Institut für Formale Methoden der Informatik

Abteilung Algorithmik

Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart

Masterarbeit

Anwendung der Well-Separated Pair Decomposition auf Sichtbarkeitsgraphen

Niklas Owens

Studiengang: Informatik

Prüfer: Prof. Dr. Stefan Funke

Betreuer: Prof. Dr. Stefan Funke

begonnen am: 17.06.2020

beendet am: 15.12.2020

Kurzfassung

Bisherige Ansätze zur Berechnung von Pfaden zwischen zwei Punkten mit minimaler euklidischer Länge in einer Umgebung mit unpassierbaren Hindernissen basieren oft auf der Konstruktion des vollständigen Sichtbarkeitsgraphen um auf diesem Routing-Algorithmen auszuführen. Problematisch an diesen Verfahren ist zum einen die hohe Berechnungszeit des vollständigen Sichtbarkeitsgraphen, zum anderen, dass die Komplexität des Sichtbarkeitsgraphen in $\mathcal{O}(n^2)$ liegt. Um dies zu vermeiden, befasst sich diese Arbeit mit der Entwicklung eines Verfahrens zur Konstruktion von Spannern für Sichtbarkeitsgraphen, auf denen kürzeste Pfade approximiert werden können. Anhand experimenteller Vergleiche mit einem naiven Algorithmus hat sich gezeigt, dass die Spannerkonstruktion deutlich schneller ist als die des vollständigen Sichtbarkeitsgraphen, wobei die Länge der approximierten Pfade im Durchschnitt nur geringfügig von der Länge der tatsächlich kürzesten Pfade abweicht.

1 Einleitung

Die Bestimmung kürzester Pfade im euklidischen Raum ist ein bekanntes Problem der algorithmischen Geometrie und findet unter anderem in der Robotik und Computergraphik Anwendung. Dabei gilt es in einer Umgebung mit polyedrischen Hindernissen einen Pfad zwischen zwei gegebenen Punkten zu berechnen, der von minimaler Länge ist und keines der Hindernisse schneidet. Ein Beispiel für eine Probleminstanz in der Ebene und den entsprechenden kürzesten Pfad ist in Abbildung 1 zu sehen. Während für die Bestimmung kürzester Wege auf Graphen bereits zahlreiche, effiziente Lösungsverfahren existieren [1], die beispielsweise zur Routenplanung in Straßennetzwerken genutzt werden können, gestaltet sich die Berechnung euklidischer, kürzester Pfade schwieriger. Bei drei oder mehr Dimensionen ist das euklidische kürzeste-Wege-Problem NP-schwer [2], diese Arbeit beschäftigt sich jedoch nur mit dem zweidimensionalem Fall.

1.1 Definition des Problems

Gegeben sei eine Menge simpler Polygone in der euklidischen Ebene sowie zwei Endpunkte $s, t \in \mathbb{R}^2$. Gesucht sei ein Polygonzug $\pi = \bigcup_{i=1}^{k-1} [p_i, p_{i+1}]$, sodass $p_1 = s, p_k = t$ und π minimale euklidische Länge hat. Die euklidische Länge eines Polygonzugs ist die Summe der euklidischen Längen seiner Strecken $\sum_{i=1}^{k-1} \|p_i - p_{i+1}\|_2$. Weiterhin darf π nicht das Innere der Polygone schneiden.

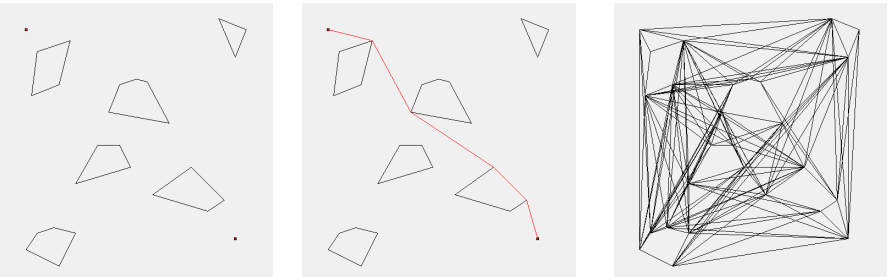


Abbildung 1: Links: Umgebung mit polygonalen Hindernissen. Mitte: Kürzester Pfad zwischen zwei gegebenen Punkten. Rechts: Zugehöriger Sichtbarkeitsgraph.

2 Verwandte Arbeiten und eigener Beitrag

2.1 Verwandte Arbeiten

Bisherige Verfahren zur Berechnung euklidischer, kürzester Pfade in der Ebene sind oftmals sehr komplex und schwer zu implementieren, wie beispielsweise der Algorithmus von Hershberger und Suri [3]. Bei diesem Verfahren wird eine vom Startpunkt ausgehende Wellenfront simuliert und darauf untersucht, wie sie mit den Hindernissen kollidiert. Praktikablere Ansätze basieren auf der Berechnung des Sichtbarkeitsgraphen der zu navigierenden Umgebung. Der Sichtbarkeitsgraph ist ein Graph, dessen Knoten die Ecken der Hindernisse sowie Start- und Endknoten des Pfades sind. Die Kanten des Sichtbarkeitsgraphen sind alle gegenseitig sichtbaren Knotenpaare. Auf dem Sichtbarkeitsgraph können dann mittels bekannter Verfahren, wie dem Dijkstra-Algorithmus [4], kürzeste Pfade berechnet werden. Abbildung 1 zeigt den entsprechenden Sichtbarkeitsgraph sowie den kürzesten Pfad zwischen zwei gegebenen Punkten für einer Umgebung. Ein wesentliches Problem hierbei ist, dass der Sichtbarkeitsgraph bei einer Umgebung mit n Polygonecken bis zu $\Theta(n^2)$ Kanten haben kann. Ansätze, die Kantenanzahl des Graphen zu reduzieren, setzen oft die zeitaufwendige Berechnung des vollständigen Graphen voraus [5]. Um dieses Problem zu umgehen, basieren viele Verfahren auf der Konstruktion von Spannern. Spanner sind Subgraphen, deren Kanten lediglich eine Teilmenge der Kanten eines Graphen sind, sodass die Pfadkosten zwischen Knoten des Spanners möglichst wenig von den tatsächlichen Pfadkosten abweichen. Ein Beispiel für einen vollständigen Sichtbarkeitsgraphen und einen Spanner ist in Abbildung 2 zu sehen. Es existieren Verfahren die euklidisch kürzeste Pfad approximieren, wie das von Aleksandrov et al [6], ohne jedoch dabei eine aussagekräftige obere Schranke für die Größe der benötigten Datenstruktur zu garantieren. Hierbei wird ein Graph konstruiert, indem zunächst die Umgebung in Dreiecke unterteilt wird und anschließend auf den Kanten der Dreiecke Steiner-Punkte berechnet werden. Diese Steiner-Punkte bilden dann die Knoten des Graphen, die durch Kanten miteinander verbunden werden.

2.2 Eigener Beitrag

Das Ziel dieser Arbeit ist die Konstruktion eines Spanners des vollständigen Sichtbarkeitsgraphen mittels Well-Separated Pair Decomposition [7]. Bei diesem Ansatz ist es nicht nötig den vollständigen Sichtbarkeitsgraphen zu berechnen. Weiterhin wird untersucht, ob sich Contraction Hierarchies [8] zur Beschleunigung von kürzesten-Pfad-Anfragen in Sichtbarkeitsgraphen eignen.

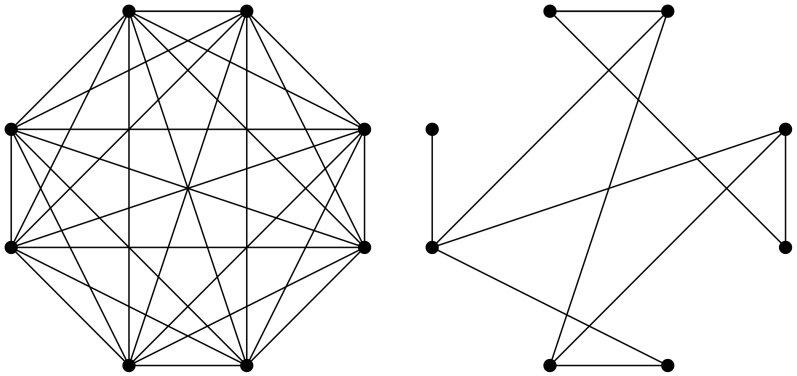


Abbildung 2: Links: Vollständiger Sichtbarkeitsgraph mit $|V| = 8$. Rechts: Möglicher Spanner.

3 Grundlagen und Definitionen

In diesem Kapitel werden für das im Weiteren vorgestellte Verfahren notwendige Grundlagen beschrieben. Bei diesen handelt es sich um Konzepte aus der Graphentheorie und algorithmischen Geometrie sowie Algorithmen zur Berechnung kürzester Pfade in Graphen.

3.1 Graphen

Um kürzeste Routen in einem Netzwerk zu berechnen, kann das Netzwerk als nichtnegativ gewichteter Graph modelliert werden.
Ein nichtnegativ gewichteter Graph $G(V, E, c)$ besteht aus einer Knotenmenge V, einer Kantenmenge E und einer Gewichtsfunktion $c\colon E \to \mathbb{R}^+$. Eine Kante $e = \{u, v\} \in E$ ist ein ungeordnetes Paar, bestehend aus den zwei Endknoten $u, v \in V$. Zwei Knoten $u, v \in V$ sind adjazent zueinander, wenn eine Kante $e = \{u, v\} \in E$ existiert. Eine Kante $e = \{u, v\} \in E$ und ihre beiden Endknoten $u, v \in E$ sind inzident zueinander. Der Grad $deg\colon V \to \mathbb{N}$ eines Knotens ist die Anzahl seiner inzidenten Kanten.
Ein s-t Pfad $\pi = (v_1, ..., v_{k+1})$ mit Pfadkosten $c(\pi) = \sum_{i=1}^{k} c(v_i, v_{i+1})$ ist eine Knotensequenz, sodass $v_1 = s, v_{k+1} = t$ und $\{v_i, v_{i+1}\} \in E$ für alle $i = 1, ..., k$.
π ist zudem ein kürzester Pfad, wenn $c(\pi) \leq c(\pi')$ für alle s-t Pfade π'. Die Kosten des kürzesten s-t Pfades in einem Graphen G werden im Weiteren auch als Distanz $d_G(s, t)$ bezeichnet.
In Abbildung 3 ist ein Beispiel für einen Graph und einen kürzesten s-t-Pfad zu sehen.
Ein Graph $G(V, E, c)$ ist euklidisch, wenn jeder Knoten $v \in V \subseteq \mathbb{R}^2$ als Punkt in der Ebene dargestellt werden kann und für alle Kanten $e = \{u, v\}$ gilt, dass $c(e) = \|v - u\|_2$.

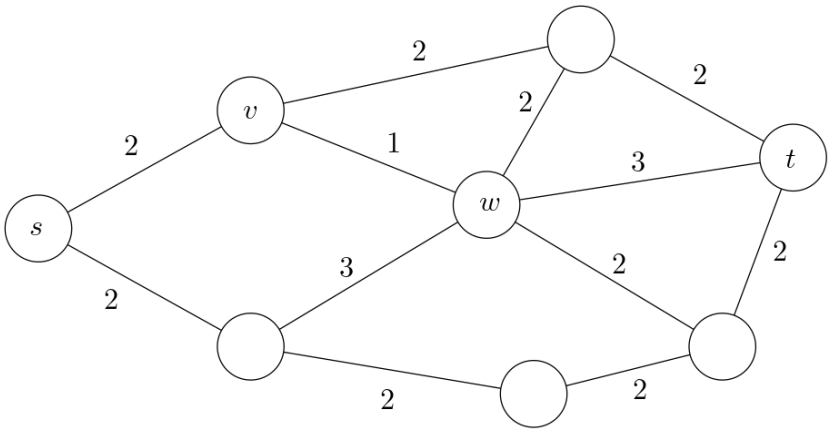


Abbildung 3: Der Pfad $\pi = (s, v, w, t)$ mit $c(\pi) = 2 + 1 + 3$ ist der kürzeste s-t-Pfad.

3.2 Sichtbarkeitsgraphen

Eine Möglichkeit, euklidisch kürzeste s-t Pfade zu bestimmen, ist es den Sichtbarkeitsgraphen der Probleminstanz zu konstruieren. Die Knoten des Sichtbarkeitsgraphen repräsentieren die Punkte s, t sowie die Ecken der Polygone. Für jedes Knotenpaar p, q existiert eine Kante $e = (p, q)$, genau dann wenn p und q gegenseitig sichtbar sind. Zwei Knoten sind gegenseitig sichtbar, wenn die Strecke $\overline{pq}$ keines der Polygone schneidet. Das Gewicht einer Kante $e = \{p, q\}$ ist der euklidische Abstand der beiden Endknoten $c(p, q) = \|q - p\|_2$.
Auf Sichtbarkeitsgraphen können euklidisch kürzeste Pfade berechnet werden, da jeder euklidisch kürzeste Pfad ein Polygonzug ist, dessen Ecken Knoten des Sichtbarkeitsgraphen sind [9].

Lemma 3.1. *Jeder euklidisch kürzeste s-t-Pfad ist ein Polygonzug.*

Beweis. Sei π ein euklidisch kürzester s-t-Pfad, der kein Polygonzug ist. Dann existiert, wie in Abbildung 4 illustriert, ein Punkt p auf π, der weder auf der Oberfläche eines Polygons noch auf einem Liniensegment von π liegt. Weiterhin gibt es einen Kreis mit Mittelpunkt p, der keines der Polygone berührt. Der Teilpfad von π, der sich innerhalb dieses Kreises befindet ist kein Liniensegment. Da der Teilpfad durch ein Liniensegment mit den Endpunkten, an denen der Teilpfad den Kreis schneidet, ersetzt werden kann und π dadurch verkürzt wird, kann π kein kürzester Pfad sein. □

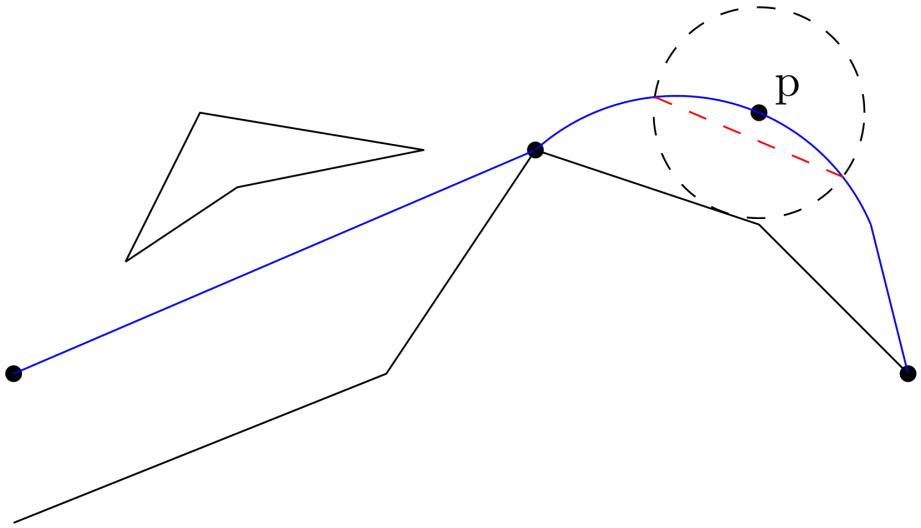


Abbildung 4: Der Teilpfad innerhalb des sich vollständig im freien Raum liegenden Kreises um den Punkt p kann durch ein kürzeres Liniensegment(rot) ersetzt werden. Der Pfad(blau) ist daher kein kürzester Pfad.

Lemma 3.2. *Jede Ecke eines euklidisch kürzesten s-t-Pfads ist ein Knoten des Sichtbarkeitsgraphen.*

Beweis. Sei π ein euklidisch kürzester s-t-Pfad und v eine Ecke von π die kein Knoten des Sichtbarkeitsgraphen ist. Befindet sich v im freien Raum beziehungsweise auf einer Kante eines Polygons, so kann ein Kreis beziehungsweise Halbkreis mit Mittelpunkt v konstruiert werden, der sich im freien Raum befindet. Der Teilpfad von π, der in diesem Kreis beziehungsweise Halbkreis liegt, kann durch ein kürzeres Liniensegment ersetzt werden. Der Pfad π ist also suboptimal, v muss daher ein Knoten des Sichtbarkeitsgraphen sein. □

Ein simpler Algorithmus um den Sichtbarkeitsgraphen für eine gegebene Menge Polygone zu bestimmen ist in Alg. 1 zu sehen.

Algorithm 1: Vollständiger Sichtbarkeitsgraph

```
   Data: Polygonmenge P
   Result: Euklidischer Graph G(V, E)
1  V ← {s, t} ∪ Ecken von P;
2  E ← ∅;
3  for p, q ∈ V × V do
4  |  if p, q sichtbar then
5  |  |  E ← E ∪ {p, q};
6  |  end
7  end
```

Zu beachten ist, dass es bei insgesamt n Knoten n Kanten gibt, aus denen die Polygone gebildet werden. Wie in Abbildung 2 illustiert, ist die Anzahl der möglichen Kanten eines Sichtbarkeitsgraphen $\binom{n}{2} \in \Theta(n^2)$. Da jede Sichtbarkeitskante n Kanten schneiden könnte, liegt die Zeitkomplexität des Algorithmus in $\mathcal{O}(n^3)$. Es existieren allerdings schnellere Algorithmen zur Berechnung des Sichtbarkeitsgraphen, beispielsweise den ausgabesensitiven Algorithmus von Ghosh und Mount [10] mit Zeitkomplexität $\mathcal{O}(n \log n + |E|)$.

3.3 Dijkstra-Algorithmus

Der Dijkstra-Algorithmus [4] ist vermutlich der bekannteste Algorithmus um kürzeste Pfade in einem nichtnegativ gewichteten Graphen zu bestimmen. Er berechnet für einen Graphen $G(V, E, c)$ und Startknoten $s \in V$ die Distanzen zu allen anderen Knoten. Zu Beginn wird für alle Knoten $v \in V \setminus \{s\}$ die vorläufig bekannte Distanz $d[v]$ auf ∞ gesetzt, für s selbst gilt $d[s] = 0$. Nun wird schrittweise, beginnend mit s, immer der unbesuchte Knoten mit der geringsten bisher bekannten Distanz besucht. Wird Knoten u besucht, werden alle inzidenten Kanten von u relaxiert. Bei der Relaxierung einer Kante $\{u, v\}$ wird überprüft, ob die bisher bekannte Distanz $d[v]$ größer als die Kosten des Pfades von s über u zu v ist. Ist das der Fall, so wird $d[v]$ entsprechend auf $d[u] + c(u, v)$ gesetzt.

Da in jedem Schritt immer der Knoten mit der geringsten Distanz besucht wird, ist sichergestellt, dass die Distanz eines einmal besuchten Knotens sich im Laufe des Algorithmus nicht mehr ändert. Bei der Suche nach dem kürzesten Pfad zu einem einzelnen Knoten t kann die Berechnung also abgebrochen werden, sobald t besucht wurde. Wird für die Datenstruktur, mit der die zu besuchenden Knoten verwaltet werden, ein Fibonacci-Heap verwendet, liegt die Zeitkomplexität des Dijkstra-Algorithmus in $\mathcal{O}(|V| \log |V| + |E|)$ [11].

Algorithm 2: Dijkstra

```
   Data: G(V, E, c), s ∈ V
   Result: Distances d
 1 foreach v ∈ V \ {s} do
 2 |  d[v] ← ∞;
 3 end
 4 d[s] = 0;
 5 foreach v ∈ V do
 6 |  PQ.insert(v, d[v]);
 7 end
 8 while PQ ≠ ∅ do
 9 |  (dist, u) ← PQ.pop();
10 |  foreach e = (u, v) ∈ E do
11 |  |  if d[u] + c(u, v) < d[v] then
12 |  |  |  d[v] ← d[u] + c(u, v);
13 |  |  |  PQ.decreaseKey(v, d[v]);
14 |  |  end
15 |  end
16 end
```

3.4 Contraction Hierarchies

Contraction Hierarchies(CH) von Geisberger et al. [8] sind eine Datenstruktur zur Beschleunigung von kürzesten-Pfad-Anfragen. Dazu wird einem Graphen $G(V, E, c)$ eine Menge von Shortcuts E' hinzugefügt, die kürzeste Pfade repräsentieren. Durch Nutzung dieser Kanten, die große Kantensequenzen des ursprünglichen Graphen darstellen können, müssen bei der Berechnung von kürzesten Pfaden weniger Kanten untersucht werden. Im Kontext von Straßennetzwerken haben sich Contraction Hierarchies als sehr effizient erwiesen. Für einen Graphen des europäischen Straßennetzwerks mit $|V| \approx 18\ 000\ 000$, $|E| \approx 42\ 200\ 000$ und $|E'| \approx 36\ 900\ 000$ konnte die durchschnittliche Berechnungszeit von kürzesten-Pfad-Anfragen von 5,6 Sekunden mit dem Dijkstra-Algorithmus auf 0,2 Millisekunden beschleunigt werden [8].

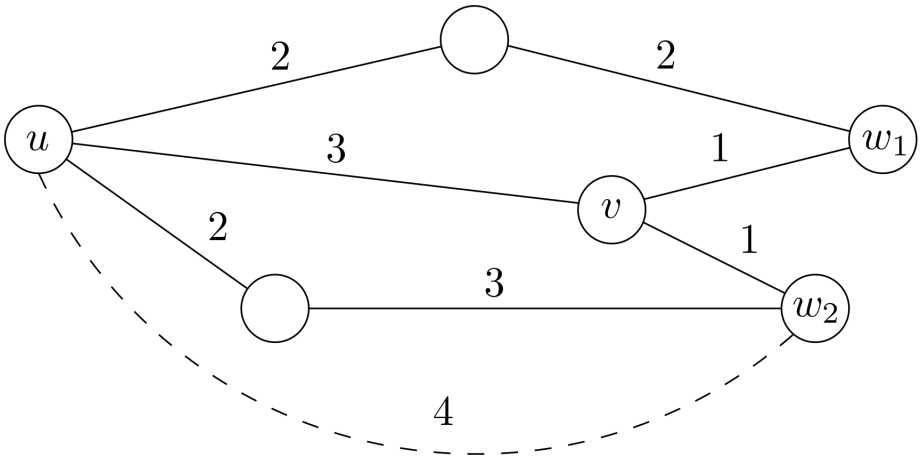


Abbildung 5: Bei der Kontraktion von v wird eine neue Kante $\{u, w_2\}$ mit $c(u, w_2) = 4$ erzeugt. Eine Kante $\{u, w_1\}$ ist nicht notwendig, da (u, v, w_1) nicht der einzige kürzeste u-w_1 Pfad ist.

3.4.1 Vorberechnungsphase

In der Vorberechnungsphase werden zunächst alle Knoten von G in einer vorgegebenen Reihenfolge kontrahiert. Bei der Kontraktion des i-ten Knotens v wird für jedes Paar $\{u, w\}$ adjazenter Knoten von v untersucht, ob (u, v, w) ein kürzester u-w-Pfad ist. Dies kann beispielsweise mit dem bereits beschriebenen Dijkstra-Algorithmus überprüft werden. Ist (u, v, w) der einzige kürzeste u-w-Pfad, so wird dem Graphen eine neue Kante $e = \{u, w\} \in E'$ mit Kantengewicht $c'(u, w) = c(u, v) + c(v, w)$ hinzugefügt. Anschließend wird v der Rang $l(v) = i$ zugeordnet und mitsamt aller inzidenten Kanten aus dem Graphen entfernt. Durch das Hinzufügen der Kanten wird sichergestellt, dass während der Vorberechnungsphase alle Distanzen zwischen noch zu kontrahierenden Knoten erhalten bleiben. Nach Kontraktion aller Knoten ist die Contraction Hierarchy von G dann $G'(V, E \cup E', c^*)$, mit

$$c^*(e) = \begin{cases} c(e), & e \in E \\ c'(e), & e \in E' \end{cases}$$

und der Rangfunktion $l \colon V \to \mathbb{N}$.

3.4.2 Kürzeste Pfade mit Contraction Hierarchies

Durch die hinzugefügten Shortcuts kann jeder kürzeste s-t-Pfad durch eine Sequenz von Aufwärtskanten, gefolgt von einer Sequenz von Abwärtskanten dargestellt werden. Eine Aufwärtskante beziehungsweise Abwärtskante $e = \{u, v\}$ ist eine Kante mit $l(u) < l(v)$ beziehungsweise $l(u) > l(v)$.

Um kürzeste s-t-Pfade auf Contraction Hierarchies zu berechnen, wird jeweils von s und t eine Variante des Dijkstra-Algorithmus ausgeführt. Diese Variante relaxiert bei Besuch eines Knotens u nur von u ausgehende Aufwärtskanten. Sei U und W die Mengen der Knoten, die bei Ausführung des von s beziehungsweise t aus gestarteten Dijkstra-Algorithmus besucht werden. Der kürzeste s-t-Pfad führt dann über den Knoten $v \in U \cap W$ der $d(s,v) + d(t,v)$ minimiert. Der Knoten v ist dann zugleich der Knoten mit dem höchsten Rang aller Knoten auf dem kürzesten s-t-Pfad. Die Distanz $d(s,t)$ ist dann $d(s,v) + d(t,v)$.

Stall-on-demand Die Berechnungszeit kann durch die Verwendung von Stall-on-demand weiter reduziert werden. Hierbei werden bei Besuch eines Knotens v alle Kanten $\{u,v\} \in E$ mit $l(u) > l(v)$ betrachtet. Gilt $d(u) + c(u,v) < d(v)$, so ist $d(v)$ nicht optimal, da ein kürzerer Pfad zu v existiert, der über den Knoten u verläuft. Die Suche von v aus kann also beendet werden. Das heißt, die vorläufige Distanz $d(v)$ wird auf $d(u) + c(u,v)$ gesetzt und die mit v inzidenten Kanten nicht weiter relaxiert, wodurch sich der Suchraum weiter verringert.

3.4.3 Kontraktionsreihenfolge

Eine hohe Anzahl von Shortcuts kann negativen Einfluss auf die Berechnungszeit kürzester Pfade haben. Eine Möglichkeit, die Anzahl der Shortcuts gering zu halten, ist es, die Knoten nach aufsteigender Kantendifferenz zu kontrahieren. Die Kantendifferenz eines Knoten v ist die Anzahl der entstehenden Shortcuts bei der Kontraktion von v minus die Anzahl der mit v inzidenten Kanten.
Diese Strategie hat jedoch den Nachteil, dass bei bestimmten Graphen die Berechnung von kürzesten Pfaden nur wenig beschleunigt wird. Sollte es sich bei dem Graphen beispielsweise um einen Pfad handeln, wie in Abbildung 6 illustriert, kann der Fall auftreten, dass immer die Endknoten des Pfades zur Kontraktion gewählt werden. In diesem Fall würden zwar keine Shortcuts, somit aber auch keine Beschleunigung entstehen.
Dies kann vermieden werden, indem in jedem Schritt der Vorberechnungsphase nicht ein einzelner Knoten, sondern eine stabile Menge des Graphen kontrahiert wird. Eine stabile Menge ist eine Teilmenge $U \subseteq V$, sodass alle Knoten von U nicht miteinander adjazent sind. Hierbei wird dann allen Knoten der stabilen Menge derselbe Rang zugewiesen.

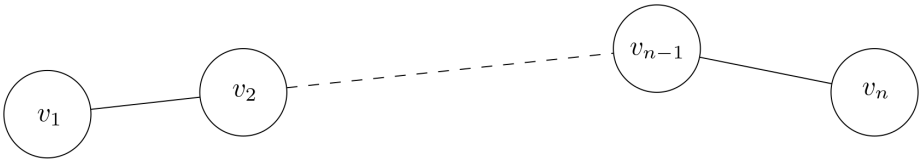


Abbildung 6: Werden die Knoten des Graphen lediglich nach aufsteigender Kantendifferenz kontrahiert, ist es möglich, dass keine Shortcuts entstehen.

3.5 Well-Separated Pair Decomposition

Die Well-Separated Pair Decomposition von Callahan und Kosaraju [7] ist eine kompakte Datenstruktur, mit der Distanzen innerhalb einer Punktmenge approximiert werden können. Dazu werden die Punkte in Paare von Teilmengen zusammengefasst, sodass die Distanz von Punkten einer Teilmenge zu Punkten der anderen Teilmenge ähnlich groß ist.
Sei $S \subset \mathbb{R}^2$ eine Punktmenge in der euklidischen Ebene, $P, Q \subset S$ disjunkte Teilmengen von S und B_P, B_Q die Bounding Boxen von P beziehungsweise Q. Die Bounding Box einer Punktmenge P ist das minimale, achsenorientierte Rechteck, das P vollständig umfasst. P und Q sind *well-separated* bezüglich dem Separationsfaktor $s > 0$, wenn zwei Kreise C_P, C_Q mit Radius r existieren, die B_P beziehungsweise B_Q enthalten und einen euklidischen Abstand von $d(C_P, C_Q) > s \cdot r$ haben. Eine Well-Separated Pair Decomposition(WSPD)

$$WSPD(S) = \{\{P_1, Q_1\}, \{P_2, Q_2\}, ..., \{P_{m-1}, Q_{m-1}\}, \{P_m, Q_m\}\}$$

von S ist eine Menge von Paaren von disjunkten Teilmengen von S wenn folgende Eigenschaften erfüllt sind:

- Für jedes Paar $p, q \in S$ existiert genau ein $i = 1, ..., m$, sodass $p \in P_i, q \in Q_i$ oder $p \in Q_i, q \in P_i$.
- P_i und Q_i sind well-separated für $i = 1, ..., m$.

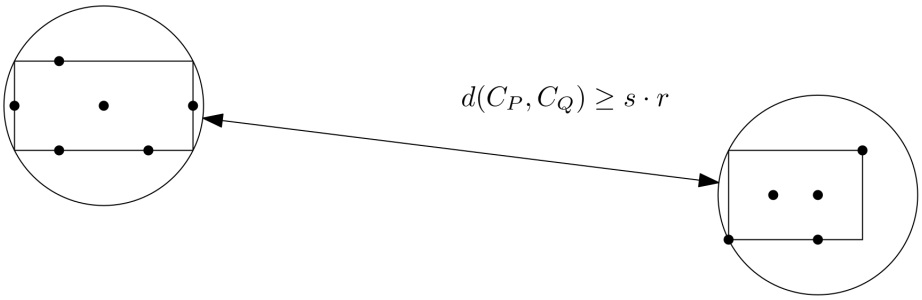


Abbildung 7: Well-separated Punktmengen.

3.5.1 Berechnung der Well-Separated Pair Decomposition

Die WSPD einer Punktmenge $S \subset \mathbb{R}^2$ kann in zwei Schritten berechnet werden. Im ersten Schritt wird ein Split Tree aufgebaut. Der Split Tree einer Punktmenge S ist ein binärer Baum, der die Bounding Box von S hierarchisch aufspaltet und abspeichert. Anschließend kann aus dem Split Tree die WSPD berechnet werden.

Split Tree Jeder Knoten u des Split Trees stellt eine Teilmenge $S_u \subseteq S$ dar. Der Split Tree wird, ausgehend von der Wurzel, die S repräsentiert, rekursiv gebildet. Sei $B_S = [x_1, x_1'] \times [x_2, x_2']$ die Bounding Box von S, $L_i(S) = x_i - x_i'$ die Länge der Bounding Box in der i-ten Dimension und $L_{max}(S) = \max_i L_i(S)$. B_S wird entlang der i-ten Dimension in zwei gleich große Bounding Boxen B_{S_1} und B_{S_2} gespalten, wie in Abbildung 8 zu sehen. Die Kindknoten sind dann die Wurzeln der Split Trees für die Punktmengen S_1 und S_2. Diese Rekursion wird fortgesetzt, bis die Punktmengen nur noch einen Punkt enthalten. Diese Punktmengen werden als Blätter repräsentiert.

WSPD Um aus dem Split Tree eine WSPD zu berechnen, werden für jeden inneren Knoten v des Split Trees dessen Kinder u und w mit zugehörigen Punktmengen S_u und S_w betrachtet. Sind S_u und S_w well-separated, so wird der WSPD das Paar (S_u, S_w) hinzugefügt. Im Fall, dass S_u und S_w nicht well-separated sind, werden die zugehörigen Bounding Boxen verglichen. Gilt $L_{max}(S_u) \leq L_{max}(S_w)$, wird rekursiv überprüft, ob die Paare (S_u, S_{w_1}) und (S_u, S_{w_2}) well-separated sind, wobei w_1 und w_2 die Kindknoten von w sind. Andernfalls werden analog die Paare (S_{u_1}, S_w) und (S_{u_2}, S_w) betrachtet.
Die Größe der WSPD einer Punktmenge $S \subset \mathbb{R}^2$ liegt bei einem Separationsfaktor $s > 0$ in $\mathcal{O}(s^2 \cdot n)$[7]. Es ist durch die WSPD also möglich, die $\Theta(n^2)$ Distanzen zwischen Punkten von S durch eine lineare Anzahl von Paaren von Punktmengen zu approximieren. Dies impliziert insbesondere, dass die Anzahl an stark unterschiedlichen Distanzen innerhalb von S in $O(n)$ liegt.

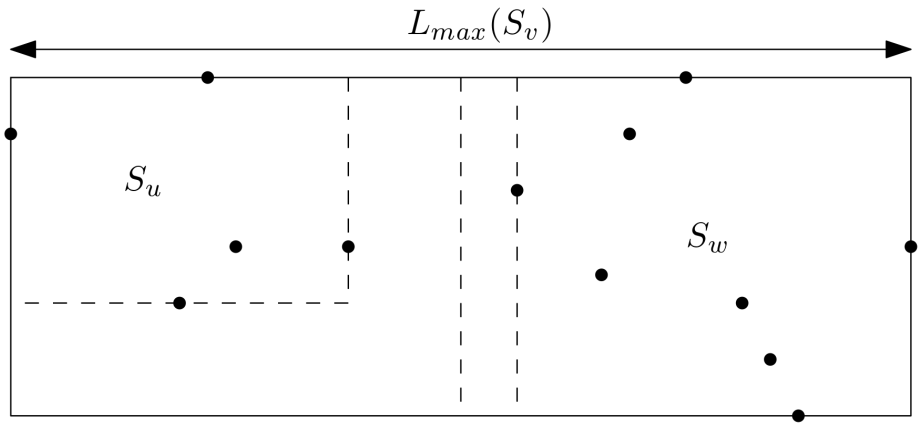


Abbildung 8: Ein Split halbiert die Bounding Box der Punktmenge eines Knotens v des Split Trees. Die Kinder u, w von v repräsentieren die Punktmengen innerhalb der zwei entstehenden Bounding Boxen.

3.5.2 Euklidischer Spanner

Sei $G(V, E, c)$ ein gewichteter Graph, dann ist $G'(V, E', c')$ mit $E' \subseteq E$ ein k-Spanner von G, wenn für jedes Paar $p, q \in V$ gilt, dass $d_{G'}(p, q) \leq k \cdot d_G(p, q)$. Für den vollständigen, euklidischen Graphen einer Punktemenge $S \subset \mathbb{R}^2$ kann mittels des in Alg. 3 beschriebenen Algorithmus aus der WSPD von S ein k-Spanner konstruiert werden. Bei einer WSPD mit Separationsfaktor $s > 4$ ist der auf diese Weise konstruierte Graph ein k-Spanner des vollständigen, euklidischen Graphen $G(S, E)$ mit $k = \frac{s+4}{s-4}$ [12].

Algorithm 3: Sichtbarkeitsspanner

Data: $WSPD(S)$
Result: $G(V, E)$
$V \leftarrow S$;
$E \leftarrow \emptyset$;
foreach $(P_i, Q_i) \in WSPD(S)$ **do**
    Wähle $u \in P_i, v \in Q_i$;
    $E \leftarrow E \cup \{u, v\}$;
end

4 Spanner für Sichtbarkeitsgraphen und beschleunigte Pfadberechnung

Im folgenden Kapitel wird ein Verfahren zur Konstruktion von Spannern für Sichtbarkeitsgraphen vorgestellt. Zudem wird darauf eingegangen, inwiefern gängige Algorithmen zur Beschleunigung kürzester Pfadberechnungen in Straßengraphen auch in Sichtbarkeitsgraphen eingesetzt werden können.

4.1 Grundidee der Spannerkonstruktion

Um den Spanner zu konstruieren, wird zunächst die WSPD der Knotenmenge des Sichtbarkeitsgraphen berechnet. Anschließend kann, auf ähnliche Weise wie in Kapitel 3.5.2 beschrieben, ein auf der WSPD basierender Spanner konstruiert werden. Nach der Berechnung des Spanners wird dessen Contraction Hierarchy berechnet.

Unterschiede zum euklidischen Spanner Bei der Konstruktion des Spanners muss beachtet werden, dass die berechneten Kanten auch korrekte Sichtbarkeitskanten sind. Anders als in Alg. 3 muss daher vor dem Hinzufügen einer Kante $\{u, v\}$ überprüft werden, ob u und v gegenseitig sichtbar sind. In dem Beispiel in Abbildung 9 sind lediglich $\{a_1, b_1\}$ und $\{a_2, b_2\}$ erlaubte Sichtbarkeitskanten für das Paar $\{\{a_1, a_2\}, \{b_1, b_2\}\}$. Es existieren dementsprechend im Spanner keine Kanten zwischen Paaren $\{P, Q\}$ der WSPD, für die es kein gegenseitig sichtbares Knotenpaar $u \in P, v \in Q$ gibt. Alg. 4 beschreibt das resultierende Verfahren zur Konstruktion des Spanners.

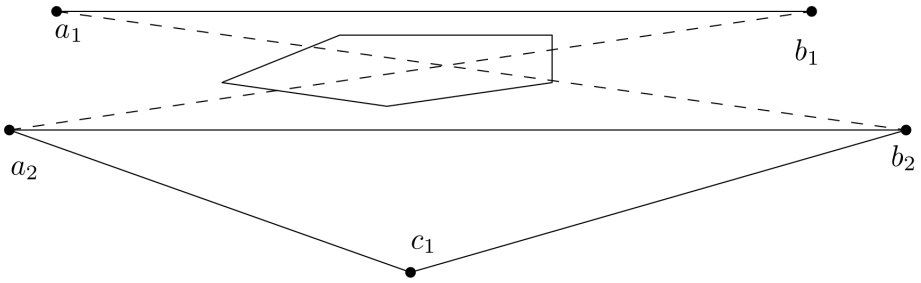


Abbildung 9: Die Kanten $\{a_1, b_2\}$ und $\{a_2, b_2\}$ sind aufgrund des polygonalen Hindernisses keine Sichtbarkeitskanten.

4.2 Test auf Sichtbarkeit

Zwei Knoten u, v sind sichtbar genau dann, wenn das Liniensegment $\overline{uv}$ keine der Kanten eines der polygonalen Hindernisse schneidet. Um die Anzahl der auf Schnitt zu testenden Polygonkanten zu reduzieren, wird die Karte zunächst in Gitterzellen unterteilt.

Wenn, wie in Abbildung 10 illustriert, für ein Paar P, Q der Well-Separated Pair Decomposition überprüft wird, ob zwei gegenseitig sichtbare Knoten $u \in P, v \in Q$ existieren, wird ein Polygon konstruiert, das alle möglichen Liniensegmente von R_P und R_Q enthält. Anschließend werden alle Gitterzellen ermittelt, die dieses Polygon schneiden. Dies kann beispielsweise mit einer Breitensuche über die Gitterzellen realisiert werden. Da nur Hinderniskanten, die in diesen Gitterzellen liegen, die Sichtbarkeit zwischen P und Q beeinflussen können, müssen alle restlichen Hinderniskanten nicht berücksichtigt werden. Dadurch kann die Berechnung einer Kante zwischen P und Q beschleunigt werden.

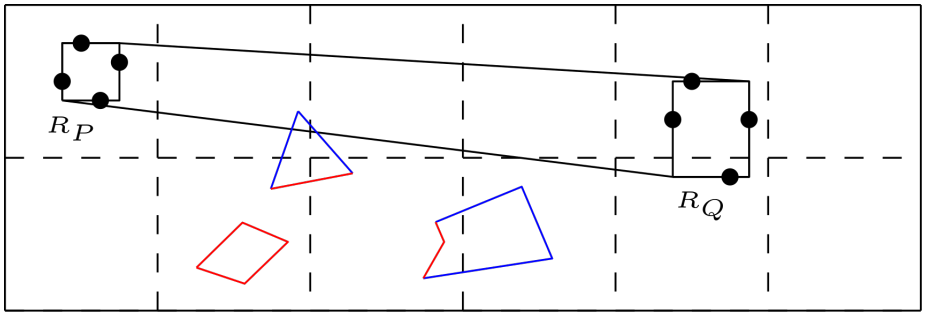


Abbildung 10: Kanten von Hindernissen(rot), die nicht in Gitterzellen liegen, die das Polygon um R_P und R_Q schneiden, können vernachlässigt werden. Lediglich die Kanten(blau), die Einfluss auf die Sichtbarkeit zwischen P und Q haben können, müssen berücksichtigt werden.

Algorithm 4: Sichtbarkeitsspanner

Data: $WSPD(S)$
Result: $G(V, E)$
1 $V \leftarrow S$;
2 $E \leftarrow \emptyset$;
3 **foreach** $(P_i, Q_i) \in WSPD(S)$ **do**
4 $\quad E \leftarrow E \cup$ Sichtbarkeitskante(P_i, Q_i);
5 **end**

4.3 Knotenpriorität

Die Reihenfolge, in der in den Zeilen 1 und 2 von Alg. 5 über die Knoten von P und Q iteriert wird, hat Einfluss auf die Approximationsgüte und Gestalt des Spanners. Insbesondere die Berechnungszeit und Anzahl der zusätzlichen Kanten der Contraction Hierarchy unterscheidet sich je nach gewählter Reihenfolge deutlich. Um eine Knotenmenge in einer bestimmten Reihenfolge zu durchlaufen, wird jedem Knoten v eine Priorität $f(v)$ zugewiesen. Knoten mit hoher Priorität werden vor Knoten mit niedriger Priorität auf Sichtbarkeit untersucht.

Algorithm 5: Kantenkonstruktion zwischen P und Q

```
Data: Well-separated pair (P, Q)
Result: Kante {u, v}
foreach u ∈ P do
    foreach v ∈ Q do
        if u, v sichtbar then
            return {u, v}
        end
    end
end
```

Im Folgenden werden drei Heuristiken vorgestellt. Die ersten beiden Heuristiken dienen zur Untersuchung, inwiefern sich die Verteilung der Kanten auf die Berechnung der CH auswirkt, während die dritte Heuristik auf die Verbesserung der Approximationsgüte des Spanners abzielt.

Minimaler Grad Knoten mit niedrigem Grad werden zuerst untersucht, die Priorität eines Knotens v ist $f(v) = -deg(v)$.

Maximaler Grad Knoten mit hohem Grad werden zuerst untersucht, die Priorität eines Knotens v ist $f(v) = deg(v)$.

In dem Beispiel in Abbildung 9 würde bei Verwendung der zweiten Heuristik die Kante $\{a_2, b_2\}$ gewählt werden, da die Knoten a_2 und b_2 durch die Adjazenz mit c_1 einen höheren Grad als a_1 und b_1 haben.

Minimaler Abstand Knoten mit geringem euklidischen Abstand zum Mittelpunkt zur anderen Knotenmenge werden zuerst untersucht, die Priorität eines Knotens v ist

$$f(v) = \begin{cases} \|v - M_{B_Q}\|_2, & v \in P \\ \|v - M_{B_P}\|_2, & v \in Q \end{cases},$$

wobei M_{B_Q} den Mittelpunkt der Bounding Box von Q bezeichnet.

4.4 Contraction Hierarchies von Sichtbarkeitsgraphen

Nachdem der Spanner berechnet wurde, wird er um seine Contraction Hierarchy erweitert. Die Contraction Hierarchy des Spanners wird, wie in Kapitel 3.4 beschrieben, auf dieselbe Art wie die von Straßengraphen berechnet. Prinzipiell wäre es auch möglich, die CH des Sichtbarkeitsgraphen zu konstruieren, würde aber dessen zeitintensive Berechnung voraussetzen. Zudem steigt durch die höhere Kantenanzahl des vollständigen Sichtbarkeitsgraphen auch die Berechnungszeit der CH. Wird die Kontraktionsreihenfolge bei der Berechnung der CH nach Kantendifferenz gewählt, werden tendenziell Knoten mit niedrigem Grad vor jenen mit hohem Grad kontrahiert. Darum könnte die Wahl der Knotenpriorität, nach welcher der Spanner konstruiert wurde, großen Einfluss auf die Gestalt der CH haben. Dies wird in Kapitel 5 genauer untersucht.

CH-Kern Bei dichten Graphen resultiert die Kontraktion von Knoten, die in der Kontraktionsreihenfolge erst spät gewählt werden, oft in unerwünscht vielen zusätzlichen Kanten. Dies kann dazu führen, dass sich die Berechnungszeit von kürzesten Pfaden aufgrund der hohen Kantenanzahl verschlechtert. Daher kann es vorteilhaft sein, bei der Berechnung der CH nicht alle Knoten des Graphen, sondern nur eine Teilmenge zu kontrahieren. Die Menge der Knoten, die nicht kontrahiert werden, wird im Weiteren CH-Kern genannt. Allen Knoten im CH-Kern wird dann der Rang $i + 1$ zugewiesen, wobei i der maximale Rang aller Knoten ist, die nicht im CH-Kern sind. Da dadurch der Fall auftreten kann, dass adjazente Knoten beide den Rang $i + 1$ haben, werden bei der Berechnung der kürzesten Pfade auch Kanten relaxiert, deren Endknoten denselben Rang haben.

5 Experimentelle Evaluation

Zur Evaluation wurden verschiedene Probleminstanzen erstellt. Für jede Instanz wurden sowohl der in Kapitel 4 vorgestellte Spanner als auch der vollständige Sichtbarkeitsgraph erstellt. Zudem wurden alle Spanner um ihre Contraction Hierarchy erweitert und mit dem vollständigen Sichtbarkeitsgraph und Spanner hinsichtlich ihrer Kantenanzahl und Konstruktionszeit verglichen. Um die Teuerung der Pfadkosten zu untersuchen, wurden für jede Karte zufällige, gleichverteilte Knotenpaare s, t erstellt und die Kosten der entsprechenden s-t-Pfade berechnet. Die Evaluation wurde mit einer C++-Implementierung mit einem einzelnen Thread auf einem i5-9500 Intel-Prozessor mit 3.0 GHz und einem Arbeitsspeicher von 64 GB durchgeführt.

5.1 Erstellung von Probleminstanzen

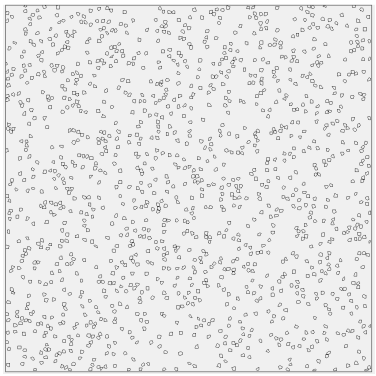
Alle Instanzen wurden erstellt, indem zufällige, konvexe Polygone auf einer rechteckigen Fläche $D = [0, 100000] \times [0, 100000]$ platziert wurden. Um ein zufälliges Polygon zu konstruieren, wird eine Menge zufälliger Punkte generiert und anschließend die konvexe Hülle der Punktmenge bestimmt. Die konvexe Hülle einer Punktmenge kann beispielsweise mit dem Gift-Wrapping-Algorithmus berechnet werden [13]. Schneidet die konvexe Hülle keines der bereits existierenden Polygone, so wird es auf dem Rechteck D als Polygon platziert. Um Probleminstanzen mit unterschiedlichen Mustern zu generieren, wurde in der Anzahl und maximalen Größe der Polygone(Abbildungen 11a, 11b, 11c) variiert. Zudem wurde der Bereich, in dem Polygone platziert werden, teilweise eingeschränkt(Abbildungen 11d, 11e). Für jede der Instanzen wurde der vollständige Sichtbarkeitsgraph mittels des in Alg. 1 beschriebenen Verfahrens erstellt. Die Parameter der verschiedenen Probleminstanzen und entsprechenden Sichtbarkeitsgraphen sind in Tabelle 1 gelistet, wobei A_P den Anteil der Fläche von D bezeichnet, der von Polygonen obstruiert ist.

Tabelle 1: Probleminstanzen und resultierende Sichtbarkeitsgraphen.

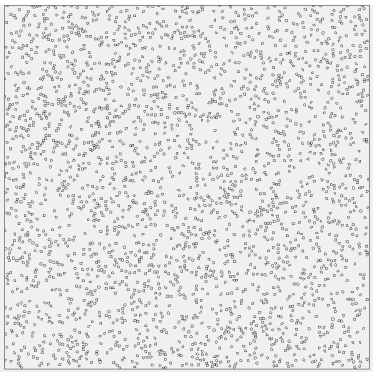
Instanz	$\|V\|$	$\|P\|$	A_P	$\|E\|$	Zeit[s]
1.1	10 139	1 306	8.2%	624 836	5 326
1.2	20 283	2 610	8.1%	1 332 268	33 398
1.3	40 173	5 184	8.0%	2 735 016	192 100
2	10 344	1 335	4.1%	911 919	6 837
3	10 007	1 290	4.0%	1 288 654	7 112

5.2 Spannerkonstruktion

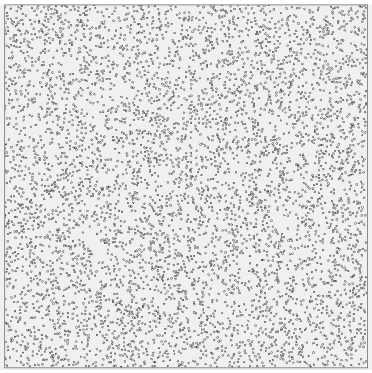
Für jede der Instanzen und Heuristiken wurde der entsprechende Spanner berechnet. Das in Kapitel 4.2 vorgestellte Gitter zur Beschleunigung der Berechnung besteht aus 20×20 Zellen.



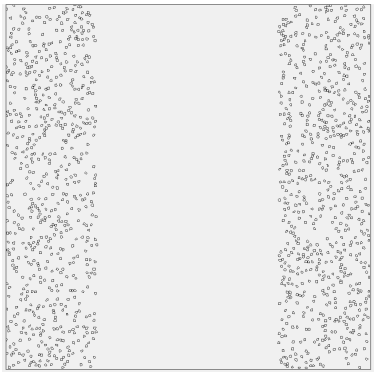
Instanz 1.1



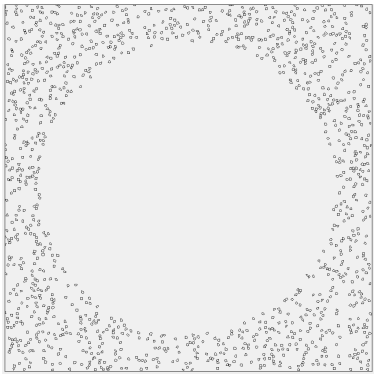
Instanz 1.2



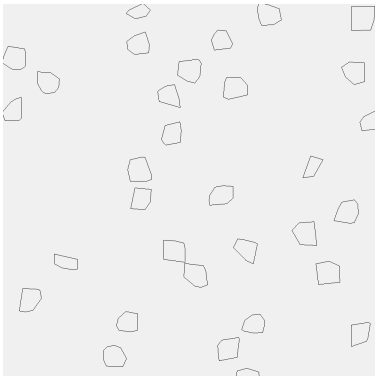
Instanz 1.3



Instanz 2



Instanz 3



Ausschnitt aus Instanz 1.1

Abbildung 11: Verschiedene Probleminstanzen

Als Separationsfaktoren der WSPD wurden 20 und 10 gewählt. Dies resultiert in insgesamt sechs verschiedenen Spannern für jede Instanz. Die Konstruktionszeit für die Spanner ist deutlich kürzer als die für die Berechnung des vollständigen Sichtbarkeitsgraphen. So benötigt der vollständige Sichtbarkeitsgraph von Instanz 1.1 ungefähr anderthalb Stunden, während der Spanner mit $s = 20$ unabhängig von der gewählten Heuristik in nur 6,5 Minuten berechnet werden kann (Tabellen 1 und 2). Bei der größten Instanz 3 verkürzt sich die Berechnung von über 53 Stunden auf 3 Stunden, was einer ähnlich hohen Beschleunigung entspricht. Einen großen Anteil an der beschleunigten Berechnungszeit hat die Nutzung des Gitters. So verkürzt sich beispielsweise die Konstruktion des Spanners für Instanz 1.1 mit Heuristik 3 und $s = 10$ von 64,5 auf knapp 8 Minuten. Die Wahl der Knotenpriorität hat wenig Einfluss auf die Konstruktionszeit für die Spanner. Lediglich Heuristik 1 wirkt sich bei den Instanzen 2 und 3 negativ auf die Berechnungszeit aus (Tabelle 2).

Tabelle 2: Benötigte Zeit für Konstruktion des Spanners bei $s = 20$ und verschiedenen Heuristiken.

			Zeit[s]		
	$\|WSPD\|$	$\|E\|$	Heuristik 1	Heuristik 2	Heuristik 3
1.1	975 072	219 306	390	386	391
1.2	2 138 246	460 657	2 057	2 043	2 056
1.3	4 639 402	938 073	12 176	12 136	12 165
2	780 868	207 353	464	408	423
3	664 509	187 136	364	295	307

Auffällig ist, dass sich ein niedrigerer Separationsfaktor je nach Instanz unterschiedlich auf die Berechnungszeit des Spanners auswirkt(Tabelle 3). Beispielsweise verschlechtert sich die Berechnungszeit bei Instanz 1.1, verbessert sich jedoch bei den Instanzen 2 und 3. Ein möglicher Grund dafür ist, dass ein großer Anteil der Berechnungszeit von Paaren der WSPD ausgemacht wird, zwischen denen keine Sichtbarkeitskante realisiert werden kann. Bei einem nicht sichtbarem Paar (A, B) der WSPD müssen alle $\Theta(|A| \cdot |B|)$ Knotenpaare auf Sichtbarkeit untersucht werden, weshalb sich insbesondere Paare von großen Knotenmengen negativ auf die Berechnungszeit auswirken können. Da dieser Fall bei Instanz 1.1 häufiger auftritt und die durchschnittliche Größe der Paare sich bei geringerem Separationsfaktor vergrößert, könnte sich so die unterschiedliche Entwicklung der Konstruktionszeit erklären.

5.3 CH-Berechnung

Alle in diesem Abschnitt gelisteten Zeitmessungen beziehen sich auf die Konstruktion der vollständigen CH der jeweiligen Spanner. Die Berechnungszeit und Menge der entstehenden Shortcuts der Contraction Hierarchies unterscheiden sich je nach Heuristik und Separationsfaktor des Spanners deutlich voneinander.

Tabelle 3: Benötigte Zeit für Konstruktion des Spanners bei $s = 10$ und verschiedenen Heuristiken.

			Zeit[s]		
	$\|WSPD\|$	$\|E\|$	Heuristik 1	Heuristik 2	Heuristik 3
1.1	312 654	132 982	484	467	472
1.2	651 734	276 044	3 048	2 992	3 011
1.3	1 355 558	562 322	19 550	19 388	19 432
2	269 042	120 930	356	318	312
3	236 302	107 983	270	224	232

Beispielsweise dauert die Berechnung der CH für den Spanner von Instanz 3 mit einem Separationsfaktor von 20 und der zweiten Heuristik 1 129 Sekunden und resultiert in 400 813 zusätzlichen Kanten. Wurde der Spanner mit der ersten Heuristik konstruiert, erweitert die CH den Spanner stattdessen um mehr als doppelt so viele Kanten und benötigt 3 410 Sekunden (Tabelle 4). Die Wahl des Separationsfaktors hat ebenfalls große Auswirkung auf die CH des Spanners. So sinkt bei $s = 10$ die Anzahl der entstehenden Shortcuts von 400 813 auf 231 987 und die Berechnungszeit von 1 129 Sekunden auf 273 Sekunden (Tabelle 5).
Allgemein auffällig ist die, im Vergleich mit Graphen von Straßennetzwerken, hohe Berechnungszeit und Anzahl von entstehenden Kanten. Während bei Straßennetzwerken die Menge von entstehenden Shortctuts meistens etwa so groß ist wie die Kantenmenge des Ausgangsgraphen, ist bei den Spannern der Sichtbarkeitsgraphen die Anzahl der berechneten Shortcuts deutlich höher. Beispielsweise verneunfacht sich die Kantenmenge des Spanners von Instanz 1.3 bei einem Separationsfaktor von 10 und Heuristik 1 durch die Erweiterung um die CH(Tabelle 5). Auch die Berechnungszeit der CH ist deutlich länger als bei Straßengraphen, so dauert beispielsweise die Berechnung der CH des Straßennetwerkes von Westeuropa lediglich fünf Minuten [1]. Dies liegt an der höheren Kantendichte von Sichtbarkeitsgraphen.

Tabelle 4: Anzahl der zusätzlichen Kanten und Berechnungszeit der CH des Spanners bei $s = 20$ und verschiedenen Heuristiken.

	Heuristik 1		Heuristik 2		Heuristik 3	
	$\|E'\|$	Zeit[s]	$\|E'\|$	Zeit[s]	$\|E'\|$	Zeit[s]
1.1	772 011	2 891	452 857	1 356	504 249	1 397
1.2	2 032 401	14 507	1 226 260	6 839	1 358 526	6 938
1.3	4 879 080	67 565	3 028 906	28 569	3 244 430	30 736
2	871 560	3 410	400 813	1 129	487 987	1 293
3	878 621	3 188	342 216	818	416 919	964

Tabelle 5: Anzahl der zusätzlichen Kanten und Berechnungszeit der CH des Spanners bei $s = 10$ und verschiedenen Heuristiken.

	Heuristik 1		Heuristik 2		Heuristik 3	
	$\|E'\|$	Zeit[s]	$\|E'\|$	Zeit[s]	$\|E'\|$	Zeit[s]
1.1	726 796	2 147	316 038	554	419 838	773
1.2	1 900 718	11 842	783 177	2 950	1 105 010	4 242
1.3	4 787 379	55 484	1 991 796	13 055	2 671 321	19 097
2	657 649	1 351	231 987	273	345,113	400
3	591 714	1 052	189 181	173	279,070	284

Rangverteilung CH Werden die Contraction Hierarchies der Spanner wie in Kapitel 3.4.3 beschrieben berechnet, ergeben sich, verglichen mit den Contraction Hierarchies von Straßengraphen, auffällige Unterschiede. Beispielsweise ist der höchste Knotenrang der CH des Spanners 347 bei Instanz 1.1 mit $|V| = 10\,139$, während es in der CH des Graphen des Saarlands mit $|V| = 595\,294$ keinen Knoten mit einem Rang von über 155 gibt. Die Abbildungen 12 und 13 zeigen die Anzahl der Knoten je nach Rang für die beiden Graphen. Es zeigt sich, dass die CH des Straßengraphen deutlich flacher ausfällt. So ist das 0,95-Quantil der Rangverteilung der CH des Saarlands Rang 8, das der CH des Spanners Rang 174. Am häufigsten tritt bei Instanz 1.1 der Rang 3 auf, bei der CH des Graphen des Saarlandes Rang 0. Dies liegt daran, dass der verwendete CH-Konstruktor zu Beginn lediglich Knoten mit einem Grad von maximal 2 kontrahiert. Solche Knoten treten im Spanner aber nur sehr selten auf, da jeder Knoten zumindest zu seinen Nachbarecken adjazent ist. Jeder Knoten im Spanner hat also einen Grad von mindestens 2.

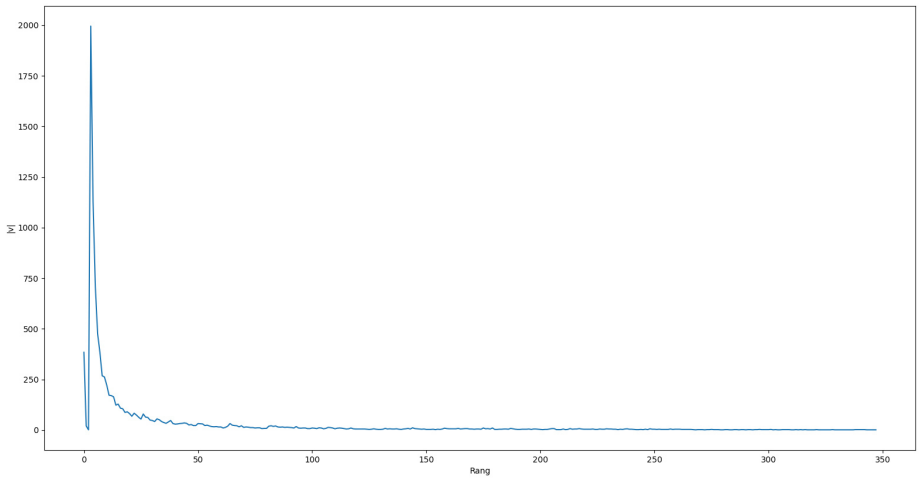


Abbildung 12: CH-Rangverteilung des Spanners für Instanz 1.1 mit Heuristik 2 und s=20.

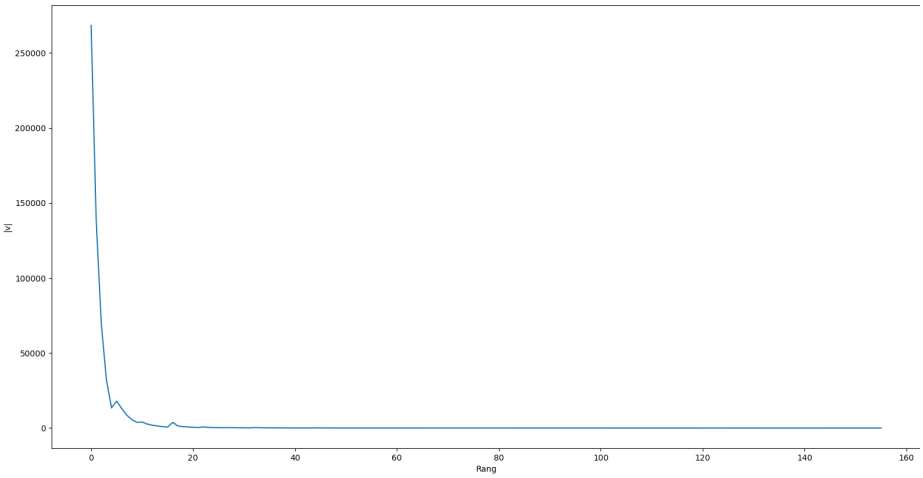


Abbildung 13: CH-Rangverteilung eines Straßengraphen des Saarlands.

5.4 Routing

Um sowohl Laufzeit für kürzeste-Pfad-Anfragen als auch Approximationsgüte der kürzesten Pfade auf Spannern zu testen, wurden für jede der Probleminstanzen 1000 zufällige, gleichverteilte s-t-Paare generiert. Anschließend wurden jeweils die entsprechenden kürzesten s-t-Pfade auf dem vollständigen euklidischen Sichtbarkeitsgraph(SG), dem Spanner und der Contraction Hierarchy des Spanners berechnet. Wie in den Tabellen 8 und 9 zu sehen ist, ist die Ausführung des Dijkstra-Algorithmus auf dem Spanner deutlich schneller als auf dem vollständigen Sichtbarkeitsgraphen, da weniger Kanten relaxiert werden. So beschleunigt sich die durchschnittliche Berechnungszeit bei Instanz 3 von 137 auf 30 Millisekunden(Tabelle 9). Durch Verwendung der CH kann die Beschleunigung weiter auf 6 Millisekunden gesenkt werden. Auffällig sind die großen Unterschiede in der Berechnungszeit je nach gewählter Heuristik, beispielsweise ist die Suche auf der CH mit Heuristik 2 teilweise doppelt so schnell wie auf der mit Heuristik 1. Für den CH-Kern wurden verschiedene Größen getestet, jedoch hat keine zu einer Beschleunigung der kürzesten-Pfad-Anfragen geführt. Die Teuerung der kürzesten Pfade ist gering, beispielsweise ist bei einem Separationsfaktor von 20 die durchschnittliche Teuerung für jede der Probleminstanzen und Heuristiken nicht größer als 0,5% (Tabelle 10). Dennoch existieren Pfade die auf dem Spanner deutlich teurer sind, deren Kosten jedoch durch Verwendung von Heuristik 3 deutlich gesenkt werden können. So verringert sich die Teuerung des Pfades mit der höchsten Teuerung bei Instanz 3 auf weniger als die Hälfte (Tabelle 8).

Tabelle 6: Durchschnittliche Berechnungszeit kürzester Pfade bei $s = 20$

	SG	Heuristik 1		Heuristik 2		Heuristik 3	
	Dijkstra	Dijkstra	CH	Dijkstra	CH	Dijkstra	CH
1.1	98ms	55ms	21ms	51ms	15ms	51ms	15ms
1.2	211ms	117ms	44ms	109ms	31ms	108ms	33ms
1.3	455ms	251ms	86ms	235ms	62ms	232ms	63ms
2	116ms	51ms	20ms	47ms	12ms	47ms	13ms
3	137ms	47ms	19ms	43ms	10ms	43ms	11ms

Tabelle 7: Durchschnittliche Berechnungszeit kürzester Pfade bei $s = 10$

	SG	Heuristik 1		Heuristik 2		Heuristik 3	
	Dijkstra	Dijkstra	CH	Dijkstra	CH	Dijkstra	CH
1.1	98ms	43ms	19ms	38ms	11ms	38ms	13ms
1.2	211ms	90ms	39ms	80ms	22ms	80ms	27ms
1.3	455ms	192ms	83ms	173ms	44ms	173ms	53ms
2	116ms	38ms	15ms	33ms	7ms	37ms	9ms
3	137ms	34ms	13ms	30ms	6ms	30ms	7ms

5.5 Zusammenfassung

Im Vergleich mit einem naiven Algorithmus zur Berechnung von vollständigen Sichtbarkeitsgraphen, hat sich gezeigt, dass die Konstruktion von Spanner mit dem vorgestellten Verfahren deutlich weniger Zeit benötigt. Die auf den Spannern berechneten Pfadkosten unterscheiden sich dabei, abgesehen von wenigen Ausreißern, nur unwesentlich von den echten Pfadkosten. Durch die Verwendung von Contraction Hierarchies kann die Zeit für die Berechnung von kürzesten Pfaden wesentlich gesenkt werden, wenn auch nicht annähernd so sehr wie bei Graphen von Straßennetzwerken. Zudem ist die Berechnungszeit der CH von Spannern deutlich höher als für jene von Straßengraphen.

Tabelle 8: Durchschnittliche und maximale Teuerung kürzester Pfade bei $s = 20$

	Heuristik 1		Heuristik 2		Heuristik 3	
Instanz	durchschn.	max.	durchschn.	max.	durchschn.	max.
1.1	0.1%	4.2%	0.2%	4.2%	0.2%	3.9%
1.2	0.1%	3.1%	0.1%	4.1%	0.2%	2.2%
1.3	0.1%	1.6%	0.1%	6.1%	0.2%	1.3%
2	0.3%	8.6%	0.3%	8.6%	0.3%	5.0%
3	0.5%	11.8%	0.5%	11.0%	0.4%	5.1%

Tabelle 9: Durchschnittliche und maximale Teuerung kürzester Pfade bei $s = 10$

	Heuristik 1		Heuristik 2		Heuristik 3	
Instanz	durchschn.	max.	durchschn.	max.	durchschn.	max.
1.1	0.4%	9.7%	0.5%	28.0%	0.6%	4.8%
1.2	0.3%	14.0%	0.4%	7.4%	0.5%	5.7%
1.3	0.3%	5.9%	0.3%	6.5%	0.4%	4.6%
2	1.0%	16.1%	1.0%	17.9%	0.8%	8.9%
3	1.7%	27.0%	1.4%	20.5%	1.1%	13.4%

6 Fazit und Ausblick

Ziel dieser Arbeit war es zu untersuchen, inwiefern sich die Konstruktion von Spannern und Nutzung von Beschleunigungsverfahren für Graphen von Straßennetzwerken zur Beschleunigung von kürzesten-Pfad-Anfragen auf Sichtbarkeitsgraphen eignen. An verschiedenen, künstlich generierten Probleminstanzen konnte experimentell gezeigt werden, dass auf der Well-Separated Pair Decomposition basierende Spanner deutlich schneller als der vollständige Sichtbarkeitsgraph berechnet werden können. Dennoch bestehen hier weitere Möglichkeiten zur Verbesserung. Da ein großer Anteil der Berechnungszeit des Spanners für die Suche nach Kanten zwischen Paaren der Well-Separated Pair Decomposition verwendet wird, zwischen denen keine Sichtbarkeitskante realisiert werden kann, könnte eine Methode, die solche Paare identifiziert und verwirft, zu einer Senkung der Berechnungszeit führen. Die Anwendung von Contraction Hierarchies auf Spanner von Sichtbarkeitsgraphen hat sich als weniger erfolgreich erwiesen. Während Contraction Hierarchies ein sehr effizientes Routing-Verfahren für Straßennetzwerke darstellen, sind sie aufgrund der hohen Kantendichte für Sichtbarkeitsgraphen weniger geeignet. Eine Möglichkeit die Kanten des Spanners zu reduzieren, ohne Distanzen zwischen Knoten wesentlich zu beeinflussen, könnte die Berechnungszeit und Anzahl der entstehenden Shortcuts verringern. Weiterhin offen bleibt, ob sich andere bekannte Routing-Algorithmen, wie beispielsweise Hub Labels [14] oder Transit Node Routing [15], zur Anwendung auf Sichtbarkeitsgraphen effizient nutzen lassen. In dem vorgestellten Verfahren wird davon ausgegangen, dass die Endpunkte des gesuchten Pfads Knoten des Spanners sind. Bei variablen Endknoten wäre eine Methodik lohnenswert, die diese möglichst effizient in den bereits berechneten Spanner integriert.

Literatur

[1] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck, "Route planning in transportation networks," in *Algorithm engineering*, pp. 19–80, Springer, 2016.

[2] J. Canny and J. Reif, "New lower bound techniques for robot motion planning problems," in *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pp. 49–60, IEEE, 1987.

[3] J. Hershberger and S. Suri, "An optimal algorithm for euclidean shortest paths in the plane," *SIAM Journal on Computing*, vol. 28, no. 6, pp. 2215–2256, 1999.

[4] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[5] S. Oh and H. W. Leong, "Edge n-level sparse visibility graphs: Fast optimal any-angle pathfinding using hierarchical taut paths," *arXiv preprint arXiv:1702.01524*, 2017.

[6] L. Aleksandrov, H. N. Djidjev, H. Guo, A. Maheshwari, D. Nussbaum, and J.-R. Sack, "Algorithms for approximate shortest path queries on weighted polyhedral surfaces," *Discrete & Computational Geometry*, vol. 44, no. 4, pp. 762–801, 2010.

[7] P. B. Callahan and S. R. Kosaraju, "A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields," *Journal of the ACM (JACM)*, vol. 42, no. 1, pp. 67–90, 1995.

[8] R. Geisberger, P. Sanders, D. Schultes, and C. Vetter, "Exact routing in large road networks using contraction hierarchies," *Transportation Science*, vol. 46, no. 3, pp. 388–404, 2012.

[9] M. De Berg, M. Van Kreveld, M. Overmars, and O. Schwarzkopf, "Computational geometry," in *Computational geometry*, pp. 1–17, Springer, 1997.

[10] S. K. Ghosh and D. M. Mount, "An output-sensitive algorithm for computing visibility graphs," *SIAM Journal on Computing*, vol. 20, no. 5, pp. 888–910, 1991.

[11] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM (JACM)*, vol. 34, no. 3, pp. 596–615, 1987.

[12] M. H. Smid, "The well-separated pair decomposition and its applications.," 2018.

[13] R. A. Jarvis, "On the identification of the convex hull of a finite set of points in the plane," *Information processing letters*, vol. 2, no. 1, pp. 18–21, 1973.

[14] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck, "A hub-based labeling algorithm for shortest paths in road networks," in *International Symposium on Experimental Algorithms*, pp. 230–241, Springer, 2011.

[15] H. Bast, S. Funke, P. Sanders, and D. Schultes, "Fast routing in road networks with transit nodes," *Science*, vol. 316, no. 5824, pp. 566–566, 2007.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Datum und Unterschrift:

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Date and Signature: