Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik I

Alexander Klein

Effiziente Berechnung einer dilationsminimalen Triangulierung

1. Februar 2006

Diplomarbeit

Betreuer: Prof. Dr. Rolf Klein

Erklärung

Mit der Abgabe der Diplomarbeit versichere ich, gemäß 19 Absatz 7 der DPO vom 15. August 1998, dass ich die Arbeit selbstständig durchgeführt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Königswinter den, 01. Februar 2006

Alexander Klein

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Dilation tritt in vielen Bereichen des täglichen Lebens auf. Sucht man sich einen Reiseweg heraus und überlegt sich, welchen Weg man am besten mit dem Auto nimmt, so trifft man eine Entscheidung über verschiedene Wege um zum Ziel zu gelangen. Da in den seltensten Fällen der direkte Weg, also die Luftlinie, befahrbar ist, muß man sich zwischen mehreren Umwegen entscheiden. Natürlich versucht man einen Weg zu nehmen, der möglichst die gleiche Länge hat, wie die Luftlinie und so unterscheiden sich die verschiedenen Wege durch ihre Güte. Die Dilation ist ein Maß für die Güte eines Weges. Sie setzt sich zusammen aus dem Umweg zwischen zwei Punkten und dem direkten Weg ($\frac{Umweg}{Luftlinie}$). Abstrahiert man die Fragestellung, so gelangt man zu einem geometrischen Graphen in der Ebene, welcher sich aus miteinander verbundenen Knoten zusammensetzt. In dieser Diplomarbeit geht es um die *graphentheoretische Dilation*. Einen Überblick zur *graphentheoretische Dilation* liefert ein Übersichtsartikel von Eppstein[Epp00]. Eine zusätzliche und ausführliche Untersuchung stellt auch die Diplomarbeit von T. Dickmeiß[Dic05] da. Im Unterschied zur *geometrischen Dilation*, wo man auch mitten auf einer Verbindungslinie / Kante starten beziehungsweise seinen Weg beenden kann, darf man bei der graphentheoretischen Dilation nur zwischen den Knotenpunkten eines Graphen hin und her wandern. Es werden nur ungerichtete Graphen betrachtet, bei denen die Kanten in beide Richtungen zu gleichen Kosten (z.B. Kantenlänge) benutzt werden können. Weitere Erkenntnisse zur gometrischen Dilation sind unter anderem auch bei A. Ebbers-Baumann, A. Grüne und R. Klein[EBGK04] zu finden.

Im Fall eines vollständigen Graphen (jeder Knoten eines Graphen ist mit jedem anderen Knoten verbunden) beträgt die Dilation 1, da es keinen Umweg gibt, der in Kauf genommen werden muß (siehe Abbildung 1.1 a). Bei Graphen mit weniger Kanten (zum Beispiel einer Triangulierung, siehe Abbildung 1.1 b) können oder müssen Umwege in Kauf genommen werden um von einem Knoten zu einem Anderen gelangen zu können und die Dilation wird 1. Sollte die Kantenzahl so weit schwinden, daß ein nicht zusammenhängender Graph (siehe Abbildung 1.1 c) entsteht, so beträgt die Dilation ∞, da man nicht mehr zwischen jeden beliebigen Punkten über die Kanten des Graphen *reisen* kann. Hat man jedoch eine Knotenmenge gegeben, welche man mit einer begrenz-

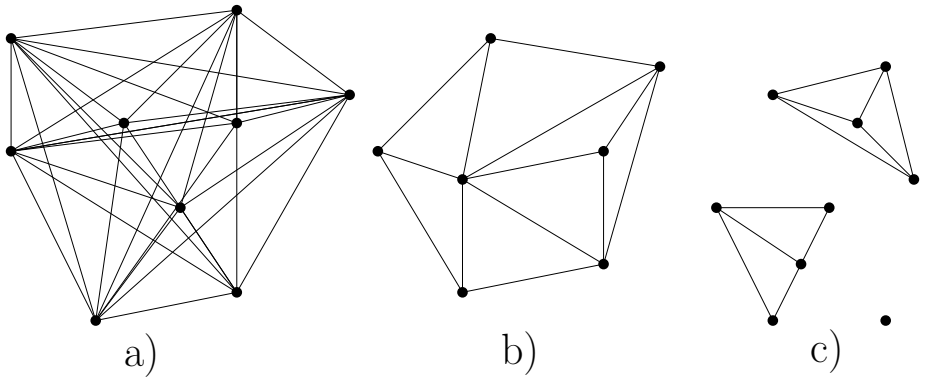


Abbildung 1.1: a) Vollständiger Graph, b) Triangulierung, c) nicht zusammenhängender Graph.

ten Anzahl von Kanten verbinden soll, so steht man vor einem Problem, sollte man nicht genug Kanten für einen vollständigen Graph zur Verfügung haben. Es gibt viele Möglichkeiten, wie man die Knoten verbinden kann und jedesmal muß man Abstriche machen, da es immer Punkte gibt, die nur über einen Umweg erreicht werden können. Für jeden möglichen Graphen gibt es mindestens ein Knotenpaar, dessen Dilation maximal ist, also der Dilation des Graphen entspricht. Wünschenswert wäre es, wenn man den Graphen finden würde, für den diese höchste Dilation möglichst gering ist. In dieser Diplomarbeit behandele ich Möglichkeiten zu einer gebenener Punktmenge eine Triangulierung mit der kleinst möglichen Dilation zu finden. Die Frage, ob es machbar ist, so eine Triangulierung mit minimaler Dilation in polynomieller Zeit zu berechnen, ist ein ungelöstes Problem innerhalb der Algorithmischen Geometrie und wird auch von Eppstein[Epp00] (Open Problem 8, p. 451) zitiert. Diese Fragestellung ist ein zentraler Punkt dieser Diplomarbeit, bei der es um die möglichst effiziente, also auch möglichst schnelle Bestimmung der Triangulierung mit kleinster Dilation geht.

Das während dieser Diplomarbeit entstandene Applet kann unter folgender Adresse genutzt werden:

http://www.geometrylab.de/

1.2 Gliederung

Bevor es *in medias res* geht, gebe ich einen kurzen Überblick über den Aufbau der Diplomarbeit:

- In Kapitel 2 gehe ich auf die grundlegensten Definition ein und erläutere zwei Theoreme.
- Kapitel 3 geht näher auf die Eigenschaften einer dilationsminimalen Triangulierung ein und behandelt mehrere Kriterien, die bei der Berechnung der gesuchten Triangulierung helfen.
- Kapitel 4 behandelt mehrere Berechnunsgverfahren und ihre Vor- und Nachteile. Außerdem wird kurz eine Greedyheuristik untersucht.

- Kapitel 5 erläutert das während der Diplomarbeit entstandene Applet und mehrere Laufzeitexperimente.

Kapitel 2

Definitionen und Begriffserklärungen

2.1 Definitionen

Definition 1 *Graph*
Unter einem Graphen *versteht man ein Tupel* $G = (V, E)$ *aus einer endlichen Knotenmenge* V *(*Verticeset*) und einer Kantenmenge* $E \subseteq V \times V$ *(*Edgeset*) .*

Im Falle dieser Diplomarbeit beschränkt sich die Menge V der Vertices auf Punkte der Ebene ($\mathbb{R}^2$) also auf *planare Graphen* . Die Kantenmenge $E = \{(p, q)|p, q \in V\} \subseteq V \times V$ ist eine Teilmenge aller möglicher Kanten zwischen den Punkten aus V und somit auch endlich. Eine Kante $e = (p, q) \in E$ definiert sich durch zwei Punkte $p, q \in V$ und ist entweder gerichtet oder *ungerichtet* . Für alle weiteren Betrachtungen gehe ich von einer ungerichteten Kantenmenge aus. Das Punktepaar p, q einer Kante e ist also ungeordnet. Ein *vollständiger Graph* enthält alle Kanten $V \times V$.

Definition 2 *Edgeflip*
Ein Edgeflip oder nur Flip einer Kante e *ist gleichbedeutend mit der Ersetzung der Kante durch ihre duale Kante* $d(e) = (p^*, q^*)$.

Ein Beispiel ist in Abbildung 2.1 zu sehen. Allerdings wird der Begriff der *dualen Kante* in mehreren Zusammenhängen gebraucht. Gebräuchlicher ist die *duale Kante* als Bestandteil eines dualen Graphens dessen Definition gleich folgt. Bespamyatnikh [Bes00] benutzt den Begriff aber auch als Synomym eine geflippte Kante. Ebenso wie Bespamyatnikh werde auch ich enie geflippte Kante als duale Kante bezeichnen. Im anders gemeinten Ausnahmefällen werde ich vorher explizit darauf hinweisen. Die duale Kante $d(e) = (p^*, q^*)$ einer Kante $e = (p, q)$ entsteht durch einen Edgeflip oder auch Flip der Kante e. Jeder Kante e einer Triangulierung, außer den Kanten der konvexen Hülle, lassen sich vier Punkte zuordnen. Erstens die zwei Punkte p, q, welche eine Kante miteinander verbindet und zweitens die zwei Punkte p^*, q^* welche auf Basis der Kante $e = (p, q)$ jeweils ein Dreieck aufspannen. Eine Kante e ist nur dann flipbar, wenn beide Dreiecke zusammen ein konvexes Polygon aufspannen. Ansonsten würde die *duale Kante* $d(e)$ von e Dreieckskanten schneiden.

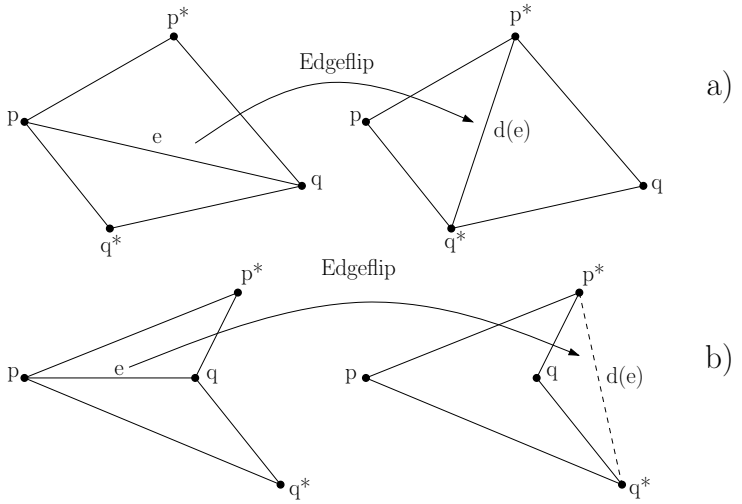


Abbildung 2.1: Kante e und zugehörige duale Kante $d(e)$, a) Die Kante e ist flipbar. b) Die Kante e ist nicht flipbar.

In Zukunft bezeichne $flip(t, e)$ einen Graphen, der durch den Edgeflip der Kante e innerhalb des Graphen t entsteht.

Definition 3 *Enumerierung*
*Der Prozess der Enumerierung (*engl. enumeration*) meint sowohl die Zählung als auch die Aufzählung von Elementen.*

Im Laufe der Diplomarbeit werden zum Beispiel zu einer Punktmenge S alle möglichen Triangulierungen über S enumeriert. Das bedeutet nichts anderes, als daß jede Triangulierung erstellt wird. Danach ist nicht nur die Anzahl der Triangulierungen sondern auch deren Aufbau bekannt.

Definition 4 *Dualer Graph*
Gegeben sein ein kreuzungsfreier, nichtleerer, zusammenhängender, endlicher geometrischer Graph G. Der zu G gehörige duale Graph G^* läßt sich wie folgt konstruieren:

- *Wähle einen Punkt p_F^* im Inneren jeder Fläche F von G. Diese Punkte sind die Knoten von G^*.*
- *Für jede Kante e von G mit angrenzenden Flächen F und F' verbinde p_F^* mit $p_{F'}^*$ mit einer Kante e^*, die nur e und sonst keine andere Kante kreuzt.*

In Abbildung 2.2 ist ein Graph G mit zugehörigem dualen Graph G^* zu sehen.

Definition 5 *Konvexe Hülle*
Die konvexe Hülle $ch(M)$ einer Punktmenge M ist die kleinste konvexe Menge in der M enthalten ist.

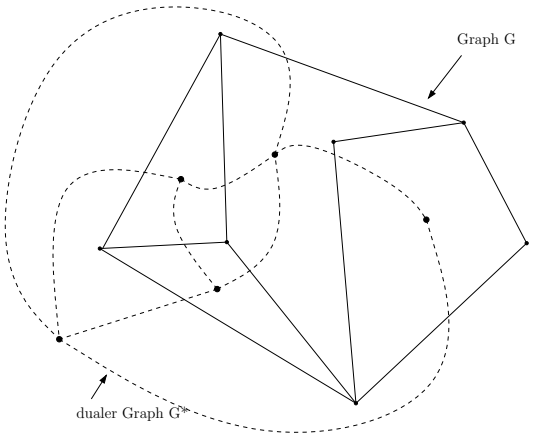


Abbildung 2.2: Graph G und dualer Graph G^* *(gestrichelt)*

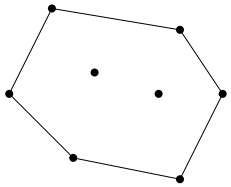


Abbildung 2.3: Beispiel für eine konvexe Hülle

Im weiteren Verlauf geht es um das Polygon der konvexen Hülle, dessen Kanten in jeder vollständigen Triangulierung der dazugehörigen Punktmenge enthalten sein müssen. Abbildung 2.3 zeigt ein Beispiel.

Definition 6 *Triangulierung*
Eine Triangulierung *einer Punktmenge S ist eine maximale Menge - bis auf die Endpunkte - paarweise disjunkter, gradliniger, kreuzungsfreier Kanten, deren Endpunkte beide in S liegen müssen.*

Einzelne Kanten haben also keine echten Schnittpunkte miteinander sondern berühren sich nur an den Punkten (Knoten). Dies kommt einer Zerlegung des Gebietes der konvexen Hülle in Dreiecke gleich (siehe dazu auch Klein [Kle98] Seiten 235f.). Die Abbildung 2.4 zeigt ein Beispiel für eine Triangulierung einer Punktmenge.

Definition 7 *Euklidische Distanz*
Die euklidische Distanz oder auch Metrik ist eine von vielen möglich Metriken um zwei Punkten p, q eine Entfernung zuordnen zu können. Sie wird für Punkte in der Ebene $p = (x, y), x, y \in \mathbb{R}^2$ wie folgt berechnet:

$$d(p,q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

Neben der euklidischen Metrik gibt es auch viele weitere Metriken, wie zum Beispiel die Manhattan Metrik oder die Karlsruhe Metrik (siehe Klein [Kle98], Seiten 239f.).

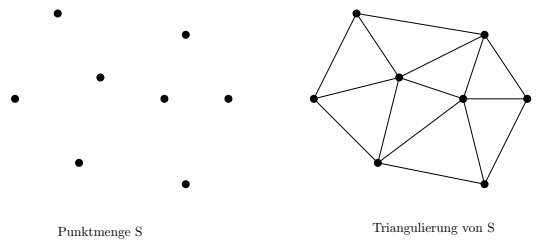


Abbildung 2.4: mögliche Triangulierung einer Punktmenge

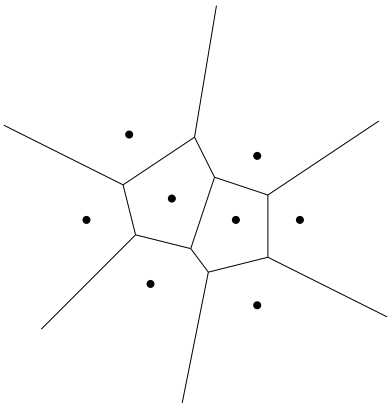


Abbildung 2.5: Beispiel für ein Voronoidiagramm

Definition 8 *Voronoi-Diagramme*
Ein Voronoidiagramm $VD(S)$ einer Punktmenge $S = (p_1, p_2, .., p_n)$ im $\mathbb{R}^2$ ist eine Einteilung der Ebene in verschiedene offene und konvexe Gebiete gleicher nächster Nachbarn. Für einen Punkt p_i ist die Voronoiregion $V(p_i)$ die Menge aller Punkte, die näher zu p_i sind, als zu jedem anderen Punkt $p_j \in S, j \neq i$.

Das Voronoidiagramm $VD(S)$ besteht aus allen Voronoiregionen $V(p_1), V(p_2)$, .., $V(p_n)$. Die verschiedenen Voronoiregionen sind durch Bisektoren von einander getrennt, welche zwischen den Punkten aus S verlaufen. Alle Punkte auf diesem Bisektoren besitzen zu den Voronoipunkten der angrenzenden Voronoiregionen denselben Abstand und können nicht einer einzigen Region zugeordnet werden. Die Bisektoren werden auch *Voronoikanten* und die Punkte, an denen die Bisektoren zusammenlaufen, *Voronoiknoten* genannt.

Definition 9 *Delaunay-Triangulierung*
Die Delaunaytriangulierung ist ein dualer Graph des Voronoidiagrammes, so daß die Knoten den Punkten aus S entsprechen und geradlinig sind.

Abbildung 2.6 zeigt ein solches Beispiel.
Die Delaunaytriangulierung weist zudem folgende Eigenschaften auf:

- Die Delaunaytriangulierung zerlegt das Gebiet der konvexen Hülle $ch(S)$ in Dreiecke.
- Der Umkreis jedes einzelnen dieser Delaunaydreiecke enthält keinen Punkt aus S.

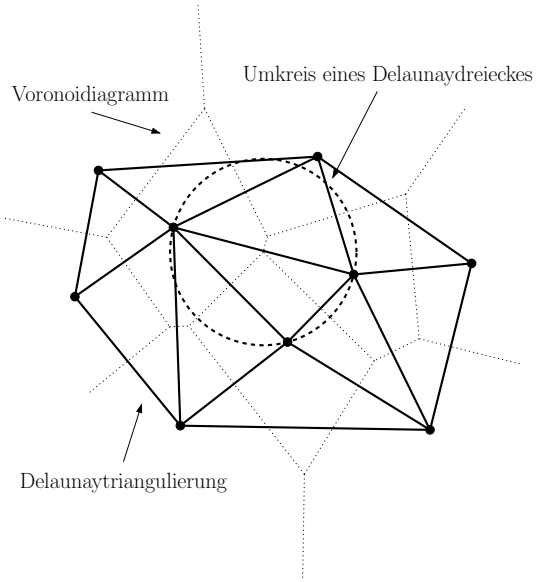


Abbildung 2.6: Delaunaytriangulierung als dualer Graph eines Voronoidiagrammes

- Die Delaunaytriangulierung vermeidet spitze Winkel innerhalb der Delaunaydreiecke.

Diese und weitere Eigenschaften werden bei Klein[Kle98] (Seiten 233f.) und Eppstein[Epp00] genauer untersucht.

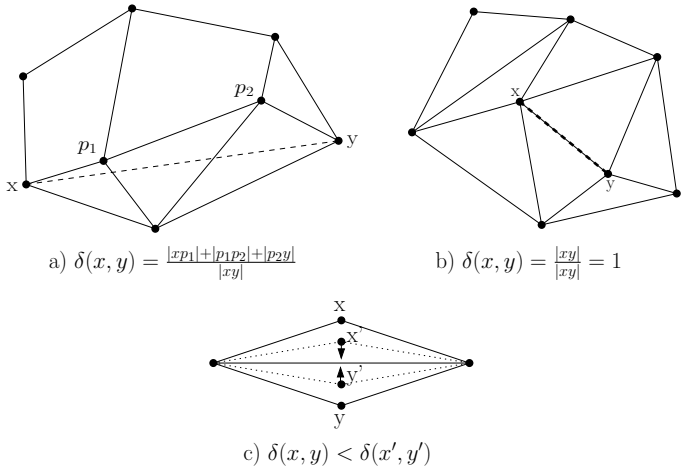
Definition 10 *Dilation*
Gegeben sei ein zusammenhängender und nicht leerer Graph $G = (V, E)$. Die Dilation ist ein Maß für den Umweg der in Kauf genommen werden muß um von einem Punkt $p \in V$ zu einem anderen Punkt $q \in V$ zu gelangen. Bezeichne $\pi_G(p, q)$ den kürzesten Pfad über die Kanten des Graphen G von p nach q. Dann berechnet sich die Dilation *$\delta_G(p, q)$ aus dem Quotient der Länge dieses kürzesten Pfades $|\pi_G(p, q)|$ und der euklidischen Distanz $d(p, q)$:*

$$\delta_G(p, q) = \frac{|\pi_G(p, q)|}{d(p, q)}.$$

Die größte auftretende Dilation $\delta(G) = \max(\delta_G(x, y)|x, y \in V)$ heißt Dilation des Graphen G.

In Abbildung 2.7 sind mehrere Beispiele für die Dilation zwischen 2 Punkten zu sehen (a)$\delta(x, y) > 1$, b) $\delta(x, y) = 1$, c) $\lim_{|xy|\to 0} \delta(x, y) = \infty$). Für jedes Punktpaar p, q mit $p \neq q$ läßt sich nun die Dilation innerhalb des Graphen G bestimmen.

Definition 11 *Minimale Dilationstriangulierung*
Gegeben sei eine Punktmenge S. Zu dieser Punktmenge existiert eine Menge aller Triangulierung $T(S)$ über S. Die Triangulierung(en)



a) $\delta(x,y) = \frac{|xp_1|+|p_1p_2|+|p_2y|}{|xy|}$ b) $\delta(x,y) = \frac{|xy|}{|xy|} = 1$

c) $\delta(x,y) < \delta(x',y')$

Abbildung 2.7: (a) Beispiel für die Dilationsbestimmung zwischen zwei Punkten x und y, (b) zwei Punkten x, y mit Dilation = 1, (c) Die Dilation kann bei sich annähernden Punkten x', y' beliebig groß werden.

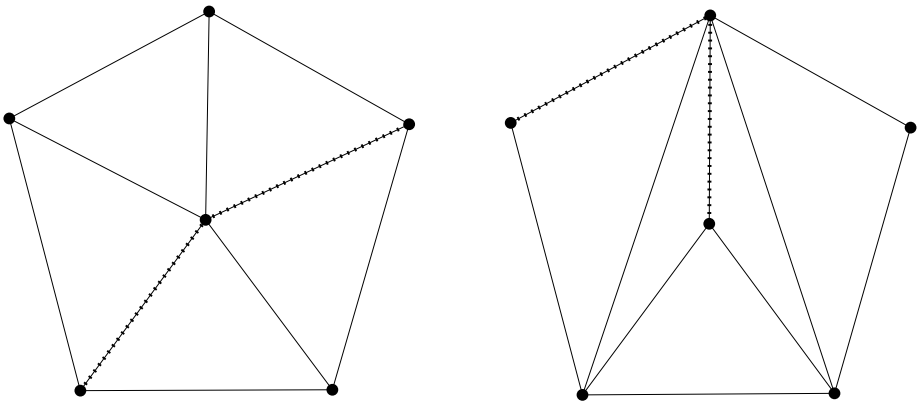
$$t_{MDT} = \{t \in T(S) | \delta(t) = \min_{t' \in T(S)} (\delta(t'))\}$$

mit der kleinsten Dilation nennen sich minimale Dilationstriangulierung(en) *von* S.

Da sie im Englischen auch als *minimum dilation triangulation(s)* bezeichnet werden und ich im Laufe der Arbeit noch öfter darauf zu sprechen komme, werde ich in Zukunft die Abkürzung *MDT* benutzen. Abbildung 2.8 zeigt ein Beispiel für eine eindeutige *MDT*. Aus der Definition und auch aus dem Beispiel 2.9 läßt sich erkennen, das die *MDT* nicht immer eindeutig sein muß, sondern daß es auch mehrere Triangulierungen mit der kleinsten Dilation geben kann. Trotzdem werde ich in Zukunft der Einfachheit halber meistens von *der MDT* sprechen.

Definition 12 *Minimaler Spannbaum*
Gegeben sei ein ungerichteter, zusammenhängender Graph $G = (V, E)$*, dessen Kanten mit Gewichten markiert sind. Gesucht ist ein Baum* B*, der den Graphen* aufspannt *und minimales Kantengewicht hat.*

Der Baum B spannt G dann auf, wenn er ein zusammenhängender Teilgraph von G ist und alle Knoten von G enthält. Im Falle dieser Diplomarbeit sind die Kanten mit ihrer geometrischen Kantenlänge gewichtet. Abbildung 2.10 zeigt ein Beispiel für einen minimalen Spannbaum. In einem minimalen Spannbaum gibt es drei Arten von Kanten, einerseits Kanten welche beidseitig nächste Nachbarn verbinden, also Punkte, die sich gegenseitig als nächste Nachbarn haben, andererseits Kanten die einseitig nächste Nachbarn verbinden und Kanten, die keine nächsten Nachbarn verbinden. Weiterhin sei darauf hingewiesen das der minimale Spannbaum nicht eindeutig sein muß, es kann also auch mehrere geben, zum Beispiel bei Punkten in einem regelmäßigen Gitter.



a) $\delta(T) = 1.05$ b) $\delta(T) = 2,19$

Abbildung 2.8: Beispiel für eine Triangulierung aus 5 Punkten mit a) der zugehörigen MDT und b) der mit maximaler Dilation. Die gestrichelten Kanten zeigen jeweils den Umweg zwischen den beiden dilationsbestimmenden Punkten auf.

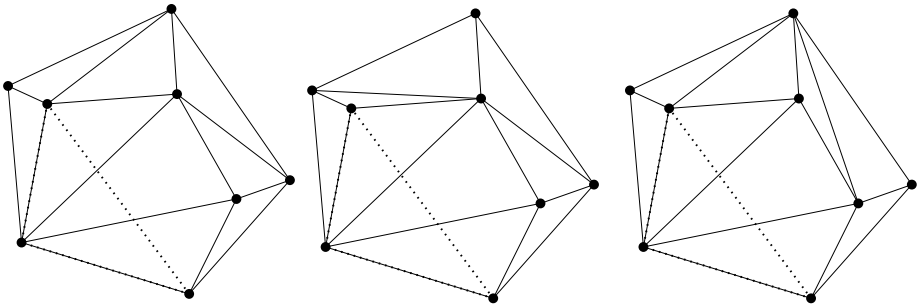


Abbildung 2.9: Drei Triangulierungen mit gleicher minimaler Dilation.

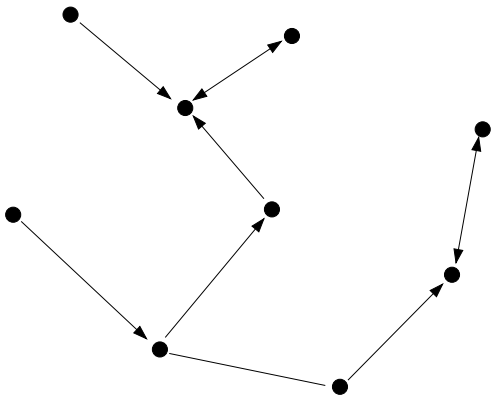


Abbildung 2.10: Beispiel für einen minimalen Spannbaum. Die Pfeile zeigen auf den nächsten Nachbarn. Es gibt also Kanten die beidseitig nächste und einseitig nächste und normale Nachbarn verbindet.

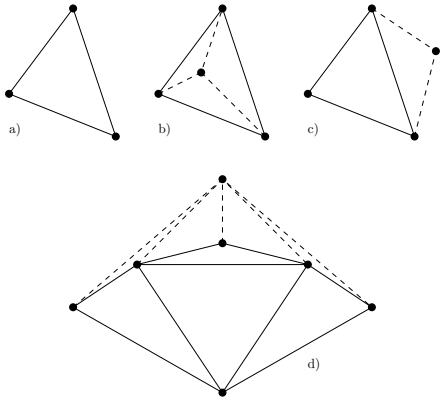


Abbildung 2.11: a) Induktionsanfang für 3 Punkte. b) Einfügen eines inneren Punktes. c) Einfügen eines äußeren Punktes, welcher die konvexe Hülle erweitert. d) Einfügen eines äußeren Punktes welcher Punkte der alten konvexen Hülle überdeckt.

2.2 Theoreme

2.2.1 Anzahl an Triangulierungskanten

Gegeben sei eine Punktmenge S mit insgesamt n Punkten. Davon liegen n^* Punkte auf der konvexen Hülle. Außerdem sei folgende Eigenschaft nicht erfüllt: Drei oder mehr Punkte sind kollinear zueinander, liegen also auf einer gemeinsamen Geraden. Dann gilt

Theorem 1 *Die Anzahl e der Kanten einer möglichen Triangulierung lässt sich für S mit folgender Formel berechnen:*

$$e = 3n - n^* - 3 \tag{2.1}$$

Eine Punktmenge S mit $n = |S|$ Punkten haben die zugehörigen Triangulierungen $O(n)$ Kanten. Erläuternde Skizzen zum folgenden Induktionsbeweis sind in Abbildung 2.11 zu finden.

Beweis per Induktion: Für $n = 3$ ist auch $n^* = 3$.

$$\begin{aligned} e &= 3n - n^* - 3 \\ &= 3. \end{aligned}$$

Für $n = 3$ ist die Formel bewiesen. Was passiert nun wenn man einen weiteren Punkt p_{neu} hinzufügt. Hier gibt es zwei Fälle zu unterscheiden. Einerseits kann p_{neu} innerhalb der alten Triangulierung liegen, oder außerhalb. Liegt p_{neu} innerhalb der alten Triangulierung so entstehen 3 neue Kanten und n^* ändert sich nicht.

$$\begin{aligned} e_{n+1} &= 3(n+1) - n^*_{n+1} - 3 \\ &= 3n + 3 - n^*_n - 3 \\ &= e_n + 3 \end{aligned}$$

Andererseits kann p_{neu} auch außerhalb der konvexen Hülle liegen. In diesem Fall gibt es wiederum zwei Möglichkeiten.

Erstens kann p_{neu} die konvexe Hülle erweitern, so dass alle Punkte der alten konvexen Hülle auch in der neuen konvexen Hülle enthalten sind. In diesem Fall werden zwei zusätzliche Kanten eingefügt und die Menge der Punkte auf der konvexen Hülle wächst um den Punkt p_{neu} an. Es gilt also $n^*_{n+1} = n^*_n + 1$.

$$\begin{aligned} e_{n+1} &= 3(n+1) - n^*_{n+1} - 3 \\ &= 3n + 3 - (n^*_n + 1) - 3 \\ &= 3n + n^*_n - 3 + 2 \\ &= e_n + 2 \end{aligned}$$

Bleibt nur noch der zweite Fall wo p_{neu} alte Punkte auf der konvexen Hülle überdeckt und diese nicht mehr in der neuen konvexen Hülle enthalten sind. Die Anzahl der überdeckten Punkte werde ich im folgendem mit n_c bezeichnet. Durch p_{neu} entstehen mindestens zwei weitere Kanten (siehe den vorherigen Fall). Zusätzlich entstehen für jeden überdeckten Punkt eine weitere Kante zwischen diesem und p_{neu}. Es entstehen also zusätzlich noch n_c neue Kanten. Insgesamt steigt also die Anzahl der Kanten um $2 + n_c$.

$$\begin{aligned} e_{n+1} &= 3(n+1) - n^*_{n+1} - 3 \\ &= 3n + 3 - (n_n + 1 - n_c) - 3 \\ &= 3n + 3 - n^*_n - 3 + 3 - 1 + n_c \\ &= e_n + 2 + n_c \\ &= 3n - n^* - 3 + 2n_c \end{aligned}$$

Damit wäre gezeigt, daß sich bei einer nach diesem Muster erstellten Triangulierung die Anzahl der Kanten berechnen lassen. Fortune [For87] zeigte, daß man von jeder beliebigen Triangulierung durch Edgeflips die Delaunaytriangulierung erreichen kann. Da Edgeflips die Kantenzahl einer Triangulierung nicht ändern folgt daraus, daß nicht nur die so erstellte Triangulierung die gleiche Kantenzahl hat wie die Delaunaytriangulierung, sondern daß jede Triangulierung die gleiche Kantenzahl besitzt und die Formel somit Gültigkeit für alle Triangulierungen der Punktmenge S hat.

□

2.2.2 Berechenbarkeit von Binärkombinationen

Theorem 2 *Gegeben sei eine Binärzahl mit n Ziffern. Gesucht ist nun die Anzahl an Kombinationen bei denen $0 \leq k \leq n$ Bits gesetzt und $n-k$ Bits nicht gesetzt sind. Dies läßt sich mittels des Binomialkoeffizienten berechnen:*

$$\#Kombinationen = \binom{n}{k} \tag{2.2}$$

Argumentativ läßt sich dieses Theorem ganz leicht begründen. Darf kein Bit einer n-elementigen Binärzahl gesetzt sein, so gibt es dafür nur eine Möglichkeit.

Für den Fall, daß nur ein Bit gesetzt sein darf, kann jedes Bit einmal gesetzt sein und es ergeben sich n Möglichkeiten. Soll hingegen jedes Bit gesetzt sein, so gibt es auch dafür nur eine Möglichkeit. Der Fall für $1 < k < n$ gesetzte Bits läßt sich mit einem Induktionsschritt begründen.

Angenommen gesucht ist die Anzahl an Kombinationen für k gesetzte Bits einer Binärzahl mit n Ziffern und man kennt die Lösung für k gesetzte Bits einer Binärzahl mit $n-1$ Ziffern. Dann ändert sich beim Schritt von der $n-1$ auf die n-stellige Binärzahl die Anzahl so, das die Zustände der $n-1$ stelligen Zahl einmal mit einer vorangestellten 0 und einmal mit einer vorangestellten 1 übernommen werden. Im Falle der vorangestellten 0 kann die Anzahl an Kombinationen für k gesetzte Bits übernommen werden. Die Null ändert ja nichts daran. Im Falle der vorangestellten 1 jedoch kommt nun ein gesetztes Bit hinzu und wir können die Kombinationen mit $k-1$ gesetzten Bits aus der $n-1$ Binärzahl übernehmen. Die unten aufgelistete rekursive Formel gibt dieses Verhalten wieder und stimmt mit der rekursiven Definition des Binomialkoeffizienten überein.

$$
\begin{aligned}
f(n,0) &= 1 \\
f(n,1) &= n \\
f(n,n) &= 1 \\
f(n,k) &= f(n-1,k) + f(n-1,k-1)
\end{aligned}
$$

□

Kapitel 3

Minimale Dilationstriangulierungen

3.1 Einleitung

Um den Begriff der Dilation besser fassen zu können und eine Vorstellung davon zu bekommen, folgen nun ein paar Beispiele an denen sich schon typische Merkmale und Eigenschaften sowie Probleme erkennen lassen. In Abbildung 3.1 sind Beispiele für Dilationsmöglichkeiten bei 4 Punkten zu sehen.

Daß die Dilation nicht kleiner sein kann als 1, dürfte allein schon aus der Berechnungsgleichung $dilation = \frac{\text{Umweg}}{\text{direkter Weg}}$ hervorgehen. Ein Umweg wird niemals kürzer sein als der direkte Weg zwischen Start- und Zielpunkt. Bei Eppstein[Epp] wird genauer auf Graphen mit Dilation 1 in der Ebene eingegangen. In Beispiel a) ist ein solches Beispiel zu sehen. In einem regelmäßigen quadratischem Gitter beträgt die Dilation hingegen $\sqrt{2}$ (Abbildung 3.1 b)) und in Beispiel c) sieht man, daß die Dilation auch beliebig groß werden kann, wenn die gekennzeichneten Punkte noch näher zusammenrücken würden.

In Abbildung 3.2 ist eine komplexere Triangulierung zu sehen, die schon eher eine Vorstellung vom Dilationsverhalten gibt. In diesem Beispiel ist der Umweg zwischen den dilationsbestimmenden Punkten gestrichelt hervorgehoben und wie man an diesem Beispiel sieht, muß der Umweg nicht nur über einen weiteren Punkt gehen sondern kann sich quer durch die gesamte Triangulierung oder zumindest über mehrere Punkte hinweg ziehen. Das läßt natürlich die Frage

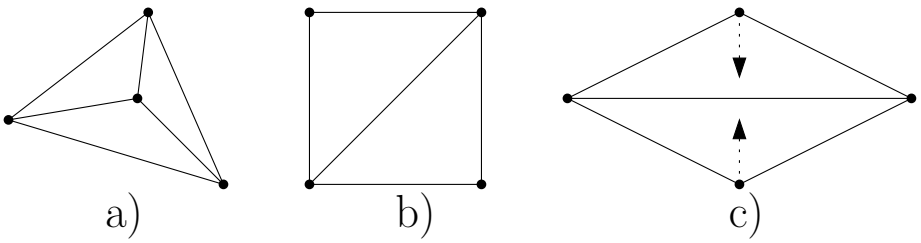


Abbildung 3.1: Dilation bei 4 Punkten: a) $dilation = 1$, b) $dilation = \sqrt{2}$, c) $dilation \to \infty$

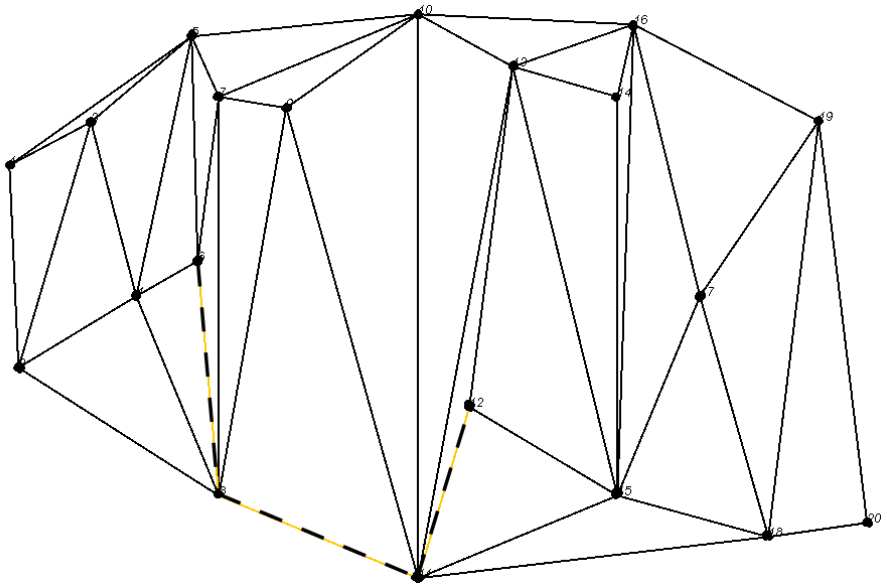


Abbildung 3.2: Ein Beispiel für größere Punktmengen mit dilationsbestimmenden Umweg(gestrichelt).

nach den Eigenschaften und Charakteristika der Dilation aufkommen, welchen ich mich nun widmen werde.

3.2 Lokale Eigenschaften

In diesem Kapitel werde ich näher auf die Dilationseigenschaften von Kanten und ihre Auswirkungen eingehen. Dabei unterscheide ich zwischen lokalen und globalen Eigenschaften. Damit verhält es sich allerdings ein wenig zwiespältig, da sich die Dilation, wie man vielleicht aus den voran gegangenen Erläuterungen erkennen kann, eher durch ein globales Zusammenspiel der Kanten auszeichnet, als ein lokales. Trotzdem gibt es Kantenkriterien, die sich sowohl lokal, als auch global überprüfen lassen. Als lokales Kriterium werde ich auf eine von Knauer und Mulzer eingeführte Exlcusion-Region [KM] näher eingehen. Der Hinderniswert einer Kante stellt hingegen eine globale Eigenschaft dar, auf die ich danach eingehen werde. Als Einleitung zur Exclusion-Region werde ich jedoch zuerst den minimalen Spannbaum einer Punktmenge S näher erläutern.

3.2.1 Minimaler Spannbaum

Ein minimaler Spannbaum einer Punktmenge S stellt, wie in Kapitel 2 schon beschrieben einen Graph mit kleinster Gesamtkantenlänge, welcher alle Punkte aus S zu einer einzigen Zusammenhangskomponente verbindet, dar. Einer der möglichen Algorithmen, um einen minimalen Spannbaum zu berechen, ist ein Greedyalgorithmus. Dieser beginnt mit einem Graphen, welcher nur die Punkte aus S enthält, und nimmt sich sukzessive die Kanten kleinster Länge, testet ob deren Hinzunahme einen Zyklus hervorruft und fügt sie, wenn dem nicht so ist,

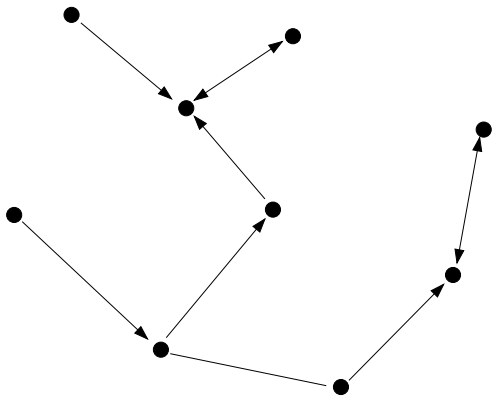


Abbildung 3.3: Beispiel für einen minimalen Spannbaum. Die Pfeile zeigen auf den nächsten Nachbarn. Es gibt also Kanten die beidseitig und einseitig nächste und normale Nachbarn verbinden.

dem Graphen hinzu. Nach und nach werden so die Punkte miteinander bis zum minimalen Spannbaum verbunden. Dabei treten besondere Kanten auf. Nämlich die Kanten, welche beidseitig eindeutige nächste Nachbarn verbinden.

Lemma 1 *Zwei Punkte $x, y \in S$ sind genau dann eindeutig nächste Nachbarn, wenn gilt: $\forall z \in S \setminus \{x, y\} : |xz| > |xy|$, und $|yz| > |xy|$.*

Siehe dazu auch Abbildung 3.3. Wie in Skizze 3.5 zu sehen ist, läßt sich aus dieser Situation folgender Schluß ziehen:

Theorem 3 *Kanten, die beidseitig eindeutig nächste Nachbarn miteinander verbinden sind Bestandteil der MDT.*

Beweis:
Angenommen die Kante $\overline{uv}$ wäre nicht im Graph $G = (S, E)$ enthalten. Dann müßte der kürzeste Weg $\pi_G(u, v)$ von u nach v über die Kanten von G über mindestens einen weiteren Punkt $c \in E$ gehen und der kürzeste Weg ließe sich wie folgt abschätzen:

$$\pi_G(u, v) \geq |uc| + |cv|$$

Da der Abstand zwischen den Punkten u, c und v, c jeweils größer d ist, folgt daraus:

$$\pi_G(u, v) > |uv| + |uv| > 2d$$

Daraus folgt, daß diese Situation für die Punkte u, v eine Dilation von $\delta_G(u, v) > \frac{2d}{d}$ erzeugen würde. Paul Chew [Che89] hat jedoch gezeigt, daß die Dreiecks-Delaunaytriangulierung in der euklidischen Metrik eine obere Dilationsschranke von 2 garantiert.

$$\delta_{MDT} \leq \delta_{\triangle - Delaunaytriangulierung} \leq 2 < \delta_G$$

Somit hat die *MDT* höchstens eine Dilation von 2, also besser als der Graph, der die Kante uv nicht enthält. Daraus folgt, daß die Kanten, welche beidseitig eindeutige nächste Nachbarn verbinden, eine Teilmenge der *MDT* darstellen.

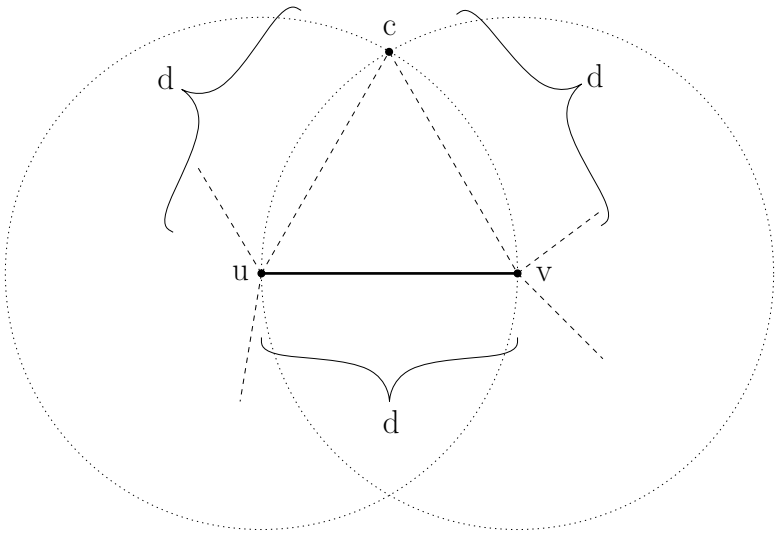


Abbildung 3.4: Eine Kante die die beidseitig eindeutigen nächsten Nachbarn u und v mit Länge $d = |uv|$ verbindet.

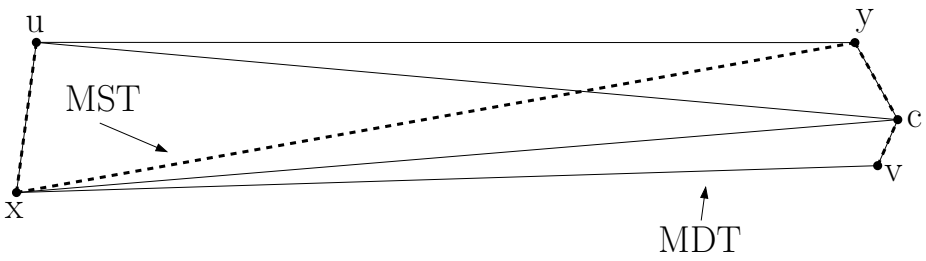


Abbildung 3.5: Der minimale Spannbaum (gestrichelt) ist nicht in der MDT enthalten da $|xy| < |uc|$ (MST) aber $|ucv| < \min(|uxv|, |uycv|)$ MDT ist.

□

Aufgrund dieser doch recht kleinen oberen Schranke für die Dilation habe ich überprüft, ob nicht vielleicht der gesamte minimale Spannbaum in der MDT enthalten ist. Damit wäre immerhin schon einmal ein größerer Teilgraph der MDT in linearer Zeit berechenbar. Leider stellte sich aber dann heraus, daß dem nicht so ist. Es lassen sich Beispiele finden, für die der minimale Spannbaum nicht in der dilationsminimalen Triangulierung enthalten ist(siehe Abbildung 3.5)

Somit scheidet dieser leider als Eigenschaft der MDT aus.

3.2.2 Exclusion Region

Den Gedanken einer Exclusion-Region, welcher beim minimalen Spannbaum schon auftauchte. verfolge ich nun mit dem Ansatz von Knauer und Mulzer

[KM] weiter. Genauer gesagt, werde ich deren Beweis skizzieren und genauer erläutern. Eine Kante kann auch für weit von einander entfernte Punkte ein großes Hindernis darstellen. Diese Fälle herauszufinden ist jedoch recht zeitaufwändig (siehe 3.3.1, $O(n^2)$ pro Kante). Viel leichter ist es zu überprüfen, ob es nicht in unmittelbarer Umgebung Punkte gibt, für die diese Kante ein Hindernis darstellt. Dies kann mittels einer Exclusion-Region überprüft werden (siehe Abbildung 3.6).

Es ist offensichtlich, daß bei einem Graph $G = (S, E)$, einer Kante $e = (u, v) \in E$ und zwei Punkten $a, b \in S \backslash \{u, v\}$, welche sehr nah am Mittelpunkt x der Kante e auf gegenübergelegenen Seite liegen, die Dilation sehr groß werden muß. Jeder Pfad $\pi_G(a, b)$ muß die Kante e umgehen. Je näher die Punkte a, b am Mittelpunkt x der Kante e liegen, um so größer die Dilation. Die Idee liegt also nahe zu überprüfen, ob Punkte in der Nähe des Kantenmittelpunktes x durch die Kante e getrennt liegen. Je kleiner der Bereich, der überprüft wird, umso größer muß die Dilation sein, wenn Punkte innerhalb der Halbkreise gefunden werden. Und genauso funktioniert auch die Exclusion-Region. Sie ist als Kreis um den Mittelpunkt einer Kante definiert und wenn in jedem der Halbkreise ein Punkt liegt, so kann man von dem Radius der Exclusion-Region auf die Dilation $\delta_G(a, b)$ schließen. Da bekannt ist, daß die Delaunaytriangulierung eine Dilation von $\delta_{Delaunay} \leq \frac{2\pi}{3\cos(\frac{\pi}{6})} \approx 2,42...$ garantiert, kann man die Exclusion-Region so groß wählen, das Punkte innerhalb der Halbkreise eine Dilation größer der Delaunaytriangulierung besitzen und eine Kante in diesem Fall nicht in der MDT vorkommen kann. Der Radius der Exclusion-Region ist proportional zur Länge der zugehörigen Kante; Schließlich stellt eine längere Kante ein größeres Hindernis da. Andererseits ist der Radius antiproportional zur Dilation. Je größer die Dilation zwischen Punkten sein soll, umso näher müssen diese am Mittelpunkt der Exclusion-Region liegen und der Radius der Exclusion-Region muß ebenso kleiner werden. Die folgenden Beweise zur Exlusion-Region sind dem Paper [KM] von Knauer und Mulzer entnommen. Es wird sich herausstellen, daß sich der Radius r einer Exclusion-Region zu einer Kante e wie folgt berechnet:

$$r = \alpha|e| \text{ mit z.B. } \alpha = \frac{1}{\delta_{Delaunay}} = \frac{3\cos(\frac{\pi}{6})}{2\pi} \approx 0,2067.$$

Dafür wird gezeigt, daß die Punkte mit kleinster Dilation auf dem Rand der Exclusion-Region liegen und welche Konstellationen von Punkten auf dem Rand die kleinste Dilation hervorrufen.

Abbildung 3.6 zeigt zwei Beispiele wie Punkte durch eine Gerade getrennt werden können. Was recht schnell auffällt, ist, daß in Beispiel a) die Dilation noch recht gering ist und vor allem noch geringer wird, wenn sich a und b in gleichbleibendem Abstand zur Kante $e = (u, v)$ auf den Punkt u zu bewegen. Je mehr es allerdings in Richtung Mittelpunkt z geht (Beispiel b), umso größer wird auch die Dilation $\delta(a, b)$. Auch wenn beide Punkte aufeinander zugehen, also sich ihr Abstand $|ab|$ und ihre Position zur Kante e sich verringert, wächst die Dilation. Dieses Verhalten lässt sich in folgender Lemma zusammenfassen (siehe auch Abbildung 3.7):

Lemma 2 *Gegeben seien ein geometrischer Graph $G = (S, E)$, eine Kante $e = (u, v) \in E$, ein Kreis D um den Mittelpunkt der Kante e und ein Punkt x, welcher auf e liegt. Sei a ein Punkt welcher innerhalb des Kreises D, aber nicht*

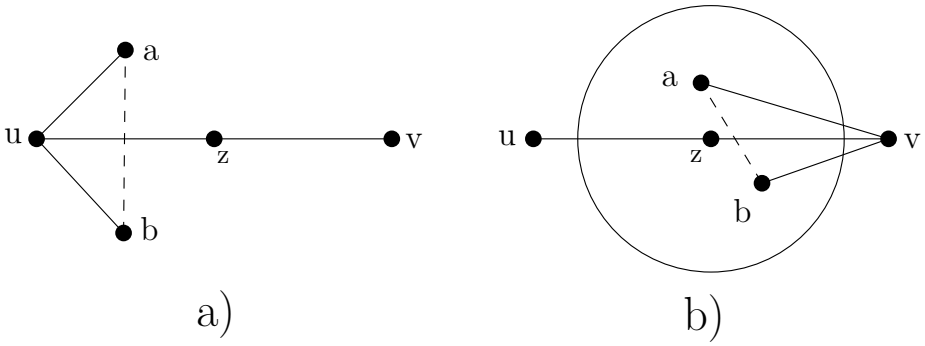


Abbildung 3.6: Verschiedene Möglichkeiten für kürzeste Pfade zwischen Punkten a und b

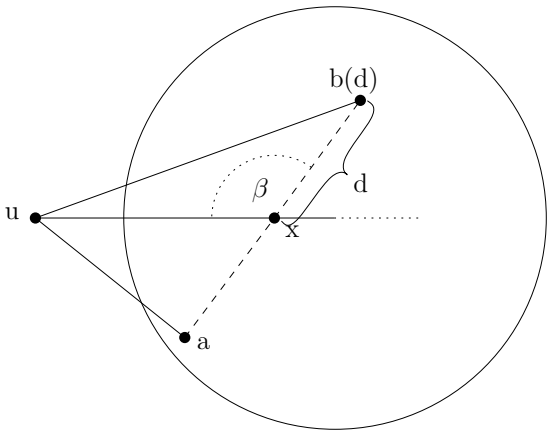


Abbildung 3.7: Skizze zu Lemma 2

auf e liegt. Für ein $d > 0$ sei $b(d)$ ein Punkt, der so auf der Verlängerung $\overline{ax}$ liegt, daß $|xb(d)| = d$ gilt. Dann fällt $\delta(a, b(d))$ monoton mit steigendem d.

Das heißt, das bei steigendem d, also je weiter sich a und $b(d)$ voneinander entfernen und näher zum Rand von D streben, die Dilation $\delta(a, b(d))$ sinkt.

Beweis
Die Dilation $\delta(a, b(d))$ berechnet sich wie folgt:

$$\delta(a, b(d)) = \frac{\min(|au| + |ub|, |av| + |vb|)}{|ax| + d}$$

Um das Lemma zu beweisen muß gezeigt werden, daß die Dilation für wachsendes d monoton fällt. Es gilt also zu überprüfen, ob sowohl $\frac{|au|+|ub(d)|}{|ax|+d}$ als auch $\frac{|av|+|vb(d)|}{|ax|+d}$ monoton fallen, wenn sich d vergrößert. Da die Argumentation aus Symmetriegründen die gleiche ist werde ich nur den Beweis für den Term $\frac{|au|+|ub(d)|}{|ax|+d}$ erläutern. Eine genauere Skizze der Situation ist in Abbildung 3.7 zu finden.

Für die Untersuchung werde ich den Term aufsplitten und als erstes den Zähler genauer betrachten. Mittels des Kosinussatzes für allgemeine Dreiecke

$$a^2 = b^2 + c^2 - 2bc\cos(\alpha)$$

läßt sich die Strecke $\overline{ub(d)}$ im Zähler ersetzen

$$|ub(d)|^2 = |ux|^2 + d^2 - 2|ux|d\cos(\beta)$$

und der gesamte Zähler ändert sich zu:

$$|ub(d)| + \sqrt{|ux|^2 + d^2 - 2|ux|d\cos(\beta)}.$$

Der Winkel β liegt, wie in Abbildung 3.7 zu sehen, gegenüber der Strecke $\overline{ub(d)}$ und wird von den Strecken $\overline{ux}$ und $\overline{xb(d)}$ eingeschlossen. Leitet man den Zähler nach d ab, so ist die erste Ableitung kleiner 1. Die erste Ableitung des Nenners $|ax| + d$ nach d beträgt 1. Somit wäre der gesamte Term $\frac{|au|+|ub(d)|}{|ax|+d}$ monoton fallend und das Lemma ist bewiesen.

□

Je weiter zwei Punkte also innerhalb einer Exclusion-Region auseinander liegen, umso kleiner wird die Dilation zwischen ihnen. Ein Punktpaar in unterschiedlichen Kreishälften der Exclusion-Region, das minimale Dilation hat, muss also mit beiden Punkten auf dem Kreisrand liegen. Um zu zeigen, welche Dilation eine Exclusion-Region mindestens bestimmt, muß nur noch gezeigt werden, in welcher Konstellation die Punkte auf dem Rand der Exclusion-Region die minimalste Dilation zu einander haben. Wenn wir also eine obere Schranke für die Dilation dank der Delaunay Triangulierung haben, so müssen wir unseren Kreisradius so anpassen, daß alle Punktepaare, die sich innerhalb des Kreises befinden eine größere Dilation als die der Delaunaytriangulierung besitzen. Wenn wir dann zu einer Kante zwei Punkte finden, die sich innerhalb der Exclusion-Region befinden, so wissen wir, daß diese Kante eine zu große Dilation erzeugt und nicht in der gesuchten MDT enthalten sein kann.

Der Radius der Exclusion-Region ist von Kante zu Kante unterschiedlich, da größere Kanten auch ein größeres Hindernis darstellen und damit bei ihnen die Exclusion-Region größer sein muss. Aus diesem Grund setzt sich der Radius einer Exclusion-Region zum einen aus einem, von der Dilation der Delaunaytriangulierung abhängigen Faktor α und zum anderen aus der Kantenlänge zusammen: $Radius = \alpha|e|$.

Die Schwierigkeit ist jetzt nur noch diesen Faktor α der bekannten oberen Dilationsschranke der Delaunaytriangulierung anzupassen. Vor allem gilt es herauszufinden, welches die minimale Dilation darstellt, die zwei Punkte auf dem Kreisrand haben können. In Abbildung 3.8 sind Möglichkeiten der Punkteverteilung auf dem Rand einer Exclusion-Region zu sehen.

Befinden sich die Punkte a, b in Abbildung 3.8 a) jeweils unendlich nahe an der Kante e, so beträgt die Dilation

$$\delta(a, b) = \frac{|e|}{2\alpha|e|} = \frac{1}{2\alpha}.$$

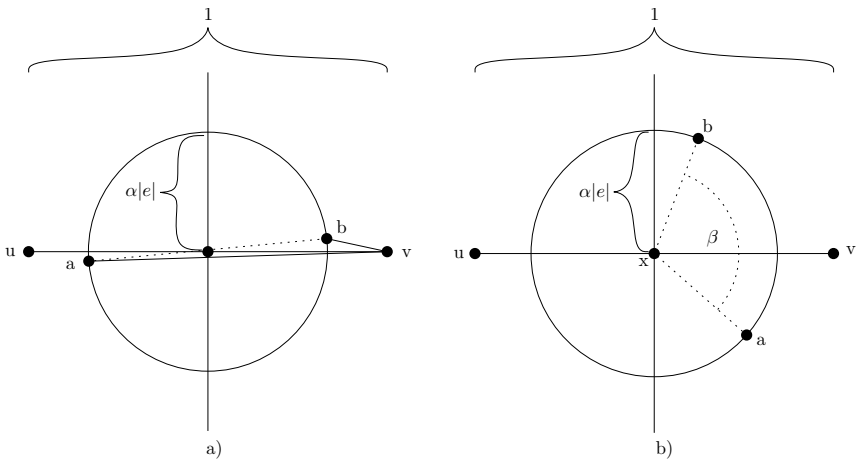


Abbildung 3.8: Mögiche Konstellationen für 2 Punkte auf dem Rand der Exclusion-Region

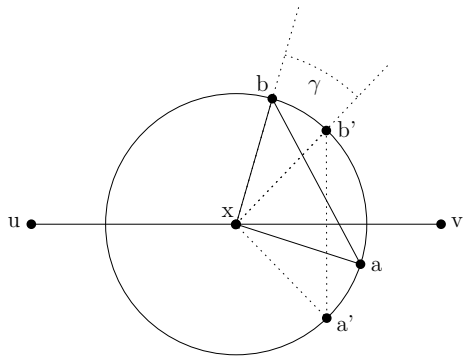


Abbildung 3.9: Skizze zur Parametrisierung

Wie sich später herausstellen wird, ist das eine der optimalen Konfigurationen. Da Knauer und Mulzer [KM] den Beweis numerisch führen, werde ich jetzt die entsprechende Parametrisierung einführen (siehe dazu Abbildung 3.8 b). Die Punkte a, b liegen dabei oBdA. auf der rechten Rand der Exclusion-Region. Deren Stellung zueinander läßt sich durch den Winkel $\beta = \angle bxa$ mit $\beta \in (0, \pi]$ beschreiben. Bei der Erklärung des Parameters γ hilft Abbildung 3.9. Der Winkel γ gibt an, um wieviel Grad man die beiden Punkte a, b um den Mittelpunkt x rotieren müßte, damit sie senkrecht übereinander stehen würden. Aus $\beta \in (0, \pi]$ folgt $\gamma \in (\frac{-\pi}{2}, \frac{\pi}{2})$. Mit Hilfe dieser beider Parameter lässt sich die Punktkonstellationen ausreichend parametrisieren.

In Abbildung 3.10 ist abzulesen wie sich $|ab|$ berechnen läßt. Da β maximal π annehmen kann und der kürzeste Weg über v geht, läßt sich Dilation zwischen a, b mittels des Kosinussatzes wie folgt berechnen (siehe Abbildung 3.10):

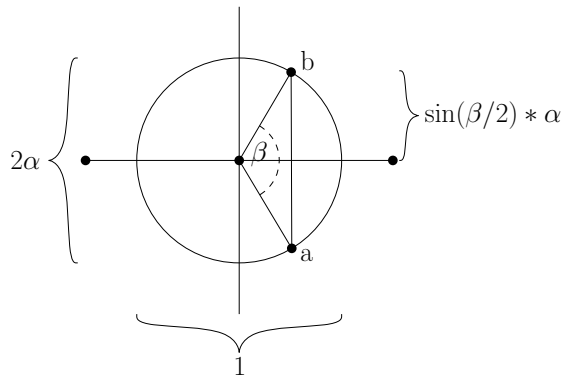


Abbildung 3.10: Weitere Skizze zur Parametrisierung

$$
\begin{aligned}
|bv| &= \sqrt{0.25 + \alpha^2 - \alpha cos(\beta/2 + \gamma)} \\
|av| &= \sqrt{0.25 + \alpha^2 - \alpha cos(\beta/2 - \gamma)} \\
|ab| &= 2\alpha \sin(\beta/2)
\end{aligned}
$$

Die Berechnung von $|bv|$ und av lassen sich wie folgt in einer Formel zusammenfassen:

$$
\begin{aligned}
f(\beta, \gamma) &= \sqrt{0.25 + \alpha^2 - \alpha cos(\beta/2 + \gamma)} \\
|bv| &= f(\beta, +\gamma) \\
|av| &= f(\beta, -\gamma)
\end{aligned}
$$

Damit ändert sich die Formel zur Berechnung der Dilation folgendermaßen:

$$\delta(\beta, \gamma) = \frac{f(\beta, \gamma) + f(\beta, -\gamma)}{2\alpha sin(\beta/2)}$$

Mittels dieser Formel können wir nun die Konstellation mit der kleinsten Dilation berechnen. Knauer und Mulzer [KM] haben genau das getan und sind dabei zu folgendem Ergebnissen gekommen. Als erstes haben sie die Formel für festes $\beta \in (0, \pi]$ optimiert.

In diesem Fall müssen zwei Unterscheidungen getroffen werden:

$$\cos(\beta/2) \leq 2\alpha \text{ und } \cos(\beta/2) > 2\alpha.$$

Für $\cos(\beta/2) \leq 2\alpha$ nimmt die Funktion $\gamma \mapsto \delta(\beta, \gamma)$ den kleinsten Wert für $\cos(\gamma) = \frac{\cos(\beta/2)}{2\alpha}$ an. Ansonsten hat sie die kleinste Dilation bei $\gamma = 0$.

In diesem Fall lautet also unsere Gleichung $\delta(0, \beta) = \frac{f(0)}{\alpha \sin(\beta/2)}$. Wiederum hat diese Funktion ihre kleinste Dilation bei $\cos(\beta/s) = 2\alpha$, und für diese Konstellation beträgt die Dilation wiederum exakt $2\alpha^{-1}$.

Bleibt also nur noch der Fall, das $\cos(\beta/2) > 2\alpha$. In diesem Fall ist die Dilation von $\delta(\beta, \gamma)$ für $\cos(x) = (a\alpha)^{-1} \cos(\beta/2)$ minimal. Dies lässt sich wiederum weiter zu $\delta(\beta, \gamma) = (2\alpha)^{-1}$ zurückverfolgen.

Das Ergebnis von Knauer und Mulzers Berechnungen ist also, daß die kleinste Dilation die in einer Exclusion Region mit Radius α erreicht werden kann $2\alpha^{-1}$ beträgt. Da die obere Schranke der Delaunaytriangulierung $\delta_{Delaunay} \leq \frac{2\pi}{3\cos(\pi/6)}$ beträgt muss der Radius einer Exclusion-Region mindestens

$$\frac{1}{2\alpha} \geq \delta_{Delaunay}, \text{ also } \alpha \geq \frac{1}{2 * \delta_{Delaunay}} \geq \frac{3\cos(\pi/6)}{4\pi}$$

betragen.

3.2.3 Adaptive Exclusion Region. Praktischer Ansatz

Die Adaptive-Exclusion-Region geht noch einen kleinen Schritt weiter als die Exclusion-Region mit fixem α. Vor allem ist sie theoretisch schlecht fassbar, vor allem was die Laufzeitvorteile betrifft, stellt sie doch eher einen ständige Anpassung der Exclusion-Regions zur Laufzeit dar.

Die Idee ist, daß man nicht mit der fixen oberen Delaunayschranke die Rechnung beginnt sondern wenn möglich schon zu Beginn auf eine bessere zurückgreift. Es lässt sich nämlich beobachten, dass die Delaunay Triangulierung meist wesentlich näher an der eigentlichen *MDT* dran liegt, als es die obere Schranke vermuten lässt. Das Vorgehen während der Berechnung der *MDT* ist folgendes, daß man für den Faktor α die jeweils zur Zeit kleinste bekannte Dilation zu Rate zieht und nicht die obere Schranke der Delaunay Triangulierung. In der Zeit $O(n \log n)$ lässt sich so zu Beginn die Delaunay Triangulierung berechnen, die in den meisten Fällen eine wesentlich bessere Näherung an die *MDT* darstellt als die obere Schranke der Delaunay Triangulierung. Dadurch umfassen die Exclusion-Regions zu Beginn einen größeren Radius und können so eher ungünstige Kanten erkennen. Sollte während des Enumerationsvorganges eine bessere Dilation gefunden werden, so stellt diese die neue Grundlage für den Faktor α dar.

- Berechne die Delaunay Triangulierung und ihre Dilation $\delta_{Delaunay}$
- $\delta_{min} = \delta_{Delaunay}$
- Beginne Enumerationsvorgang
  - Ist $\delta_{aktuell} < \delta_{min}$ so aktualisiere $\delta_{min} = \delta_{aktuell}$ als auch $\alpha = \frac{1}{2*\delta_{aktuell}}$.

Dadurch wächst die Exclusion-Region monoton und der Algorithmus muss wesentlich weniger Triangulierungen überprüfen.

3.3 Globale Eigenschaften

Die bisher betrachteten Eigenschaften der Exclusion-Region betrafen immer das direkte Kantenumfeld und waren dementsprechend lokaler Natur. Global betrachtet läßt sich einer Kante jedoch noch eine weitere Eigenschaft zuweisen: Ihr Hinderniswert.

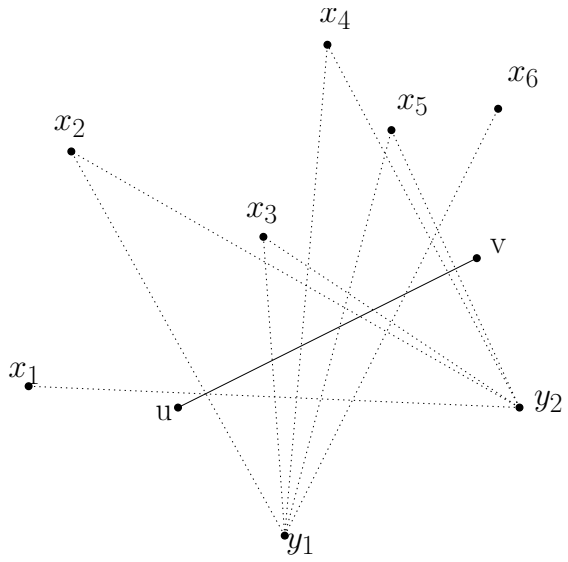


Abbildung 3.11: Beispiel für die Berechnung des Hinderniswertes einer Kante

3.3.1 Hinderniswert von Kanten

Der Hinderniswert einer Kante entspricht der maximalen Dilation die ihr Vorhandensein hervorruft. Durch ihre Existenz blockiert sie vielleicht den direkten Weg zwischen Punkten und erzwingt deshalb einen Umweg, welcher sich in Form des Dilationswertes auch messen läßt. Angenommen eine Kante $k = (u, v)$, mit $u, v \in S$ stellt für die Punkte $x, y \in S\backslash\{u, v\}$ einer Punktmenge S das größte Hindernis dar. So berechnet sich der Hinderniswert der Kante k aus der auftretenden Dilation

$$\delta(k) := \max_{x,y \in S} \delta(x, y) = \max_{x,y \in S} \min\left(\frac{|xu| + |uy|}{|xy|}, \frac{|xv| + |vy|}{|xy|}\right).$$

In Abbildung 3.11 ist ein Beispiel verzeichnet. Dabei werden nur die Punktpaare für die Kante (u, v) überprüft, bei denen (u, v) im direkten Weg(gestrichelt) liegt. Zur Bestimmung des Hinderniswertes einer Kante $e = (u, v)$, mit $u, v \in S$ geht man alle denkbaren Punktpaare $(x, y)S\backslash\{u, v\}$ durch und bestimmt jeweils zwischen diesen Punkten mit der Kante e als eventuelles Hindernis den Hinderniswert. Dabei spielen die anderen möglichen Kante bei der Berechnung keine Rolle und werden herausgenommen. Die einzigen Kanten die bei der Berechnung betrachtet werden sind $\overline{uv}, \overline{ux}, \overline{uy}, \overline{vx}$ und $\overline{vy}$. Der Hinderniswert ist eine wesentlich mächtigere Eigenschaft als die Exclusion-Region, bezieht sie doch auch Punkte mit ein, die nicht in der lokalen Umgebung einer Kante liegen.

In Abbildung 3.12 ist ein Beispiel zu sehen, in welchem Punkte außerhalb der Exclusion-Region eine höhere Dilation verursachen, als die Exclusion-Region erkennen kann, da es nicht nur darauf ankommt, wie nah Punkte in der Nähe eines Kantenmittelpunktes liegen, sondern auch, wie nah sie generell an einer Kante liegen.

Allerdings ist der Hinderniswert auch isoliert von anderen Kanten, da bei seiner Betrachtung der zugrundeliegende Graph nur aus den Punkten und den

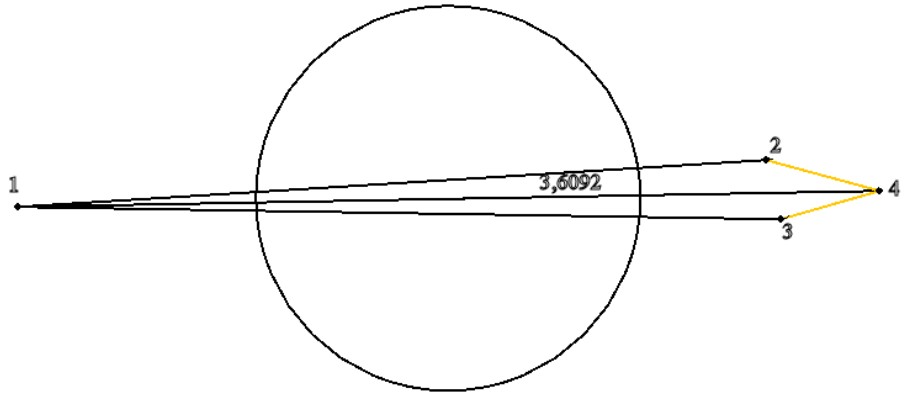


Abbildung 3.12: $\delta(p_1, p_4) = 3.6092$. Die Exclusion-Region erkennt die hohe Dilation zwischen p_2 und p_3 nicht.

oben genannten Kanten besteht (siehe Abbildung 3.13). Ein Dilationweg zwischen Punkten, der über mehrere Kanten verläuft, kann somit nicht erfasst werden. Würde man diese miteinbeziehen, wäre man beim Problem der MDT-Findung angelangt.

Die höhere Mächtigkeit des Hinderniswertes gegenüber der Exclusion-Region hat allerdings auch ihren Preis. Der Berechnungsaufwand ist wesentlich höher als eine Überprüfung der Exclusion-Region, bei der die Überprüfung pro Kante $O(n)$ in Anspruch nimmt als für alle Kanten $O(n^3)$. Gegeben sei eine Punktmenge S mit n Punkten. Dann gibt es insgesamt $O(n^2)$ verschiedene Kanten. Für jede dieser Kanten muß für jedes Punktpaar die Dilation überprüft werden. Innerhalb des Javaprogrammes ist das wie folgt gelöst:

Für jede mögliche Kante:	$n^2/2 = O(n^2)$
Für jedes Punktepaar	$n^2/2 = O(n^2)$
Berechne den Hinderniswert	$O(1)$

Wie man sieht ist dies ein reiner Brute Force Ansatz, welcher jedoch die Komplexität der Berechnung wiederspiegelt. Bei $n = |S|$ Punkten, ergeben sich $\frac{n(n-1)}{2}$ ungerichtete mögliche Kanten und die Laufzeit für die Berechnung aller Hinderniswerte der Punktmenge S und der darüber verlaufenden möglichen Kanten beträgt:

$$O(n^2) * O(n^2) * O(1) = O(n^4).$$

Angewandt wird der Hinderniswert unter anderem während der Bitwise-Enumeration 4.3. Mit seiner Hilfe wird die Kandidatenmenge der zu enumerierenden Kanten drastisch gesenkt und trotz des hohen Berechnungsaufwandes bei der Initialisierung des Hinderniswertes führt es zu einer spürbaren Performanceverbesserung während des eigentlichen Enumerationsvorganges.

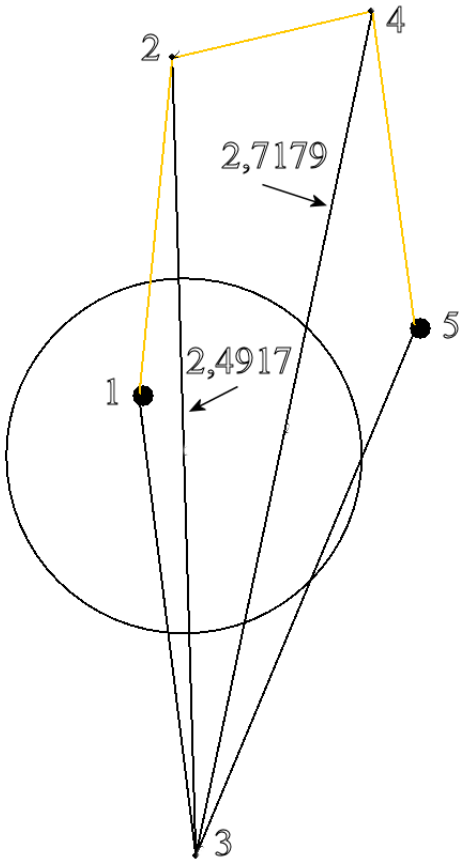


Abbildung 3.13: $\delta(p_1, p_5) = 3.1705....$ Die Exclusion-Region erkennt die hohe Dilation nicht. Auch der Hinderniswert der Kanten $\delta_{Kante(2,3)} = 2,4917$ und $\delta_{Kante(3,4)} = 2,7179$ kann die reale Dilation nur zum Teil erfassen.

Kapitel 4

Berechnungsverfahren

4.1 Einleitung

In diesem Kapitel stelle ich zwei Verfahren vor, welche es jeweils ermöglichen, die *MDT* zu einer Punktmenge S zu ermitteln. Danach gehe ich noch kurz auf eine Greedyimlementierung ein und diskutiere die Problematik, warum ein solcher Ansatz sich als schwierig herausstellt.

Kurzbeschreibung Bespamyatnikh

Beim ersten Algorithmus den ich vorstelle, handelt es sich um einen Enumerationsalgorithmus. Bespamyatnikh[Bes00] erstellte diesen Algorithmus als *Reverse-Search Algorithmus*, welcher es ermöglicht alle Triangulierungen einer Punktmenge S mit $n = |S|$ Punkten aufzuzählen. Unter diesen Triangulierungen ist dann auch die *MDT*. Pro Triangulierung benötigt der Algorithmus $O(\log \log n)$. Auf die Arbeitsweise und Definition eines *Reverse-Search Algorithmus* werde ich vorher kurz eingehen.

Kurzbeschreibung Bitwise Enumeration

Der zweite Algorithmus ist hingegen kein vollständiger Enumerationsalgorithmus mehr. Es werden nicht alle Triangulierungen enumertiert sondern nur eine Menge von Kandidaten für die *MDT*. Der *Bitwise-Enumeration Algorithmus* erstellt ein optimiertes Set von Kanten, welche Kandidaten für die *MDT* darstellen. Über einen Bitstring, bei dem jedes Bit eine Kante des Kandidatensets repräsentiert, werden dann alle *MDT*-Kandidaten enumeriert, so daß auf diesem Weg die *MDT* gefunden wird.

Kurzbeschreibung Greedyalgorithmus

Der letzte Algorithmus ist ein Versuch mittels eines Greedyalgorithmus die *MDT* zu finden. Recht schnell wird jedoch deutlich werden, das sich das ganze als recht kompliziert darstellt. Mittels Beispielen werde ich näher darauf eingehen und einen Ausblick geben, welche Schritte zu einer Lösung führen könnten. Dieser Greedyalgorithmus verdeutlicht noch einmal die Schwierigkeit und Komplexität einer Heuristik.

4.2 Reverse Search Enumeration nach Bespamyatnikh

Dieser Algorithmus dient dazu alle Triangulierungen einer Punktmenge aufzuzählen und basiert auf dem *Reverse-Search Mechanismus* von Avis und Fukuda[AF96]. Im folgendem werde ich zuerst eine kurze Einführung zur Funktionsweise eines *Reverse-Search Algorithmus* geben um dann die Brücke zur Realisierung von Bespamyatnikh zu schlagen. Zu Anfang der Diplomarbeit hatte ich meine Hoffnung auf diesen Algorithmus als wichtiges Hilfsmittel zur Problemlösung gelegt. Warum dem leider nicht so ist, werde ich später erklären (siehe Kapitel 4.2.4). Immerhin enumeriert er zuverlässig und effizient alle möglichen Triangulierungen einer Punktmenge S und benötigt dafür $O(\log \log n)$ pro Triangulierung.

4.2.1 Wie arbeitet ein Reverse Search Algorithmus

Reverse-Search Algorithmen sind eine effiziente Methode um alle Elemente einer Menge zu enumerieren. Dabei handelt es sich um eine graphenbasierte Technik. Dazu müssen die aufzuzählenden Elemente in Relation zu einander stehen, so daß sich daraus ein Graph erstellen lässt. Dieser Graph G ist dann wie folgt definiert:

$$G = (V, E) \text{ mit } V \equiv \text{Knotenmenge und } E \equiv \text{Kantenmenge.}$$

Wenn es um die Enumeration aller Triangulierungen einer Punktmenge S geht, ist V äquivalent zur Menge aller möglichen Triangulierungen und E äquivalent zur Menge aller Edgeflip-Nachbarschaftsbeziehungen. Zwischen zwei Triangulierungen besteht eine Edgeflip-Nachbarschaftsbeziehung, wenn man je eine der beiden Triangulierungen durch einen Edgeflip in die jeweils andere Triangulierung überführen kann. Die Menge

$$E = \{\{X, Y\} | \forall X \neq Y \in V \text{ und } \exists \text{ Kante } e \in X : Edgeflip(X, e) = Y\}$$

entspricht also der Menge aller benachbarten Triangulierungen. Da die Relationen auf einem Edgeflip basieren, ist der dazugehörige Triangulierungsbaum ein ungerichteter Graph. Eine Teilmenge von E stellt die Lösungsmenge L dar. Die Bedeutung der *Lösungsmenge* ist in diesem Zusammenhang allerdings nicht offensichtlich, sondern bedarf in einer näheren Erklärung (siehe auch Abbildung 4.1).

Die eigentliche Zielsetzung eines *Reverse-Search Algorithmus* ist die Enumeration von Elementen. Wenn ich im folgenden von einer Lösung, Lösungsmenge oder auch Solution spreche, ist damit jedoch etwas anderes gemeint. Die zentrale Idee des *Reverse-Search Verfahrens* ist es, sich Elemente zu suchen, in welche jeder Knoten aus V umgeformt werden kann. Diese Elemente werden als Lösungselemente oder zusammenfassend Lösungsmenge L, mit $L \subset V$ bezeichnet. Der Hinweg ist die Umformung jedes Knotens aus $V \setminus L$ zu den Lösungselementen.

Geht man aber nun den umgekehrten Weg der Umformung beginnend bei der Lösungsmenge, so gelangt man wieder auf dem Rückweg zu allen Knoten $S \setminus L$. Daher auch die Bezeichnung *Reverse Search*. Man geht also von der Lösungsmenge den Weg der Umformung zurück, da nicht diese Lösungselemente

das eigentliche Ziel sind, sondern die Knoten $S \setminus L$. Führt man das für jedes Element $l \in L$ durch, so enumeriert man sämtliche Elemente von $V \setminus L$. Die Enumeration aller Elemente reduziert sich also auf das Finden einer passenden Lösungsmenge sowie der dazugehörigen Umformung. Wenn in Zukunft von einer Lösungsmenge die Rede ist, sind nicht alle Elemente von V gemeint, deren Enumeration das eigentliche Ziel darstellt, sondern so sind damit die Ausgangselemente des *Reverse-Search Algorithmus* gemeint. In Abbildung 4.1 ist ein Beispiel zu sehen. Die einzelnen Punkte repräsentieren die Triangulierungen einer Punktmenge und sind untereinander verbunden, falls zwischen ihnen eine Edgeflipbeziehung besteht. Die schwarzen Punkte stellen die Elemente der Lösungmenge dar und die dicker gezeichneten Kanten verweisen auf den Weg der Umformungsfunktion von den Triangulierungen zu den Lösungselementen. Wie man in diesem Beispiel sieht, wird durch eine sinnvoll gewählte Umformungsfunktion die Menge der Triangulierungen wie ein Baum aufgespannt. So besteht immer ein eindeutiger Weg von einer Triangulierung zu dem dazugehörigen Lösungselement und der aufgespannte Umformungsgraph jedes Lösungselementes ist zusammenhängend, zyklenfrei und eindeutig.

Nun aber zurück zur allgemeinen Beschreibung eines Reverse-Search Algorithmus. Wir haben nun einen ungerichteten Graph bestehend aus vielen Elementen unter denen sich auch die Lösungselemente befinden. Mittels der allgemein definierten *lokalen Suchfunktion f (locale search function)* gelangt man von einer Triangulierung Schritt für Schritt zum dazugehörigen Lösungselement.

$$f(v) = w \text{ mit } v \in V \setminus L, w \in V \setminus \{v\}$$

Die *lokale Suchfunktion* liefert zu einem Element der Knotenmenge V einen Nachfolger.Gilt zusätzlich folgendes Kriterium, so nennt man f eine *endliche lokale Suchfunktion (finite local search function)*:

$$\forall v \in V \setminus L : \exists k \geq 0, \text{ so das } f^k(v) \in L$$

Eine lokale Suchfunktion f ist also dann endlich, wenn man für jedes Element der Ausgangsmenge nach endlicher Anwendung von f zu einem Element der Lösungsmenge gelangt. Kennt man die Funktion f so kann man von jedem beliebigen Element $v \in V$ zur Lösung gelangen. Das Tripel (G, L, f) nennt sich dann *local search* oder *finite local search*.

procedure LocalSearch(G,S,f, v_0: vertex of G)
v := v_0
while $v \notin L$ do
v := f(v)
endwhile
output v

Für jedes Element der Vertexmenge induziert die lokale Suchfunktion einen Pfad zu einem Element der Lösung. Daraus ergibt sich für jedes Element der Lösungsmenge eine Zusammenhangskomponente, deren Struktur einem Baum entspricht. In jeder Komponente gibt es genau ein Lösungselement, welches die Wurzel dieser Zusammenhamgskomponente darstellt. Eine solche Zusammenhangskomponente nennt sich Trace. Abbildung 4.1 zeigt ein Beispiel einer

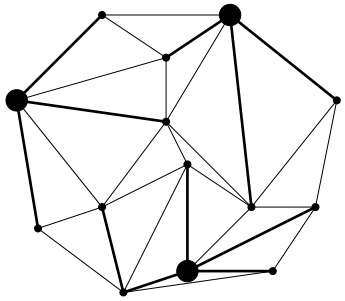


Abbildung 4.1: Beispiel für eine Reverse-Search-Situation. Die Vertices aus V ergeben mit den Kantenbeziehungen E einen ungerichteten Graphen. Die Elemente der Lösungsmenge sind durch größere Punkte hervorgehoben. Es gibt 3 Traces, welche als Wurzel je ein Element der Lösungsmenge L enthalten. Die Baumstruktur der Traces entsteht dadurch, dass jedem Knoten genau ein Vorgänger zugeordnet ist, welcher näher zu einem Element der Lösungsmenge steht.

Vertexmenge V, deren Lösungselemente $l \in L$ und den zugehörigen durch f induzierten Traces.

An dieser Stelle setzt der Grundgedanke eines *Reverse-Search Algorithmus* ein. Für jedes Element der Lösungsmenge existiert ein Trace, und wenn man von jedem Element der Lösungsmenge den dazugehörigen Trace traversiert, dann enumeriert man auch jedes Element von $V \setminus L$.

Im Falle dieser Diplomarbeit wird die Lösungsmenge nur aus einem Element bestehen und somit gibt es auch nur einen Trace, welcher traversiert werden muss. Die Traversierung läuft über die Umkehrfunktion einer nach bestimmten Kriterien definierten Kantenflip-Operation. In Abbildung 4.2 ist ein Beispiel dieser Situation zu sehen.

4.2.2 Umsetzung von Bespamyatnikh

Die nun folgenden Ausführungen fassen die Arbeit von Bespamyatnikh [Bes00] zusammen, welcher einen *Reverse-Search Algorithmus* implementiert um alle Triangulierungen einer Punktmenge aufzuzählen. Dazu muss das Tripel (G, L, f) der endlichen lokalen Suche für eine Punktmenge S genauer spezifiziert werden. Der Triangulierungsgraph G setzt sich aus der Vertexmenge V aller möglicher Triangulierungen für S und der Kantenmenge E zusammen. Eine Kante zwischen zwei Triangulierungen existiert dann, wenn man eine Triangulierung mittels eines Edgeflips in die andere überführen kann. Dann fehlt nur noch die Spezifikation der Suchfunktion f und einer geeigneten Lösungsmenge L. Am einfachsten ist eine einelementige Lösungsmenge, denn dann existiert auch nur ein einziger Trace, der traversiert werden muss.

Es ist also nach einer Triangulierung gesucht, welche sich durch Edgeflip-Operationen von jeder anderen Triangulierung erreichen lässt. Fortune [For87] zeigte, daß es möglich ist von jeder Triangulierung in $O(n^2)$ Delaunay-Edgeflips zur Delaunaytriangulierung zu gelangen. Auf einen *Reverse-Search Algorithmus* übertragen heißt das, dass die Delaunaytriangulierung die Lösungsmenge darstellt und die Delauny-Edgeflips die lokale Suchfunktion $f(x)$ darstellen. Um

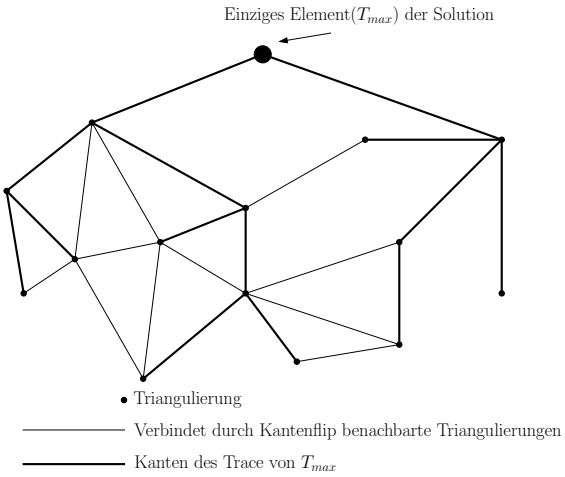


Abbildung 4.2: Beispiel für einen Trace im Falle von Bespamyatnikh mit der Triangulierung Tmax als Lösung.

alle Triangulierungen zu enumerieren würde man beim *Reverse-Search* bei der Delaunaytriangulierung starten und den umgekehrten Weg gehen. Ursprünglich sind Delaunay-Edgeflips derart, das durch den Edgeflip spitze Winkel eliminiert werden. Geht man also den umgekehrten Weg, so werden mit den Edgeflips spitze Winkel *erzeugt*. Genaueres ist bei Fortune [For87] nach zulesen.

Bespamyatnikh[Bes00] wählt nicht die Delaunaytriangulierung als einziges Lösungselement beziehungsweise Ausgangspunkt des Reverse-Search, sondern stattdessen die lexikographisch grösste Triangulierung. Dazu werden die Punkte der Punktmenge $S = \{p_1, p_2, .., p_n\}$ mit einer Ordnung versehen. Anhand dieser Ordnung lassen sich nun die Punkte, Kanten und Triangulierungen jeweils in eine lexikographische Reihenfolge bringen.

Aus Gründer der Übersichtlichkeit werden die Kanten und Sequenzen im folgenden als geordnete Tupel dargestellt. Obwohl es sich dabei immer noch um ungerichtete Kanten handelt, lassen sich so die einzelnen Kanten besser vergleichen. Eine Sequenz von Punkten $(p_{i_1}, p_{i_2}, ..., p_{i_k}, p_{i_{k+1}})$ ist kleiner als eine andere Sequenz $(p_{j_1}, p_{j_2}, ..., p_{j_k}, p_{j_{k+1}})$ wenn $i_1 = j_1, i_2 = j_2, ..., i_k = j_k$ und $i_{k+1} < j_{k+1}, k \geq 0$. So ist auch eine Kante $e(p_i, p_j), i < j$ kleiner als eine andere Kante $e(p_k, p_l), k < l$ wenn die Sequenz (p_i, p_j) lexikographisch kleiner ist als die Sequenz (p_k, p_l). Auf diesem Weg lassen sich dann auch Triangulierungen als Kantenmenngen vergleichen und in eine lexikographische Reihenfolge setzen. Eine Kantenmenge $(e_{i_1}, e_{i_2}, ..., e_{i_k}, e_{i_{k+1}})$ ist dann kleiner als eine Kantenmenge $(e_{j_1}, e_{j_2}, ..., e_{j_k}, e_{j_{k+1}})$ wenn $e_{i_1} = e_{j_1}, e_{i_2} = e_{j_2}, ..., e_{i_k} = e_{j_k}$ und $e_{i_{k+1}} < e_{j_{k+1}}$. Eine Triangulierung wird dabei durch ihr sortierte Kantensequenz dargestellt.

Mittels $F(T)$ wird im folgenden die Menge der flipbaren Kanten einer Triangulierung T bezeichnet. Dann sind $F_+(T), F_-(T)$ die Menge der flipbaren Kanten welche die lexikographische Ordnung von T vergrößern beziehungsweise verkleinern. T_{max} ist die Triangulierung mit maximaler lexikographischer Ordnung, für diesen *Reverse-Search Algorithmus* stellt sie die Lösungsmenge da.

Das erste Problem, welches sich stellt, ist in folgender Beobachtung zusammengefaßt:

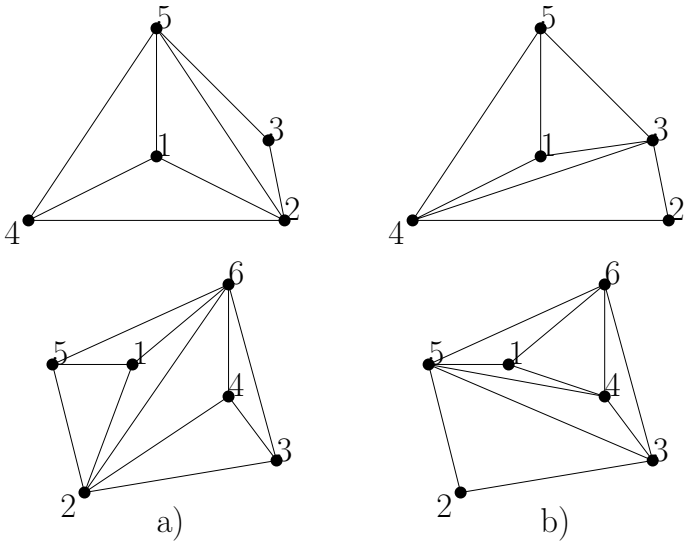


Abbildung 4.3: a) Triangulierungen mit $F_+(T) = \emptyset$, b) dazugehörige T_{max}

Beobachtung(Deadlock):
Für jede Punktmenge S mit mehr als 5 Punkten lässt sich eine Konstellation von Punkten, Kanten und dazugehöriger Triangulierung $T \neq T_{max}$ finden, so daß $F_+(T) = \emptyset$.

Ein paar Beispiele sind in Abbildung 4.3 zu sehen.

Im folgenden werde ich zeigen, wie solche Deadlocks vermieden werden können, indem die Ordnung der Punkte ein bestimmtes Kriterium erfüllt. Bei einer Punktmenge S mit n Punkten bezeichne $S(i), 1 \leq i \leq n$ die Punktmenge $\{p_i, p_{i+1}, ..., p_n\}$, also die Punkte i bis n. Die Idee ist, daß Deadlocks vermieden werden können, wenn für jedes $1 \leq i \leq n$ der Punkt p_i auf der konvexen Hülle von $S(i)$ liegt. Dies ist zum Beispiel der Fall, wenn man die Punkte anhand ihrer X-Koordinaten aufsteigend sortiert und benennt.

In Abbildung 4.4 ist ein Beispiel von 4 Punkten und den zugehörigen Teilmengen $S(i)$ aufgeführt. Die einzelnen Untermengen $S(i)$ induzieren für T_{max} ebenso die Teilgraphen $T_{max}(i)$.

Das folgende Lemma charakterisiert die Struktur der Triangulierung T_{max} und seiner Teilgraphen $T_{max}(i)$ mit deren Hilfe ich zeigen werde, daß es für jede Triangulierung einen Edgeflip gibt, welcher die lexikographische Ordnung vergrößert.

Lemma 3 *Der Graph $T_{max}(i), i \in \{1, 2, ..., n\}$ ist eine Triangulierung von $S(i)$.*

Beweis:
Den folgenden Beweis werde ich per Induktion führen, wobei die Besonderheit darin besteht, daß ich den Schritt von $S(i)$ zu $S(i+1)$ mache. Trotz zunehmendem i verkleinert sich dabei $S(i)$. Bei jedem Schritt werden der Punkt p_i sowie alle Kanten, die von p_i ausgehen, gelöscht. Aber nun zum Induktionsanfang:

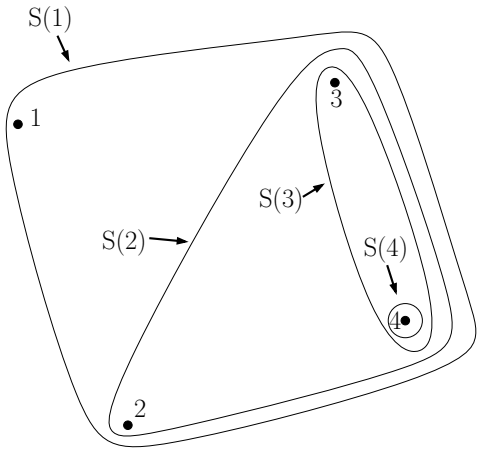


Abbildung 4.4: Beispiel für die Sortierung der Punkte zur Vermeidung von Deadlocks

Eindeutig ist $T_{max}(1) = T_{max}$ eine Triangulierung von $S(1) = S$. Die Frage ist nur wie es sich mit dem nächsten Schritt verhält.

Sei für $1 < i < n$ $T_{max}(i)$ eine Triangulierung von $S(i)$. Beim Schritt von i zu $i+1$ mit $S(i+1) = S(i) \setminus p_i$ verkleinert sich auch $T_{max}(i)$ zu $T_{max}(i+1)$ um die Kanten $\{(p_i, p_{i_1}), ..., (p_i, p_{i_k})\}$. Dabei bezeichnen die Indizes i_1 bis i_k die Punkte, mit denen p_i in $T_{max}(i)$ verbunden war.

In Abbildung 4.5 sind das alle Kanten zwischen p_i und $p_{i_1}, p_{i_2}, p_{i_3}, p_{i_4}$. Diese Kanten gehörten zu T_{max}. Die Frage ist aber, wie es sich mit der Kantenkette (p_{i_1}, p_{i_2}) bis $(p_{i_{k-1}}, p_{i_k})$ verhält, die nun in $S(i+1)$ zur konvexen Hülle $ch(S(i+1))$ gehört. In Abbildung 4.5 sind das die Kanten zwischen p_{i_1} bis p_{i_4}. Sollte auch diese zu T_{max} gehören, so stellt auch $T_{max}(i+1)$ eine Triangulierung da und das Lemma wäre bewiesen. Ansonsten hat sich beim Schritt von $T_{max}(i)$ zu $T_{max}(i+1)$ nichts verändert.

Angenommen es wäre nicht so, zum Beispiel für $j = 2$ wäre (p_{i_2}, p_{i_3}) nicht in T_{max} enthalten. Dann würden von p_j aus im Winkelbereich zwischen p_j und p_{j+1} noch weitere Punkte liegen, mit denen p_i verbunden gewesen wäre. Sei p_m der Punkt, welcher am weitesten von p_i entfernt auf dem Pfad von p_j nach p_{j+1} liegt. Die Kante zwischen p_i und p_m wäre damit flipbar und würde die lexikographische Ordnung der Triangulierung vergrößern, da p_i ja der Punkt mit der kleinsten Ordnung in $S(i)$ ist und deshalb die duale Kante von (p_i, p_m) eine lexikographische höhere Ordnung haben muß. Dies widerspricht jedoch der nicht vergrößerbaren Ordnung von T_{max} und somit wäre das Lemma bewiesen.

□

Diese wichtige Eigenschaft wird beim Beweis des nächsten Theorems eine zentrale Rolle spielen.

Theorem 4 *Für jede Triangulierung $t \neq T_{max}$ ist $F_+(t) \neq \emptyset$.*

Für jede Triangulierung t ungleich T_{max} gibt es also einen Flip, welcher die lexikographische Ordnung von t vergrößert. Deadlocks sind damit ausgeschlossen.

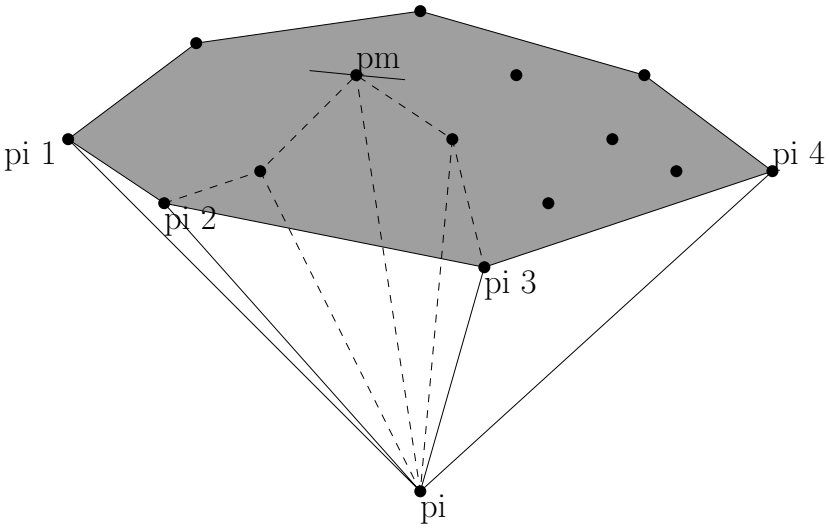


Abbildung 4.5: Graph $T_{max}(i)$ und der Graph $T_{max}(i+1)$(dunkel)

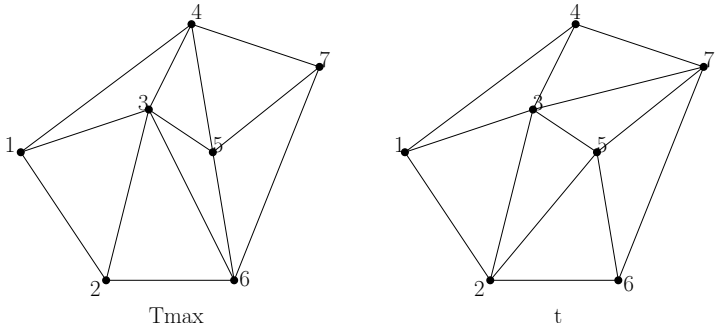


Abbildung 4.6: Punktmenge mit 7 Punkte, dem dazugehörigen T_{max} und einer Beispieltriangulierung t

Beweis:
Gegeben sei eine Punktmenge, die lexikographisch größte Triangulierung T_{max} von S und eine weitere Triangulierung $t \neq T_{max}$ von S. Ein Beispiel ist in Abbildung 4.6 zu sehen. Die dazugehörige Kantenmenge $t \backslash T_{max}$ ist in Abbildung 4.7 a) zu sehen.
Die Triangulierung T_{max} läßt sich auch wie folgt zerlegen:

$$T_{max} = (T_{max} \setminus T_{max}(1)) \cup (T_{max} \setminus T_{max}(2)) \cup ... \cup (T_{max} \setminus T_{max}(n)).$$

Dabei sind die einzelnen Mengen jeweils Teilmengen der nächst größeren:

$$(T_{max} \setminus T_{max}(1)) \subset (T_{max} \setminus T_{max}(2)) \subset ... \subset (T_{max} \setminus T_{max}(n)).$$

Eine Kante $e = (p_i, p_j) \in T_{max}, 1 < i < j \leq n$ taucht das erste mal in der Menge $(T_{max} \setminus T_{max}(i+1))$ auf. Das heißt, das sie in den Mengen $(T_{max} \setminus T_{max}(1)), ..., (T_{max} \setminus T_{max}(i))$ nicht vorkommt.

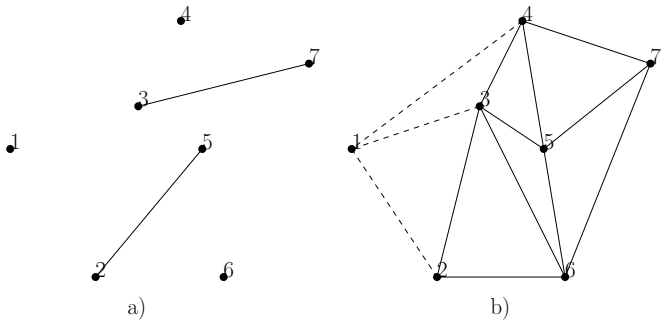


Abbildung 4.7: a) $t \backslash T_{max}$ b) $T_{max}(2)$, fehlende Kanten zu T_{max}(gestrichelt).

Sei $e = (p_i, p_j), i < j$ die kleinste lexikographische Kante mit $e \in t \setminus T_{max}$. In diesem Beispiel ist $e = (p_2, p_5)$. Dann muß t alle Kanten von $T_{max} \setminus T_{max}(i)$ enthalten. Im Beispiel wären das die Kanten von $T_{max} \setminus T_{max}(2)$ (siehe dazu auch $T_{max}(2)$ in Abbildung 4.7 b)).

Analog zum Beweis von Lemma 3 liegt mit e eine Kante vor, welche den Punkt p_i zum gedachten Punkt p_m verbindet (siehe Abbildung 4.5). Dem ist so, da e nicht in T_{max} enthalten ist, aber von $p_{i=2}$ aus in $T_{max}(i+1)$ hineinragt und es auch hier wieder eine Kantenkette auf der konvexen Hülle $ch(T_{max}(i+1))$ gibt, deren Weg die Kante e kreuzt. Daher läßt sich auch hier wieder aus den gleichen Gründen argumentieren, daß die Kante e flipbar sein muß und dieser Flip die aktuelle Ordnung vergrößert. Im Beispiel in Abbildung4.6 würde sich die Kante $e = (2, 5)$ mit der Kante $(3, 6) \in T_{max}$ schneiden. Also ist $F_+(t) \neq \emptyset$ solange $t \neq T_{max}$ und das Theorem damit bewiesen.

□

Wie die Deadlocks umschifft werden können, ist also geklärt. Was fehlt ist die Suchfunktion f, mit deren Hilfe wir den Trace für unser Problem erstellen können.

Dazu gehe ich erstmal auf den genauen Aufbau eines Triangulierungsbaumes $T = (V, E)$ ein. Ein Triangulierungsbaum ist ein Graph mit Baumstruktur, dessen Knoten einzelne Triangulierungen repräsentieren (Beispiel siehe Abbildung 4.8). Wenn ich in Zukunft von einer einzelnen Triangulierung, also einem Element des Triangulierungsbaumes T sprechen werde, so werde ich diese mit t oder $T(v)$ also als Knoten $v \in V$ mit $V = \{$ Menge aller Triangulierungen der Punktmenge $S\}$ bezeichnen. Die Triangulierung T_{max} bildet die Wurzel von T. Für jeden Knoten $v \in V$ sind dessen Kinder durch Edgeflips so bestimmt, daß der umgekehrte Flip die lexikographisch größte Trangulierung des Kindes, nämlich v ergibt. Wegen Theorem 4 sind alle Triangulierungen in T enthalten und T ist ein Baum. Weiterhin bezeichne $f_{max}(t)$ die Kante einer Triangulierung t, deren Edgeflip die lexikographisch größte Nachfolgetriangulierung produziert. In Abbildung 4.8 ist ein Triangulierungsbaum für eine Menge von 4 Punkten abgebildet.

Nun sind nur noch zwei Dinge zu klären. Wie lassen sich einerseits die Kante $f_{max}(t)$ ermitteln, und zweitens zu einer Triangulierung t alle direkten Kinder

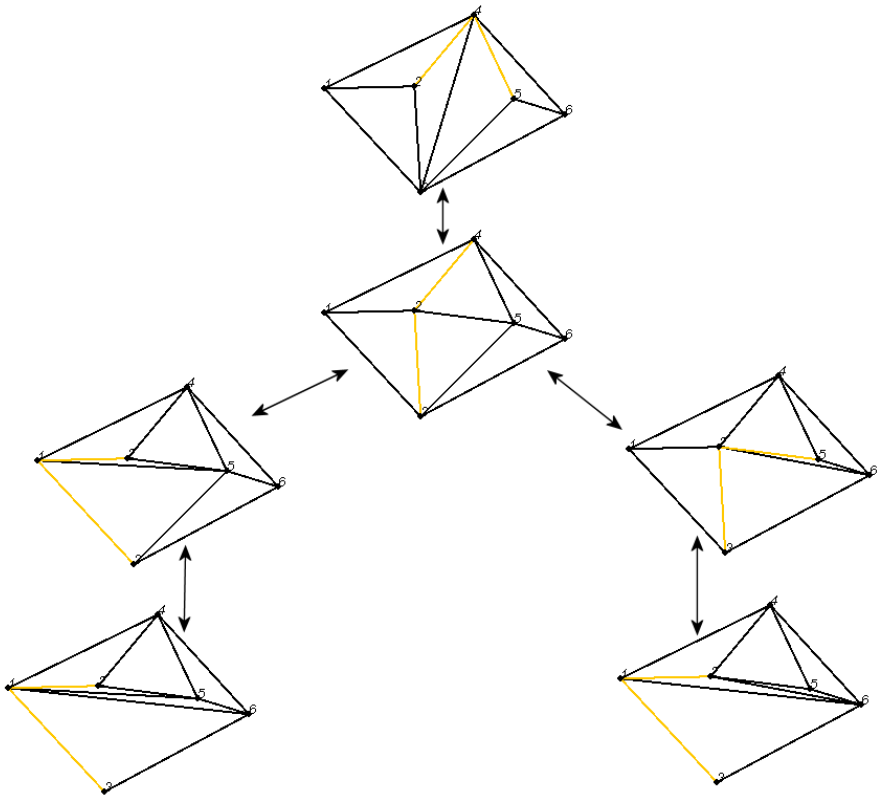


Abbildung 4.8: Ein Triangulierungsbaum für 6 Triangulierungen mit T_{max} als Wurzel des Baumes. Die helleren Kanten geben jeweils den Umweg zwischen den dilationsbestimmenden Punkten an.

berechnen.

Lemma 4 ***Elternkante***
Für eine Triangulierung $t \neq T_{max}$ ist die Kante $f_{max}(t)$ die lexikographisch kleinste flipbare Kante aus $t \backslash T_{max}$.

Beweis:
Erstens wissen wir laut Theorem 4, daß für jede Triangulierung $t \neq T_{max}$ die Menge $F_+(t)$ mindestens eine flipbare Kante enthält. Sei nun $e = (p_a, p_b), a < b$ die lexikographisch kleinste Kante aus $t \setminus T_{max}$. Sei nun $e_1 = (p_i, p_j), i < j$ eine beliebige andere flipbare Kante aus t.

Wenn der Index $i < a$ ist, muß e_1 ein Punkt auf der konvexen Hülle $ch(S(i))$ sein und ein Edgeflip von e_1 würde die lexikographische Ordnung verringern. Sollte $i > a$ sein so gibt es zwei Möglichkeiten. Entweder läßt ein Edgeflip von e_1 die Umgebung zu p_a unangetastet oder ein Edgeflip resultiert in einer Kante, die in p_a endet oder anfängt (Die Trianguliergen sind ungerichtet). In beiden Fällen würde ein Edgeflip von e in einer Triangulierung mit größerer Ordnung münden als ein Edgeflip von e_1. Sollte $i = a$ sein, so würde wiederum ein Edgeflip von e eine Triangulierung größerer Ordnung erstellen, da e lexikographisch kleiner ist als e_1.

□

Was wir noch benötigen, ist ein Kriterium wie sich zu einer Triangulierung t alle direkten Kinder berechnen lassen.

Lemma 5 ***direkte Kindkante***
Sei v ein Knoten des Triangulierungsbaumes T und sei $(p_a, p_b), a < b$ die Kante $f_{max}(T(v))$ die zum Vater von $T(v)$ führt. Eine flipbare Kante e, deren duale Kante $d(e) = (p_c, p_d), c < d$ sei, erstellt ein direktes Kind, falls:

- *1) $c < a$ oder*
- *2) $a = c$ und $d < b$ oder*
- *3) $a = c, d > b$ und (p_a, p_b) ist nicht flipbar in $flip(T(v), e)$ und wenn (existierend) die lexikographisch zweit kleinste flipbare Kante in $T(v) \backslash T_{max}$ lexikographisch größer ist als (p_c, p_d).*

Beweis:
In Abbildung 4.9) ist eine erläuternde Skizze aufgeführt. Die duale Kante $d(e) = (p_c, p_d)$ ist die Kante mit deren Flip man vom Kind $flip(T(v), e)$ zur aktuellen Triangulierung $T(v)$ gelangt. Diese Kante $d(e)$ ist also gleich mit $f_{max}(flip(T(v), e))$. Und mittels der drei Kriterien läßt sich überprüfen, ob die duale Kante $d(e)$ im Kind wirklich alle Voraussetzungen erfüllt um die f_{max} Kante zu sein. Siehe dazu auch Abbildung 4.9. Laut dem vorigen Lemma zeichnet sich eine f_{max}-Kante dadurch aus, daß sie die lexikographisch kleinste flipbare Kante aus dem Kantenset $T \backslash T_{max}$ ist. In dieser Situation führt das zu folgenden Schlussfolgerungen.

Die Kante (p_a, p_b) welche die f_{max}-Kante der aktuellen Triangulierung darstellt ist auch im Kind vorhanden. Da allerdings angenommen wird, das eine

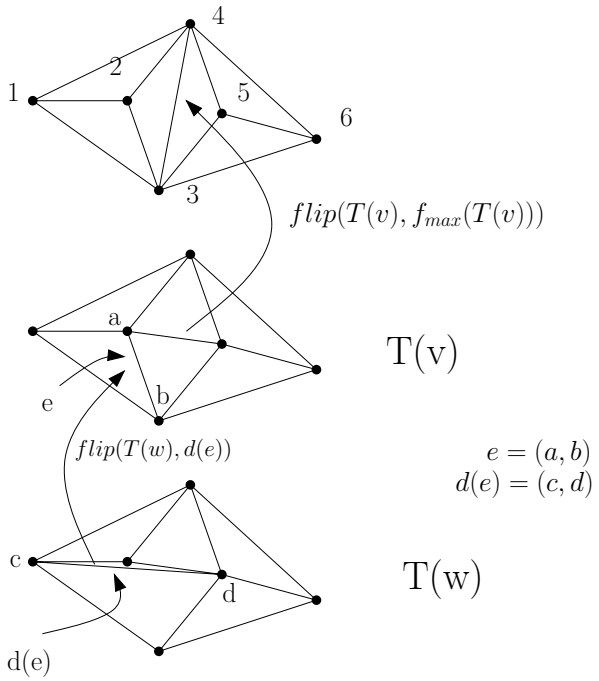


Abbildung 4.9: Erklärende Skizze zum Lemma: direkte Kindkante

andere Kante, nämlich die Kante $d(e) = (p_c, p_d)$ im Kind die f_{max}-Kante ist, so muß entweder $d(e)$ im Kind die lexikographisch kleinste Kante sein. In diesem Fall muß sie kleiner sein, als die Kante (p_a, p_b), welches mit den Kriterien 1) und 2) überprüft wird. Oder aber die Kante (p_a, p_b) ist deshalb im Kind nicht die f_{max}-Kante, da sie dort nicht flipbar ist. Erst ein Flip der Kante $d(e)$ könnte zum Beispiel die Umgebung so verändern, das (p_a, p_b) flipbar wird. Wenn es in diesem Fall im Kind noch weitere flipbare Kanten gibt, so muß die zweit kleinste Kante nicht nur im Kind existieren, sondern auch in der aktuellen Triangulierung, der vermuteten Vatertriangulierung. Auch hier muß sie die zweitkleinste flipbare Kante sein, da $d(e)$ ausscheidet und dafür (p_a, p_b) an erster Stelle als $f_{max} - Kante$ nachgerückt ist. Und dieser letzte Fall wird mittels des Kriteriums 3) überprüft.

□

Und damit wäre sowohl dieses Lemma als auch die Machbarkeit eines Reverse-Search-Algorithmus über diese spezielle Ordnung von Punkten bewiesen. Im folgenden gehe ich näher auf den Algorithmus sowie dessen Laufzeit ein.

4.2.3 Laufzeit

Den Enumerationsalgorithmus, welcher von Bespamyatnikh in seinem Paper[Bes00] vorgestellt wird, werde ich im folgenden genauer unter die Lupe nehmen, und die Laufzeit erläutern. Der Algorithmus enumertiert einen Triangulierungsbaum T zu einer Punktmenge $S, n = |S|$ beginnend bei der Wurzel T_{max} nach dem *depth-*

first-search-Schema. Eine einzelne Triangulierung aus dem Triangulierungsbaum T sei mit $T(v), v$ ist Triangulierung von S bezeichnet.

Die Datenstruktur, in welcher der Algorithmus zur Laufzeit eine Triangulierung speichert, sieht wie folgt aus. Zu den einzelnen Elementen der Triangulierung werden zusätzliche Verweise abgespeichert, so daß auf die Kanten / Dreiecke benachbart zu anderen Kanten / Dreiecken und auf benachbarte Punkte in konstanter Zeit zugegriffen werden kann. Weiterhin wird davon ausgegangen, daß Kanten, benachbart zu einem Punkt, in einer doppelt verketteten Liste vorliegen, so daß in konstanter Zeit auf alle Kanten in und gegen dem Uhrzeigersinn um diesen Punkt zugegriffen werden kann. Während der Enumeration behält der Algorithmus immer nur eine Triangulierung im Speicher vor. Zusätzlich zu dieser Triangulierung $T(v)$ existieren noch weitere Datenstrukturen.

T_1 = ausbalancierter Suchbaum mit allen flipbaren Kanten von $T(v)$
T_2 = statisch, ausbalancierter Suchbaum mit den Kanten von T_{max}
T_3 = ausbalancierter Suchbaum mit den flipbaren Kanten von $T(v) \backslash T_{max}$
T_4 = ausbalancierter Suchbaum mit den dualen Kanten von T_3

Der Zugriff auf alle Datenstrukturen T_1, T_2, T_3, T_4 nimmt infolge der Baumstruktur $O(\log n)$ Zeit in Anspruch. Außerdem sind Verweise von den Kanten aus T_3 auf ihre dualen Kanten in T_4 und umgekehrt gesetzt. Die Datenstruktur T_2 erlaubt es in $O(\log n)$ feststellen zu können, ob eine Kante in T_{max} enthalten ist oder nicht. Mittels T_1 und T_2 läßt sich in $O(\log n)$ eine Triangulierung $flip(T(v), e)$ zu einer Triangulierung $T(v)$ durch den Edgeflip einer Kante $e \in T(v)$ erstellen. Zur Erinnerung: Während der Laufzeit wird immer nur die aktuelle Triangulierng $T(v)$ im Speicher behalten. Wechselt man zum Beispiel zu einem Kind von $T(v)$, so wird die interne Triangulierung per Edgeflip in das Kind geändert.

Bevor der Enumerationsvorgang allerdings starten kann, müssen die Datenstrukturen initialisiert werden.

Schritt	Beschreibung	Laufzeit
	Initialisierungsvorgang	
I-1	Punkte nach den X-Koordinaten sortieren	$O(n \log n)$
I-2	Den Punkten eine Ordnung zuweisen	$O(n)$
I-3	Erstelle T_{max} und speichere in T_2 und als $T(v)$ ab	$O(n \log n)$
I-4	Erstellung von T_1	$O(n)$
I-5	Erstellung von T_3	$O(n \log n)$
I-6	Erstellung von T_4	$O(n)$
	Berechnungsvorgang pro Triangulierung	
B-1	Überprüfe, ob $T(v)$ das erste Mal besucht wird	$O(1)$
B-2	Wenn ja, dann erste Kante aus T_4 ist nächste Testkante	$O(1)$
B-3	Wenn nein, nächste Kante aus T_4 ist nächste Testkante	$O(1)$
B-4	Erfüllt die Testkante Lemma 5?	$O(1)$
B-5	Wenn ja, enumeriere $Flip(T(v), Testkante)$ als Kind	$O(1)$
B-6	Wenn nein, fahre mit B-3 fort	$O(1)$

In Schritt $I-2$ wird mittels eines vertikalen Sweeps von links nach rechts den Punkten eine aufsteigende Ordnung zugewiesen. Die Triangulierung T_{max}

wird mittels eines vertikalen Sweeps von rechts nach links durchgeführt (siehe auch [Bes00]. Da für alle Kanten Verweise auf die benachbarten Dreiecke abgespeichert sind, läßt sich so für jede Kante in konstanter Zeit testen, ob sie flipbar ist oder nicht und die Erstellung von T_1 benötigt $O(n)$ Zeit. Da in $O(\log n)$ überprüft werden kann, ob eine Kante in T_{max} enthalten ist, läßt sich T_3 mit $O(n \log n)$ Zeitaufwand erstellen. Für die $O(n)$ Kanten in T_3 läßt sich in konstanter Zeit die duale Kanten zu T_3 in T_4 erstellen. Der gesammte Initialisierungsvorgang nimmt dann $3O(n \log n) + 3O(n) = O(n \log n)$ Zeit in Anspruch.

Ein Berechnungsvorgang pro Triangulierung sieht dagegen wie folgt aus: Als erstes wird geprüft, ob dieser Triangulierung $T(v)$ des Triangulierungsbaumes T das erste Mal besucht wird. Dies ist der Fall, wenn die vorher besuchte Triangulierung der Vater der jetzigen Triangulierung ist. Ansonsten war die vorherige Triangulierung ein Kind der aktuellen Triangulierung. Sollte die aktuelle Triangulierung zum ersten Mal enumeriert werden, so wird für die erste Kante in T_4 geprüft, ob sie Bestandteil einer gültigen Nachfolgetriangulierung ist. Wenn die aktuelle Triangulierung hingegen schon einmal besucht worden war, so testet man die nächste Kante aus T_4 als beim letzten Besuch. Wenn man das nächste Kind enumeriert wird, müssen insgesamt $O(1)$ Updates in der Triangulierung $T(v)$ der internen Datenstruktur,T_1, T_3 und T_4 durchgeführt werden. Schließlich ändert sich nur eine Kante und eventuell deren Umgebung, da nun angrenzende Kanten flipbar oder nicht mehr flipbar geworden sein können. Da ein Update dieser Datenstrukturen je $O(\log n)$ Zeit in Anspruch nimmt, kostet die Enumeration jeder Triangulierung $O(\log n)$.

P. van Emde Boas stellt in seinem Paper eine Datenstruktur vor, die es ermöglicht, die Zeit des Updatevorganges auf $O(\log \log n)$ zu reduzieren. Die Datenstruktur ermöglicht es eine Menge von Schlüssel $[1...U]$ so abzuspeichern, das Einfüge- und Löschoperationen in $O(\log \log U)$ Zeit durchgeführt werden können. Zusätzlich sind die Schlüssel in einer sortierten Liste abgespeichert, welche es ermöglicht Vorgänger und Nachfolgerelemente in $O(1)$ zu finden. Wenn man jeden der Datenstrukturen T_1, T_2, T_3, T_4 in diesem Format ablegt und $U = \binom{n}{2}$ setzt, so reduziert sich der Enumertionsaufwand pro Triangulierung auf $O(\log \log n)$.

4.2.4 Problematik

Der Algorithmus sieht vielversprechend aus und enumeriert auch zuverlässig alle Triangulierungen und findet damit auch die *MDT*. Die Frage ist allerdings, wie einfach er sich auf die Belange dieser Diplomarbeit zuschneiden lässt, denn im Grunde interessiert hier nur die *MDT*. Und hier kommen die Eigenschaften der *MDT* ins Spiel, auf welche ich in Kapitel 3.2 näher eingegangen bin. Alle diese Eigenschaften sind Kantenbezogen und der Enumerationsalgorithmus von Bespamyatnikh traversiert einen Triangulierungsbaum. Der Enumerationsvorgang läßt sich dann optimieren, wenn sich Teile des Triangulierungsbaumes abschneiden lassen und somit Traversierungsschritte gespart werden können. Dazu müssen alle Elemente des abgeschnittenen Teilbaumes ein Kriterium verletzen oder erfüllen, welches den Optimierungsvorgang zuläßt und daher müssen Eigenschaften, welche bei einer Triangulierung festgestellt werden auch auf alle Kinder übertragtbar sein, damit ein verletztes oder erfülltes Kriterium auch bei ihnen den gleichen Zustand hat. Da es sich bei den bisherigen Eigenschaften vor allem um Kanteneigenschaften wie die Exclusion Region oder den Hinder-

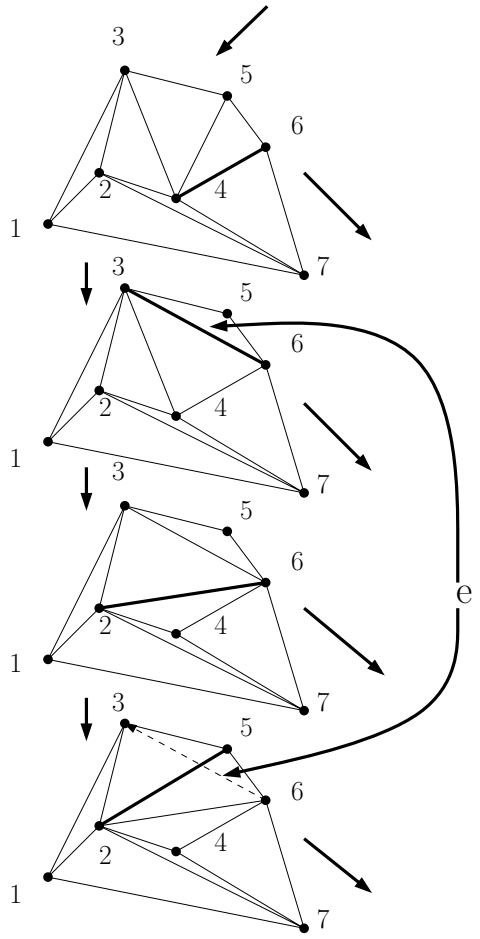
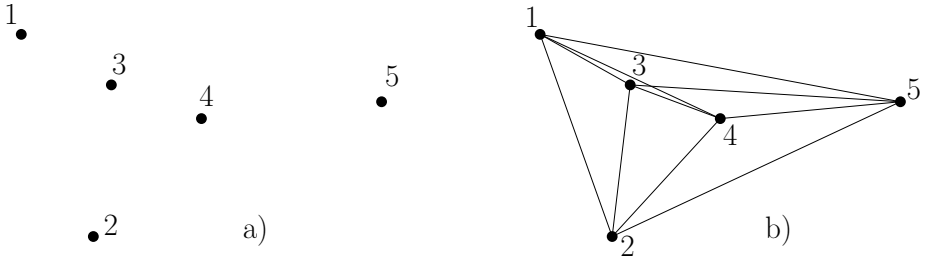


Abbildung 4.10: Ausschnitt aus einem Triangulierungsbaum. Die f_{max}-Kanten sind jeweils dunkel hervorgehoben. Die Kante e verschwindet wieder aus einem Kind.

nisswert einer Kante handelt, müssen Kanten, die das entsprechende Kriterium verletzen auch in allen Kindern vorkommen, damit ein Teilbaum abgeschnitten werden kann. Leider ist das bei diesem Verfahren nicht der Fall, wie sich in Abbildung 4.10 leicht erkennen läßt.

Leider läßt sich mittels der bis jetzt gefunden MDT-Eigenschaft der *Reverse-Search Algorithmus* nicht verbessern und ein anderer Ansatz muß beschritten werden. Ein möglichen Ausweg stellt die Bitwise Enumeration dar, welche einen konsequent optimierten Brute-Force Ansatz mit Reduktion der Suchmenge darstellt.



Kanten: (1,1) (1,2) (1,3) (1,4) (1,5) (2,3) (2,4) (2,5) (3,4) (3,5) (4,5)
Bitstring von a) = 0 0 0 0 0 0 0 0 0 0
Bitstring von b) = 1 1 1 1 1 1 1 1 1 1

Abbildung 4.11: a) Punktmenge S, b) Alle möglichen Kanten von S, eine Liste der möglichen Kanten und des daraus resultierenden Bitstrings

4.3 Bitwise Enumeration

Dieses Enumerationsverfahren beruht auf einem Brute Force Ansatz und wurde im Zuge dieser Diplomarbeit ausgearbeitet. Allerdings muss ich dazu sagen, das es nicht alle möglichen Triangulierungen enumeriert sondern alle Triangulierungen, welche Kandidaten für eine mögliche MDT darstellen.

4.3.1 Ansatz

Die grundlegende Idee ist folgende: Wenn man sich überlegt, daß Triangulierungen aus unterschiedlichen Kantenkombinationen bestehen, liegt die Idee nahe, alle Kantenkombinationen auszuprobieren um alle Triangulierungen zu enumerieren. Dazu wird als erstes das Set S_K aller möglichen Kanten basierend auf der zugrundeliegenden Punktmenge S gebildet. Dies geschieht recht einfach, indem man alle Punkte paarweise miteinander verbindet.

$$S_K = \{(p_i, p_j) : 1 \leq i < j \leq |S|\}$$

Um nun alle möglichen Zusammensetzungen dieser Kantenmenge zu erstellen wird ein Bitstring der Länge $|S_K|$ erstellt, wobei jedes Bit dieses Strings eine Kante aus S_K repräsentiert. Jeder Kantenkombination kann man nun einen solchen Bitstring zuordnen, wobei ein gesetztes Bit für das Vorhandensein einer Kante steht. Ist das Bit nicht gesetzt, kommt auch die Kante in dieser Kombination (oder auch Konfiguration) nicht vor. Angefangen bei 0 lässt sich nun der Bitstring komfortabel bis zu $2^{|S_K|} - 1$ hochzählen und man enumeriert garantiert jede mögliche Kombination der Kanten und damit auch die Teilmenge aller Triangulierungen.

Am Beispiel (siehe Abbildung 4.11) einer Punktmenge mit fünf Punkten läßt sich das Vorgehen veranschaulichen und ein erster Optimierungsansatze erkennen.

Die Punkte werden mit Indizes versehen, die Menge aller möglichen Kanten $S_K$werden erstellt, wobei es bei n Punkten $\binom{n}{2} = \frac{n(n-1)}{2}$ mögliche Kanten gibt. Danach wird ein Bitstring der Länge $L = |S_K|$ erstellt und jedem Bit eine Kante aus S_K zugewiesen. In Abbildung 4.12 ist ein Beispiel abgebildet.

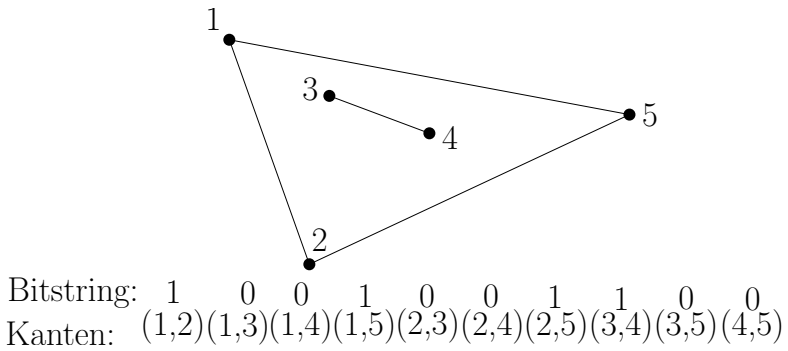


Abbildung 4.12: Beispiel für die Interpretation eines Bitstrings

Und bei diesem Beispiel wird auch sofort ein Problem des Ansatzes deutlich: Dieser Bitstring repräsentiert leider keine gültige Triangulierung. Der Vorteil dieses Verfahrens gegenüber dem Ansatz von Bespamyatnikh liegt allerdings in der Möglichkeit radikal optimieren zu können, womit ich die Pfade des Brute-Force-Ansatzes verlasse. Fairerweise muß dazu gesagt werden, das das Ziel von Bespamyatnikh die Enumerierung aller Triangulierungen und nicht die Findung einer speziellen Triangulierung darstellt.

4.3.2 Modifikationen

Hochzählen der k-Bitkonfigurationen

Bis zu diesem Punkt haben wir einen reinen Brute Force Ansatz, welcher mit einem immens hohen Overload zu kämpfen hat. Leicht lässt sich erkennen das bereits die erste Konfiguration 00...000 mit keiner gesetzten Kante keine gültige Triangulierung darstellt. Wie allerdings in 1 schon bewiesen hat eine Triangulierung für eine Punktmenge S immer eine feste Kantenzahl, die sich problemlos berechnen lässt. Wir wissen also bereits im vor raus, wie viele Kanten unsere gesuchten Triangulierungen haben. Also wissen wir auch wie viele Bits im hochzuzählendem Bitstring gesetzt sein müssen. Eine große Zeiteinsparung ist es also, nur die Konfigurationen auszutesten, welche die benötigte Anzahl an Kanten haben. Diese Konfigurationen werde ich in Zukunft als k-Bitkonfigurationen bezeichnen, wobei k die Anzahl der benötigten Kanten bezeichnet.

Und hier bietet sich ein weiterer Ansatz zur Optimierung. Es wäre optimal, wenn man von einer k-Bitkonfiguration direkt zur nächsten hochzählen könnte, ohne die Zwischenkonfigurationen berücksichtigen zu müssen. Genau das ist auch die erste Optimierung, auf welche ich nun eingehen werde. Wie schon in Theorem **??** gezeigt, läßt sich die Anzahl der k-Bitkonfigurationen im voraus berechnen und wenn man sich folgendes Beispiel vor Augen hält wird deutlich, wie viele Bitkonfigurationen sich mit diesem Ansatz auslassen lassen.

> Angenommen man hätte eine 25-stellige Binärzahl. Das resultiert in $2^{25} = 33.554.432$ möglichen Konfigurationen, die alle ausgetestet werden müßten. Betrachtet man aber nur die wirklich interessanten k-Bitkonfiguration mit $k = 20$ so reduziert sich die Anzahl der zu

testenden Konfigurationen auf $\binom{25}{20} = 53130$, das sind gerade mal $0,0016\%$ der möglichen Konfigurationen.

Das Verfahren werde ich im folgenden näher erläutern, wobei es am einfachsten ist sich den Startzustand, also die kleinste k-Bitkonfiguration anzuschauen.

Der Startzustand eines Bitstrings B einer Länge von n_B Bits und k benötigten Kanten, ist folgender:

$$\underbrace{00..00}_{n_B - kBits}\ \underbrace{11..11}_{kBits}$$

Die einzelnen Bits lassen sich durch Indizes ansprechen, wobei B_0 das Bit mit der Wertigkeit 2^0 und B_{n_B-1} das Bit mit der Wertigkeit 2^{n_B-1} adressiert.

Die Frage ist also, welches ist die nächstgrößere Konfiguration, welche wiederum k gesetzte Bits enthält. Dazu betrachten wir zunächst wie man überhaupt von einer beliebigen Binärzahl auf die nächsthöhere Binärzahl kommt, ohne daß eine Bedingung an die Anzahl von gesetzten Einsen gestellt wird. Sollte das kleinste Bit $B_0 = 0$ sein, so ersetzt man es durch 1 und ist fertig.

$$101\underbrace{0}_{B_0} \Rightarrow 101\underbrace{1}_{B_0}$$

Andernfalls sucht man sich das erste Bit B_i, so dass $B_i = 0$ ist und $\forall j : 0 \leq j < i \Rightarrow B_j = 1$ gilt, und setzt B_i auf 1. Allerdings verdoppelt sich durch jedes weitere Bit der Wert der Binärzahl und man hat nun zur ursprünglichen Binärzahl 2^i anstatt einer 1 addiert und muss dementsprechend den Wert $2^i - 1$ subtrahieren. Dieses entspricht dem Löschen der Bits B_{i-1} bis B_0.

$$1\underbrace{0}_{B_{i=2}}11 \Rightarrow (1\underbrace{1}_{B_{i=2}}11) \Rightarrow 11\ \underbrace{00}_{B_{i-1}B_0}$$

Das Auffinden der nächsthöheren Binärzahl wäre damit beschrieben, leider besitzt diese aber nicht mehr die gewünschte Anzahl an gesetzten Bits. Denn sollte der erste Fall vorgelegen haben, so enthält die neue Binärzahl $k + 1$ statt der gewünschten k Bits. Das heißt, es muss noch mindestens eine Addition folgen, die dann jedoch vom Typ 2 sein wird. Die Lösung ist also im zweiten Fall zu suchen.

Im zweiten Fall fehlen uns nach der Addition $i - 1$ Bits. Das Bit B_i war das erste nicht gesetzte Bit, daher waren die Bits B_{i-1} bis B_0 gesetzt. Für eines dieser i Bits wurde B_i gesetzt, bleiben also noch $i - 1$ Bits. Diese $i - 1$ fehlenden Bits können wir auf die Bits der kleinsten Wertigkeit verteilen um die nächst höhere Konfiguration mit passender Anzahl an gesetzten Bits zu erhalten.

$$101\underbrace{0}_{B_{i=3}}\ \underbrace{111}_{\text{i gesetzte Bits}} \Longrightarrow 101\underbrace{1}_{B_i}000, \text{ es fehlen i-1=2 Bits}$$

$$101\underbrace{1}_{B_{i=3}}000 \quad \underbrace{\Longrightarrow}_{\text{setze die letzten i-1 Bits}} \quad 10011011$$

Und dieses Prinzip lässt sich auf alle Ausgangskonfigurationen verallgemeinern. Die Nachfolgerkonfiguration einer Binärzahl mit k gesetzten Bits lässt sich daran erkennen, daß sich beim Hochzählen ein Bit nach links verschiebt und es

so möglich wird die restlichen gesetzten Bits auf die kleinsten Wertigkeiten zu verteilen. Und genau diese Beobachtung ermöglicht es von der aktuellen Konfiguration auf die nächste zu schließen. Man sucht nach dem kleinsten gesetzten Bit B_{next}, dessen höherwertiges Nachbarbit nicht gesetzt ist, setzt dieses Nachbarbit auf 1 und verteilt die restlichen $i_{next} - i_{smallest}$ niederen Bits auf die kleinsten Stellen. Der Index $i_{smallest}$ bezeichnet dabei die Position des kleinsten gesetzten Bits. Der Algorithmus, welcher das Problem löst, ist im folgenden als Pseudocode angegeben.

procedure CountUpToNextBitConfiguration(Bitstring B, $n_B = |B|$ **)**

$i_{smallest} = min\{i \geq 0 \vert B_i = 1\}$	$O(n_B - k) = O(n_B)$
if $i_{smallest} = n - k$ then 'Endzustand. Abbruch'	$O(1)$
$i_{next} = min\{i \geq i_{smallest} \vert B_{i+1} = 0\}$	$O(1)$
Setze $B_{i_{next}+1} = 1$	$O(1)$
Lösche $B_{i_{smallest}}$ bis $B_{i_{next}}$	$O(k-1) \leq O(n_B)$
Setze B_0 bis $B_{i_{next}-i_{smallest}-1}$	$O(n_B - 2k - 1) = O(n_B)$
return B	$= 3O(n_B) = O(n_B)$

Um die Arbeitsweise des Algorithmus zu erläutern, sind im folgenden alle Zustände für ein Kantenset S_K mit $n_B = 5$ Kanten und einer benötigten Kantenzahl von $k = 3$ aufgelistet:

$$00111 \Rightarrow 01011 \Rightarrow 01101 \Rightarrow 01110 \Rightarrow 10011$$
$$\Rightarrow 10101 \Rightarrow 10110 \Rightarrow 11001 \Rightarrow 11010 \Rightarrow 11100$$

Und hier folgt der Übergang von 10110 zu 11001 im Detail:

Bitstring $B = 10110$
Bestimme Indizes: $10 \underbrace{1}_{B_{i_{next}}} \underbrace{1}_{B_{i_{smallest}}} 0$
$i_{next} = 2, i_{smallest} = 1$
Setze $B_{i_{next}+1} = B_3$: 11110
Lösche $B_{i_{smallest}} = B_1$ bis $B_{i_{next}} = B_2$: 11000
Setze B_0 bis $B_{i_{next}-i_{smallest}-1} = B_{2-1-1} = B_0$: 11001

Korrektheitsbeweis des Algorithmus Um den Beweis führen zu können werde ich ein paar Definitionen zur Hilfe nehmen. Sei erstens die Menge B der k-Bitkonfigurationen wie folgt definiert:

$$B = \{w \in \{0,1\}^n | \#_w 1 = k\}$$

Also ist B die Menge von Bitstrings der mit jeweils der gleichen Länge n_B mit k gesetzten Bits und $n_B - k$ nicht gesetzten Bits. Weiterhin sei das maximale Element b_{max} der Menge B wie folgt definiert:

$$b_{max} = 1^k 0^{n_B - k}$$

Der Algorithmus sei eine Funktion f welche zu einer k-Bitkonfiguration nach der oben erläuterten Weise den Nachfolger liefert. Für den Beweis der Vollständigkeit und der Korrektheit sind drei Punkte zu beweisen:

- $f : B \backslash b_{max} \rightarrow B$
  Es gilt sich also zu vergewissern, daß der gelieferte Nachfolger wiederum ein Bitstring der Länge n mit k gesetzten Bits darstellt. Für b_{max} kann natürlich kein Nachfolger gefunden werden.

  Der Algorithmus ändert nicht die Länge des übergebenen Bitstrings. Daher bleibt nur noch zu zeigen, dass er auch nicht die Anzahl an k gesetzten Einsen ändert. Gesetzt werden die Bits B_0 bis $B_{i_{next}-i_{smallest}-1}$ und $B_{i_{next}+1}$. Zusammen sind das $i_{next} - i_{smallest} + 1$ Bits. Gelöscht werden die Bits $B_{i_{next}}$ bis $B_{i_{smallest}}$. Auch das sind $i_{next} - i_{smallest} + 1$ Bits, womit der erste Punkt bewiesen wäre.

- $f(x) > x, x \in B \backslash b_{max}$
  Hier ist zu zeigen, daß der gelieferte Nachfolger eine größere Bitfolge als sein Vorgänger darstellt. Auch das ist schnell gezeigt.

  Der Algorithmus setzt das Bit $B_{i_{next}+1}$, ein Bit welches vorher noch nicht gesetzt war. Sozusagen werden $2^{i_{next}+1}$ hinzu addiert. Alle danach durchgeführten Operationen finden auf niderwertigen Bist, den Bits $B_{i_{next}}$ bis B_0 statt. Selbst wenn alle diese Bits gelöscht würden, was nicht der Fall ist, würde dadurch der Wert $2^{i_{next}+1} - 1$ subtrahiert werden. Der Nachfolger ist damit mindestens um eins größer als sein Vorgänger. Damit wäre auch dieser Punkt bewiesen.

- $\forall x : \neg \exists y \in B : x < y < f(x)$
  Bleibt als letztes nur noch zu zeigen, dass der Algorithmus auch wirklich den nächsten Nachfolger liefert und nicht weitere gültige k-Bitkonfigurationen überspringt.

  Sei $y \in B \setminus \{x\}$. Zusätzlich seien folgende Indices definiert:

$$
\begin{aligned}
i_{diff} &:= \max_{0 \leq i < n_B} x_i \neq y_i \\
i_{smallest} &:= \min_{0 \leq i < n_B} x_i = 1 \\
i_{next} &:= \min_{i_{smallest} < i < n_B} x_{i+1} = 0
\end{aligned}
$$

  Die k-Bitkonfiguration y ist genau dann $> x$, wenn $y_{i_{diff}} = 1$ und $x_{i_{diff}} = 0$ gilt. Denn bis dahin waren die höherwertigen Bits laut Definition gleich und mit dem Unterschied an der Stelle i_{diff} können auch die niederwertigen Bits aufgrund von $2^i > \sum_{j=0}^{i-1} 2^j$ daran nichts ändern.

  Daraus folgt $i_{diff} > i_{next}$. Warum dem so ist, darauf gehe ich jetzt ein. Der Fall $i_{diff} = i_{next}$ ist per Definition ausgeschlossen, da $x_{i_{next}} = 1$.

  Sollte $i_{diff} < i_{next}$ sein, so folgt $i_{diff} < i_{smallest}$, da ja $y_{i_{diff}} = 1$ und $x_{i_{diff}} = 0$ gelten. Diesen Wert kann x_{i_diff} für $i_{diff} < i_{next}$ aber nur für $< i_{smallest}$ annehmen, da per Definition der Bereich zwischen i_{next} und $i_{smallest}$ bei x mit lauter Einsen gesetzt ist. Das heißt aber, daß x und y bis zu $i_{smallest}$ übereinstimmen. Da jedoch x, y beide die gleiche Anzahl an gesetzten Einsen haben und im Bereich $x_{i_{smallest}-1} ... x_0$ alle Bits nicht gesetzt sind, so hat auch y keine Bits mehr über, die gesetzt werden könnten. Von daher ist auch $i_{diff} < i_{next}$ widerlegt.

Also muß $i_{diff} > i_{next}$ sein. Sollte $i_{diff} = i_{next} + 1$ sein, so hat y die gleiche Charakteristik wie $f(x)$, welches sich ja auch erst an der Stelle $i_{next} + 1$ von x unterscheidet. Da $f(x)$ jedoch die kleinste Zahl in dieser Situation bildet, da alle restlichen Einsen auf die niederwertigsten Bits gesetzt werden, folgt daraus $y \geq f(x)$.

Sollte $i_{diff} > i_{next} + 1$ sein, so ist y erst recht größer als $f(x)$, da $f(x)$ in diesem Fall bis einschließlich zum Index i_{diff} mit x übereinstimmt und beide wegen $y_{i_{diff}} = 1$ und $x_{i_{diff}} = f(x)_{i_{diff}} = 0$ kleiner sind als y.

Damit ist gezeigt, daß für ein $y > x$ erstens $i_{diff} > i_{next}$ gílt und daraus folgt $y \geq f(x)$, womit die Behauptung bewiesen wäre, daß zwischen x und $f(x)$ keine weitere k-Bitkonfiguration existieren kann und $f(x)$ der wirkliche Nachfolger zu x ist.

□

Kollisionsabfrage

Der Übergang zwischen den k-Konfigurationen bietet noch weitere Optimierungsmöglichkeiten und ist mit dem gerade vorgestellten CountUp-Algorithmus noch längst nicht ausgeschöpft. Einen weiteren wesentlichen Anteil nehmen die Bitkonfigurationen ein, welche zwar die richtige Anzahl an gesetzten Bits besitzen, aber bei denen sich die enthaltenen Kanten überschneiden (im folgenden wird der Ausdruck *Kollision* von mir synonym gebraucht). Da eine Triangulierung eine maximale Menge sich nicht schneidender Kanten ist, sind auch solche Konfigurationen ein Overload.

Leider scheint es nicht möglich zu sein einen Hochzählalgorithmus mit einer Laufzeit von $O(\#Bits^2)$ pro Schritt zu finden, welcher den nächsthöheren Triangulierungsstring liefert. Im Applet sieht der Vorgang folgendermaßen aus, daß während der Erstellung des Kantensets S_K die Kollisionen zwischen den Kanten registriert werden. Für jede Kante wird dann regisrtiert, mit welchen anderen Kanten sie kollidiert. Die Ergebnisse werden in eine sortierte Liste pro Kante gespeichert. Dort ist dann verzeichnet, mit welchen anderen Kanten eine Überschneidung stattfindet. Während des Enumerationsprozesses wird dann für jede vorkommende Kante überprüft, ob eine der mit ihr in Kollision tretenden Kanten in der aktuellen Konfiguration vorhanden beziehungsweise gesetzt sind. Sollte sich die aktuelle Konfiguration nicht als Triangulierung (die Triangulierungskanten sind kreuzungsfrei) herausstellen, wird zur nächsten k-Bitkonfiguration hochgezählt bis entweder eine gültige Triangulierung vorliegt oder der Endzustand 11..1100..00 erreicht ist. Dieser Überprüfungsvorgang dauert dann am längsten, wenn die Triangulierung 00..0011..11 vorliegt und die Triangulierungskanten jeweils mit den $n-k$ übrigen Kanten in Konflikt stehen. Dann finden $k * (n-k) = O(n^2)$ Überprüfungen statt, bis festgestellt wird, ob es sich um eine Triangulierung handelt.

Die Anzahl an Bitkonfigurationen mit k gesetzten Bits, welche sich während der Bitwiseenumeration nicht als Triangulierung herausgestellt haben und einzusparende Konfigurationen darstellen, wird am Ende des Prozesses in der Statuskonsole ausgegeben.

Der enorme Effekt dieser Optimierungsmöglichkeit, also eine Aussparung dieser Kollisionsgraphen bei der Enumeration, wird durch experimentelle Ergebnisse im nächsten Kapitel belegt. Eine möglichst einfache und effiziente Me-

thode der Kollisionserkennung und Vermeidung würde eine weitere wesentliche Performancesteigerung bewirken.

Entfernung unnötiger Kanten

Eine weitere Optimierung ist die Reduzierung des Kantensets S_K. Dabei kann man Kanten aus zwei Gründen aus S_K ausschließen. Einerseits wenn man weiß, dass sie unter keinen Umständen in der gesuchten MDT vorkommen können. Andererseits gibt es auch Kanten, von denen man schon im Voraus sagen kann, daß sie in der MDT vorkommen müssen. Gesucht sind also Kantenkriterien, welche Kanten zu einer von drei Teilmengen zuweißt: Kanten die nicht (Kantenset S_{NOT}) oder auf jedenfall (Kantenset S_{MUST}) enthalten sind und die Menge der Kanten, deren Zugehörigkeit nicht sicher ist (Kandidatenset S_K).

Zu den Kriterien, welche Kanten aus der MDT ausschließen können, gehören die Exclusion Region sowie das wesentlich mächtigere Obstacle-Value (Hindernisswert) Kriterium. Beide sind auch im Applet implementiert, wobei der gleichzeitige Einsatz keinen Laufzeitvorteil gegenüber der alleinigen Nutzung des Hindenisswertes bringt.

Auch zu den Elementen der Menge S_{MUST} liegen Kriterien vor. Die Kanten der konvexen Hülle j der zugrundeliegenden Punktmenge sind Elemente von S_{MUST}. Die konvexe Hülle der Punktmenge $S, n = |S|$ ist allen Triangulierungen von S gemeinsam, also auch der gesuchten MDT. Dabei hat die konvexe Hülle mindestens 3, maximal $n-1$ Kanten. Mittels der in **??** vorgestellten Formel lässt sich dann von Fall zu Fall die Anzahl der gespaarten Binärkombinationen berechen. Ein theoretische Aussage lässt sich da schon schwieriger treffen. Auch zu dieser Optimierung befinden sich im nächsten Kapitel experimentelle Ergebnisse, die die Auswirkung dieser Optimierung in der Praxis verdeutlichen. Zweitens sind auch die Kanten Elemente von S_{MUST} die, wie in Kapitel 3.2.1 gezeigt, beidseitig eindeutige nächste Nachbarn miteinander verbinden. Zur Klarstellung sei nochmal darauf hingewiesen, daß diese Optimierung in der Initialisierungsphase des Algorithmus stattfindet unhd nicht zur Laufzeit. Im Gegensatz zur nächsten Optimierung.

Vorzeitiger Abbruch der Enumeration

Es gibt noch eine Laufzeitoptimierung, denn das Ziel ist es ja, die $MDTs$ möglichst früh zu finden. Um dies zu begünstigen werden die übrig gebliebenen Kanten des Kantensets S_K vor dem Start der Enumeratiob so neu geordnet, dass die aussichtsreicheren Kandidaten im hochzuzählendem Bitstring von den kleinsten Bits repräsentiert werden. Somit gehören sie zu den Kanten, welche als Erste enumeriert werden. Dazu nutzen wir den Hindernisswert einer Kante, welcher in der Initialisierungsphase des Algorithmus bereits berechnet wurde.

Mit Hilfe dieses Wertes wurde schon das Kantenset S_K reduziert, die übrig gebliebenen Kanten sind die, mit einem Hinderniswert kleiner der bisherigen bekannten kleinsten Dilation: Der Dilation der explizit berechneten Delaunaytriangulierung. Sollte während des Enumerationsprozesses eine Triangulierung mit kleinerer Dilation auftauchen, so haben wir eine neue obere Schranke für den Hinderniswert der Kanten. Sind die Kanten dann nach ihrem Hinderniswert sortiert, so daß die Kanten mit dem größten Hinderniswert auch durch die höchstwertigsten Bits repräsentiert werden, so kann diesmal S_K während

der Laufzeit reduziert werden. Dazu wird das kleinste Bit Bit B_j herausgesucht, welches die neue obere Schranke verletzt. Alle höherwertigen Bits werden die Schranke auch verletzen, da die Kanten nach ihrem Hinderniswert sortiert vorliegen. Es können nun zwei Fälle auftreten. Erstens kann es sein, daß der Bitstring noch nicht bis zu B_j hochgezählt wurde. In diesem Fall kann der Bitstring um die Bits $B_{n-1}...B_j$ verkürzt werden und im günstigsten Fall werden damit $\binom{n}{k} - \binom{j}{k}$ k-Bitkonfigurationen gespart.

Im zweiten Fall tauchte B_j schon in vergangenen k-Bitkonfigurationen auf und alle folgenden Konfigurationen würden mindestens ein Bit von $B_{n-1}...B_j$ enthalten. Da diese jetzt aber einen zu hohen Obstacle-Value besitzen, können die aktuelle und alle zukünftigen Konfigurationen keine Kandidaten für die *MDT* sein und der Enumerationsvorgang kann abgebrochen werden. Die *MDT* wurde schon gefunden.

Abschließend bleibt zu sagen, daß wir während der Enumeration unser Wissen über die bisher gefundenen Triangulierungen einfließen lassen können und es zu erwarten ist, das die obere Schranke im Laufe der Enumeration weiter sinken kann, wenn bessere Triangulierungen gefunden werden. Einerseits sind zwar die Kanten welche noch nicht mit in die Berechnung eingeflossen sind, die mit dem höheren Hindernisswert. Andererseits gibt es wie schon in Kapitel **??** gezeigt ein globales Zusammenspiel zwischen den Kanten, welches auch solche Kanten mit schlechterem Hindernisswert mit einbezieht.

Beide Optimierungen, sowohl das Ausschließen als auch die Identifizierung als *MDT*-Kante, führen bei kleineren Punktmengen meistens zur sofortigen Findung der *MDT*. Was man natürlich bedenken muss, ist, daß sich nicht nur das Kandidatenset S_K verringert, sondern auch die Anzahl an benötigten gesetzten Bits. Aus den gesuchten k-Bitkonfigurationen werden (k-$|S_{MDT}|$)-Bitkonfigurationen. Um sich den Effekt der Reduktion von S_K vor Augen zu führen, sei nochmal darauf hingewiesen, daß jedes weitere Bit, also jede weitere Kante, eine Verdoppelung des Wertebereichs des hochzuzählenden Bitstrings mit sich bringt und damit eine Vergrößerung der möglichen k-Bitkonfigurationen. Auch sollte man bei diesem Verfahren und steigender Punktmenge nicht den Effekt eines weiteren Punktes auf die Menge möglicher Kanten unterschätzen. Jeder weitere Punkt zur bestehenden Punktmenge der Größe n führt zu einer Vergrößerung der Kantenmenge S_K um n Kanten. Um ein Gefühl für die Auswirkungen zu bekommen sei hier auf die experimentellen Ergebnisse im nächsten Kapitel hingewiesen.

4.3.3 Laufzeit des Bitwise-Enumeration Algorithmus

Gegeben sei eine Punktmenge S mit $n = |S|$ Punkten. Auf der Suche nach der *MDT* verfährt der Bitwise-Enumeration Algorithmus wie folgt:

Schritt	Beschreibung	Laufzeit
	Initialisierung	
I-1	Berechnung der konvexen Hülle	$O(n \log n)$
I-2	Berechnung der Delaunaytriangulierung	$O(n \log n)$
I-3	Berechnung der Anzahl an Triangulierungskanten k	$O(1)$
I-4	Erstellung des Kandidatensets S_K	$O(n^2)$
I-5	Extraktion der konvexen Hülle aus S_K nach S_{MUST}	$O(n)$
I-6	Berechnung des Hinderniswertes jeder Kante $k \in S_K$	$O(n^4)$
I-7	Sortiere S_K anhand der Hinderniswerte	$O(n^2 \log n^2)$
I-8	Berechnung der Kollisionen aller Kanten aus S_K	$O(n^4)$
	Bitwise Enumeration (Laufzeit pro Schritt)	
B-1	Berechung der nächste k-Bitkonfiguration	$O(n^2)$
B-2	Kollisionscheck. Bei Kollision fahre mit Schritt B-1 fort	$O(n^2)$
B-3	Berechnung der Dilation für diese Triangulierung	$O(n^2)$
B-4	eventuelle Verbesserung der oberen Schranke	$O(1)$
B-5	Abbruch, wenn obere Schranke verletzt wird	$O(1)$
B-6	Abbruch, wenn letzte Konfiguration erreicht	$O(1)$
B-7	Fahre mit B-1 fort	

Die verschiendensten optimale Wege die konvexe Hülle und die Delaunaytriangulierung zu berechnen können bei [Kle98] nachgeschlagen werden. Die Berechnung des Kandidatensets S_K und die Bestimmung der Hinderniswerte sind wahrscheinlich nicht optimal und könnten weiter optimiert werden. Insgesamt nimmt die Initialisierung $2O(n \log n) + O(n^2) + O(n) + 2O(n^4) + O(n^2 \log n^2) = O(n^4)$ Zeit in Anspruch. Der Berechnungsvorgang hingegen nimmt im Worstcase für eine k-Bitkonfiguration $3O(n^2) = O(n^2)$ in Anspruch. Da es bei n Punkten für eine Triangulierung mit k Kanten insgesamt $\binom{n}{k}$ mögliche k-Bitkonfigurationen gibt, ist also spätestens nach $O(\binom{n}{k}n^2)$ Schritten der Berechnungsvorgang erfolgreich abgeschlossen.

Für $k = n/2$ lassen sich die Anzahl an Triangulierungen wie folgt abschätzen:

$$\binom{n}{k} = \frac{n * ... * (\frac{n}{2} + 1)}{(\frac{n}{2})!} = \frac{n * ... * (\frac{n}{2} + 1)}{\frac{n}{2} * ... * 1} \geq 2^{\frac{n}{2}}$$

Zu einer Punktemenge $S, n = |S|$ und Triangulierungen mit k Kanten kann die dazugehörige $MDT(S)$ in insgesamt $O(n^4) + O(\binom{n}{k}n^2)$ berechnet werden. Für $k = n/2$ ergibt das eine Laufzeitabschätzung von $O(n^4) + O(2^{\frac{n}{2}}n^2)$, also nicht in polynomieller Zeit und in polynomiellen Platz (jede der $2^{\frac{n}{2}}$ Triangulierungen benötigt Speicherplatz) lösbar.

4.4 Ein möglicher Greedyalgorithmus

Eher als Heuristik, gehe ich nun noch kurz auf einen Greedyansatz ein. Warum ein Greedyalgorithmus im Falle der Dilationsberechnung so problematisch ist, werde ich danach diskutieren.

Ein Greedyalgorithmus baut sich die Lösung Stück für Stück zusammen indem er jedes Element nach bestimmten Kriterien auswählt, zur Lösung hinzufügt und diesen Schritt solange wiederholt, bis das Ergebnis vollständig ist. Das Vorgehen, welches sich in diesem Fall anbietet, ist es solange Kanten auszuwählen, bis eine vollständige Triangulierung erreicht ist. Über den Hinderniswert einer Kante lassen sich zudem schnell günstige Kanten von Ungünstigen trennen und ein Entscheidung treffen. Für eine Punktmenge $S, n = |S|$ mit Kantenmenge $K = \{(i,j), 1 \leq i \neq j \leq n\}, |K| = \frac{n(n-1n)}{2}$, mit $n_T = \#Triangulierungskanten$ (siehe Therorem 1) und zu Anfang nur aus der Punktmenge bestehenden Triangulierung $T = T_K = (\emptyset, S)$ läßt sich ein Greedyalgorithmus zum Beispiel wie folgt implementieren:

Laufzeit:

Berechne den Hinderniswert für alle Kanten $k \in K$:	$O(n^4)$
Für jedes Punktpaar $x, y \in S, x \neq y$:	$O(n^2)$
Berechne den Hindernisswert für k, x und y	$O(1)$
Sortiere alle Kanten anhand ihres Hinderniswertes	$O(n^2 \log n^2)$
	$= O(n^4)$
Solange wie T unvollständig (Laufzeit pro Kante):	
Extrahiere die Kante $k \in K, \delta_k < \delta_e, \forall e \in K \neq k$	$O(1)$
Überprüfe k auf Schnitt mit $\forall j \in T_K$	$O(n_T^2)$
Kein Schnitt, dann $T_k = T_k \cup k$	$O(1)$
	$= O(n_T^2)$

Im WorstCase und mit nicht optimiertem Bruteforece-Ansatz ergibt sich eine Laufzeit von

$$O(n^4) + O(n_T^2) = O(n^4).$$

Also läßt sich in immerhin polynomieller Zeit eine Greedytriangulierung berechnen die der *MDT* zumindest nahe kommen könnte. Denn wie sich bei vorherigen Überlegungen schon dargelegt hat, ist der Hinderniswert einer Kante eine isolierte Kanteneigenschaft und ermöglicht daher auch kein Zusammenspiel mehrerer Kanten bei der Findung der kleinstmöglichen Dilation. Demensprechend ist auch das Ergebnis des Greedyansatzes fehlerbehaftet. Ein Beispiel ist in Abbildung 4.13 abgebildet.

In diesem Fall findet der Greedyalgorithmus nicht die optimale Lösung, da die Kanten mit dem kleinstem Hinderniswert in Kombination eine höhere Dilation verursachen, als andere Kanten. Es gibt natürlich auch Fälle, in denen das Ergbenis korrekt ist, allerdings hatte ich gehoft, das man den Greedyalgorithmus entsprechend modifizieren könnte um die korrekte *MDT* zu bestimmen. Ein Ansatz, welchen ich anfangs recht zuversichtlich verfolgt hatte, geht alle Kanten der berechneten Triangulierung durch und überprüft ob ein Flip der flipbaren Kanten zu einer besseren Triangulierung führen würde. Um bei diesem Ansatz

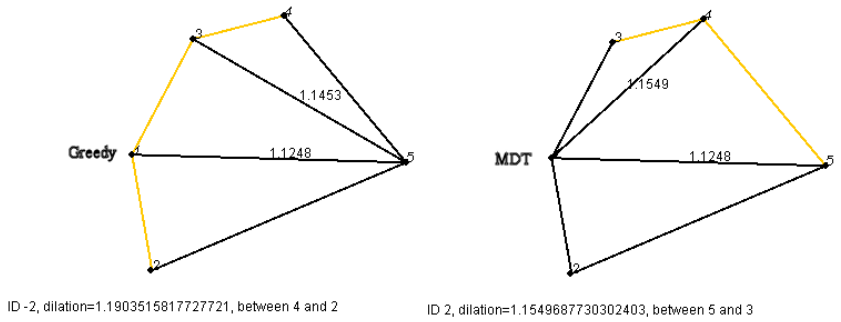


Abbildung 4.13: Die Greedytriangulierung betrachtet Kanten nur isoliert.

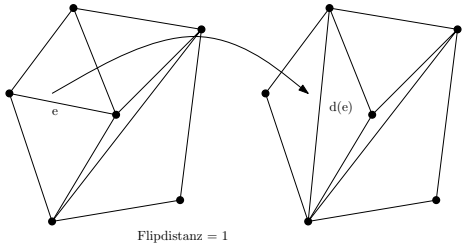


Abbildung 4.14: Beispiel für zwei benachbarte Triangulierungen mit Flipdistanz Eins.

aber eine Laufzeitverbesserung zu erhalten und nicht im Worst-Case bei rekursiven Flips alle Triangulierungen zu enumerieren, stellt sich die Frage wieviel Flips die Greedytriangulierung und die wirkliche MDT auseinander liegen. Eine Möglichkeit für ein Maß wäre die Flipdistanz zwischen zwei Triangulierungen

Flipdistanz:
Die Flipdistanz zweier Triangulierungen gibt die minimal benötigte Anzahl an Edgeflips an, die benötigt wird, um von der einen Triangulierung über Edgeflipoperationen die andere Triangulierung zu erstellen. Zwei benachbarte Triangulierung haben eine Flipdiszanz von Eins, zwei gleiche Triangulierungen eine Flipdistanz von Null. Siehe dazu auch Abbildung 4.14

Um es kurz zu machen: Die modifizierte Version des Greedyalgorithmus, welcher nur eine Ebene tief die Kanten der Greedytriangulierung flippt, führt auch zu keinem besseren Ergebnis (Abbildung 4.15). Eine Herausforderung ist es dabei auch die richtige Reihenfolge der Flips zu berechnen, da diese wie gesehen(4.10) das eigentliche Problem darstellt.

Im Javaprogramm sind sowohl der normale Greedyalgorithmus als auch seine modifizerte Version implementiert, welche das Ergebnis wie folgt zu verbessern versucht:

Für jede flipbare Kante $k \in T$:
Wenn $\delta_{d(k)} < \delta_k$ ersetze in T die Kante k durch $d(k)$.

Es können also *MDTs* gefunden werden, die durch die iterative Überprüfung jeder flipbaren Kanten und deren eventuellem Flip erreicht werden können. Pro-

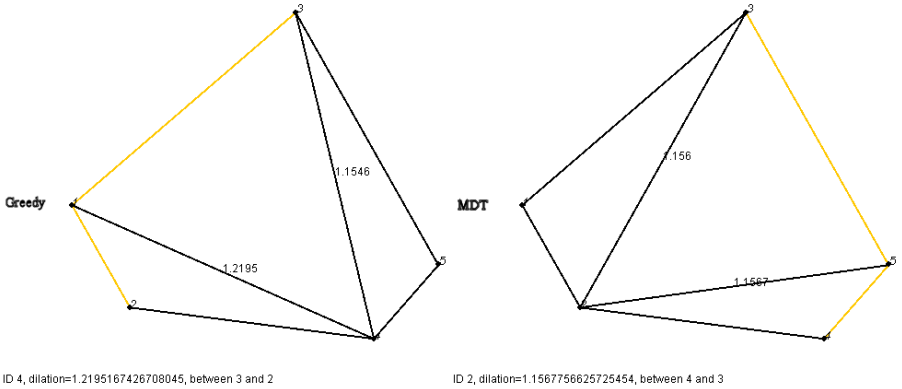


Abbildung 4.15: Die Greedytriangulierung hat zur *MDT* eine Flipdistanz von Zwei.

blematisch wird es, wenn mehrere Flips in bestimmter Reihenfolge benötigt werden. Mehr als ein Tropfen auf den heissen Stein Stellt diese Modifikation also nicht dar. Doch um die Problematik zu veranschaulichen und eine Vorstellung von der Greedytriangulierung zu bekommen ist diese Option im Programm enthalten.

Ausblick:

Um zu einer Verbesserung gegenüber einer Enumerierung ala Bespamyatnikh zu kommen, muß also gezeigt werden, das die Flipdistanz zwischen Greedytriangulierung und *MDT* kleiner ist als die Anzahl aller Triangulierungen der entsprechenden Punktmenge. Nur dann, kann man sicher sein, das der modifizierte Greedy schneller zu einer Lösung kommt, vielleicht sogar mit einer berechenbaren oberen Schranke. Vielleicht gibt es neben dem Hinderniswert noch weitere Kriterien, die umfassendere Rückschlüsse auf die resultierende Dilation schliessen lassen. Damit wäre vielleicht auch ein Greedyansatz machbar. Anstatt der Flipdistanz könnten auch andere Verfahren die Greedytriangulierung als Ausgangspunkt für eine Verbesserung zur *MDT* nehmen. In vielen Experimenten hat sich zudem gezeigt, das die Greedytriangulierung in vielen Fällen eine kleinere Dilation aufweist als die Delaunaytriangulierung, in vielen Fällen aber auch mit ihr übereinstimmt.

Kapitel 5

Die Java-Applikation

5.1 Einleitung

Im Laufe der Diplomarbeit ist ein Javaprogramm entstanden, welches die drei theoretischen Ansätze praktisch umsetzt. So ist es möglich nicht nur die theoretischen Laufzeiten zu vergleichen sondern auch in der Praxis die Auswirkungen verschiedenster Gegebenheiten zu untersuchen, ob es dabei um spezifische Punktkonstellationen oder Auswirkungen der einzelnen Optimierungen geht, welche sich teilweise schlecht theoretisch abschätzen lassen.

In diesem Kapitel gehe ich als erstes auf das Programm ein und biete einen kurzen Überblick über die Möglichkeiten und Funktionalität des Programmes. Danach werde ich auf verschiedene experimentelle Ergebnisse eingehen um die Auswirkungen, der im letzten Kapitel besprochenen Optimierungen bezüglich der Bitwise Enumeration, in der Praxis mit Leben füllen zu können.

5.2 Erklärung der GUI

Im folgenden gehe ich auf den Aufbau des Applets ein, welches sich in einzelne Panels aufteilt, die logisch in verschiedenen Anwendungsbereiche des Programmes aufgeteilt sind. Allgemein kann man im Zeichenbereich des Programmes mit der linken Maustaste Punkte setzten und mit der rechten Maustaste Punkte löschen. Das Programm verbindet die Punkte automatisch zum T_{max}. Außerdem befindet sich oben im Applet eine Textzeile, welche Informationen über den aktuellen Zustand des Programmes liefert. Also zum Beispiel Mausposition und Anzahl an Triangulierungen im Speicher oder den Status des aktuell laufenden Enumerationsvorganges. Weitere Informationen zur aktuell dargestellten Triangulierung werden als Textnachricht unten links im Zeichenbereich des Applets angegeben.

5.2.1 Auflistung der Panels und Funktionalität

Das Hauptmenu(Abbildung 5.2.1):

Wenn man das Programmm (zum Beispiel über die Html-Seite (ïndex.html")) startet, bietet sich einem ein Anblick wie in Abbildung 5.2.1 zu sehen. In diesem Fall sind mit der Maus schon Punkte gesetzt und ein Suchvorgang wurde

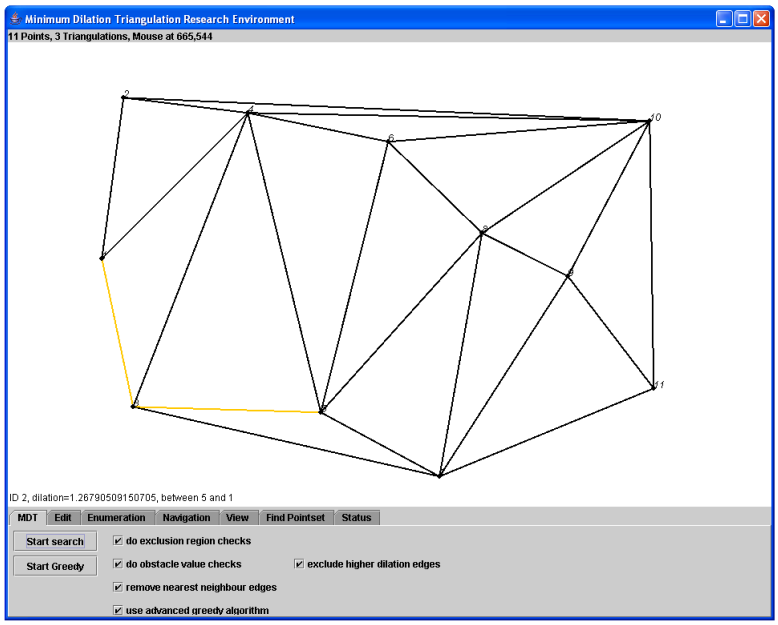


Abbildung 5.1: Main Panel

gestartet. Das Ergebnis ist nun zu sehen. Es folgt nun eine Beschreibung der einzelnen Buttons und Optionen der einzelnen Panels.

- **Start Search:** Startet einen Suchvorgang nach der aktuellen *MDT* mittels des Bitwise-Enumeration Algorithmus. Die zu verwendenden Optimierungen lassen sich über die Optionen in diesem Panel steuern.

- **Start Greedy:** Startet einen Suchvorgang mittels des verbessrten Greedyalgorithmus. Mittels dieser Option kann entschieden werden, ob der erweiterte Ansatz oder der unmodifizierte Ansatz verwendet werden soll.

- **(do exclusion checks)-Option:** Ist diese Option gesetzt, so wird für jede Kante das Exclusion-Region-Kriterium überprüft. Diese Option hat nur Auswirkungen auf den Bitwise-Enumeration-Algorithmus.

- **(do obstacle value checks)-Option:** Ist diese Option gesetzt, so wird für jede Kante der Hindernisswert berechnet. Sollte dieser höher sein, als die Dilation der zu Beginn erstellten Delaunaytriangulierung, so wird die Kante bei der Initialisierung des Algorithmus aus der Kandindatenmenge ausgeschlossen. Diese Option hat nur Auswirkungen auf den Bitwise-Enumeration-Algorithmus.

- **(remove nearest neighbour edges)-Option:** Ist diese Option gesetzt, so werden zu Beginn der Berechnung die Kanten des Minimum Spanning Tree aus dem Kandidatenset entfernt und werden als in der *MDT* enthalten angenommen. Diese Option hat nur Auswirkungen auf den Bitwise-Enumeration-Algorithmus.

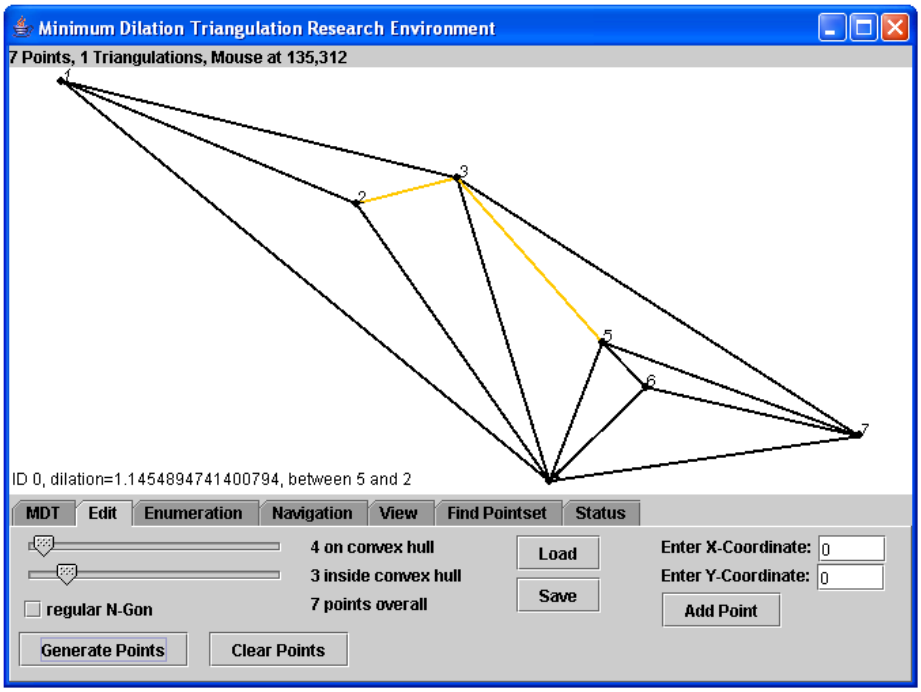


Abbildung 5.2: Edit Panel

- **(exclude higher dilation edges)-Option:** Ist diese Option gesetzt so kann der Enumerationsvorgang frühzeitig abgebrochen werden, da erkannt werden kann, ob es Sinn macht weiter zu suchen. So gesehen ist diese Option das Laufzeitpendant zur *obstacle-value-check Option*, welche in der Initialisierungsphase greift. Hier jedoch, werden Kanten, die sich nachträglich als nicht MDT-fähig herausstellen während der Laufzeit von der Enumeration ausgeschlossen. Mehr darüber findet sich in Kapitel 4.3.2. Diese Option hat nur Auswirkungen auf den Bitwise-Enumeration-Algorithmus.

Generierung und Management von Punktmengen(Abbildung 5.2):

Im *Edit-Panel* können nach bestimmten Vorgaben automatisch Punktmengen generiert, geladen und gespeichert werden. Die Einstellungen in diesem Panel nehmen auch Einfluß auf die Berechnung die das "Find PointsetPanel ermöglichen.

- **Slider:** Mit diesen beiden Slidern lassen sich die Anzahl der Punkte einstellen, die eine neue Punktmenge nach der Generierung haben soll. Mit dem oberen Slider lassen sich die Punkte auf der konvexen Hülle einstellen und mit dem unteren Slider die inneren Punkte.

- **(regular N-Gon)-Option:** Ist diese Option gesetzt, so werden nur Punktmengen generiert, deren äußere Punkte gleichmäßig auf einem imaginären Kreis verteilt werden und es entsteht ein reguläres N-Gon.

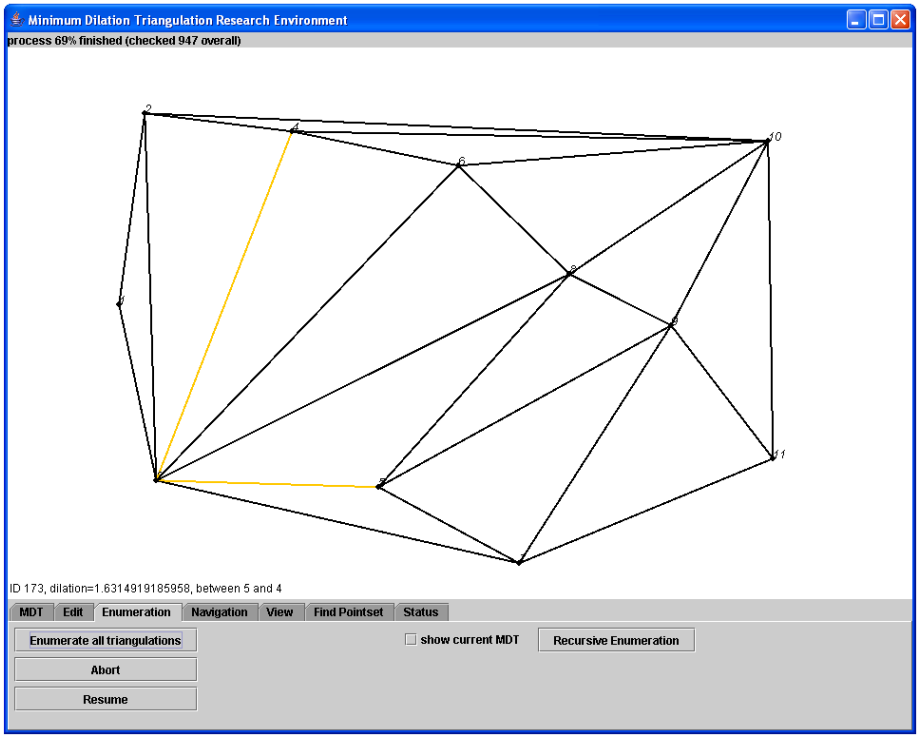


Abbildung 5.3: Enumeration Panel

- **Generate Points :** Mit diesem Button lassen sich nach den obigen Vorgaben automatisch Punktmengen erstellen.
- **Clear Points:** Mit diesem Button löscht man alle aktuellen Punkte.
- **Load:** Mit diesem Button lassen sich einmal abgespeicherte Punktmengen wieder laden. Dabei ist folgendes zu beachten. Es kann nur das programmeigene Format geladen werden, es besteht aus einer bloßen Auflistung der Punktkoordinaten. Auch wird vor dem Ladevorgang einer Datei die aktuelle Punktmenge gelöscht. Möchte man diese behalten, sollte man vorher speichern.
- **Save:** Mit Hilfe dieses Buttons kann die aktuelle Szene im programmeigenen Format gespeichert werden.
- **Add Point:** Ein neuer Punkt wird an der Stelle eingefügt, welche in den beiden Eingabefeldern über dem Button spezifiziert ist.

Enumeration nach Bespamyatnikh(Abbildung 5.3):

Im *Enumeration-Panel* kann die aktuelle Punktmenge nach dem Reverse-Search-Algorithmus von Bespamyatnikh enumeriert werden. Dabei gibt es eine iterative Variante, welche sich jederzeit unterbrechen und wieder aufnehmen läßt, dafür aber etwas langsamer ist und eine rekursive Variante, welche schneller ist, dafür

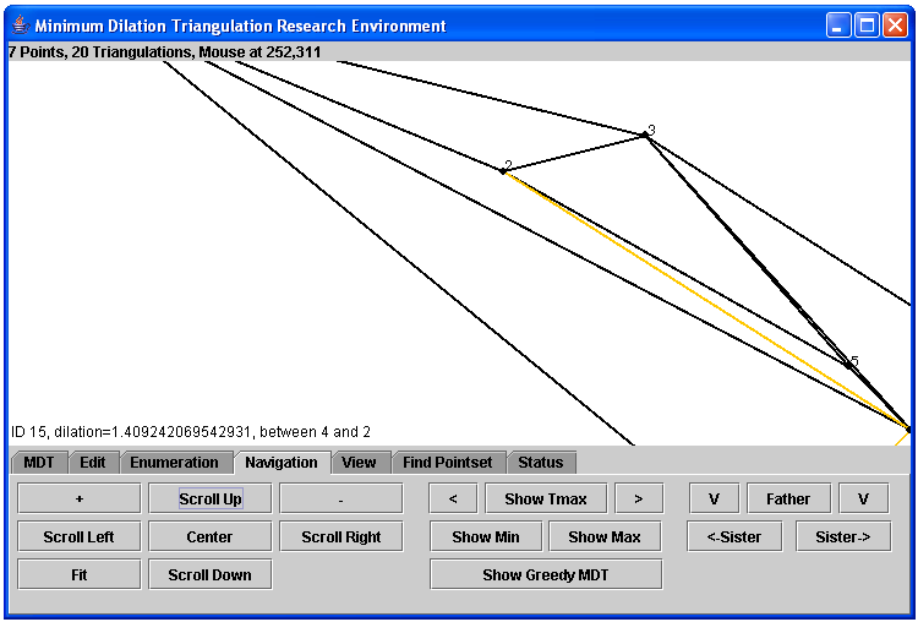


Abbildung 5.4: Navigation Panel

aber auch das gesamte Applet in Anspruch nimmt und sich nicht abbrechen läßt.

- **Enumerate all triangulations:** Hiermit läßt sich der iterative Enumerationsvorgang starten.
- **Abort:** Hiermit läßt sich der aktuelle iterative Enumerationsvorgang abbrechen.
- **Resume:** Mit diesem Button läßt sich ein abgebrochener iterativer Enumerationsvorgang wieder aufnehmen.
- **(Show Current MDT)-Option:** Ist diese Option gesetzt, so wird während des Enumerationsvorganges die aktuelle *MDT* angezeigt und nicht die aktuell bearbeitete. So läßt sich der Fortschritt des Vorganges beobachten.
- **Recursive Enumeration:** Dieser Button startet den rekursiven Enumerationsvorgang, welcher zwar schneller ist, als der iterative, aber sich nicht abbrechen läßt und das gesamte Applet blockiert.

Ansichtsoptionen und Triangulierungstraversierung(Abbildung 5.4):

Das *Navigation-Panel* bietet die Möglichkeit sowohl die Darstellung und den Fokus auf die aktuelle Triangulierung zu ändern als auch die Ansicht zwischen den sich im Speicher befindlichen Triangulierungen zu wechseln. Um zwischen Triangulierungen wechseln zu können, müssen vorher natürlich welche enumeriert worden sein.

- **+ / - Button:** Mittels dieser Buttons läßt sich die aktuell betrachtete Triangulierung heran- oder herauszoomen.
- **Scroll-Up- / Scroll-Down-Button:** Mittels dieser Buttons läßt sich der Blick auf die aktuell betrachtete Triangulierung nach oben oder unten verschieben.
- **Scroll-Left- / Scroll-Right-Button:** Mittels dieser Buttons läßt sich der Blick auf die aktuell betrachtete Triangulierung nach links oder rechts verschieben.
- **Center-Button:** Dieser Button zentriert die aktuelle Triangulierung in der Bildschirmmitte aber behält den aktuellen Zoom bei.
- **Fit-Button:** Dieser Button zentriert die aktuelle Triangulierung in der Bildschirmmitte und verändert den Zoom, so daß die Triangulierung den gesamten Blickbereich ausfüllt.
- **Show-Tmax-Button:** Mit diesem Button schaltet die Ansicht auf die Triangulierung T_{max} um.
- **← Button, Button → :** Mit Hilfe dieser Buttons kann man zwischen benachbarten Triangulierungen hin- und herschalten. Allerdings betrifft das nur die benachbarte Lage im Speicher und hat keine Aussage über wirkliche Nachbarschaftsbeziehungen der Triangulierungen.
- **Show-Min-Button:** Mittels dieses Buttons kann man sich die aktuell gefundene *MDT* anzeigen lassen. ACHTUNG: Wenn man den Enumerationsvorgang abgebrochen hat, wird nur die vorläufig gefundene *MDT*, welche nicht unbedingt die Richtige sein muß, angezeigt.
- **Show-Max-Button:** Mittels dieses Buttons kann man sich die aktuell gefundene Triangulierung mit der größten Dilation anzeigen lassen. ACHTUNG: Wenn man den Enumerationsvorgang abgebrochen hat, wird nur die vorläufig gefundene *MDT*, welche nicht unbedingt die Richtige sein muß, angezeigt.
- **v-Button, Father-Button, v-Button:** Mittels dieser drei Buttons kann man die bisher gefundenen Triangulierungen anhand ihrer Nachbarschaftsverhältnisse traversieren. Der linke Button zeigt das erste Kind, der rechte Button das letzte Kind und der mittlere Button zeigt den Vater der aktuellen Triangulierung an. Diese Buttons machen nur Sinn, wenn man zuvor den Enumerationsalgorithmus nach Bespamyatnikh ausgeführt hat.
- **← Sister-Button, Sister-Button →:** Auch diese Buttons machen nur Sinn nach einer Enumeration nach Bespamyatnikh. Mit diesen Buttons schaltet man zwischen den Geschwistern der aktuellen Triangulierung hin und her.

Aktivierbare Zusatzinformationen(Abbildung 5.5):

Im *View-Panel* läßt sich einstellen welche Eigenschaften gerade im Zeichenbereich dargestellt werden sollen.

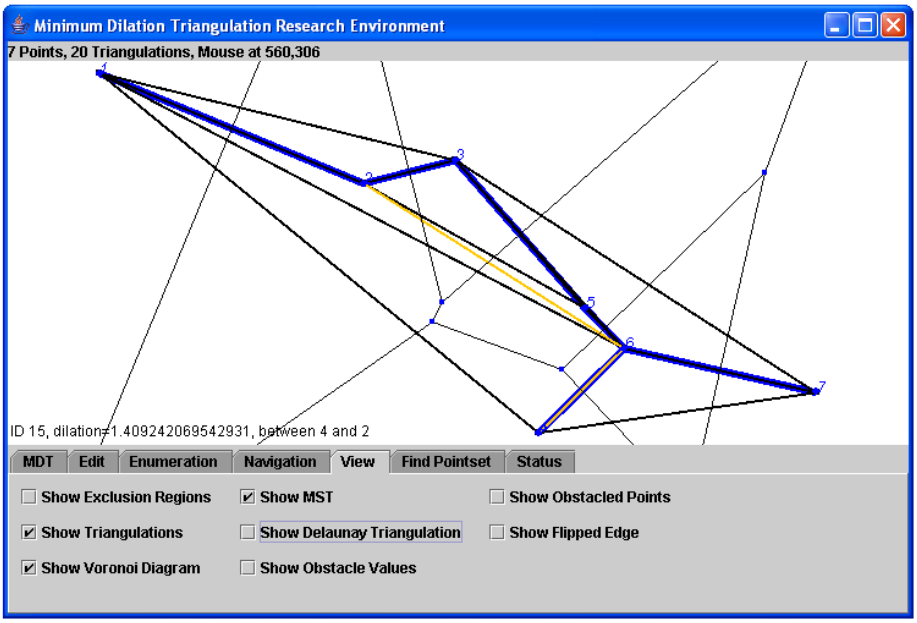


Abbildung 5.5: View Panel

- **(Show Exclusion Regions)-Option:** Ist diese Option ausgewählt, so wird einem immer zu der Kante, die jeweils unter dem Mauszeiger liegt, die zugehörige Exlucion-Region ($\alpha = \frac{3\cos(\frac{\pi}{6})}{2\pi} \approx 0,2067$) angezeigt.
- **(Show Triangulations)-Option:** Ist diese Option ausgewählt, werden über die Punktmenge S die Kanten der aktuell betrachteten Triangulierung gezeichnet.
- **(Show Voronoi Diagramm)-Option:** Ist diese Option ausgewählt so wird im Hintergrund schwach das zu den Punkten gehörigen Voronoidiagramm dargestellt.
- **(Show MST)-Option:** Ist diese Option ausgewählt, so wird ein aktueller Minimum Spanning Tree angezeigt.
- **(Show Delaunay Triangulation)-Option:** Ist diese Option ausgewählt, so wird die aktuelle Delaunaytriangulierung(hellgrau) angezeigt.
- **(Show Obstacle Value)-Option:** Ist diese Option ausgewählt, so wird zu allen Kanten, die einen Hindernisswert ungleich Null haben, dieser neben der Kanten angezeigt.
- **(Show Obstacled Points)-Option:** Ist diese Option ausgewählt, so werden immer zu der Kante unter dem Mauszeiger die Punkte markiert, für welche diese Kante das größte Hindernis darstellt und die mit der Kante den Hindernissert verursachen.
- **(Show Flipped Edge)-Option:** Ist diese Option ausgewählt, so wird immer zu der Kante, die unter dem Mauscursor liegt, die duale Kante oder Flipped-Edge angezeigt.

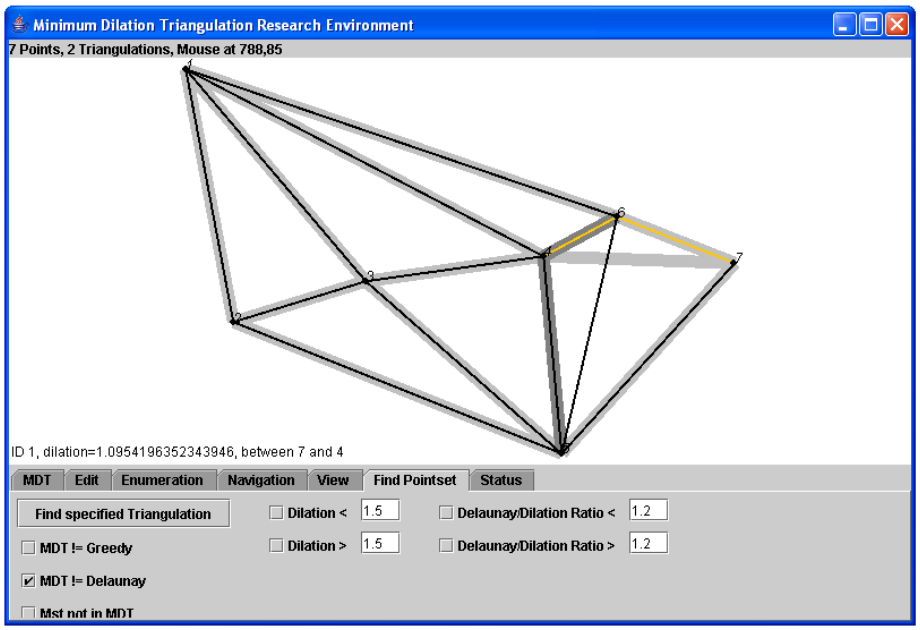


Abbildung 5.6: FindPointset Panel

Suche spezifischer Punktmengen(Abbildung 5.6):

Im *Find-Panel* kann man sich auf die Suche nach Punktmegen begeben, deren *MDT* bestimmte Kriterien erfüllen. Dabei werden die Eckdaten der Punktmenge wie Anzahl der Punkte auf der konvexen Hülle und im Inneren und ob es ein reguläres N-Gon sein soll, aus dem *Edit-Panel* genommen. Der Suchvorgang ist nicht optimiert und von daher kann es bis zu einem Ergebnis etwas länger dauern. Die einzelnen Optionen können auch miteinander kombiniert werden.

- **Find Triangulation Button:** Dieser Button startet den Suchvorgang nach den vorher festgelegten Kriterien.
- **(MDT not Greedy)-Option:** Sucht nach einer *MDT*, die unter anderen unterschiedlich zur *Greedy-MDT* ist.
- **(MDT not Delaunay)-Option:** Sucht nach einer *MDT*, die unter anderen unterschiedlich zur Delaunaytriangulierung ist.
- **(MST not in MDT)-Option:** Sucht nach einer *MDT*, bei der der minimale Spannbaum nicht enthalten ist.
- **(Dilation kleiner / größer...)-Option:** Sucht nach einer *MDT* deren Dilation kleiner beziehungsweise größer ist als der eingegebene Wert.
- **(Dilation / Delaunay Ration kleiner / größer als...)-Option:** Sucht nach einer *MDT* bei der der Quotient aus Dilation der *MDT* durch Dilation der Delaunaytriangulierung den eingegebenen Wert unter- beziehungsweise überschreitet.

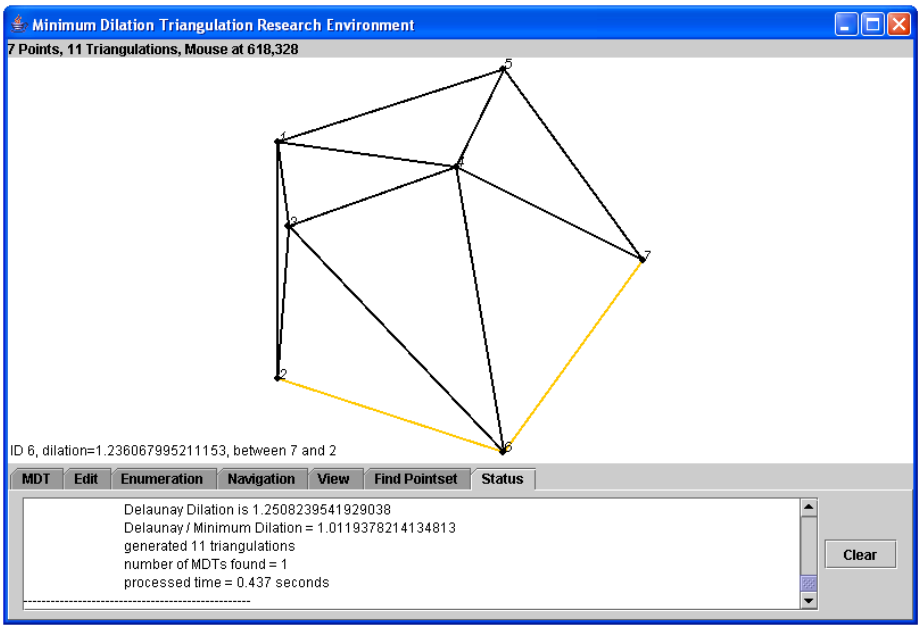


Abbildung 5.7: Status Panel

Backgroundinformationen(Abbildung 5.7):

Im *Status-Panel* befindet sich ein Textfenster, welches ausführlichere Informationen über die vergangenen Operationen liefert. Unter anderem werden hier die Eckdaten der einzelnen Enumerationsvorgänge wie zum Beispiel die Dilationen von *MDT*, MaxDT und Delaunaytriangulierung aufgelistet. Mittels des "ClearButtons läßt sich der Inhalt des Textfensters jederzeit löschen.

5.3 Experimentelle Ergebnisse und offene Fragen

Es folgen nun einige Laufzeitmessungen, die mit dem Javaapplet durchgeführt worden sind. Dazu wurden zu gleichen Punktmengen einerseits der Algorithmus von Bespamyatnikh verwendet und zusätzlich der Bitwise-Enumeration Algorithmus mit verschiedenen Optionen. So läßt sich auch gut erkennen, was für eine Auswirkung die einzelnen Optimierungen in der Praxis haben. In den Tabellen sind die Laufzeiten in Sekunden angegeben und die Algorithmuseinstellungen wie folgt abgekürzt:

- Bespamyatnikh - Der Enumerationsalgorithmus von Bespamyatnikh.
- Bitwise Pure - Bitwise-Enumeration Algorithmus ohne Optimierung.
- Bitwise MST - Bitwise-Enumeration Algorithmus mit Reduzierung um die entfernbaren MST-Kanten.
- Bitwise E.R. - Bitwise-Enumeration Algorithmus mit Exclusion-Region.

- Bitwise O.V. - Bitwise-Enumeration Algorithmus mit Optimierung des Kandidatensets über den Hinderniswert (engl. obstacle value) der Kanten.

- Bitwise Exclude- Bitwise-Enumeration Algorithmus mit der *exclude-higher-dilation-edges*-Optimierung, Siehe dazu auch die Beschreibung des *MDT-Panels*.

- Bitwise All - Bitwise-Enumeration Algorithmus mit allen Optimierungen (MST, Exclusion-Region, Hinderniswert, ExcludeEdges).

Mit der Optimierung von *Bitwise Exclude* ist der Abschnitt 4.3.2 gemeint. Dieser besagt, das die Kanten ihren Hinderniswerten nach aufsteigen im Bitstring repräsentiert werden und die Kanten, mit dem kleinsten Hinderniswert als erste enumeriert werden. Die obere Dilationsschranke (zu Anfangs auf die obere Dilationsschranke der Delaunaytriangulierung gesetzt) paßt sich während des Enumerationsvorganges der bis dato kleinst gefundenen Dilation an. So kann, wenn eine Kante im Bitstring einen Hinderniswert höher als die obere Dilationsschranke hat, der Enumerationsvorgang früher beendet werden, da in diesem Fall alle weiteren Triangulierungen eine höhere Dilation hätten und die *MDT* demnach schon gefunden wurde.

Laufzeitmessung für Punktmengen ohne innere Punkte

Algorithmus	Pointset 5.8	Pointset 5.9	Pointset 5.10	Pointset 5.11
Bespamyatnikh	2,040s	7,864s	31,882s	143,66s
Bitwise Pure	1,593s	5,969s	19,547s	115,5s
Bitwise MST	1,492s	5,969s	19,547s	115,5s
Bitwise E.R.	0,953s	5,969s	19,547s	115,5s
Bitwise O.V.	0,031s	0,313s	0,313s	0,98s
Bitwise Exclude	0,078s	0,234s	0,329s	1,02s
Bitwise All	0,047s	0,312s	0,313s	0,95s

Hier läßt sich erstens erkennen, wieviel schneller die Bitwise-Enumeration ist, wenn es um die MDT-Findung geht. Im Extremfall konnte hier die MDT in knapp einer Sekunde gefunden werden, wohin gegen die vollständige Enumeration aller Triangulierungen mittels des Algorithmus von Bespamyatnikh oder mittels *Bitwise Pure* knapp 2 Minuten braucht. Worauf ich noch hindeuten möchte, ist, daß sich die Laufzeit von *Bitwise Pure* und *Bitwise MST* drastisch dem Algorithmus von Bespamyatnikh annähert und wie man in den folgenden Tabellen erkennen kann, sogar bald überholt und scheinbar exponentiell ansteigt. Teilweise war sie so groß, daß der Vorgang von mir abgebrochen wurde und daher in diesem Fällen keine Zeitdauer erfaßt wurde.

Laufzeitmessung für Punktmengen mit nur 3 äußeren Punkten

Algorithmus	Pointset 5.12	Pointset 5.13	Pointset 5.14	Pointset 5.15
Bespamyatnikh	1,66s	5,86s	17,91s	421,5s
Bitwise Pure	1,38s	5,17s	1616,24s	-
Bitwise MST	1,33s	4,99s	245,35s	-
Bitwise E.R.	1,24s	1,94s	2,3s	25,02s
Bitwise O.V.	0,24s	1,08s	0,36s	1,2s
Bitwise Exclude	0,03s	0,56s	0,38s	1,67s
Bitwise All	0,03s	0,58s	0,36s	1,25s

Das die Laufzeit von *Bitwise Pure & MST* so drastisch ansteigt läßt sich dadurch erklären, das alle k-Bitkonfigurationen getestet werden müßen, da die Menge der MDT-Kandidatenkanten nicht reduziert wird und daher bei ansteigender Punktzahl n auch die Menge der k-Bitkonfigurationen explosionsartig ansteigt ($\binom{n}{k}$). Zusätzlich liegt bei diesen Punktmengen die Besonderheit vor, daß die konvexe Hülle jeder Punktmenge nur aus 3 Punkten besteht. Aus Kapitel 2.2.1 folgt der Schluß , daß Triangulierungen von Punktmengen mit kleinerer konvexer Hülle mehr Kanten haben als Triangulierungen von Punktmengen gleicher Punktzahl aber größerer konvexer Hülle. Dadurch läßt sich auch unter anderem der erhöhte Zeitbedarf der Punktmengen 5.12, 5.13, 5.14, 5.15 gegenüber den Punktmengen 5.8, 5.9, 5.10, 5.11 erklären.

Laufzeitmessung für Punktmengen mit ausgeglichenem Verhältnis zwischen inneren und äußeren Punkten

Die Punktmengen der letzten beiden Laufzeitbetrachtungen hatten gegensätzlichen Charakter, da sowohl Punktmengen mit minimaler konvexer Hülle (3 Punkte auf der konvexen Hülle) als auch maximaler konvexer Hülle (keine inneren Punkte) vorlagen. Die nun folgenden Punktemengen gehen den Mittelweg und haben ein ausgeglichenes Verhältnis zwischen inneren und äußeren Punkten.

Algorithmus	Pointset 5.16	Pointset 5.17	Pointset 5.18	Pointset 5.19
Bespamyatnikh	0,78s	5,04s	32,67s	97,12s
Bitwise Pure	0,63s	5,2s	419,79s	-
Bitwise MST	0,12s	0,25s	323,71s	-
Bitwise E.R.	0,38s	1,58s	4,59s	5,74s
Bitwise O.V.	0,03s	0,03s	0,05s	0,28s
Bitwise Exclude	0,03s	0,05s	0,05s	0,28s
Bitwise All	0,03s	0,03s	0,06s	0,28s

Auch die Auswirkungen der einzelnen Optimierungen lassen sich jetzt besser abschätzen. Der Hinderniswert einer Kante hat nicht nur eine größere Aussagekraft über eine Kante als die Exclusion-Region, sondern auch eine wesentlich bessere Laufzeitbeschleunigung. Dabei konkuriert *Bitwise O.V.* mit *Bitwise Exclude*, welches, wie in Kapitel 4.3.2 beschrieben, versucht durch eine Anpassung der oberen Dilationsschranke während der Laufzeit, den Enumerationsvorgang so früh wie möglich mit einem korrekten Ergebnis zu terminieren und nicht unnötige Triangulierungen mit betrachten zu müßen. Dazu muß aber auch gesagt werden, daß die Abschätzung über den Abbruch der MDT-Suche auch über den Hinderniswert läuft. Auf jedenfall bestätigen auch die Laufzeitergebnisse die Vorteile der Bitwise-Enumeration gegenüber dem Ansatz von Bespa-

myatnikh. Offen ist allerdings, ob es für die Laufzeit des Bitwise-Enumeration Algorithmus eine bessere obere Schranke gibt, als die Abschätzung über den Binomialkoeffizienten. Auch könnte das Verfahren wesentlich beschleunigt werden, wenn man die k-Bitkonfigurationen, welche keine Triangulierungen darstellen, aussparen könnte, ohne sie umständlich auf Kollisionen überprüfen zu müssen. Ansonsten bleibt die Frage, ob sich die MDT in polynomieller Zeit finden läßt, immer noch ungeklärt, aber mittels der in dieser Diplomarbeit vorgestellten Verfahren lassen sich für kleine Punktmengen in akzeptabler Zeit korrekte Ergebnisse finden. Eine weitere interessante Frage, in wie weit sich das Prinzip der Exclusion-Region und des Hinderniswertes auf höher dimensionale Räume übertragen oder erweitern lassen. Genauso wäre es interessant, wie in diesen Fällen der Bitwise-Enumeration Algorithmus auf höher dimensionale Räume übertragbar und anwendbar wäre. Es bleibt auf jedenfall noch viel zu tun und es werden sicherlich noch weiter spannende Aspekte der Dilation untersucht und mehr Licht ins Dunkel gebracht.

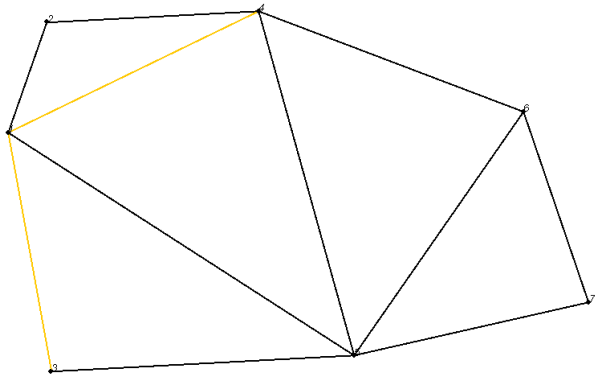


Abbildung 5.8: 7 Punkte (keine Inneren Punkte vorhanden.)

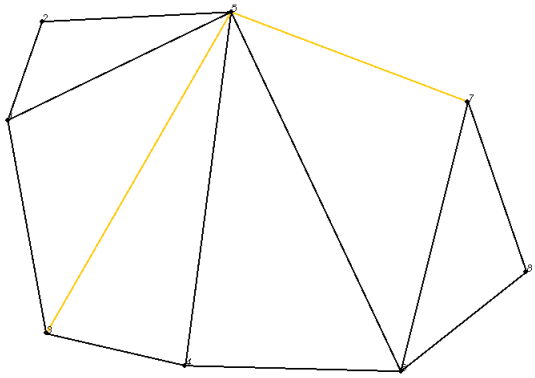


Abbildung 5.9: 8 Punkte (keine Inneren Punkte vorhanden.)

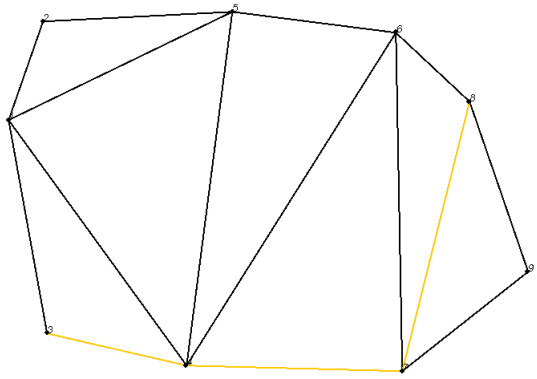


Abbildung 5.10: 9 Punkte (keine Inneren Punkte vorhanden.)

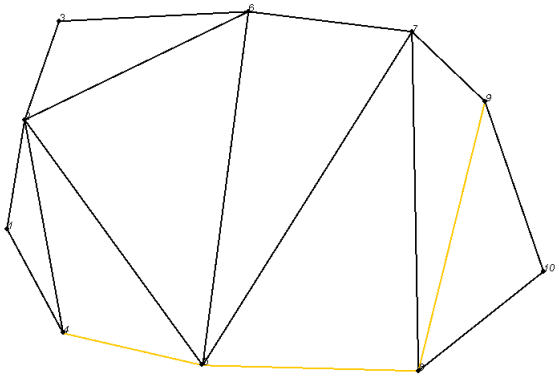


Abbildung 5.11: 10 Punkte (keine Inneren Punkte vorhanden.)

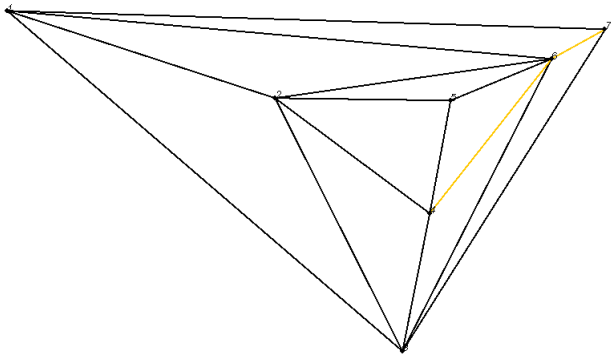


Abbildung 5.12: 7 Punkte bestehend aus 3 äußeren und 4 inneren Punkten

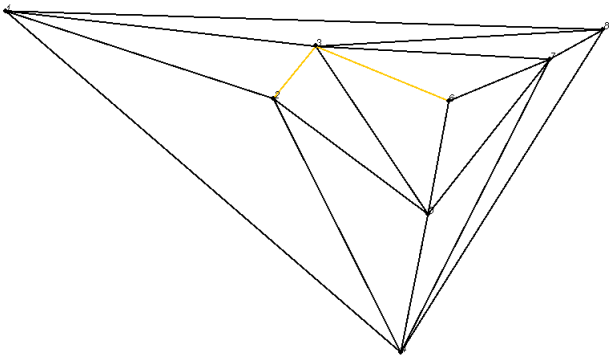


Abbildung 5.13: 8 Punkte bestehend aus 3 äußeren und 5 inneren Punkten

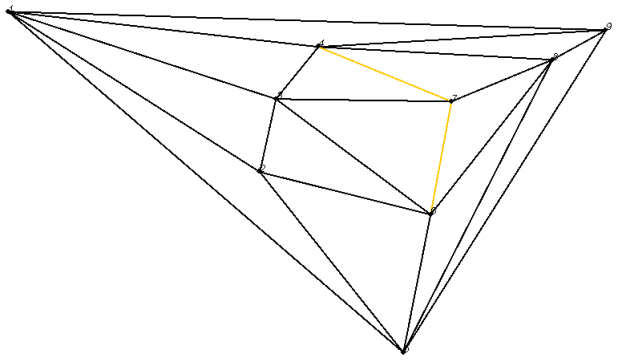


Abbildung 5.14: 9 Punkte bestehend aus 3 äußeren und 6 inneren Punkten

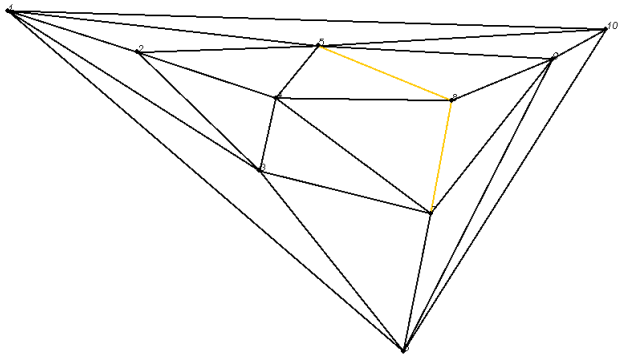


Abbildung 5.15: 10 Punkte bestehend aus 3 äußeren Punkten und 7 inneren Punkten

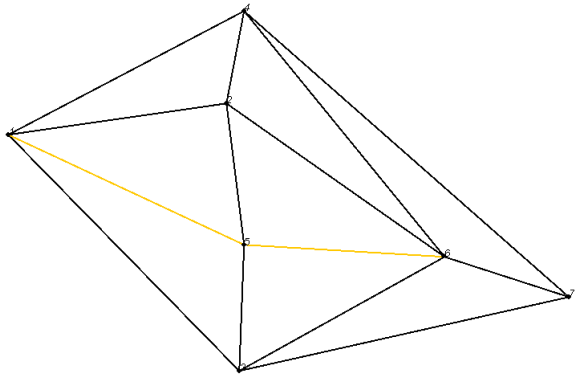


Abbildung 5.16: 7 Punkte bestehend aus 4 äußeren und 3 inneren Punkten

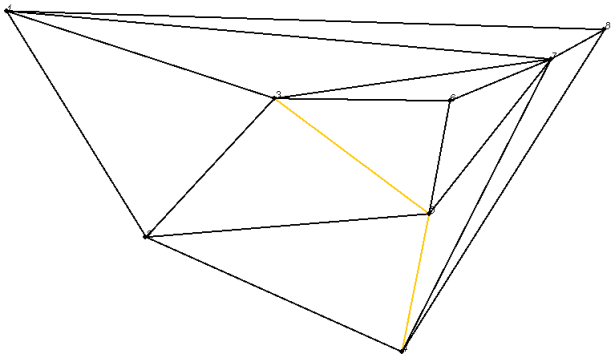


Abbildung 5.17: 8 Punkte bestehend aus 4 äußeren und 4 inneren Punkten

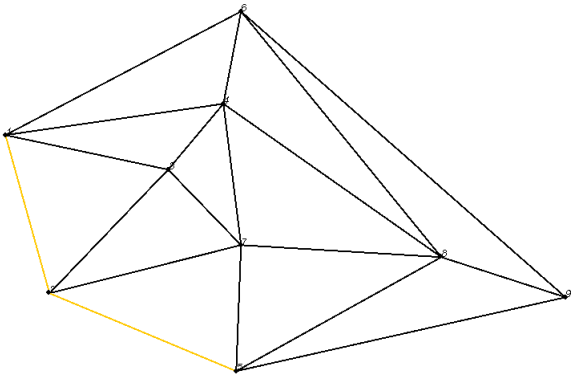


Abbildung 5.18: 9 Punkte bestehend aus 5 äußeren und 4 inneren Punkten

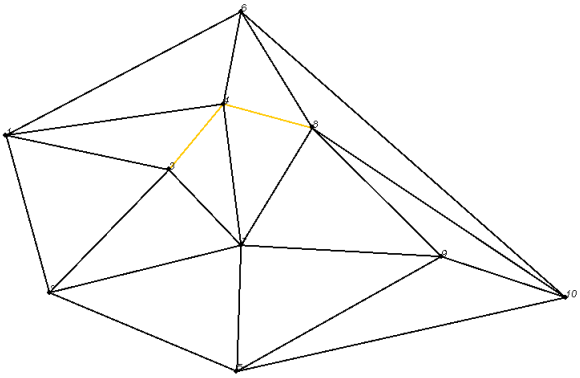


Abbildung 5.19: 10 Punkte bestehend aus 5 äußeren und 5 inneren Punkten

Literaturverzeichnis

[AF96] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Appl. Math.*, 65:21–46, 1996.

[Bes00] Sergei Bespamyatnikh. An efficient algorithm for enumeration of triangulations. 10th Annual Fall Workshop on Computational Geometry., 2000.

[Che89] L. P. Chew. There are planar graphs almost as good as the complete graph. *J. Comput. Syst. Sci.*, 39:205–219, 1989.

[Dic05] T. Dickmeiß. *Zur graphtheoretischen Dilation der Delaunay-Triangulation und verwandter Graphen.* PhD thesis, Institut für Informatik, Abteilung 1, Universität Bonn, 2005.

[EBGK04] Annette Ebbers-Baumann, Ansgar Grüne, and Rolf Klein. On the geometric dilation of finite point sets. *accepted by Algorithmica*, 2004. accepted.

[Epp] D. Eppstein. The geometry junkyard: Dilation-free planar graphs. *http://www.ics.uci.edu/ eppstein/junkyard/dilation-free.*

[Epp00] David Eppstein. Spanning trees and spanners. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[For87] S. J. Fortune. A note on Delaunay diagonal flips. Manuscript, AT&T Bell Lab., Murray Hill, NJ, 1987.

[Kle98] Rolf Klein. *Kurs 1840: Algorithmische Geometrie.* FernUniversität Hagen, Fachbereich Informatik, 1998.

[KM] C. Knauer and M. Mulzer. An exclusion region for minimum dilation triangulation. *EWCG 2005 Eindhoven.*

Index