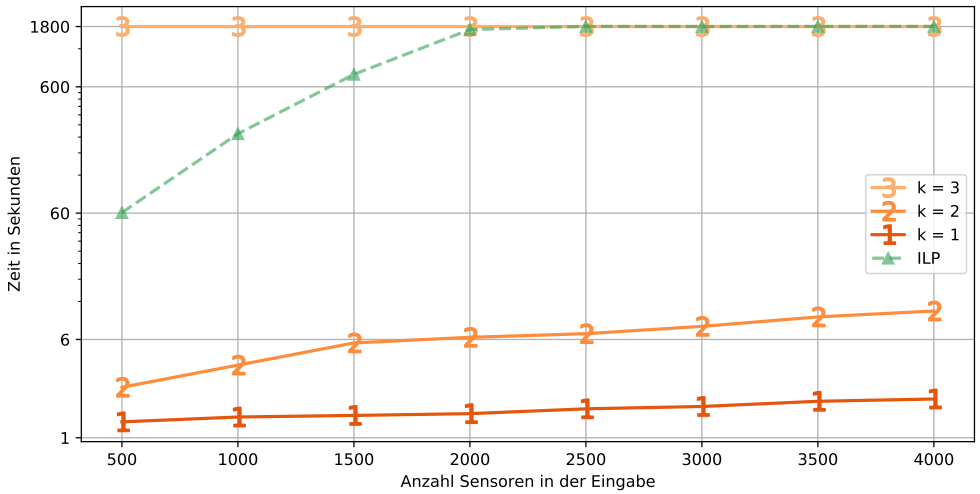


Abb. 5.31: Laufzeiten des Voronoi-Local-Search-Algorithmus für verschiedene k und des ILPs für das Voronoi-Cover-Problem (Teststrecke 4).

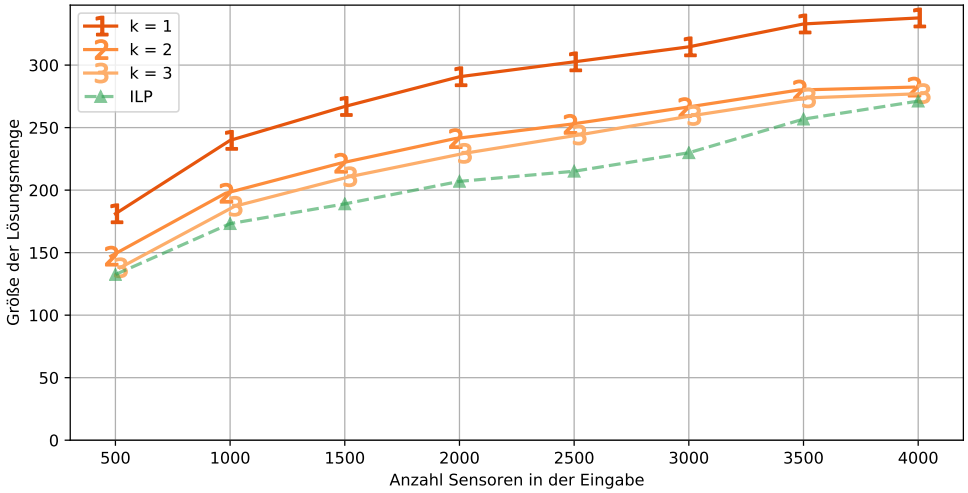


Abb. 5.32: Qualität der Lösung des Voronoi-Local-Search-Algorithmus für verschiedene k und des ILPs für das Voronoi-Cover-Problem (Teststrecke 4).

Im Gesamtüberblick über die vier Teststrecken kann gesagt werden, dass der Voronoi-Local-Search-Algorithmus das Voronoi-Cover in den vorliegenden Testinstanzen nie schlechter als um einen Faktor von 1,25 approximiert hat. Generell gilt, je größer die Sensoren-Anzahl in der Eingabe und je größer die maximale Sendereichweite gesetzt wird, desto früher wird der Voronoi-Local-Search-Algorithmus für $k = 2$ sowohl in der Laufzeit als auch in der Qualität der Lösung besser als das ILP.

6 Fazit

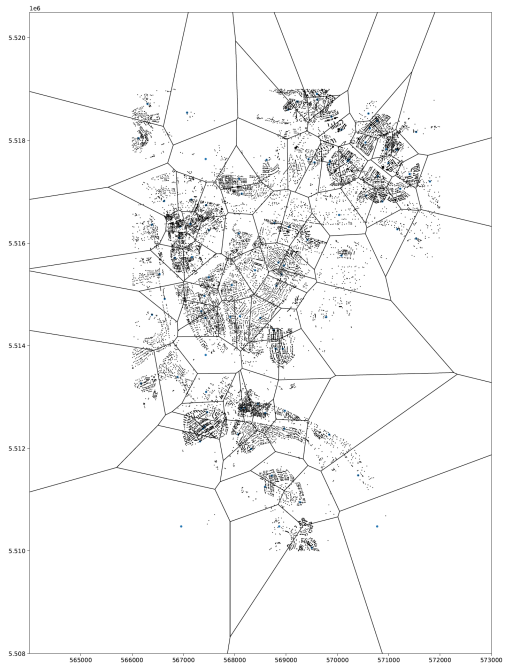
Aus den gesammelten Tests aus Kapitel 5 geht hervor, dass der Voronoi-Local-Search-Algorithmus für die getesteten Instanz-Größen und Sendereichweiten mit einem Wert von $k = 2$ einen guten Abtausch zwischen Laufzeit und Lösungsqualität bietet. Der Voronoi-Local-Search-Algorithmus mit $k = 3$ hat für größere Instanzen innerhalb des Zeitlimits meist keine Verbesserung mehr gefunden. Mit $k = 1$ ist der Voronoi-Local-Search-Algorithmus zwar sehr schnell, jedoch konnte ein zusätzlicher Schritt mit $k = 2$ die Lösungsmenge in den meisten Fällen noch merklich verkleinern. Zudem hat der Voronoi-Local-Search-Algorithmus mit $k = 2$ die optimale Lösung des Voronoi-Cover-Problems im Durchschnitt mit einem Faktor von etwa 1,24 approximiert. Außerdem hat der Voronoi-Local-Search-Algorithmus das Set-Cover-Problem um einen Faktor von etwa 1,7 mal besser approximiert als der `dnet`-Algorithmus, welcher nach Angaben der Autoren einen durchschnittlichen Approximationsfaktor von 1,3 hat. Nach unseren Tests hat der Voronoi-Local-Search-Algorithmus mit $k = 2$ einen durchschnittlichen Approximationsfaktor von etwa 1,25 für das Set-Cover-Problem. Mithilfe der Tests konnten wir weiterhin bestätigen, dass die entwickelten Beschleunigungsmethoden aus Kapitel 4 den Voronoi-Local-Search-Algorithmus für $k \geq 2$ schneller gemacht haben, ohne dass sich dabei die Qualität der Lösung merklich verschlechtert hat. Jedoch könnten zukünftig auch noch Tests mit anderen Städten oder auch mit Modellierungen ländlicher Gebiete und zusätzlich anderen Instanz-Größen durchgeführt werden.

Ein weiterer Vorteil des Voronoi-Local-Search-Algorithmus, welcher bislang nicht benannt wurde, ist, dass die Robustheit des Netzwerks von dem Benutzer selbst gewählt werden kann. Die Robustheit eines Netzwerks beschreibt, wie viele Sensoren in der Zukunft noch neu in das Netzwerk eingefügt werden können, ohne dass neue Gateways aufgestellt oder bestehende Gateways neu platziert werden müssen. Angenommen ein Gateway könnte in der Realität 300 Sensoren versorgen. Dann könnte der Benutzer des Algorithmus das Limit auf 150 Sensoren pro Gateway setzen. Dadurch könnten in der Zukunft theoretisch genau so viele Sensoren neu in das Netzwerk eingespeist werden, wie zur Ausführung des Algorithmus bereits im Netzwerk vorhanden waren. Dies gilt jedoch nur in der Annahme, dass die neu eingefügten Sensoren in der Ebene ähnlich verteilt sind wie die bereits bestehenden Sensoren.

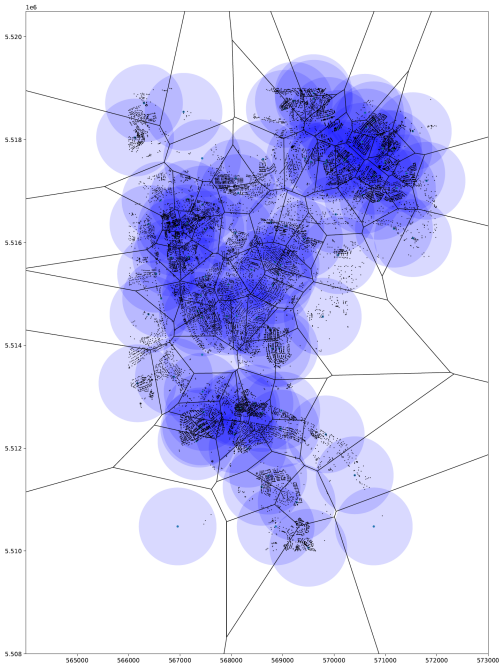
Abschließend kann in Abbildung 6.1 eine Instanz mit 20.000 Sensoren in der Eingabe und dessen Lösung des Voronoi-Local-Search-Algorithmus für $k = 2$ betrachtet werden. Auf den Achsen sind die Koordinaten innerhalb des UTM32-Koordinatensystems aufgetragen. Für diese Instanz wurde ein Sensoren-Limit pro Gateway von 300 und eine Sendereichweite von 750 Metern verwendet. Die Lösungsmenge dieser Instanz beinhaltet 96 Elemente. Mithilfe der beiden Abbildungen 6.1c und 6.1d ist zu sehen, dass mehrere Kreise in Abbildung 6.1c meist nur dann überlappen, wenn die Voronoi-Zellen zusammen

über 300 Sensoren enthalten würden, oder die Kreise jeweils andere Sensoren abdecken, welche von sonst keinem Kreis abgedeckt werden. Das spiegelt die Hauptfunktion des Algorithmus wieder.

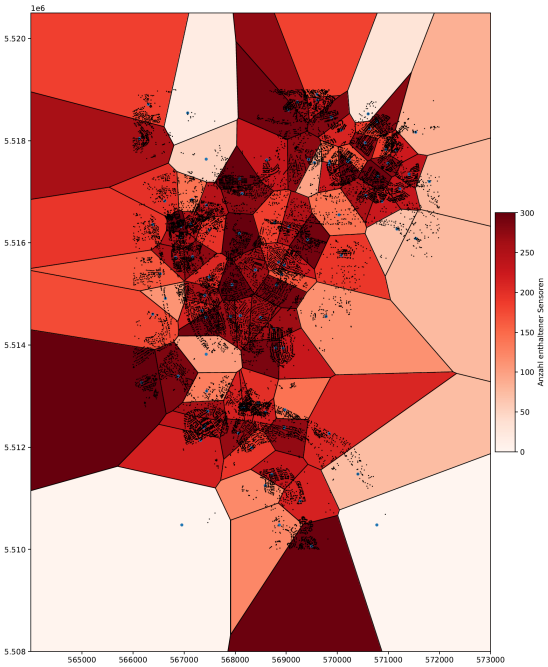
(a) Sensoren-Menge der Eingabe



(b) Lösung inklusive Voronoi-Diagramm



(c) Lösung inklusive Sendereichweiten und Voronoi-Diagramm



(d) Lösung inklusive Voronoi-Diagramm als Heatmap

Abb. 6.1: Darstellung einer Sensoren-Eingabe eines Würzburg-Modells mit 20.000 Sensoren. Die Lösung des Voronoi-Local-Search-Algorithmus enthält dabei 96 Gateways, welche jeweils maximal 300 Sensoren enthalten können. Die maximale Sendereichweite beträgt 750 Meter.

7 Ausblick

Im Folgenden werden weitere Betrachtungsmöglichkeiten für die Problemstellung dieser Arbeit oder mögliche Erweiterungen für den Voronoi-Local-Search-Algorithmus beschrieben.

7.1 Geometrisches Set-Multi-Cover

Eine mögliche Erweiterung zum Set-Cover-Problem ist, dass jeder Punkt durch mindestens m Kreise aus der Lösungsmenge abgedeckt sein muss. Dieses Problem wird auch als geometrisches Set-Multi-Cover-Problem bezeichnet.

Besonders für die Robustheit des Netzwerks wäre ein solches Set-Multi-Cover von Vorteil. Sollte ein Gateway ausfallen, stehen einem Sensor noch $m - 1$ andere Gateways zur Kommunikation zu Verfügung.

Falls ein Kreis dabei auch öfters verwendet werden darf, ist eine naive Lösung für das geometrische Set-Multi-Cover die Lösung für das geometrische Set-Cover zu berechnen und jeden Kreis aus der Lösungsmenge m mal zu verwenden. Wenn man dabei für das geometrische Set-Cover einen Algorithmus mit einer $\mathcal{O}(a)$-Approximation für ein beliebiges a benutzt, erhält man für das geometrische Set-Multi-Cover-Problem eine $\mathcal{O}(m \cdot a)$-Approximation. Die Laufzeit ist dabei identisch zur Laufzeit des geometrischen Set-Cover Algorithmus.

Ein theoretisch besserer Algorithmus ist, das geometrische Set-Cover für die Eingabe zu lösen und daraufhin alle Punkte, welche schon n mal abgedeckt werden, aus der Eingabe zu löschen. Falls jeder Kreis nur einmal verwendet werden darf, können zusätzlich auch die benutzen Kreise aus der Eingabe gelöscht werden. Anschließend wird der geometrische Set-Cover Algorithmus auf der neuen Eingabe ausgeführt. Das wird solange wiederholt, bis jeder Punkt n mal abgedeckt ist.

7.2 Abstandsminimierung

Eine andere Betrachtungsweise des Problems wäre es, den Abstand zwischen Sensoren und Gateways möglichst zu minimieren. Dabei könnte der Abstand auf verschiedene Arten minimiert werden.

- Der durchschnittliche (quadratische) Abstand von Sensoren zu dem nächsten Gateway.
- Der maximale (quadratische) Abstand eines Sensors zum nächsten Gateway.

Dadurch würden die Sensoren und Gateways (im Durchschnitt) weniger Energie für die Kommunikation benötigen. Eine solche Abstandsminimierung wird meist durch Heuristiken erreicht [LGS+15]. Jedoch wurde dies (nach unserem Wissensstand) noch nicht in Zusammenhang mit einem Set-Cover-Ansatz erforscht.

7.3 Einbeziehung von Relief-Daten der Region

Zusätzlich zur Sendereichweite der Gateways und Sensoren könnte mittels Relief-Daten gearbeitet werden, um die Erreichbarkeit zwischen Sensoren und Gateways praxisnäher zu modellieren. Dabei muss ein Sensor nicht nur in Sendereichweite eines Gateways liegen, sondern sie müssen sich zusätzlich noch über die Luftlinie sehen können. Ein Gateway und ein Sensor sehen sich dabei, wenn zwischen dem Sensor und dem Gateway keine Erhöhung der Landschaft liegt, sodass eine direkte Linie zwischen den beiden durch die Erde verlaufen würde. Für diese Arbeit wurde eine solche Möglichkeit für die Modellierung implementiert, jedoch ist die Annahme, dass die Gateways durch Kreise repräsentiert werden können, dadurch nicht geeignet. Daher müsste die Wahl des Ausgangsalgorithmus für eine solche Instanz neu überdacht werden.

7.4 Ausführliche Tests zur Robustheit des Netzwerks

Wie bereits in Kapitel 6 beschrieben, kann in der Theorie die Robustheit des Netzwerks mithilfe des Voronoi-Local-Search-Algorithmus vom Benutzer gewählt werden. Um diese Theorie jedoch auch in der Praxis zu simulieren, könnten ausführliche Tests nach dem folgenden Schema durchgeführt werden. Sei das Limit der Sensoren-Anzahl pro Gateway in der Praxis ℓ und $x \geq 0$ die mithilfe der Tests zu bestimmende Größe für die Skalierung des Sensoren-Limits für die Gateways. Der Testdatensatz wird zufällig in zwei Gruppen von Sensoren A und B aufgeteilt. Dabei sei $g_A = |A|/(|A| + |B|)$. Auf der ersten Gruppe der Sensoren wird der Voronoi-Local-Search-Algorithmus mit einem Sensoren-Limit von $\ell \cdot (g_A - x)$ ausgeführt. Danach wird die zweite Gruppe in die Ebene eingefügt. Erfüllt das Netzwerk die Bedingungen für das Voronoi-Cover-Problem mit einem Sensoren-Limit von ℓ, war der Test erfolgreich. Anhand des Wertes von x, für welchen eine gewünschte Prozentzahl der Tests erfolgreich ist, kann demnach bestimmt werden, wie robust das Netzwerk in der Praxis wirklich ist. Leider konnten keine ausführlichen Testreihen mehr hierzu durchgeführt werden. Jedoch haben wir einen ersten Test mit der Instanz aus Abbildung 6.1 mit $x = 0$ aufgeführt. Darin existieren 20.000 Sensoren, welche zufällig in zwei Gruppen mit je 10.000 Sensoren aufgeteilt wurden. Das Ausführen des Voronoi-Local-Search-Algorithmus für eine der beiden Sensoren-Gruppen mit einem ℓ von 150 ergab, dass 94 Gateways benötigt werden, um jeden Sensor abzudecken. Diese Lösung kann auch in Abbildung 7.1 betrachtet werden. Nachdem die andere Sensoren-Gruppe wieder in die Modellierung aufgenommen wurde, wurden 5 dieser Gateways mehr als 300 Sensoren zugewiesen. Diesen 5 Gateways wurden 305, 326, 336, 308 und 301 Sensoren zugewiesen. Die Modellierung mit beiden Sensoren-Gruppen und $\ell = 300$ kann in Abbildung 7.2 betrachtet werden. Dabei wurden die Voronoi-Zellen der Gateways, welchen zu viele Sen-

soren zugeordnet wurden, blau eingefärbt. Somit müsste in ausführlichen Tests der Wert von x erhöht werden, damit dieser Test erfolgreich werden würde. Jedoch ist es auch gut möglich, dass der Test mit $x = 0$ zufälligerweise erfolgreich sein könnte, da sowohl die Wahl der Sensoren-Gruppen zufällig ist, als auch der Voronoi-Local-Search-Algorithmus eine Zufallskomponente beinhaltet. Zudem hat sich die generelle Farbverteilung über die Voronoi-Zellen nicht stark verändert, was die Behauptung, dass mithilfe des Wertes von ℓ die zukünftige Robustheit des Netzwerks angepasst werden kann, bekräftigt.

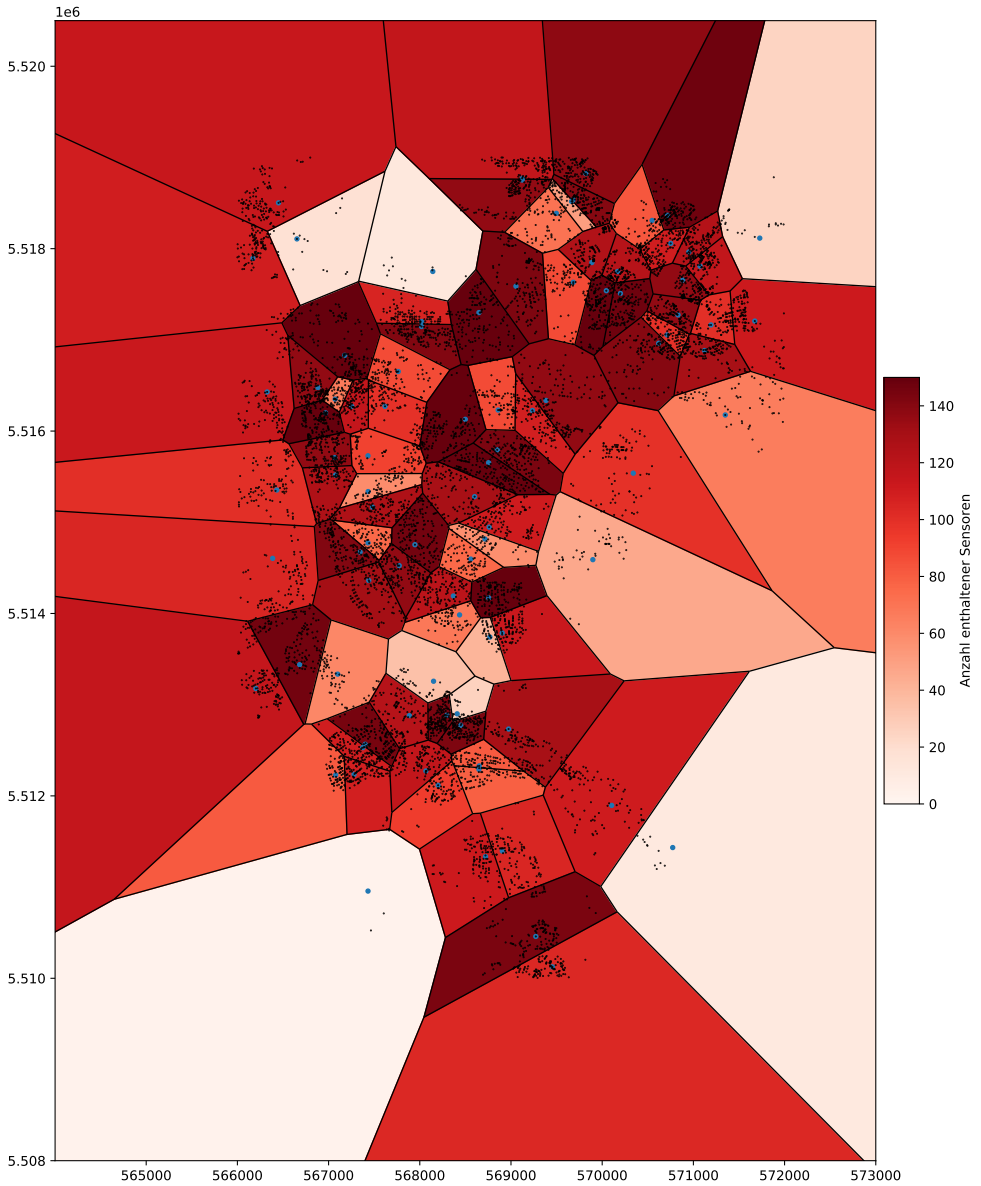


Abb. 7.1: Heatmap über die Lösung des Voronoi-Local-Search-Algorithmus mit einer Sensoren-Gruppe aus 10.000 Sensoren und einem ℓ von 150.

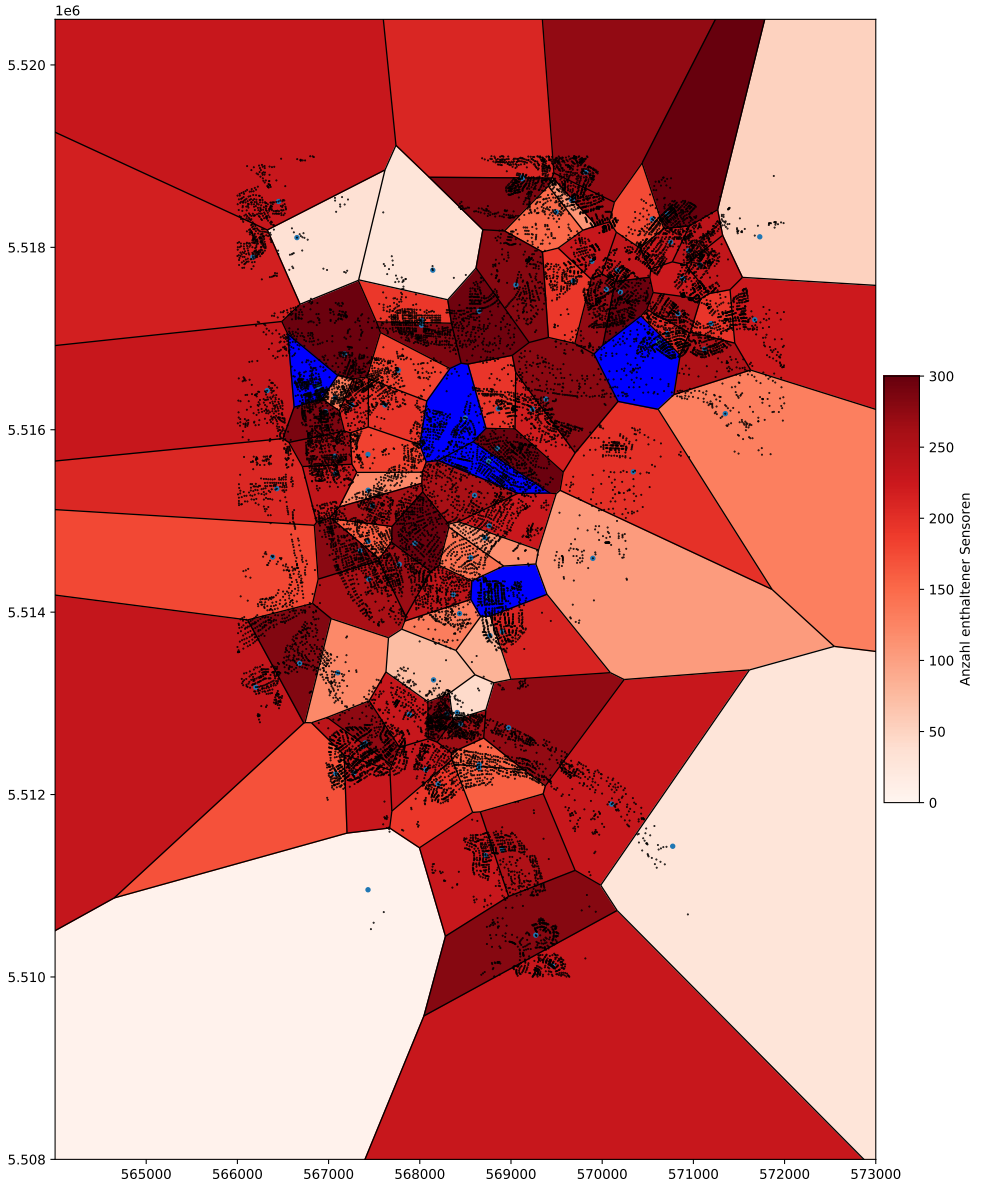


Abb. 7.2: Heatmap über die Lösung des Voronoi-Local-Search-Algorithmus mit beiden Sensoren-Gruppe und einem ℓ von 300. Blaue Flächen sind dabei die Voronoi-Zellen der Gateways mit mehr als 300 zugeordneten Sensoren.

Danksagung

Zum Schluss möchte ich mich bei einigen Personen für Ideen und Betreuung bedanken. Dr. Joachim Spoerhase danke ich für die Idee der Beschleunigungstechnik aus Kapitel 4.1. Jan Pfister möchte ich für die fruchtbaren Diskussionen zu dieser Arbeit danken. Bei Frank Loh und Prof. Dr. Tobias Hoßfeld möchte ich mich für die Problemstellung und für die Einführung in die technischen Aspekte bedanken.

Besonderer Dank gilt Prof. Dr. Alexander Wolff und Johannes Zink für die vielen Betreuungstreffen und das stets hilfreiche Feedback.

Literaturverzeichnis

[AP20] Pankaj Agarwal und Jiangwei Pan: Near-linear algorithms for geometric hitting sets and set covers. *Discrete & Computational Geometry*, 63(2):1–23, 2020. https://doi.org/10.1007/s00454-019-00099-6.

[BMR18] Norbert Bus, Nabil H. Mustafa und Saurabh Ray: Practical and efficient algorithms for the geometric hitting set problem. *Discrete Applied Mathematics*, 240:25–32, 2018. https://doi.org/10.1016/j.dam.2017.12.018, Linear Optimization.

[Chv79] Vašek Chvátal: A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979. https://doi.org/10.1287/moor.4.3.233.

[Fei98] Uriel Feige: A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998. https://doi.org/10.1145/285055.285059.

[FPT81] Robert J. Fowler, Michael S. Paterson und Steven L. Tanimoto: Optimal packing and covering in the plane are NP-complete. *Information Processing Letters*, 12(3):133–137, 1981. https://doi.org/10.1016/0020-0190(81)90111-3.

[JMM+03] Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi und Vijay V. Vazirani: Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM*, 50(6):795–824, 2003. https://doi.org/10.1145/950620.950621.

[JMS02] Kamal Jain, Mohammad Mahdian und Amin Saberi: A new greedy approach for facility location problems. In: *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, STOC '02, Seite 731–740. Association for Computing Machinery, 2002. https://doi.org/10.1145/509907.510012.

[Kar72] Richard M. Karp: *Reducibility among combinatorial problems*, Seiten 85–103. Springer US, 1972. https://doi.org/10.1007/978-1-4684-2001-2_9.

[LGS+15] Stanislav Lange, Steffen Gebert, Joachim Spoerhase, Piotr Rygielski, Thomas Zinner, Samuel Kounev und Phuoc Tran-Gia: Specialized heuristics for the controller placement problem in large scale SDN networks. In: *Proceedings of the 27th International Teletraffic Congress*, Seite 210–218. IEEE Computer Society, 2015. https://doi.org/10.1109/ITC.2015.32.

[LGZ+15] Stanislav Lange, Steffen Gebert, Thomas Zinner, Phuoc Tran-Gia, David Hock, Michael Jarschel und Marco Hoffmann: Heuristic approaches to the controller placement problem in large scale SDN networks. *IEEE Transactions on Network and Service Management*, 12(1):4–17, 2015. https://doi.org/10.1109/TNSM.2015.2402432.

[MR10] Nabil H. Mustafa und Saurabh Ray: Improved results on geometric hitting set problems. *Discrete and Computational Geometry*, 44(4):883–895, 2010. https://doi.org/10.1007/s00454-010-9285-9.

[Vaz13] Vijay V Vazirani: *Approximation algorithms*. Springer Science & Business Media, 2013. https://doi.org/10.1007/978-3-662-04565-7.

[VC15] Vladimir N. Vapnik und Alexey Y. Chervonenkis: *On the uniform convergence of relative frequencies of events to their probabilities*, Seiten 11–30. Springer International Publishing, 2015. https://doi.org/10.1007/978-3-319-21852-6_3.

Erklärung

Hiermit versichere ich die vorliegende Abschlussarbeit selbstständig verfasst zu haben, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, und die Arbeit bisher oder gleichzeitig keiner anderen Prüfungsbehörde unter Erlangung eines akademischen Grades vorgelegt zu haben.

Würzburg, den 1. März 2021

.....
Dominique Bau