Diplomarbeit

Algorithms and Implementations for Disk Induced Flows

Michel Stöcklin

Professor:
Emo Welzl

Betreuer:
Joachim Giesen, Matthias John

28. Februar 2002

Inhaltsverzeichnis

Kapitel 1

Motivation

Hauptziel meiner Diplomarbeit war die Erzeugung eines Programmes zur Zerlegung der Ebene in Zellen, von denen jede einem Minimum bzw. Maximum eines durch die Menge der gewichteten Punkte, die als Eingabe dient, definierten dynamischen Systems, genannt Fluss, zugeordnet werden kann.
Eine praktische Bedeutung einer solchen Zerlegung in drei Dimensionen könnte die Zerlegung eines Makromoleküles in einzelne Teile sein, wenn die Punkte der Eingabe den Positionen der Atome und die Gewichte, die zu den Punkten gehören, den jeweiligen Atomradien entsprechen. Somit liesse sich eine Approximation der Form des Makromoleküles auf effiziente Weise berechnen. Die Form eines Makromoleküles ist deshalb von Interesse, weil sie seine Funktionalität bestimmt, die möglichen Interaktionen mit anderen Makromolekülen.
Um die gesteckten Ziele zu erreichen, musste ich mich zuerst in die von Joachim Giesen und Matthias John entwickelte Theorie der Disk induzierten Flusssysteme einlesen. Die im Pseudocode vorliegenden Algorithmen habe ich in praktisch implementierbare Algorithmen übersetzt und diese dann in mein Programm integriert.

Kapitel 2

Theoretische Grundlagen

Dieses Kapitel enthält die theoretischen Grundlagen, auf die die späteren Algorithmen aufbauen.
Eine Delaunay-Triangulierung auf gewichteten Punkten bildet die Basis für weitere Berechnungen. Eine solche Triangulierung wird auch reguläre (Delaunay-)Triangulierung genannt. Ich werde zuerst auf die gewichtete Delaunay-Triangulierung und deren duale Darstellung, den gewichteten Voronoi-Komplex, der auch als Power-Diagramm bezeichnet wird, eingehen. Später wende ich mich den kritischen Punkten und dem Fluss zu.

2.1 Disks, Diagramme und kritische Punkte

Disk: Disk ist die Bezeichnung für die bereits erwähnten gewichteten Punkte.
Eine Disk ist ein Paar $(z, r) \in \mathbb{R}^3$, wobei $z \in \mathbb{R}^2$ der Mittelpunkt der Disk und $r \in \mathbb{R}$ der Radius ist. Der Bereich, der von der Disk eingenommen wird, ist

$$\{x \in \mathbb{R}^2 : \|x - z\| \leq r\}.$$

Power-Abstand: Der Power-Abstand eines Punktes $x \in \mathbb{R}^2$ zu einer Disk (z, r) ist $\pi(x) = \|x - z\|^2 - r^2$.

Power-Bisektor: Gegeben seien zwei Disks in der Ebene. Der Power-Bisektor dieser beiden Disks ist die Menge aller Punkte, die den selben

Power-Abstand zu beiden Disks haben. Der Power-Bisektor ist eine Gerade, orthogonal zum Liniensegment, welches die Mittelpunkte der beiden Disks verbindet.

Gewichtetes Voronoi-Diagramm: Gegeben sei eine Menge B von Disks. Die Voronoi-Zelle einer Disk b_i ist gegeben durch

$$V_i = \{x \in \mathbb{R}^2 : \forall b_j \in B,\ \pi_i(x) \leq \pi_j(x)\}.$$

Die Mengen V_i sind entweder konvexe Polygone oder die leere Menge, da Punkte die denselben Power-Abstand zu zwei Disks haben auf einer Geraden zu liegen kommen. Abgeschlossene Liniensegmente, die Teil von zwei oder mehr Voronoi-Zellen sind, werden Voronoi-Kanten genannt. Punkte, die von drei oder mehr Voronoi-Zellen geteilt werden, nennt man Voronoi-Knoten. Die Bezeichnung Voronoi-Objekt kann sowohl für eine Voronoi-Zelle, eine Voronoi-Kante als auch für einen Voronoi-Knoten verwendet werden. Das Voronoi-Diagramm der Menge B ist die Gesamtheit der Voronoi-Zellen, Voronoi-Kanten und Voronoi-Knoten.

Gewichtetes Delaunay-Diagramm: Das Delaunay-Diagramm einer gegebenen Menge B von Disks ist zum Voronoi-Diagramm der selben Menge B dual. Die konvexe Hülle von drei oder mehr Mittelpunkten definiert eine Delaunay-Zelle, falls die Schnittmenge der zugehörigen Voronoi-Zellen nicht leer ist und falls es keine grössere Menge von Mittelpunkten gibt, die die vorherigen Mittelpunkte enthält und für die ebenfalls gilt, dass die Schnittmenge der zugehörigen Voronoi-Zellen nicht leer ist. Analog dazu bildet die Verbindung zwischen zwei Mittelpunkten eine Delaunay-Kante, falls die Schnittmenge der zugehörigen Voronoi-Zellen nicht leer ist. Jeder Mittelpunkt wird Delaunay-Knoten genannt. Die Bezeichnung Delaunay-Objekt kann sowohl für eine Delaunay-Zelle, eine Delaunay-Kante als auch für einen Delaunay-Knoten verwendet werden. Das Delaunay-Diagramm definiert eine Zerlegung der konvexen Hülle aller Mittelpunkte. Falls die Disks sich in allgemeiner Lage befinden, so ist die Zerlegung eine Triangulierung.

Es kann hilfreich sein, eine zusätzliche Disk mit Mittelpunkt im Unendlichen und Radius 0 einzuführen. Das führt zu einem Voronoi-Knoten im Unendlichen am Ende jeder unbegrenzten Voronoi-Kante.

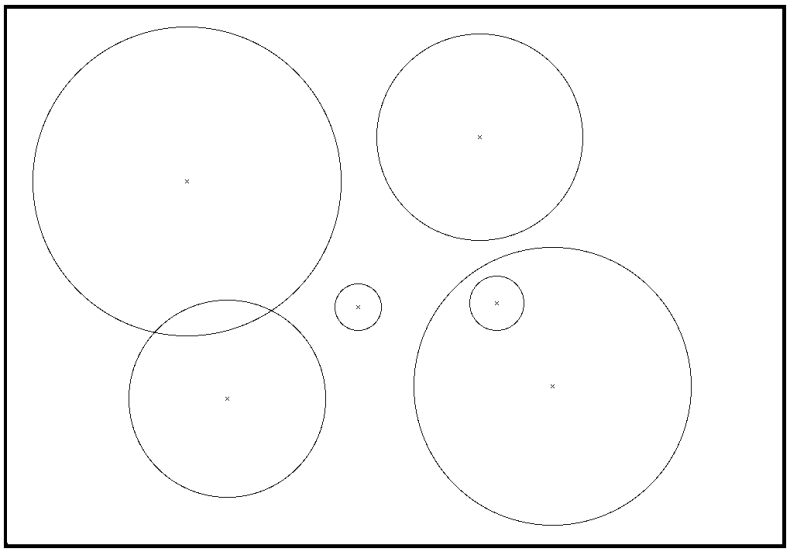


Abbildung 2.1: Eine Menge von Disks in der Ebene.

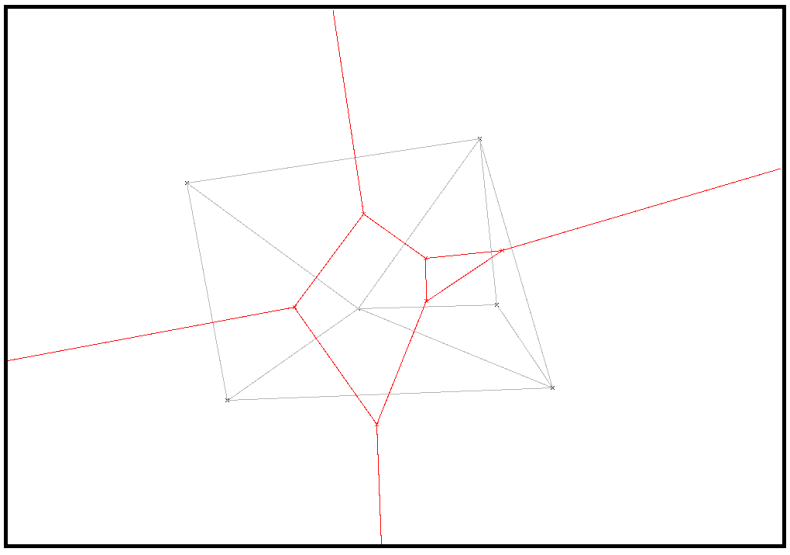


Abbildung 2.2: Die zur obigen Menge von Disks gehörenden Voronoi- und Delaunay-Diagramme.

Power-Höhenfunktion: Gegeben sei eine Menge B von Disks in $\mathbb{R}^2$. Die Power-Höhenfunktion ist gegeben durch

$$h(x) = \min\{\pi_i(x) : b_i \in B\}.$$

Die Funktion h ist überall glatt ausser an den Punkten, die den selben Power-Abstand zu zwei oder mehr Disks aufweisen, d.h. ausser auf Punkten, die auf der Grenze zwischen verschiedenen Voronoi-Zellen liegen.

Reguläre und kritische Punkte: Eine glatte Funktion auf einer glatten Mannigfaltigkeit M wird regulär oder nicht kritisch an einem Punkt x genannt, falls der Gradient an dieser Stelle nicht verschwindet. Alle anderen Punkte nennt man kritisch.
Obwohl die Power-Höhenfunktion nicht glatt ist, lässt sich die Terminologie der kritischen Punkte dennoch auf sinnvolle Weise verwenden.
Die kritischen Punkte der Power-Höhenfunktion können mit einen Index versehen werden. Kritische Punkte mit Index 0, 1 bzw. 2 nennt man Minimum, Sattelpunkt bzw. Maximum.

Lemma 1: Gegeben sei eine Menge B von Disks. Die kritischen Punkte der Power-Höhenfunktion h sind die Schnittpunkte von Voronoi-Objekten V und deren dualen Delaunay-Objekten σ. Der Index der kritischen Punkte entspricht der Dimension von σ.

In degenerierten Fällen ist es möglich, dass kritische Punkte zusammenfallen, d.h. dass beispielsweise ein Minimum auch ein Sattelpunkt sein könnte. Im Folgenden werden solche degenerierten Fälle ausgeschlossen.

Korollar 1: Die Power-Höhenfunktion die zu einer Menge B von Disks gehört und die Power-Höhenfunktion die zur dualen Menge C von Disks gehört haben dieselben kritischen Punkte. Der Index eines kritischen Punktes der dualen Menge von Disks beträgt 2 minus dem Index desselben Punktes der primalen Menge von Disks, d.h. Minima und Maxima werden vertauscht.

2.2 Advancing Dynamical Systems

Advancing Dynamical System: Gegeben sei eine Mannigfaltigkeit M. Eine stetige Abbildung

$$\Phi : [0, \infty) \times M \to M$$

wird Advancing Dynamical System oder Fluss auf M genannt, falls für alle $x \in M$ und alle $s, t \in [0, \infty)$ die folgenden Eigenschaften gelten:

$$\Phi(0, x) = x.$$

$$(\Phi(t, \Phi(s, x)) = \Phi(t + s, x).$$

Gegeben sei $x \in M$. Die Kurve

$$\Phi_x : [0, \infty) \to M,\ t \mapsto \Phi(t, x)$$

heisst Flusslinie oder Orbit von x. Ein Punkt $x \in M$ heisst Fixpunkt von Φ, falls $\Phi(t, x) = x$ für alle $t \geq 0$.

Stabile und instabile Mannigfaltigkeiten: Gegeben sei ein Advancing Dynamical System Φ auf einer Mannigfaltigkeit M. Die stabile Mannigfaltigkeit $S(x)$ eines kritischen Punktes $x \in M$ enthält alle Punkte, die asymptotisch in x hineinfliessen, d.h.

$$S(x) = \{y \in M : \lim_{t \to \infty} \Phi_y(t) = x\}.$$

Gegeben sei eine Umgebung U von x und $V(U)$ die Menge aller Punkte, die auf einem Orbit liegen, welcher seinen Ursprung in U hat, d.h.

$$V(U) = \{y \in M : \exists z \in U,\ t \in [0, \infty) \text{ so dass } \Phi_z(t) = y \text{ oder } \lim_{t \to \infty} \Phi_z(t) = y\}.$$

Dann ist die instabile Mannigfaltigkeit $U(x)$ gegeben als Schnittmenge aller solcher Mengen $V(U)$.

Treiber: Da das Voronoi-Diagramm von B eine Zerlegung der Ebene ist, liegt jeder Punkt $x \in \mathbb{R}^2$ auf irgendeinem Voronoi-Objekt. Gegeben sei ein Punkt $x \in \mathbb{R}^2$ auf irgendeinem Voronoi-Objekt. Sei V das Voronoi-Objekt geringster Dimension, auf dem x liegt. Sei σ das duale Delaunay-Objekt von

V und $y = \text{argmin}_{z \in \sigma} \|x - z\|$, also derjenige Punkt auf σ der x am nächsten liegt. Dieser Punkt y wird Treiber von x genannt.

Disk induzierter Fluss: Gegeben sei eine Menge B von Disks. Der induzierte Fluss Φ ist wie folgt gegeben: Sei V das Voronoi-Objekt geringster Dimension, auf dem x liegt. Angenommen x liege auf dem Schnittpunkt von V und dessen dualem Delaunay-Objekt. In diesem Fall gilt:

$$\Phi(t, x) = x, \ t \in [0, \infty).$$

Anderenfalls sei y der Treiber von x. Falls y auf einem Delaunay-Objekt geringerer Dimension als σ liegen sollte, dann ersetzt man V durch das duale Voronoi-Objekt des Delaunay-Objektes geringster Dimension. Sei R der Strahl mit Ursprung in x und Richtung $x - y$. Sei z der erste Punkt auf R für den $\text{argmin}_{z \in \tau} \|x - z\|$ sich von y unterscheidet, wobei τ das duale Delaunay-Objekt des Voronoi Objektes geringster Dimension ist, auf dem z liegt. Ein solcher Punkt z muss in $\mathbb{R}^2$ nicht existieren. In diesem Fall liegt z im Unendlichen. Der induzierte Fluss ist:

$$\Phi(t, x) = x + t\frac{x - y}{\|x - y\|}, \ t \in [0, \|z - x\|].$$

Für $t > \|z - x\|$ ist der induzierte Fluss gegeben durch die definierten Eigenschaften des Flusses:

$$\Phi(t, x) = \Phi(t - \|z - x\| + \|z - x\|, x) = \Phi(t - \|z - x\|, \Phi(\|z - x\|, x)).$$

Wegen der gewählten Parametrisierung der Orbits Φ_x muss man die Definition der stabilen Mannigfaltigkeiten der Situation anpassen. Die stabile Mannigfaltigkeit $S(x)$ eines kritischen Punktes $x \in \mathbb{R}^2$ des Disk induzierten Flusses Φ enthält alle Punkte, deren Orbit x beinhaltet, d.h.

$$S(x) = \{y \in \mathbb{R}^2 : \exists t \in [0, \infty) \text{ so dass } \Phi_y(t) = x\}.$$

Beobachtung 1: Es gilt Folgendes:
Die Fixpunkte von Φ sind die kritischen Punkte der Power-Höhenfunktion.
Die Orbits von Φ sind stückweise lineare Kurven.

Lemma 2: Der Fluss Φ hat keine geschlossenen Orbits.

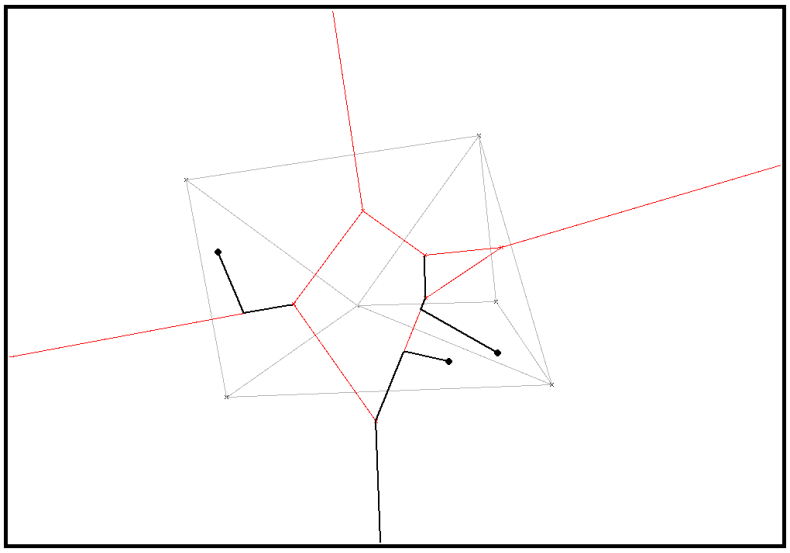


Abbildung 2.3: Einige Flusslinien des von der gegeben Menge von Disks induzierten Flusses.

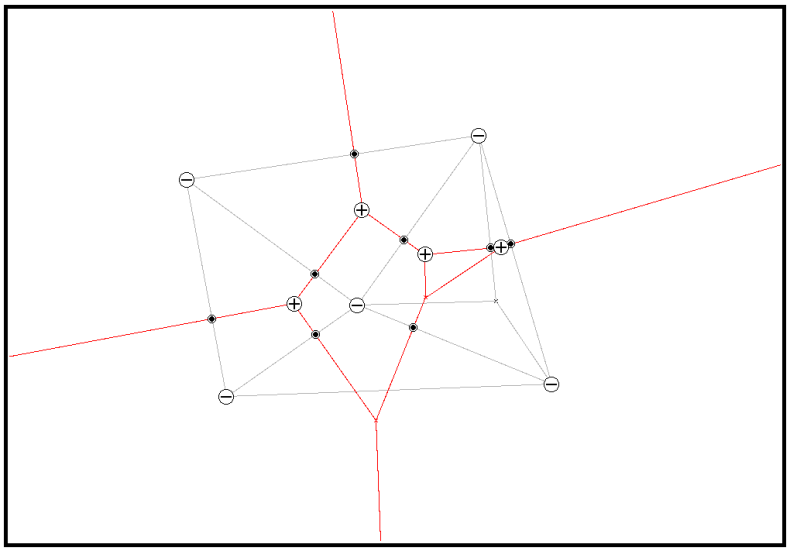


Abbildung 2.4: Die kritischen Punkte des obigen Flusses.

Die folgenden technischen Konstrukte sollen lediglich der Vereinfachung dienen: Es wird eine Ordnung auf den Indizes der Disks angenommen, so dass der Disk Radius eine monoton wachsende Funktion des Disk Index ist. Sei $\mathcal{V}$ die Menge aller Voronoi-Objekte des Voronoi-Diagrammes. Mit jedem $x \in \mathbb{R}^2$ wird eine Funktion

$$V_x : [0, \infty) \to \mathcal{V}$$

assoziiert, die jedem Zeitpunkt t das Voronoi-Objekt geringster Dimension zuordnet, das $\Phi_x(t)$ enthält.

Lemma 3: Gegeben sei die Menge M der Minima von Φ. Die Menge

$$\mathcal{M} = \{U(m) : m \in M\}$$

bedeckt ganz $\mathbb{R}^2$, d.h. jedes $x \in \mathbb{R}^2$ ist in mindestens einer stabilen Mannigfaltigkeit $U(m)$ eines Minimums m von Φ enthalten.

Etwas entsprechendes lässt sich auch für Maxima und Sattelpunkte beweisen. Dazu wird folgendes Lemma benötigt.

Lemma 4: Gegeben seien $t' \in [0, \infty)$ und $x \in \mathbb{R}^2$, so dass jede genügend klein gewählte Umgebung von t' durch die Funktion V_x exakt drei Voronoi-Objekte zugeordnet bekommt. In diesem Fall ist $V_x(t')$ entweder eine Voronoi-Kante oder ein Voronoi-Knoten.
Falls es sich bei $V_x(t' \pm \varepsilon)$ um Voronoi-Zellen handelt, dann liegen die dualen Delaunay-Knoten von $V_x(t' - \varepsilon)$ und $V_x(t' + \varepsilon)$ beide auf derjenigen Seite des Power-Bisektors, die zum Delaunay-Knoten gehört, der der Disk mit dem grösseren Index entspricht.

Interessant sind jene Punkte $x \in \mathbb{R}^2$, die nicht ins Unendliche fliessen.

Lemma 5: Gegeben sei K die Menge aller Punkte, die nicht ins Unendliche fliessen, d.h.

$$K = \mathbb{R}^2 - \{x \in \mathbb{R}^2 : \forall n \in \mathbb{N}\ \exists t_n \geq 0 \text{ so dass } \forall t > t_n\ \|\Phi_x(t) - 0\| > n\}.$$

Gegeben sei die Menge S aller Sattelpunkte, die Menge M aller Maxima sowie die Menge M' aller Minima von Φ. Die Menge

$$\mathcal{M} = \{S(y) : y \in S \cup M\}$$

bedeckt $K - M'$, d.h. jedes $x \in K$, das kein Minimum ist, ist in genau einer stabilen Mannigfaltigkeit $S(y)$ eines Sattels oder Maximums y von Φ enthalten.

Lemma 6: Gegeben sei ein Sattelpunkt s von Φ. Die instabile Mannigfaltigkeit $U(s)$ von s ist eine stückweise lineare Kurve. Gleiches gilt im Allgemeinen nicht für die stabilen Mannigfaltigkeiten von Sattelpunkten. D.h. die stabilen Mannigfaltigkeiten von Sattelpunkten müssen keine Kurven sein, sondern können eine offene Untermenge von $\mathbb{R}^2$ beinhalten.

Die instabilen Mannigfaltigkeiten verschiedener Sattelpunkte können sich entweder in einem Punkt treffen oder sie können bis ins Unendliche führen. Die einzigen Punkte, wo sich zwei solche instabile Mannigfaltikeiten treffen können, müssen auf den Grenzen von Voronoi-Zellen liegen, d.h. im Inneren von Voronoi-Kanten oder auf Voronoi-Knoten. Wenn ein Punkt auf dem sich zwei oder mehr instabile Mannigfaltigkeiten treffen kein Maximum oder Sattelpunkt ist, dann nennt man ihn einen Steiner-Knoten.

Im Gegensatz zu instabilen Mannigfaltigkeiten, müssen stabile Mannigfaltigkeiten von Sattelpunkten keine eindimensionalen Kurven sein. Auf Situationen, in denen eine stabile Mannigfaltigkeit degenerieren kann, wird später eingegangen. Es werden ausgedünnte stabile Mannigfaltigkeiten eingeführt, die immer eindimensional sind, um solche degenerierten Fälle zu behandeln. Diese ausgedünnten stabilen Mannigfaltigkeiten werden später verwendet, um das sogenannte Max-Diagramm einer Menge von Disks zu erstellen.

Lemma 7: Gegeben sei ein Sattelpunkt s von Φ. Falls die stabile Mannigfaltigkeit $S(s)$ von s keinen Voronoi-Knoten enthält, so ist closure($S(s)$) eine stückweise lineare Kurve.

Im folgenden wird die degenerierte Situation betrachtet, in der die stabile Mannigfaltigkeit eines Sattelpunktes einen Voronoi-Knoten enthält.

Lemma 8: Gegeben sei ein Voronoi-Knoten v, der nicht in seiner dualen Delaunay-Zelle enthalten ist, d.h. ein Voronoi-Knoten, der kein Maximum ist. Gegeben sei weiter eine ausreichend kleine Umgebung U von v und

$$S(v) = \{x \in U : v \in \Phi_x([0, \infty)) \cap U\}.$$

Die Menge $S(v)$ ist vollständig enthalten in einem abgeschlossenen Halbraum, dessen Grenze v enthält.
Die Strahlen, welche $S(v)$ begrenzen, sind in Strahlen, die v mit Delaunay-Knoten der Grenzen der zu v dualen Delaunay-Zelle verbinden, enthalten.

Join: Ein Join ist ein Voronoi-Knoten v, der in der stabilen Mannigfaltigkeit eines Sattelpunktes liegt, so dass die Schnittmenge jeder offenen Umgebung von v mit der Menge der Punkte, welche nach v fliessen, eine offene Untermenge von $\mathbb{R}^2$ enthält.

Ausgedünnte stabile Mannigfaltigkeiten: An jedem Join f eines Sattelpunktes s von Φ wird die stabile Mannigfaltigkeit von s auf folgende Weise verändert:
Es ist zu beachten, dass die Strahlen, die die winkelförmige Region A, welche nach f fliesst, begrenzen, nicht zusammenfallen können. Man betrachte dazu Lemma 8 und die Definition eines Joins. Sei R_l der linke und R_r der rechte begrenzende Strahl von A. Die ausgedünnte stabile Mannigfaltigkeit $T(s)$ von s schliesst alle Punkte von $S(s)$ aus, die nach f fliessen, aber deren Fluss ausser f keinen Punkt von R_l oder R_r enthält, d.h. an jedem Join f wird folgende Menge von $S(s)$ subtrahiert:

$$\{x \in S(s) : f \in \Phi_x([0, \infty)), \Phi_x([0, \infty)) \cap (R_l \cup R_r) = \{f\}\}.$$

Entfernt man alle diese Mengen aus $S(s)$, so bleibt eine eindimensionale Menge $T(s)$ übrig, die in stückweise lineare Kurven zerlegt werden kann. Man betrachte dazu Lemma 7.

Instabiler Zerlegungsgraph: Gegeben sei der folgende Graph G: Seine Knotenmenge besteht aus allen Maxima, Sattelpunkten und Steiner-Knoten. Zwei Knoten sind durch eine Kante verbunden, falls eine instabile Mannigfaltigkeit $U(s)$ eines beliebigen Sattelpunktes s existiert, so dass beide Knoten in $U(s)$ enthalten sind und es keine anderen Knoten auf $U(s)$ gibt, die dazwischen liegen. Ein zusätzlicher Knoten im Unendlichen wird der

Knotenmenge von G hinzugefügt und Kanten, die diesen Knoten mit allen Knoten in G verbinden, welche unter Φ auf einer Geraden ins Unendliche fliessen, werden der Kantenmenge von G hinzugefügt. Der Graph G heisst zu einem Disk induzierten Fluss gehörender instabiler Zerlegungsgraph. Sei S die Menge der Sattelpunkte von Φ und $U(S) = \bigcup_{s \in S} U(s)$. $U(S)$ nennt man die geometrische Realisierung von G.

Lemma 9: Der instabile Zerlegungsgraph von Φ ist planar, d.h. er kann frei von Überkreuzungen in eine Kugeloberfläche eingebettet werden.

Stabiler Zerlegungsgraph: Gegeben sei der folgende Graph G : Seine Knotenmenge besteht aus allen Minima, Sattelpunkten und Joins. Zwei Knoten sind durch eine Kante verbunden, falls eine ausgedünnte stabile Mannigfaltigkeit $T(s)$ eines beliebigen Sattelpunktes s existiert, so dass beide Knoten in closure($T(s)$) enthalten sind und es keine anderen Knoten auf $T(s)$ gibt, die dazwischen liegen. Es ist nötig, den Abschluss der ausgedünnten stabilen Mannigfaltigkeiten zu nehmen, weil sonst die Minima nicht darin enthalten wären. Der Graph G heisst zu einem Disk induzierten Fluss gehörender stabiler Zerlegungsgraph. Sei S die Menge der Sattelpunkte von Φ und $T(S) = \bigcup_{s \in S} \text{closure}(T(s))$. $T(S)$ nennt man die geometrische Realisierung von G.

Lemma 10: Der stabile Zerlegungsgraph von Φ ist planar, d.h. er kann frei von Überkreuzungen in eine Fläche eingebettet werden.

2.3 Min- und Max-Regionen

Anstatt direkt mit den instabilen Mannigfaltigkeiten von Minima und den stabilen Mannigfaltigkeiten von Maxima zu arbeiten, kann man sogenannte Min- und Max-Regionen definieren, welche angenehmere Eigenschaften besitzen.

Min- und Max-Regionen: Gegeben sei ein Advancing Dynamical System mit endlich vielen Fixpunkten. Für ein Minimum m dieses Systems nennt man den Abschluss der instabilen Mannigfaltigkeit $U(m)$ die Min-Region. Für ein Maximum m dieses Systems nennt man den Abschluss des Inneren der stabilen Mannigfaltigkeit $S(m)$ die Max-Region.

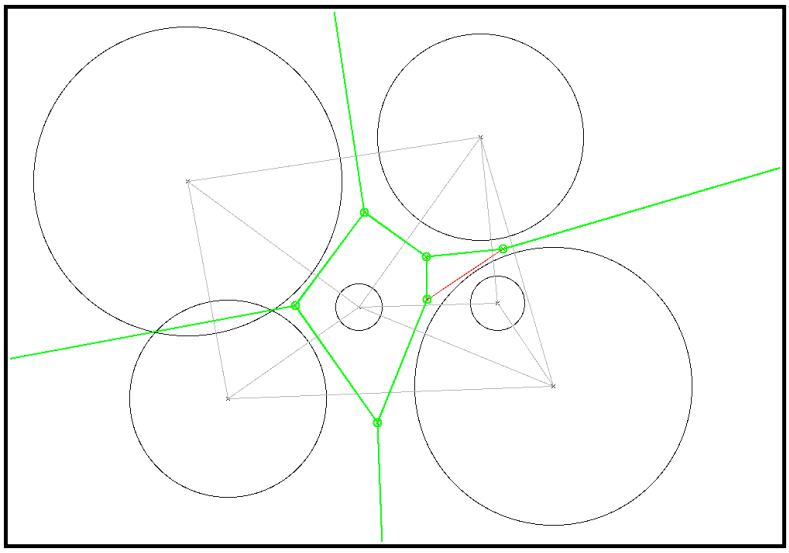


Abbildung 2.5: Die Min-Regionen einer Menge von Disks in der Ebene.

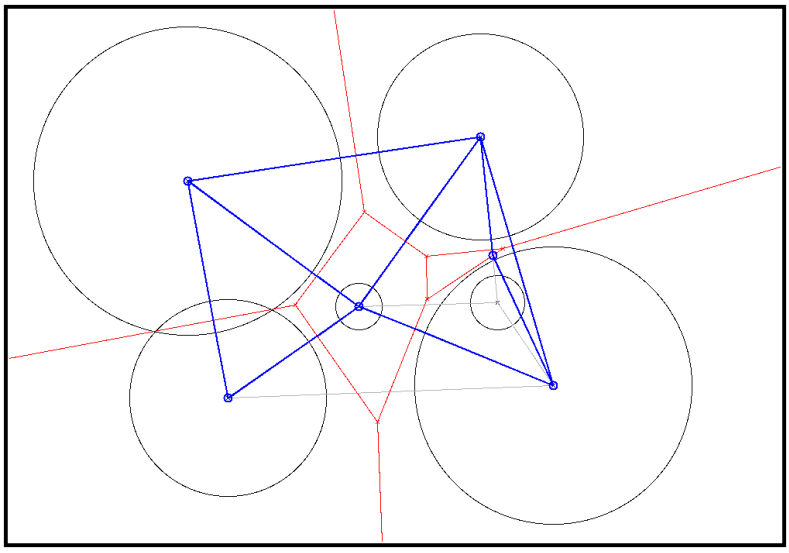


Abbildung 2.6: Die Max-Regionen der selben Menge von Disks.

Join-Region: Gegeben sei der zu einem Disk induzierten Fluss Φ gehörende stabile Zerlegungsgraph G. Da G mit Hilfe von Φ konstruiert wurde, kann man die Kanten von G, gemäss der Richtung des Flusses entlang den Kanten, orientieren. Jeder Join hat drei inzidente Kanten in G. Die geometrische Realisierung von G zerlegt die Ebene in abgeschlossene Regionen. Eine solche Region nennt man Join-Region, falls sie einen Join f und zwei zu f hingerichtete Kanten auf ihren Grenzen hat.

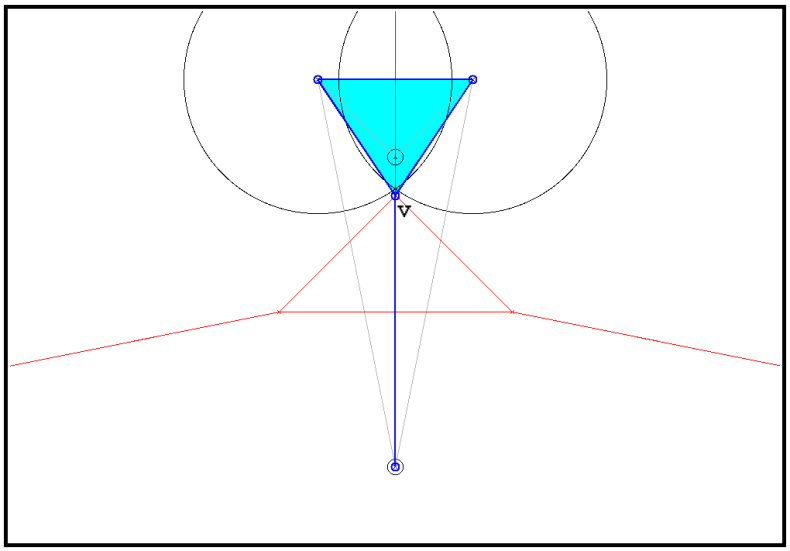


Abbildung 2.7: Die Join-Region eines Joins v.

Theorem 1: Gegeben sei ein Disk induzierter Fluss Φ. Die Min-Regionen von Φ sind genau die Abschlüsse der Regionen der geometrischen Realisierung des zu Φ gehörenden instabilen Zerlegungsgraphen.

Theorem 2: Gegeben sei ein Disk induzierter Fluss Φ. Die Max- und Join-Regionen von Φ sind genau die Abschlüsse der Regionen der geometrischen Realisierung des zu Φ gehörenden stabilen Zerlegungsgraphen.

Korollar 2: Die Min-Regionen von Φ sind Polygone mit Knoten auf den Maxima und Sattelpunkten von Φ und auf Steiner-Knoten. Steiner-Knoten sind entweder Voronoi-Knoten, die keine Maxima sind oder sie

liegen im inneren von Voronoi-Kanten.

Korollar 3: Die Max- und Join-Regionen von Φ sind Polygone.

Korollar 4: Der zu Φ gehörende instabile Zerlegungsgraph ist zusammenhängend.

Korollar 5: Der zu Φ gehörende stabile Zerlegungsgraph ist zusammenhängend.

Folgender Algorithmus, der eine Menge B von Disks als Eingabe verlangt, lässt sich zur Berechnung der Min-Regionen von Φ bzw. deren Grenzen verwenden:

```
MinDiagram(B)
// The basic data structures.
 1  V, E := ∅.
 2  compute Voronoi- and Delaunay diagram of B.
 3  compute the drivers of all Voronoi edges and
    Voronoi vertices.
 4  for each Voronoi vertex v do
 5     flag(v) := false.
 6  end for
 7  for each Voronoi egde e do
 8     if driver(e) is not a Delaunay vertex
 9        A(e) := ∅; type(e) := 1
10     else
11        type(e) := 2
12     end if
13  end for
// Computing first parts of the unstable decomposition
// graph of B.
14  compute the saddles S of φ.
15  for each s ∈ S do
16     vw := Voronoi edge that contains s.
17     V := V ∪ {v, w}; A(vw) := A(vw) ∪ {v, w}
18     for each x ∈ {v, w} do
19        while x is not a maximum or a saddle or a
```

```
                point at infinity or a Voronoi vertex
                with flag(x) = true do
20          if x is a Voronoi vertex do
21            flag(x) := true
22          end if
23          y := first point on the orbit of φ_x that lies
                 on a Voronoi edge e not incident to x;
                 a point at infinity if such a Voronoi
                 edge does not exist.
24          if xy is a Voronoi edge do
25            A(xy) := A(xy) ∪ {x, y}
26          else
27            E := E ∪ {xy}
28            if type(e) = 1 and y is not a Voronoi
                 vertex do
29              A(e) := A(e) ∪ {y}
30              y := the Voronoi vertex incident to e
                       that y flows into.
31              A(e) := A(e) ∪ {y}
32            end if
33          end if
34          V := V ∪ {y}; x := y
35        end while
36      end for
37    end for
// Finishing the computation of the unstable
// decomposition graph of B.
38    for each Voronoi edge e do
39      if type(e) = 1 do
40        V := V ∪ A(e)
41        sort A(e) along e.
42        for i := 1 to |A(e)| − 1 do
43          E := E ∪ {A(e)[i]A(e)[i + 1]}
44        end for
45      end if
46    end for
```

Folgender Algorithmus, der eine Menge B von Disks als Eingabe ver-

langt, lässt sich zur Berechnung der Max- und Join-Regionen von Φ bzw. deren Grenzen verwenden:

```
MaxDiagram(B)
 1  E, V, F := ∅
 2  compute Voronoi- and Delaunay diagram of B.
 3  compute the saddles S of φ.
 4  for each s ∈ S do
 5    V := V ∪ {s}
 6    vw := Delaunay edge which contains s.
 7    F := F ∪ {sv, sw}
 8    while F ≠ ∅ do
 9      choose xy ∈ F
10      F := F − {xy}.
11      x' := first intersection point of the segment
                from x to y with a Voronoi edge e that
                intersects xy in only one point;
                or y if such a point does not exist.
12      V := V ∪ {x'}; E := E ∪ {xx'}
13      if x' ≠ y do
14        if x' lies in the interior of e do
15          yy' := Delaunay edge dual to e.
16          F := F ∪ {x'y'}
17        else
18          (z, z') := Join(x')
19          F := F ∪ {x'z, x'z'}
20        end if
21      end if
22    end while
23  end for
24  extract the regions of G = (V, E).

Join(v)
 1  V := ∅
 2  for all Voronoi cells c incident to v in
            counter clockwise order around v do
 3    x := Delaunay vertex dual to c.
 4    if vx crosses c do
```

```
5       V := V ∪ {x}
6     end if
7   end for
8   (y, y') := the tuple of consecutive points in the
                sequence V[1], ..., V[|V|], V[1] that
                maximizes the smaller angle spanned
                by the segments vy and vy'.
9   return (y, y').
```

2.4 Dualität

Strenge Dualitätsbedingung: Eine Menge B von Disks in $\mathbb{R}^2$ erfüllt die strenge Dualitätsbedingung, falls der Mittelpunkt jeder Disk in B innerhalb seiner dualen Voronoi-Zelle liegt.

Gabriel-Graph: Gegeben sei eine Menge von Disks B in der Ebene. Der Gabriel-Graph von B ist wie folgt gegeben: Seine Knoten sind die Mittelpunkte der Disks von B und seine Kanten sind gegeben durch Delaunay-Kanten, die ihre duale Voronoi-Kante schneiden. Die Kanten des Gabriel-Graphen nennt man Gabriel-Kanten. Aus Lemma 1 folgt, dass jede Gabriel-Kante einen Sattelpunkt von Φ_B enthält und umgekehrt jeder Sattelpunkt in einer Gabriel-Kante enthalten ist.

Orthogonale Disk: Zwei Disks $b_i = (z_i, r_i)$ und $b_j = (z_j, r_j)$ sind orthogonal, falls gilt

$$\|z_i - z_j\|^2 = r_i^2 + r_j^2.$$

Duale Diskmenge: Gegeben sei eine Menge von Disks B in der Ebene. Die Menge der dualen Disks C ist wie folgt gegeben: Diese neuen Disks haben ihre Mittelpunkte in den Voronoi-Knoten des durch B definierten Voronoi-Diagrammes. Sei y_j der j-te Voronoi-Knoten und s_j^2 der minimale Power-Abstand zu einer Disk in B. Für eine Menge von Disks in allgemeiner Lage gibt es jeweils drei Disks, die zu einem Voronoi-Knoten einen minimalen

Power-Abstand haben. Es gilt

$$C = \{c_j = (y_j, s_j)\}$$

für alle Knoten y_j des durch die Disks in B definierten Voronoi-Diagrammes. Zusätzlich stellt man sich für jede unbegrenzte Voronoi-Kante einen Voronoi-Knoten im Unendlichen in Richtung der unbegrenzten Seite der Kante vor. Der Radius der an dieser Stelle entstehenden dualen Disk ist dann ebenfalls unendlich und wird so gewählt, dass die duale Disk orthogonal zu den beiden ursprünglichen Disks ist, die die Delaunay-Kante begrenzen, welche zur oben erwähnten unbegrenzten Voronoi-Kante dual ist.
Die Menge der dualen Disks C besteht also aus Disks, die auf den Voronoi-Knoten der ursprünglichen Diskmenge liegen und die jeweils zu den ursprünglichen Disks, die auf den Delaunay-Knoten der zum entsprechenden Voronoi-Knoten dualen Delaunay-Zelle liegen, orthogonal sind.

Eine interessante Eigenschaft der dualen Diskmenge ist, dass sie Delaunay-Triangulierung und Voronoi-Diagramm der ursprünglichen Diskmenge, ausser am Rand, vertauscht.

Lemma 11: Gegeben sei eine Menge von Disks B, die die strenge Dualitätsbedingung erfüllen und eine zusätzliche Disk im Unendlichen. Sei C die Menge der dualen Disks und seien Φ_B und Φ_C die Advancing Dynamical Systems die zu B bzw. C gehören. Die geometrische Realisierung des zu Φ_C gehörenden instabilen Zerlegungsgraphen ist der Gabriel-Graph von B.

Theorem 3: Gegeben sei eine Menge von Disks B, die die strenge Dualitätsbedingung erfüllen und eine zusätzliche Disk im Unendlichen. Sei C die Menge der dualen Disks und seien Φ_B und Φ_C die Advancing Dynamical Systems die zu B bzw. C gehören. Die Max-Regionen von Φ_B sind genau die Min-Regionen von Φ_C.

Korollar 6: Die Max-Regionen von Φ_B und die Min-Regionen von Φ_C sind eine Menge von benachbarten Delaunay-Zellen in B, getrennt durch Gabriel-Kanten.

Theorem 4: Gegeben sei eine Menge von Disks B, die die strenge Dualitätsbedingung erfüllen und Φ_B der von B induzierte Fluss. Die Min-Regionen von Φ_B sind die Voronoi-Zellen von B.

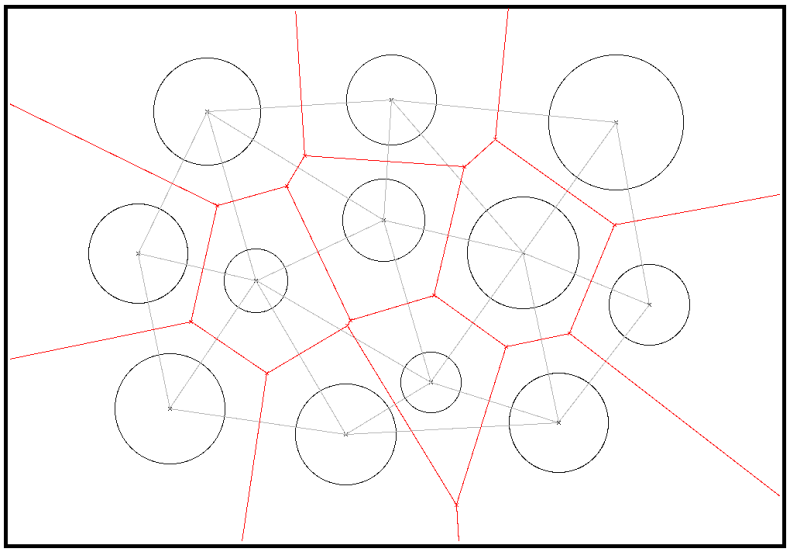


Abbildung 2.8: Eine Menge von Disks, welche die strenge Dualitätsbedingung erfüllen, und die zugehörigen Voronoi- und Delaunay-Diagramme.

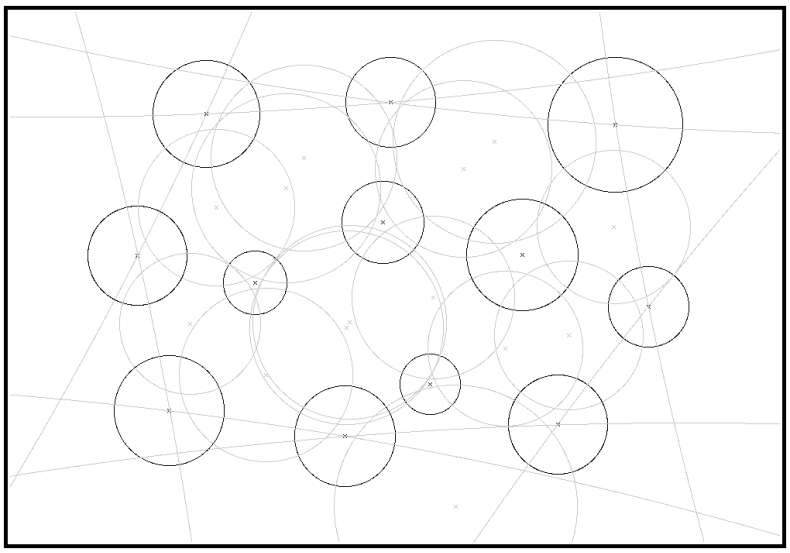


Abbildung 2.9: Die obige Menge von Disks sowie die duale Diskmenge.

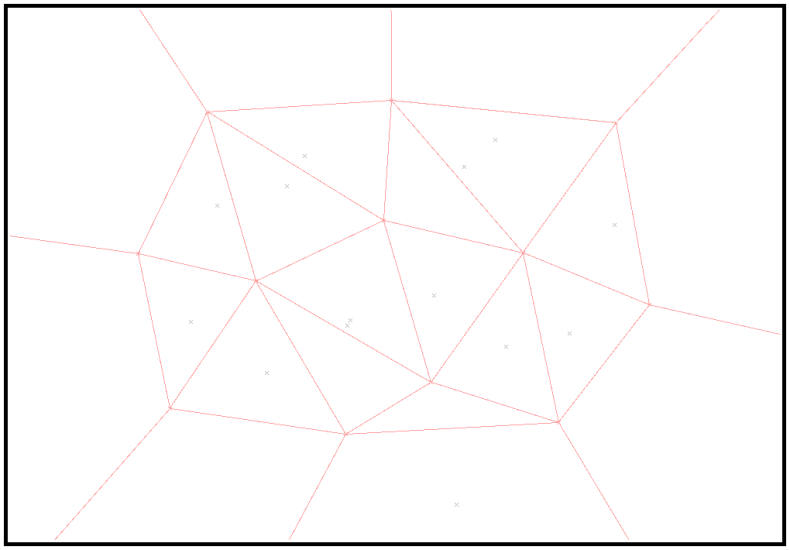


Abbildung 2.10: Das Voronoi-Diagramm der dualen Diskmenge.

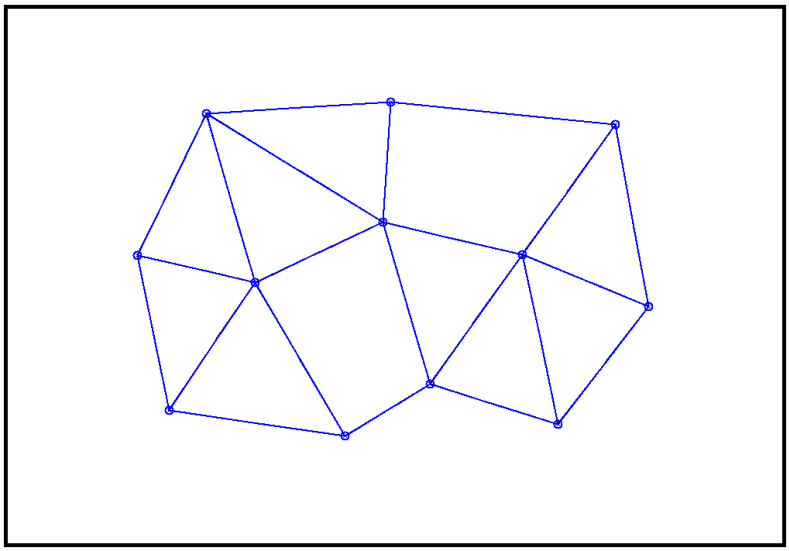


Abbildung 2.11: Das Max-Diagramm der ursprünglichen Diskmenge.

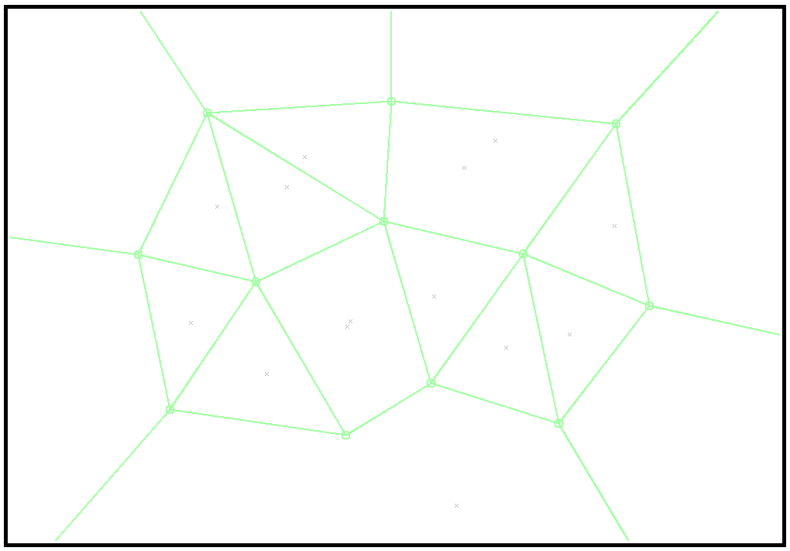


Abbildung 2.12: Das Min-Diagramm der dualen Diskmenge.

Kapitel 3

Implementierung der Algorithmen

Für die Implementierung der beschriebenen Algorithmen mit C++ habe ich die CGAL-Library verwendet. Die CGAL-Library liefert Algorithmen und Datenstrukturen für diverse geometrische Berechnungen.
Die reguläre Delaunay-Triangulierung und das duale Voronoi-Diagramm sind die Grundlagen für weitere Berechnungen. Für beides stellt die CGAL-Library Algorithmen und Datenstrukturen zur Verfügung, die ich verwendet und erweitert habe, um möglichst optimal darauf aufbauen zu können.
Der folgende Text soll einen Überblick über mein Programm bieten. Für Details betrachte man den vollständigen Quellcode.

3.1 Gewichtete Delaunay-Triangulierung und gewichteter Voronoi-Komplex

CGAL stellt in der Klasse `Regular_Triangulation_2` die benötigten Datenstrukturen und Algorithmen zur Verfügung, um die Delaunay-Triangulierung einer Menge von gewichteten Punkten in zwei Dimensionen zu berechnen, sowie um den dualen Voronoi-Komplex zu berechnen.
Mein Programm enthält zunächst einmal die Klassen `face_base_2` zur Repräsentierung der Delaunay-Dreicke und `vertex_base_2` für die Delaunay-Knoten. Es ist hilfreich, diese Klassen selber zu erstellen, um darin zusätzliche Daten abspeichern zu können. Für die gewichteten Punkte wird jeweils der quadrierte Radius abgespeichert.

Delaunay-Kanten werden nicht direkt durch Kanten-Objekte repräsentiert, sondern durch ein Objekt der Klasse `face_base_2`, das für eines der beiden Delaunay-Dreiecke steht, die zur Kante inzident sind, sowie ein Index $i \in \{0, 1, 2\}$, der die Kante eindeutig bestimmt.
Die Dualität zwischen Voronoi- und Delaunay-Objekten erleichtert die Programmierung. Voronoi-Knoten, -Kanten und -Gebiete werden immer durch Objekte ihrer dualen Delaunay-Dreiecke, -Kanten bzw. -Knoten repräsentiert.
Parallel zur Speicherung der gewichteten Punkte in der Klasse `vertex_base_2` werden sie auch noch im globalen Vektor `wpoints` gespeichert. Das hat den Vorteil, dass man bei einer Veränderung der Punkt-Daten lediglich diesen Vektor aktualisieren muss und davon ausgehend die Delaunay-Triangulierung neu aufbauen kann.
Der Aufbau der in der Variablen `rt` gespeicherten Delaunay-Triangulierung erfolgt inkrementell. `rt` ist von der Klasse `Regular_Triangulation_2`. Diese Klasse stellt die Prozedur `insert` zur Verfügung, die einen gewichteten Punkt als Eingabe verlangt und diesen Punkt der Delaunay-Triangulierung hinzufügt. Mit dieser Prozedur lässt sich ein Punkt nach dem anderen der Triangulierung hinzufügen.

3.2 Halbkanten-Datenstruktur

Bevor ich auf die Prozeduren zur Berechnung der Min- und Max-Diagramme eingehe, möchte ich zuerst die Halbkanten-Datenstruktur vorstellen, die ich erstellt habe, um die beiden Diagramme zu repräsentieren.
Halbkanten-Datenstrukturen sind sehr gut geeignet, um darin planare Graphen mit ungerichteten Kanten, wie eben die Min- bzw. Max-Diagramme, abzulegen. Jede ungerichtete Kante des ursprünglichen Graphen wird dabei durch ein Paar von Halbkanten repräsentiert. Diese beiden Halbkanten verbinden die selben zwei Punkte, wie die ursprüngliche Kante. Die beiden zeigen in gegensätzliche Richtungen. Knoten des ursprünglichen Graphen werden durch Knoten der Halbkanten-Datenstruktur repräsentiert. Eine Halbkante enthält einen Zeiger auf ihren sogenannten Basis-Knoten, den Knoten der Halbkanten-Datenstruktur, von dem aus die gerichtete Halbkante ausgeht. Desweiteren enthält sie einen Zeiger auf ihre Zwillings-Halbkante, jene entgegengesetzt gerichtete Halbkante die zur Repräsentierung der selben Kante des ursprünglichen Graphen dient. Zuletzt enthält sie noch einen Zeiger auf

die nächste Halbkante, die wie folgt definiert ist. Die nächste Halbkante h2 einer Halbkante h1 ist immer eindeutig. Basis-Knoten v2 von h2 ist derjenige Knoten, der auch Basis-Knoten der Zwillings-Halbkante von h1 ist. h2 repräsentiert jene ursprüngliche Kante, die auf die von h1 repräsentierte Kante folgt, wenn man alle zum durch v2 repräsentierten Knoten inzidenten Kanten des ursprünglichen Graphen im Gegenuhrzeigersinn ordnet. Ein Knoten der Halbkanten-Datenstruktur enthält einen Zeiger auf einen Punkt, der die Koordinaten des zu repräsentierenden Knotens des ursprünglichen Graphen enthält. Ausserdem enthält er einen Zeiger auf eine ausgehende Halbkante. Die ausgehende Halbkante ist nicht eindeutig. Sie kann unter allen Halbkanten, die den Knoten als Basis-Knoten haben, beliebig gewählt werden.

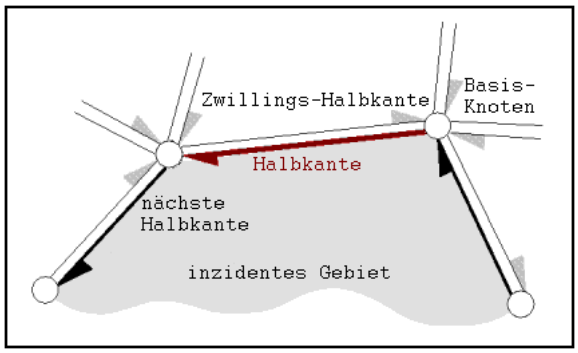


Abbildung 3.1: Ein Ausschnitt aus einer Halbkanten-Datenstruktur.

Ein Vorteil dieser Repräsentierung ist, dass man jetzt, wenn man bei einer Halbkante startet und jeweils zur nächsten Halbkante geht, am Ende wieder bei der selben Halbkante angelangt und dabei die Grenzen des Gebietes verfolgt hat, das an die Halbkante grenzt, sofern die besagte Halbkante an ein endliches Gebiet grenzt. Das Verfolgen der Gebietsgrenzen wiederum ist hilfreich, um den Flächeninhalt des Gebietes zu berechnen.
Den Code für die Halbkanten-Datenstruktur habe ich in einem separaten File `plangraph.h` gespeichert, um die Unabhängigkeit vom eigentlichen Programm und somit die Wiederverwendbarkeit zu erhöhen. Es besteht aus drei Template-Klassen. Eine für die Halbkanten (`halfedge`), eine für die Knoten der Halbkanten-Datenstruktur (`hvertex`) und eine für den gesammten Graphen, der sich aus diesen beiden Elementen zusammensetzt (`plangraph`). Alle drei Template-Klassen sind parametrisiert mit einer Klasse, die die Art

der verwendeten Punkte festlegt. Die dadurch erreichte Flexibilität ist später wichtig, z.B. um Punkte einfügen zu können, die ein zusätzliches Flag besitzen das anzeigt, ob es sich dabei um endliche oder unendliche Punkte handelt. Die Klasse `plangraph` enthält alle Halbkanten und Knoten des Graphen, gespeichert durch zwei Vektoren von Zeigern auf die Halbkanten bzw. die Knoten die untereinander ebenfalls durch Zeiger verknüpft sind. Die Klasse besitzt unter anderem die Prozeduren `insert_edge` und `insert_vertex` zum Einfügen einer neuen Kante bzw. eines neuen Knotens. `insert_vertex` verlangt lediglich einen Zeiger auf einen Punkt als Eingabe und erzeugt daraus einen Knoten der Halkanten-Datenstruktur und speichert diesen, wobei der Zeiger auf die ausgehende Kante zunächst nicht gesetzt wird. `insert_edge` verlangt Zeiger auf zwei Knoten der Halkanten-Datenstruktur, welche die End-Knoten der einzufügenden Kante repräsentieren. Die Prozedur erzeugt daraus zwei Halbkanten mit den richtigen Verknüpfungen, die gespeichert werden. Die richtige Wahl der nächsten Kante einer Halbkante erwies sich dabei als schwierigste Aufgabe, da sie wegen der zahlreichen zu berücksichtigenden Spezialfälle die grösste Fehlerquelle darstellte.

3.3 Min-Diagramm

Mein Algorithmus zur Berechnung des Min-Diagrammes basiert auf dem Pseudocode aus Kapitel 2.3.
Ich habe der Klasse `face_base_2`, die der Repräsentierung der Delaunay-Dreiecke dient, einige Variablen hinzugefügt. Die Variable `vor` dient der Speicherung des Voronoi-Knotens jedes (endlichen) Delaunay-Dreieckes. Dies ist sinnvoll aus Gründen der Effizienz, damit der selbe Voronoi-Knoten nicht mehrfach neu berechnet werden muss. `visited[3]` ist ein Array von Boolean-Werten, jeweils einem pro Kante, der sicherstellt, dass Daten, die zu einer bestimmten Kante gehören nur in einem der beiden zur Kante inzidenten Delaunay-Dreicke abgespeichert werden. CGAL stellt keine Objekte für die Repräsentierung von Delaunay-Kanten zur Verfügung. Delaunay-Kanten werden jeweils durch ein Delaunay-Dreieck und einen Index $i \in \{0, 1, 2\}$, der dem Index des der Kante im Dreieck gegenüberliegenden Knotens entspricht, repräsentiert. Anfangs werden alle Arrayelemente `visited[i]` auf `false` initialisiert. Für Fälle, in denen zwei endliche Delaunay-Dreicke benachbart sind, wird willkürlich in einem dieser Dreiecke die zur dazwischen liegenden Kante gehörende Variable `visited[i]` auf `true` gesetzt, was bedeutet,

dass die Daten der gemeinsamen Kante nicht in diesem Dreieck gespeichert werden. Das Array `A[3]` enthält die zu den Kanten des Delaunay-Dreiecks gehörenden Vektoren von Zeigern auf Knoten der Halbkanten-Datenstruktur. Diese Vektoren entsprechen den im Pseudocode als $A(e)$ bezeichneten Hilfsmengen einer Kante e, die dazu dienen, Knoten des Min-Diagrammes, die auf dieser Kante liegen, bis zur am Ende erfolgenden Sortierung zwischenzuspeichern.
`mindiagram` ist eine Variable der im File `plangraph.h` enthaltenen Klasse `plangraph` und dient der Speicherung aller Halbkanten-Objekte, die zum Min-Diagramm gehören. Sie ist eine der wenigen globalen Variablen meines Programmes. Weitere globale Variablen sind die Vektoren von Punkten `maxima` und `saddles`, in denen wie die Namen schon vermuten lassen die Maxima und Sattelpunkte des Flusses gespeichert werden.
Wie schon erwähnt verlangen die im File `plangraph.h` enthaltenen Template-Klassen eine Punkt-Klasse als Parameter, die die Art der verwendeten Punkte festlegt. Zu diesem Zweck habe ich die Klasse `myhalfpoint` erstellt, welche ausser einem gewöhnlichen CGAL-Punkt noch ein Flag enthält. Dieses Flag gibt an, ob es sich bei dem Punkt um einen endlichen Punkt, oder um einen Punkt im Unendlichen handelt. Solche unendlichen Punkte sind notwendig, weil sich die Kanten des Min-Diagrammes teilweise bis ins Undendliche erstrecken. Die Koordinaten eines unendlichen Punktes sind endlich, werden aber so gross gewählt, dass er ausserhalb des im nächsten Kapitel genauer beschriebenen Graphik-Fensters zu liegen kommt. Somit ensteht bei der Ausgabe auf dem Bildschirm der Eindruck einer sich ins Unendliche erstreckenden Kante.
Ich habe zahlreiche kleine Prozeduren erstellt, die geometrische Berechnungen ausführen, die zur Berechnung des Min-Diagrames häufig benötigt werden. Diese Prozeduren liefern unter anderem den Treiber einer Voronoi-Kante oder eines Voronoi-Knotens, den Sattelpunkt einer Delaunay-Kante, falls vorhanden, oder überprüfen, ob ein Voronoi-Knoten ein Maximum ist. Als Eingabe verlangen sie jeweils ein Delaunay-Objekt, auch falls es sich eigentlich um eine Berechnung handelt, die sich auf ein Voronoi-Objekt bezieht. In diesem Fall übergibt man der Prozedur einfach das zum Voronoi-Objekt duale Delaunay-Objekt.
Die Prozedur `calculate_mindiagram` enthält den Algorithmus zur Berechnung des Min-Diagrammes. Es wird wie schon erwähnt in der globalen Variable `mindiagram` abgespeichert.
Zuerst werden die Variablen `mindiagram`, `maxima` und `saddles` initialisiert.

Dann wird über alle endlichen Delaunay-Dreiecke iteriert, um die dualen Voronoi-Knoten zu berechnen und im Dreieck abzuspeichern, um die Arrays `A[i]` jedes Dreiecks zu initialisieren und um zu überprüfen, ob es sich beim dualen Voronoi-Knoten eines Delaunay-Dreiecks um ein Maximum handelt und ihn in diesem Fall dem Vektor `maxima` hinzuzufügen. Danach folgt eine weitere Iteration über alle endlichen Delaunay-Dreiecke. Diesmal wird für jedes Dreieck über alle drei Kanten iteriert. Die bereits erwähnte Variable `visited[i]` wird für jede Kante auf `false` initialisiert. Ausserdem wird für jede Kante überprüft, ob sie einen Sattelpunkt enthält und falls dies der Fall ist wird der Sattelpunkt dem Vektor `saddles` hinzugefügt.
Nun folgt die im Pseudocode beschriebene Iteration über alle Sattelpunkte s des Flusses Φ. Praktisch geschieht dies mit Hilfe einer Iteration über alle endlichen Delaunay-Dreiecke, die eine Iteration über alle Kanten des aktuellen Dreiecks beinhaltet. Für jede solche Delaunay-Kante wird nun überprüft, ob sie einen Sattel enthält. In diesem Falle erfolgen dann weitere Berechnungen, sofern die selbe Kante nicht schon einmal vom anderen inzidenten Delaunay-Dreieck aus besucht wurde, was mit Hilfe der Variablen `visited[i]` leicht zu überprüfen ist. Die weiteren Berechnungen im Falle eines Sattelpunktes dienen nun zunächst der Speicherung der beiden Endpunkte der zur aktuellen Delaunay-Kante dualen Voronoi-Kante und der Speicherung von Zeigern auf diese Punkte in der globalen Variable `mindiagram` sowie im Array `A[i]` der aktuellen Delaunay-Kante. Diese Endpunkte müssen nicht zwingendermassen endlich sein. Da eine Iteration über endliche Delaunay-Dreiecke aber nur erfolgt, falls die für die Existenz von mindestens einem endlichen Dreieck erforderliche Anzahl von drei Knoten erreicht wird, kann es aber im Falle einer Iteration keine beidseitig unbegrenzten Voronoi-Kanten geben, d.h. es ist mindestens einer der Endpunkte endlich. Im Falle eines unendlichen Endpunktes wird nun ein Punkt auf dem Strahl, ausgehend vom endlichen Endpunkt hin zum unendlichen Endpunkt der Voronoi-Kante, gewählt und das Flag dieses Punktes auf unendlich gesetzt. Bevor ein neuer Punkt in das Min-Diagramm bzw. in ein Array `A[i]` eingefügt wird, ist es jeweils wichtig, zu überprüfen, ob dieser Punkt nicht bereits enthalten ist. Zuerst wird überprüft, ob im Min-Diagramm bereits ein Punkt mit den selben Koordinaten existiert. Falls nicht wird der Zeiger auf den neuen Punkt eingefügt. Ansonsten wird mit dem Zeiger auf den bereits existierenden Punkt weitergearbeitet. Für `A[i]` wird nun überprüft, ob dieser Zeiger bereits enthalten ist. Falls nicht wird er eingefügt.
Für jeden dieser Endpunkte der Voronoi Kante wird nun eine Schleife zur

Verfolgung des Flusses ausgeführt. Am Anfang der Schleife wird zuerst überprüft, ob es sich bei dem Punkt, auf dem man sich gerade befindet, um einen Punkt im Unendlichen, um ein Maximum oder um einen Sattelpunkt handelt. Falls einer dieser drei Fälle zutrifft, was sich mit den Hilfsprozeduren die ich erstellt habe leicht überprüfen lässt, dann wird die Schleife verlassen. Ansonsten wird der Fluss weiterverfolgt. Am Anfang dieser Schleife befindet man sich jeweils entweder im Innern einer Voronoi-Kante, oder auf einem Voronoi-Knoten. Falls man sich auf einem Voronoi-Knoten befindet, wurde die Variable `on_vor_vertex` bereits zuvor, beim Erreichen dieses Knotens, auf `true` gesetzt, ansonsten auf `false`. Falls man sich auf einem Voronoi-Knoten befindet, muss nun weiter unterschieden werden, ob der Fluss von dort weiter einer Voronoi-Kante entlang verläuft oder durch eine Voronoi Zelle hindurch führt. Diese Aufgabe erfüllt die Hilfprozedur `drivertype`, die als Eingabe das zum aktuellen Voronoi-Knoten, der kein Maximum sein darf, duale Delaunay-Dreieck verlangt und daraus einen Integer-Wert $i \in \{-1, 0, 1, 2\}$ berechnet und zurückgibt. Die berechnete Integer-Zahl dient der Unterscheidung des Typs des Treibers dieses Voronoi Knotens. Der Treiber wird in Kapitel 2.2 unter „Disk induzierter Fluss“ definiert. Der Typ des Treibers einer Voronoi-Kante kann entweder true oder false sein. Er ist true, falls der Schnittpunkt der beideitig ins Unendliche verlängerten Voronoi-Kante mit der beidseitig ins Unendliche verlängerten dualen Delaunay-Kante auf der Delaunay-Kante zu liegen kommt, sonst false. Für Voronoi-Kanten mit einem Treiber vom Typ true ist der Treiber gerade der besagte Schnittpunkt. Für Voronoi-Kanten mit einem Treiber vom Typ false ist der Treiber derjenige der beiden zur Delaunay-Kante inzidenten Delaunay-Knoten, der dem Schnittpunkt am nächsten liegt. Der Treiber eines Delaunay-Knotens wird mit der Prozedur `calculate_driver` berechnet, die als Eingabe ebenfalls das zum aktuellen Voronoi-Knoten, der kein Maximum sein darf, duale Delaunay-Dreieck verlangt. Für einen beliebigen Punkt auf einer Voronoi-Kante führt der Fluss von dort aus weiter entlang der Kante, genau dann wenn der Treiber dieser Voronoi-Kante vom Typ true ist. Ansonsten führt der Fluss durch eine Voronoi-Zelle hindurch. Etwas komplizierter ist es aber, falls man sich in einem Voronoi-Knoten befindet, denn der ist zu drei Voronoi-Kanten inzident. Der Treiber eines Voronoi-Knotens ist der am nächsten liegende Treiber der drei inzidenten Voronoi-Kanten. Liefert die Prozedur `drivertype` für einen Voronoi-Knoten $i = -1$ zurück, so ist der entsprechende Treiber vom Typ false. Liefert sie $i \in \{0, 1, 2\}$, so ist der entsprechende Treiber vom Typ true. i entspricht dann ausserdem dem Index der Delaunay-Kante auf der

der Treiber liegt, d.h. jener Delaunay-Kante, die zur Voronoi-Kante dual ist, auf der der Fluss vom Voronoi-Knoten aus weiter verläuft. Für eine Voronoi-Kante die in der Richtung des Flusses, d.h. vom Treiber weg, begrenzt ist, ist keine Koordinatenberechnung nötig, denn man gelangt über die Kante zu einem weiteren Voronoi-Knoten, dessen Koordinaten bereits in seinem dualen Delaunay-Dreieck gespeichert sind. Ansonsten gelangt man zu einem Punkt im Unendlichen.
Falls man sich auf einem Voronoi-Knoten befindet und `drivertype` $i = -1$ zurückliefert, d.h. dass der entsprechende Treiber vom Typ false ist, dann muss man den Fluss vom Voronoi-Knoten durch die Voronoi-Zelle hindurch dem Strahl entlang, der vom Voronoi-Knoten aus in die entgegengesetzte Richtung des Treibers verläuft, weiterverfolgen, bis man wieder auf eine Voronoi-Kante oder auf einen Voronoi-Knoten trifft. Man schneidet also diesen Strahl mit allen Voronoi-Kanten, die die Voronoi-Zelle begrenzen, durch die der Fluss verläuft. Um alle diese Voronoi-Kanten zu erhalten, iteriert man über die Delaunay-Kanten, die zum dualen Delaunay-Knoten der Voronoi-Zelle inzident sind und nimmt deren duale Voronoi-Kanten. Der zur Voronoi-Zelle duale Delaunay-Knoten ist gerade der Treiber. CGAL stellt einen Zirkulator zur Verfügung, mit dessen Hilfe sich diese Aufgabe einfach erledigen lässt. Die resultierende Schnittmenge enthält auf diese Weise natürlich auch den Voronoi-Knoten, von dem man ausgeht. Leider musste ich feststellen, dass die Koordinaten der Schnittpunkte, welche ich mit Hilfe der von CGAL zur Verfügung gestellten Schneide-Operation berechnet habe, nicht ausreichend präzise waren, dass ich den unerwünschten Schnittpunkt hätte herausfiltern können, indem ich seine Koordinaten mit denen des Voronoi-Knotens verglich. Deshalb musste ich jeweils darauf achten, die Schneide-Operationen mit den beiden Voronoi-Kanten, die zum Voronoi-Knoten inzident sind, von dem man ausgeht wegzulassen. Falls man keinen Schnittpunkt erhält, so bedeutet das, dass die Voronoi-Zelle offen ist und der Fluss ins Unendliche verläuft.
Befindet man sich am Anfang der Schleife nicht auf einem Voronoi-Knoten, so befindet man sich im Inneren einer Voronoi-Kante. Auch für die Voronoi-Kanten habe ich eine Prozedur `drivertype` erstellt, die die duale Delaunay-Kante als Eingabe verlangt und einen Boolean-Wert zurückgibt, der dem Typ des Treibers der Voronoi-Kante entspricht. Die Prozeduren `driver_of_type_false` und `driver_of_type_true` verlangen ebenfalls die zu einer Voronoi-Kante duale Delaunay-Kante als Eingabe und berechnen daraus den Treiber der Voronoi-Kante. `driver_of_type_false` berechnet Trei-

ber vom Typ false und `driver_of_type_true` berechnet Treiber vom Typ true. Für einen Treiber vom Typ true verfolgt man den Fluss, der in diesem Fall entlang der Kante verläuft, vom aktuellen Punkt aus vom Treiber weg bis zu einem der beiden Voronoi-Knoten dieser Kante, falls sie in diese Richtung begrenzt ist, oder bis ins Unendliche, falls sie in dieser Richtung unbegrenzt ist.
Für einen Treiber vom Typ false verfolgt man den Fluss, der vom aktuellen Punkt auf einer Voronoi-Kante aus in der entgegengesetzten Richtung des Treibers durch die Voronoi-Zelle hindurch verläuft, bis man bei einem Punkt auf einer Voronoi-Kante ankommt, die diese Voronoi-Zelle begrenzt bzw. auf einem Voronoi-Knoten. Ich habe hier wieder das selbe Verfahren verwendet, um alle Voronoi-Kanten zu erhalten, die an die Voronoi-Zelle grenzen. Auch hier muss man darauf achten, dass man nicht mit der Voronoi-Kante schneidet, auf der der Ausgangspunkt liegt und auch hier kann es wieder passieren, dass man keinen Schnittpunkt erhält, was bedeutet, dass der Fluss vom aktuellen Punkt aus ins Unendliche fliesst ohne dabei eine weitere Voronoi-Kante oder einen Voronoi-Knoten zu erreichen.
Ein Zeiger auf den Punkt auf dem man nun angelangt ist wird, ob er nun endlich oder unendlich ist, in der Knotenmenge der Variablen `mindiagram` abgespeichert. Die Kante vom Ausgangspunkt zu dem Punkt bei dem man nun angelangt ist wird aber nur direkt in `mindiagram` abgespeichert, falls sie nicht auf einer Voronoi-Kante liegt. Ansonsten werden Zeiger auf die beiden Endpunkte der Kante in das Array `A[i]` eingefügt, dass zu der Voronoi-Kante gehört, auf der die Kante liegt. Der Grund dafür ist, dass man verhindern kann, dass sich überschneidende Kanten des Min-Diagrammes, die auf ein und der selben Voronoi-Kante liegen, entstehen, wenn man alle Knoten des Min-Diagrammes die auf einer Voronoi-Kante liegen erst später sortiert und die entsprechenden Kanten einfügt.
Nun beginnt die Schleife zur Verfolgung des Flusses erneut, wobei der Ausgangspunkt zuerst auf jenen Punkt gesetzt wird, bei dem man angelangt ist. Zuletzt müssen jetzt nur noch die Punkte in den Arrays `A[i]` geordnet und die entprechenden Kanten in das Min-Diagramm eingefügt werden. Dazu wird wieder über alle endlichen Delaunay-Dreicke und für jedes Delaunay-Dreieck über alle inzidenten Delaunay-Kanten iteriert. Die Sortierung erfolgt ganz einfach nach aufsteigender x-Koordinate der Punkte, bzw. nach aufsteigender y-Koordinate, falls es sich um eine senkrechte Voronoi-Kante handelt und die x-Koordinaten der Punkte alle identisch sind. Jeweils zwei aufeinanderfolgende Punkte bilden jetzt eine Kante, die in `A[i]` eingefügt wird.

3.4 Max-Diagramm

Mein Algorithmus zur Berechnung des Max-Diagrammes basiert auf dem Pseudocode aus Kapitel 2.3.
Er verwendet im Wesentlichen die selben Datenstrukturen und Hilfsprozeduren, die auch zur Berechnung und Speicherung des Min-Diagrammes verwendet werden.
Alle Halbkanten-Objekte, die zum Max-Diagramm gehören, werden in der globalen Variable `maxdiagram` abgespeichert. `maxdiagram` ist eine Variable der im File `plangraph.h` enthaltenen Klasse `plangraph`.
Die Prozedur `calculate_maxdiagram` enthält den Algorithmus zur Berechnung des Max-Diagrammes.
Zuerst werden die Variablen `maxdiagram`, `maxima` und `saddles` initialisiert. Dann wird über alle endlichen Delaunay-Dreiecke iteriert, um die dualen Voronoi-Knoten zu berechnen und im Dreieck abzuspeichern und um zu überprüfen, ob es sich beim dualen Voronoi-Knoten eines Delaunay-Dreiecks um ein Maximum handelt und ihn in diesem Fall dem Vektor `maxima` hinzuzufügen.
Die im Pseudocode beschriebene Iteration über alle Sattelpunkte s des Flusses Φ erfolgt mittels einer Iteration über alle endlichen Delaunay-Kanten und einer Überprüfung, ob die Kante einen Sattelpunkt enthält. Eine endliche Delaunay-Kante muss in CGAL nicht unbedingt durch ein endliches Delaunay-Dreieck und einen Index $i \in \{0, 1, 2\}$ repräsentiert sein, sondern eventuell auch durch ein durch ein unendliches Delaunay-Dreieck und einen Index $i \in \{0, 1, 2\}$. Dies spielt aber für meinen Algorithmus keine weitere Rolle.
Ich habe die im Pseudocode verwendete `while`-Schleife und die Menge F durch eine Rekursion ersetzt. Für jeden Sattelpunkt s wird jeweils ein Zeiger auf diesen Punkt in die Punktenge der Variablen `maxdiagram` eingefügt. Nun wird die rekursive Hilfsprozedur `max_recursion` zweimal aufgerufen. Zuerst mit dem Zeiger auf den aktuellen Sattelpunkt, dem Punkt-Index des einen zur aktuellen Delaunay-Kante, die den Sattelpunkt enthält, inzidenten Delaunay-Knotens sowie dem durch die aktuelle Delaunay-Kante definierten Delaunay-Dreieck, dann noch einmal mit dem Zeiger auf den aktuellen Sattelpunkt, dem Punkt-Index des anderen zur aktuellen Delaunay-Kante, die den Sattelpunkt enthält, inzidenten Delaunay-Knotens sowie dem durch die aktuelle Delaunay-Kante definierten Delaunay-Dreieck als Eingabe. Der erste Eingabe-Parameter der Prozedur `max_recursion` ist immer ein Zeiger auf

einen Punkt x auf einer Delaunay-Kante. Der zweite und dritte Parameter repräsentieren einen Delaunay-Knoten y, der der im Moment aktuellen Treiber ist. Zuerst wird nun ein neuer Punkt x' berechnet. x' ist der Schnittpunkt des Liniensegments von x nach y mit einer Voronoi-Kante, die xy in einem einzigen Punkt schneidet, oder falls so ein Schnittpunkt nicht existiert gleich y. Um alle Voronoi-Kanten zu erhalten, die möglicherweise einen Schnittpunkt mit dem Segment xy haben können, iteriert man über die Delaunay-Kanten, die zum aktuellen Treiber y inzident sind und nimmt deren duale Voronoi-Kanten. Falls kein solcher Schnittpunkt ungleich y existiert, wird der neue Punkt x' auf y gesetzt. Dann wird ein Zeiger auf diesen neuen Punkt x' der Punktmenge und eine Kante zwischen den Punkten x und x' der Variablen `maxdiagram` hinzugefügt.
Falls x' gleich y ist die Prozedur an dieser Stelle beendet. Ansonsten wird zwischen den Fällen unterschieden, wo x' im Innern einer Voronoi-Kante liegt oder wo x' gerade ein Voronoi-Knoten ist. Falls x' im Inneren einer Voronoi-Kante liegt, wird zunächst der Delaunay-Knoten y' ermittelt, der zusammen mit y die Menge der Punkte bildet, die zu der Delaunay-Kante inzident sind, welche zur Voronoi-Kante dual ist, die x' enthält. Nun wird die rekursive Prozedur `max_recursion` erneut aufgerufen mit einem Zeiger auf den Punkt x' als ersten, dem Knoten-Index von y' im Delaunay-Dreieck, dass die Kante yy' enthält, als zweiten und dem Delaunay-Dreieck, dass die Kante yy' enthält, als dritten Eingabe-Parameter.
Falls x' aber gerade auf einem Voronoi-Knoten *vor* zu liegen kommt, so müssen die Berechnungen folgen, die im Pseudocode in der Prozedur `JOIN` zusammengefasst sind. In meinem Programm, wird im Boolean-Array `crosses[3]` jeweils `crosses[i]` auf `true` gesetzt, falls das Liniensegment vom Voronoi-Knoten *vor* zum Delaunay-Knoten, der Teil des zum Voronoi-Knoten dualen Delaunay-Dreiecks ist und den Index i hat, die zu diesem Delaunay-Knoten duale Voronoi-Zelle schneidet, ansonsten auf `false`. Da für die gewichtete Delaunay-Triangulierung wie CGAL sie erstellt die Anzahl solcher Schnitte kleiner als drei sein muss, wird für jeden Delaunay-Knoten z des zum Voronoi-Knoten *vor* dualen Delaunay-Dreiecks, dessen Variable `crosses[i]` gleich `true` ist die rekursive Prozedur `max_recursion` erneut aufgerufen mit einem Zeiger auf x' als ersten, dem Index des Delaunay-Knotens z im zum Voronoi-Knoten *vor* dualen Delaunay-Dreieck als zweiten sowie dem zum Voronoi-Knoten *vor* dualen Delaunay-Dreieck als dritten Eingabe-Parameter.

3.5 Duale Diskmenge

Die Prozedur `calculate_orthodisks` enthält die Anweisungen zur Berechnung der Menge der zu den Disks im Vektor `wpoints` dualen Disks und zur Speicherung dieser dualen Diskmenge im Vektor `wpoints`. Bevor `calculate_orthodisks` aufgerufen wird, muss also eine Kopie des ursprünglichen Vektors `wpoints` angelegt werden. Ausserdem muss die gewichtete Delaunay-Triangulierung des ursprünglichen Vektors `wpoints` von Disks bereits erstellt sein.
Zunächst werden die dualen Disks berechnet, die auf endlichen Voronoi-Knoten der ursprünglichen Diskmenge liegen. Dazu werden mit einem Face-Iterator alle endlichen Delaunay-Dreiecke der urspünglichen Delaunay-Triangulierung durchlaufen. Die Position des zum jeweiligen Delaunay-Dreieck dualen Voronoi-Knotens ergibt die Position der dualen Disk. Damit die neue Disk zu den drei Disks, welche das zum Voronoi-Knoten, der die Position der neuen Disk definiert, duale Delaunay-Dreieck begrenzen, orthogonal ist, muss das Gewicht der neuen Disk dem Power-Abstand zu diesen drei ursprünglichen Disks entsprechen. Der Power-Abstand von der Position des Voronoi-Knotens zu den drei Disks, die das duale Delaunay-Dreieck begrenzen, ist gemäss Definition des Voronoi-Diagrammes für alle drei Disk identisch. Deshalb habe ich willkürlich einfach die erste zur Berechnung des Power-Abstandes gewählt.
Nun müssen noch die dualen Disks berechnet werden, die auf Voronoi-Knoten liegen, die man sich für unbegrenzte Voronoi-Kanten im Unendlichen auf der unbegrenzten Seite der Voronoi-Kante vorstellt. Um die unbegrenzten Voronoi-Kanten zu erhalten, wird noch einmal über alle Delaunay-Dreiecke und für jedes dieser Dreiecke über alle inzidenten Delaunay-Kanten iteriert. Die dualen Voronoi-Kanten dieser Delaunay-Kanten werden dann auf Begrenztheit überprüft. Es können hier übrigens keine beidseitig unbegrenzten Voronoi-Kanten auftreten, da eine solche Kante lediglich für eine Menge von genau zwei Disks entsteht, für die es aber keine Delaunay-Dreiecke gibt, die die Grundlage für meine Iteration bilden. Praktisch werden die Koordinaten der neuen Disks im Vergleich zur Theorie endlich gewählt. Ich habe die Koordinaten auf der unbegrenzten Voronoi-Kante, sehr weit vom endlichen Voronoi-Knoten der selben Kante entfernt gewählt. Dadurch ensteht bei der graphischen Darstellung der Eindruck einer sich ins Unendliche erstreckenden Kante. Das Gewicht dieser neuen Disks wird auf den Power-Abstand gesetzt, die die Position der neuen Disk zu den beiden ursprünglichen Disks

hat, welche die Delaunay-Kante begrenzen, die zur unbegrenzten Voronoi-Kante dual ist, auf der die neue Disk liegt. Damit ist die neue Disk zu diesen beiden ursprünglichen Disks orthogonal.
Zuletzt wird ausgehend vom Vektor `wpoints`, der jetzt die duale Menge von Disks enthält, die gewichtete Delaunay-Triangulierung neu aufgebaut.
Auf der durch die Prozedur `calculate_orthodisks` erzeugten Menge von dualen Disks lassen sich nun alle Diagramme erstellen, wie sie für die primale Menge von Disks erstellt werden können. Wie in Kapitel 2 erwähnt entspricht die Delaunay-Triangulierung der dualen Menge von Disks, ausser am Rand, dem Voronoi-Diagramm der ursprünglichen Menge von Disks und umgekehrt.
Die ursprüngliche Diskmenge kann jeder Zeit wieder mit Hilfe der zu Anfang erstellten Kopie des ursprünglichen Vektors `wpoints` hergestellt werden.
Auf unerwartete Probleme stiess ich, als ich zum ersten Mal das Min-Diagramm einer dualen Diskmenge erstellen wollte. Es war an mehreren Stellen ganz offensichtlich inkomplett oder falsch. Das Problem bestand darin, dass ich die Prozedur welche das Min-Diagramm berechnet nicht dafür konzipiert hatte, das mehrere Delaunay-Dreiecke einen Voronoi-Knoten mit den selben Koordinaten besitzen. Genau dies ist aber der Fall für alle Delaunay-Dreicke der dualen Diskmenge, die im Voronoi-Diagramm der ursprünglichen Diskmenge noch gemeinsam ein Voronoi-Gebiet bildeten. Oder anders ausgedrückt jedes Voronoi-Gebiet der ursprünglichen Diskmenge, welches mehr als drei Ecken hat, wird zu mehreren Delaunay-Dreiecken der dualen Diskmenge deren Voronoi-Knoten alle die selbe Position haben. Da es für die Prozedur, die das Min-Diagramm berechnet aber einen Unterschied macht, in welchem dieser Voronoi-Knoten man sich am Ende eines Schleifendurchlaufes der Flussverfolgung befindet, musste ich die entsprechenden Änderungen vornehmen, damit bei mehrereren Möglichkeiten der richtige Voronoi-Knoten gewählt wird. Der richtige heisst dabei, entweder ein Voronoi-Knoten mit identischer Position, der innerhalb seines dualen Delaunay-Dreiecks liegt, der also ein Maximum ist, oder falls es keinen solchen gibt, derjenige Voronoi-Knoten mit identischer Position, dessen Treiber am nächsten liegt. Nach diesen Änderungen liess sich das Min-Diagramm auch für duale Diskmengen ohne Probleme berechnen.

3.6 Flächenberechnung

Mit Hilfe der Prozedur `calculate_area` lässt sich der Flächeninhalt eines beliebigen endlichen Teilbereichs eines Min- oder Max-Diagrammes berechnen. Als Eingabeparameter verlangt die Prozedur lediglich den Pointer auf eine beliebige Halbkante, die zur Menge der Halbkanten gehört, die den gewünschten Teilbereich einschliessen, sowie eine Boolean-Variable, die angibt, ob es sich um ein Min- oder um ein Max-Diagramm handelt. Für unbegrenzte Bereiche wird ein Flächeninhalt von `-1` zurückgegeben. Zusätzlich zum Flächeninhalt berechnet die Prozedur auch noch die Koordinaten des zum Teilbereich gehörenden Extremums sowie die Anzahl der Knoten des Bereiches. Für das Max-Diagramm werden jedoch nur die Knoten gezählt, die Eckpunkte des Bereiches sind. Sattelpunkte, die auf den Kanten des Bereiches liegen, werden hier nicht mitgezählt.
Bevor die Prozedur `calculate_area` aufgerufen wird, muss aber noch eine der beiden Hilfsprozeduren `calculate_min_positions` bzw. `calculate_max_positions` aufgerufen werden, welche die Koordinaten aller Minima bzw. Maxima berechnen, die im Min- bzw. Max-Diagramm innerhalb eines endlichen Teilbereiches liegen können. Aus all diesen Koordinaten kann die Prozedur `calculate_area` dann später diejenigen des Extremums aussuchen, welches zum Teilbereich gehört. Jede dieser Koordinaten wird in der Prozedur `calculate_min_positions` bzw. `calculate_max_positions` in einem Vektor `min_positions` bzw. `max_positions` von Paaren jeweils zusammen mit einem Boolean-Flag abgespeichert. Das Boolean-Flag dient als Indikator dafür, ob an dieser Stelle schon einmal ein Flächeninhalt berechnet wurde. Damit kann verhindert werden, dass bei einer späteren graphischen Ausgabe der berechneten Flächeninhalte, auf der Position des zugehörigen Extremums des Teilbereichs, Werte für den selben Flächeninhalt mehrmals übereinandergeschrieben werden.
Die Prozedur `calculate_area` setzt zuerst ihre lokale Variable `area_position` entweder auf `min_positions` oder auf `max_positions`, je nachdem, ob es sich um eine Flächenberechnung in einem Min- oder einem Max-Diagramm handelt, was durch den zweiten Eingabeparameter der Prozedur angegeben wurde. Für die Flächenberechnungen in Max-Diagrammen der dualen Diskmenge werden die für die ursprüngliche Diskmenge berechneten Koordinaten der Minima verwendet und für die Flächenberechnungen in Min-Diagrammen der dualen Diskmenge werden die für die ursprüngliche Diskmenge berechneten Koordinaten der Maxima verwendet, weil diese

Koordinaten jeweils übereinstimmen.
Nun wird eine Schleife durchlaufen, die mit der durch den Zeiger, welcher erster Eingabeparameter der Prozedur `calculate_area` ist, gelieferten Halbkante beginnt und solange zur jeweils nächsten Halbkante weiterläuft, bis entweder eine Halbkante mit unendlichem Basis-Knoten oder wieder die Halbkante erreicht wird, bei der die Schleife begann. Eine Halbkante mit einem unendlichen Basis-Knoten bedeutet, dass der Teilbereich, der berechnet werden soll, unbegrenzt ist. Deshalb wird in diesem Fall die Scheife abgebrochen und der Flächeninhalt auf `-1` gesetzt. Ansonsten wird für jeden Schleifendurchgang der Basis-Knoten der aktuellen Halbkante einem zu Anfang leeren Polygon hinzugefügt. Am Ende der Schleife werden für das berechnete Polygon die Koordianten desjenigen Minimums bzw. Maximums gesucht, das innerhalb dieses Polygons liegt. Dazu wird der Vektor `min_positions` bzw `max_positions` durchlaufen. Wurde für den zum ermittelten Extremum geörenden Teilbereich schon einmal der Flächeninhalt berechnet, was durch das im Vektor `min_positions` bzw `max_positions` gespeicherte zugehörige Boolean-Flag angezeigt wird, so wird der Flächeninhalt auf `-1` gesetzt, um eine mehrmalige Darstellung zu verhindern. Ansonsten wird der Flächeninhalt des erstellten Polygons berechnet und das entsprechende Boolean-Flag des Extremums im Vektor `min_positions` bzw `max_positions` gesetzt, um anzuzeigen, dass dieser Flächeninhalt schon einmal berechnet wurde.
Wie schon erwähnt entstehen bei der Berechnung der dualen Diskmenge Delaunay-Knoten die im Unendlichen liegen. Praktisch erhalten diese Delaunay-Knoten aber sehr gross gewählte endliche Koordinaten. Um zu verhindern, dass für die Teilbereiche des dualen Max-Diagrammes, die an einen solchen unendlichen Delaunay-Knoten grenzen, wie für gewöhnliche Teilbereiche ein endlicher Flächeninhalt berechnet wird, musste ich an der entsprechenden Stelle in der Prozedur zur Berechnung des Max-Diagrammes dafür sorgen, dass diese Delaunay-Knoten als Knoten der Halbkanten-Datenstruktur eingefügt werden, deren Flag anzeigt, dass es sich dabei um Knoten im Unendlichen handelt. Somit berechnet die Prozedur `calculate_area` für diese Teilbereiche ein Flächeninhalt von `-1` wie es für unbegrenzte Bereiche nötig ist.

Kapitel 4

Eine graphische Schnittstelle

Zum Testen der Algorithmen habe ich zusätzlich noch eine Benutzeroberfläche erstellt. Diese dient sowohl zur Eingabe der Menge von Disks, zur Manipulation dieser Daten, als auch zur graphischen Darstellung der verschiedenen im letzten Kapitel erwähnten Diagramme. Zur Erstellung der Benutzeroberfläche habe ich die LEDA-Library verwendet.
Die globale Variable `window` ist ein Zeiger auf ein Obkjekt der Klasse `leda_window`. Dieses beinhaltet die Gesamtheit aller Graphik-Objekte.
Die graphische Oberfläche besteht aus einem grossen Graphik-Fenster und einem kleineren darüberliegenden Panel das diverse Buttons zur Steuerung der grapischen Ausgabe enthält. Ausserdem gibt es auch noch die Buttons `LOAD` zum Laden einer Menge von Disks aus einem File, `SAVE` zum Speichern der aktuellen Menge von Disks in ein File, `CLEAR` zum Löschen der gesamten aktuellen Menge von Disks, `ABOUT` zur Darstellung von Informationen über das Programm selbst und `EXIT` zum Verlassen des Programmes.
Die Main-Prozedur meines Programmes besteht im wesentlichen aus einer Endlosschleife, in der ständig überprüft wird, ob entweder ein Button gedrückt wurde oder eine Manipulation der Daten im Graphik-Fenster erfolgt ist. Falls dies der Fall ist, werden die nötigen Prozeduren aufgerufen.

4.1 Eingabe und Manipulation der Daten

Die Ausgangsdaten des Programmes, die Disks, müssen zunächst im globalen Vektor `wpoints` gespeichert werden. Bei einer Veränderung der Punkt-

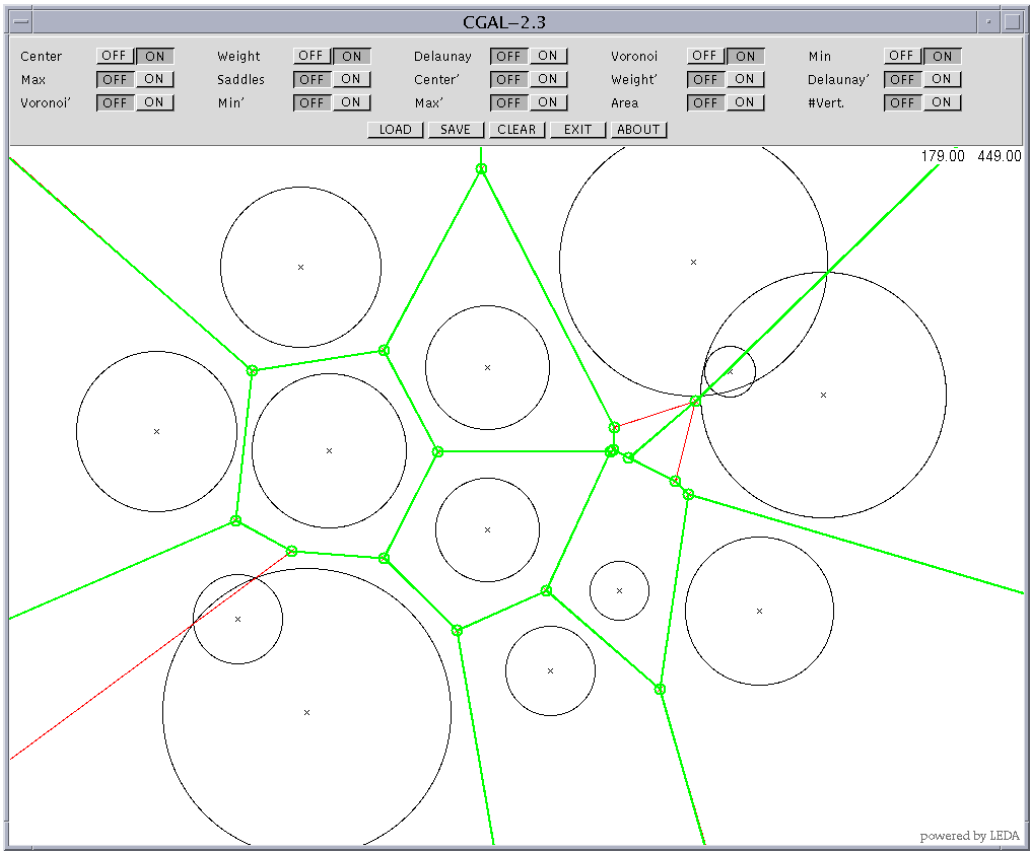


Abbildung 4.1: Die graphische Benutzeroberfläche.

Daten wird jeweils lediglich dieser Vektor aktualisiert und davon ausgehend die Delaunay-Triangulierung neu aufgebaut.
Die Eingabe der gewichteten Punkte, der Disks, kann auf zwei Arten erfolgen. Einerseits verfügt das Programm über Lade- und Speicher-Funktionen für Diskdaten-Files. Diese Files sind gewöhnliche Textfiles, die als erstes die Anzahl der Disks enthalten und danach entsprechend viele Disk-Tripel, bestehend aus x-Koordinate, y-Koordinate und quadriertem Gewicht. Die zweite Möglichkeit der Eingabe von Disks ist über das Graphik-Fenster. Ein Klick mit dem linken Maus-Button in das Graphik-Fenster ruft die Prozedur `get_input` auf. Die Koordinaten des Mittelpunktes der neuen Disk werden auf die Koordinaten des Mauszeigers zum Zeitpunkt des Klicks gesetzt und

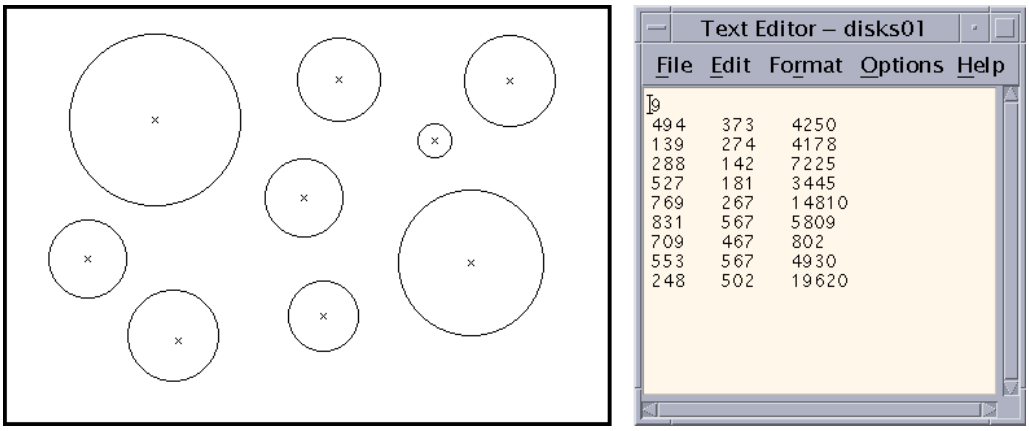


Abbildung 4.2: Eine Menge von Disks und das zugehörige Diskdaten-File.

man kann jetzt durch einen zweiten Klick den Radius definieren.
Ich habe dem Programm auch Editier- und Lösch-Funktionen hinzugefügt. Ein Mausklick mit dem mittleren Button im Graphik-Fenster ruft die Prozedur `edit_disk` auf. Befand sich der Mauszeiger zum Zeitpunkt des Klicks über dem Radius einer Disk oder ausreichend nahe am Radius einer Disk, so kann man nun durch einen weiteren Klick diesen Radius neu bestimmen. Befand sich der Mauszeiger zum Zeitpunkt des Klicks ausreichend nahe beim Mittelpunkt einer Disk, so kann man nun durch einen weiteren Klick die neue Position des Mittelpunktes dieser Disk bestimmen. Kommen mehrere Disks in Frage, so wird eine willkürlich gewählt. Ein rechter Mausklick im Graphik-Fenster ruft die Prozedur `delete_disk` auf. Es werden alle Disks gelöscht, deren Mittelpunkt sich ausreichend nahe bei der Position des Mauszeigers zum Zeitpunkt des Klicks befindet.

4.2 Graphische Darstellung

Wann immer die zugrunde liegende Menge der gewichteten Punkte verändert wird, muss die Delaunay-Triangulierung ausgehend vom Vektor `wpoints` neu aufgebaut werden und danach die Prozedur `redraw` aufgerufen werden, die alle gewünschten (d.h. durch Aktivierung des jeweiligen Buttons selektierten) Objekte im Graphik-Fenster ausgibt. Die Prozedur `redraw` wird auch aufgerufen, wenn ein Button geklickt wurde. Jeder Button zur Steuerung der

graphischen Ausgabe besitzt ein zugehöriges Boolean-Flag, das `true` ist für einen aktiven Button und `false` für einen inaktiven Button.

4.2.1 Disks

Die Buttons `Center` und `Weight`, die sich ein oder ausschalten lassen, steuern, ob die Mittelpunkte bzw. die Gewichte der Disks im Graphik-Fenster ausgegeben werden sollen.
Disk-Mittelpunkte werden als kleine schwarze Kreuze dargestellt und Disk-Gewichte als Kreise um den Disk-Mittelpunkt mit einem Radius, der dem Radius der Disk entspricht.
Der Programmcode dafür befindet sich in der Prozedur `redraw`, die aufgerufen wird, wenn die Menge der gewichteten Punkte verändert oder ein Button geklickt wurde, und wird nur ausgeführt, wenn der entsprechende Button aktiv ist.
Die Buttons `Center` und `Weight` sind die einzigen Buttons, die beim Starten des Programmes aktiv sind.

4.2.2 Gewichtete Delaunay-Triangulierung und gewichteter Voronoi-Komplex

Die graphische Ausgabe der Delaunay-Triangulierung bzw. des Voronoi-Komplex wird durch die Buttons `Delaunay` und `Voronoi` gesteuert.
Glücklicherweise existieren entsprechende Schnittstellen, so dass die Variable `rt`, die die gewichtete Delaunay-Triangulierung enthält, direkt an das LEDA-Window übergeben werden kann. Die Delaunay-Triangulierung wird grau gezeichnet.
Zur Ausgabe des Voronoi-Komplexes werden zunächst mit Hilfe eines Face-Iterators alle Delaunay-Dreiecke durchlaufen, um die dualen Voronoi-Knoten zur graphischen Darstellung an das LEDA-Window zu übergeben. Dann werden mit der Prozedur `draw_dual` der Variable `rt`, der das LEDA-Window als Eingabe-Parameter übergeben wird, auch noch die Voronoi-Kanten dargestellt. Der Voronoi-Komplex wird rot gezeichnet.
Mit dem Button `Saddles` lässt sich die Darstellung der Sattelpunkte aktivieren. Ein Sattelpunkt ist ein Schnittpunkt einer Delaunay-Kante mit ihrer dualen Voronoi-Kante. An der entsprechenden Stelle in der Prozedur `redraw` wird für jede Delaunay-Kante mit Hilfe der Prozedur `has_saddle`, die die jeweilige Kante als Eingabe verlangt, getestet, ob sie einen Sattelpunkt

enthält. Falls dies der Fall ist, dann wird der Sattelunkt mit der Prozedur `calculate_saddle`, der man die Delaunay-Kante, welche den Sattelpunkt enthält, als Eingabe übergibt, berechnet. Die Sattelpunkte werden als kleine violette Kreise dargestellt.

4.2.3 Min- und Max-Diagramm

Die Buttons `Min` und `Max` dienen der Steuerung der graphischen Ausgabe des Min- bzw. des Max-Diagrammes.
Sowohl das Min- wie auch das Max-Diagramm sind in einer globalen Variablen, `mindiagram` bzw. `maxdiagram`, der selben Klasse `plangraph` gespeichert. Aus diesem Grund sind sich die Codeabschnitte in der Prozedur `redraw` zur Darstellung der beiden Diagramme auch sehr ähnlich.
Zunächst wird jeweils die entsprechende Prozedur zur Berechnung des jeweiligen Diagrammes, `calculate_mindiagram` bzw. `calculate_maxdiagram`, aufgerufen. Danach wird der Vektor durchlaufen, indem die Zeiger auf die Halbkanten des Min- bzw Max-Diagrammes gespeichert sind. Um eine solche Halbkante darzustellen wird jeweils das Liniensegment vom Basis-Knoten dieser Halbkante zum Basis-Knoten der nächsten Halbkante gezeichnet. Zuletzt wird noch der Vektor durchlaufen, in dem alle Knoten gespeichert sind. Diese werden jeweils als kleine Kreise dargestellt. Für das Max-Diagramm werden jedoch nur die Knoten dargestellt, die Eckpunkte des aktuellen Bereiches sind. Sattelpunkte, die auf den Kanten des Bereiches liegen, werden hier nicht dargestellt.
Unendliche Punkte, d.h. Punkte deren Flag auf `0` gesetzt ist, werden genau so behandelt wie endliche Punkte mit Flag `1`. Der Unterschied liegt darin, dass die Koordinaten der unendlichen Punkte so gross gewählt wurden, dass sie ausserhalb des Graphik-Fensters zu liegen kommen. Diese Punkte werden somit zwar selber nicht dargestellt, aber die Kante, die sie mit endlichen Punkten verbindet schon. Damit wird der Eindruck einer sich ins Unendliche erstreckenden Kante erreicht.
Das Min- und Max-Diagramm werden mit etwas stärker ausgezogenen Linien als die Delaunay-Triangulierung oder der Voronoi-Komplex in grün bzw. blau gezeichnet.

4.2.4 Duale Diskmenge

Zu jedem der bisher erwähnten Buttons, ausser zum Button `Saddles`, habe ich einen analogen Button zur Darstellung des selben Diagrammes der dualen Diskmenge erstellt. Es ist überflüssig, die Sattelpunkte der dualen Diskmenge darstellen zu können, da sie mit den Sattelpunkten der ursprünglichen Diskmenge übereinstimmen. Die neuen Buttons tragen die Namen `Center'`, `Weight'`, `Delaunay'`, `Voronoi'`, `Min'` und `Max'`. Wann auch immer mindestens einer dieser Buttons aktiv ist, wird in der Prozedur `redraw` nach den Anweisungen, welche der Darstellung der Diagramme der ursprünglichen Menge von Disks dienen, der Vektor der ursprünglichen Disks `wpoint` gesichert und dann die Prozedur `calculate_orthodisks` zur Berechnung der dualen Diskmenge aufgerufen. Zusätzlich wird die Boolean-Variable `restore`, welche am Anfang der Prozedur `redraw` jeweils auf `false` initialisiert wird, auf `true` gesetzt, um anzuzeigen, dass am Ende der Prozedur `redraw` eine Wiederherstellung des ursprünglichen Vektors `wpoints` und der zugehörigen gewichteten Delaunay-Triangulierung nötig sein wird.
Die Anweisungen in der Prozedur `redraw`, welche der Darstellung der Diagramme der dualen Diskmenge dienen, entsprechen im Wesentlichen ihren Gegenstücken zur Darstellung der Diagramme der ursprünglichen Diskmenge. Es werden für die Diagramme der dualen Diskmenge Farben des selben Farbtons wie für die Diagramme der ursprünglichen Diskmenge verwendet, jedoch mit geringerer Sättigung.
Am Ende der Prozedur `redraw` wird, falls notwendig, d.h. falls die Boolean-Variable `restore` auf `true` gesetzt wurde, der Vektor der ursprünglichen Diskmenge `wpoints` mit Hilfe der erstellten Sicherungskopie wieder hergestellt und darauf aufbauend die Delaunay-Triangulierung wieder aufgebaut.

4.2.5 Flächenberechnung

Mit den Buttons `Area` und `#Vert.` lassen sich zu jedem Min- oder Max-Diagramm der ursprünglichen oder dualen Diskmenge die Flächeninhalte bzw. die Anzahl der Knoten aller Teilbereiche dieser Diagramme ausgeben. Diese Werte werden auf der Position des zum jeweiligen Teilbereichs gehörenden Extremums ausgeben.
Zunächst werden in der Prozedur `redraw` falls notwendig die Prozeduren `calculate_min_positions` bzw. `calculate_max_positions` aufgerufen, um die Positionen der Extrema zu berechnen an denen die Werte für Flächenin-

halt und Kantensumme später ausgegeben werden sollen.
Dann folgt jeweils ein Programmabschnitt für die Min- und Max-Diagramme der ursprünglichen und der dualen Diskmenge. Diese Abschnitte werden jeweils nur dann ausgeführt, wenn der Button für die Darstellung des entsprechenden Min- bzw. Max-Diagrammes aktiv ist und wenn zusätzlich entweder der Button `Area` oder `#Vert.` aktiv ist. Diese Programmabschnitte enthalten Programmschleifen, in denen jeweils jede Halbkante des entsprechenden Min- bzw. Max-Diagrammes durchlaufen wird. Für jede dieser Halbkanten wird dann mit der Prozedur `calcualte_area` der Flächeninhalt, die Position des zugehörigen Extremums sowie die Anzahl der Knoten des an die Halbkante angrenzenden Teilbereiches berechnet. Da bei wiederholter Berechnung für den selben Teilbereich, wie für unbegrenzte Bereiche, jeweils nur noch ein Flächeninhalt von `-1` zurückgegeben wird, kann eine wiederholte Darstellung von Werten für den selben Bereich vermieden werden. Es werden nur Werte für Bereiche mit Flächeninhalt ungleich `-1` dargestellt.
Je nachdem, welche der Buttons `Area` und `#Vert.` aktiv sind, wird entweder nur der Flächeninhalt, nur die Knotensumme oder aber beides hintereinander dargestellt. Die Flächeninhalte und Knotensummen von Min-Diagrammen werden in grün ausgegeben, jene von Max-Diagrammen in blau.

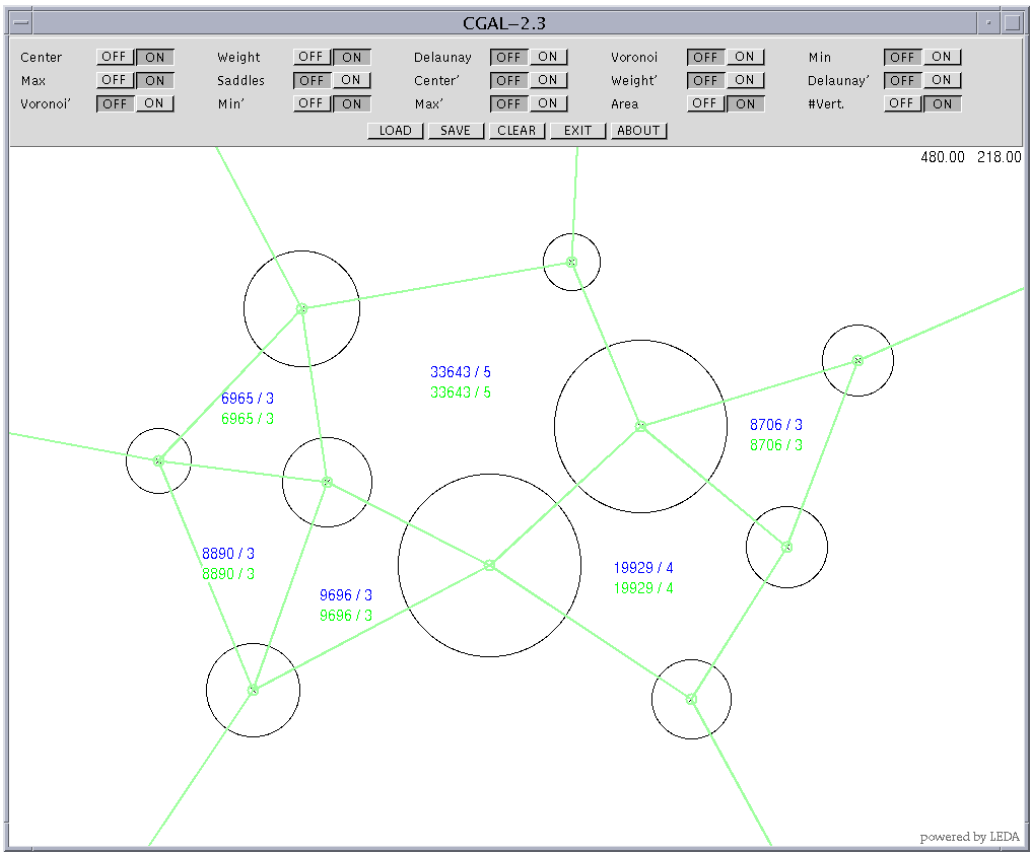


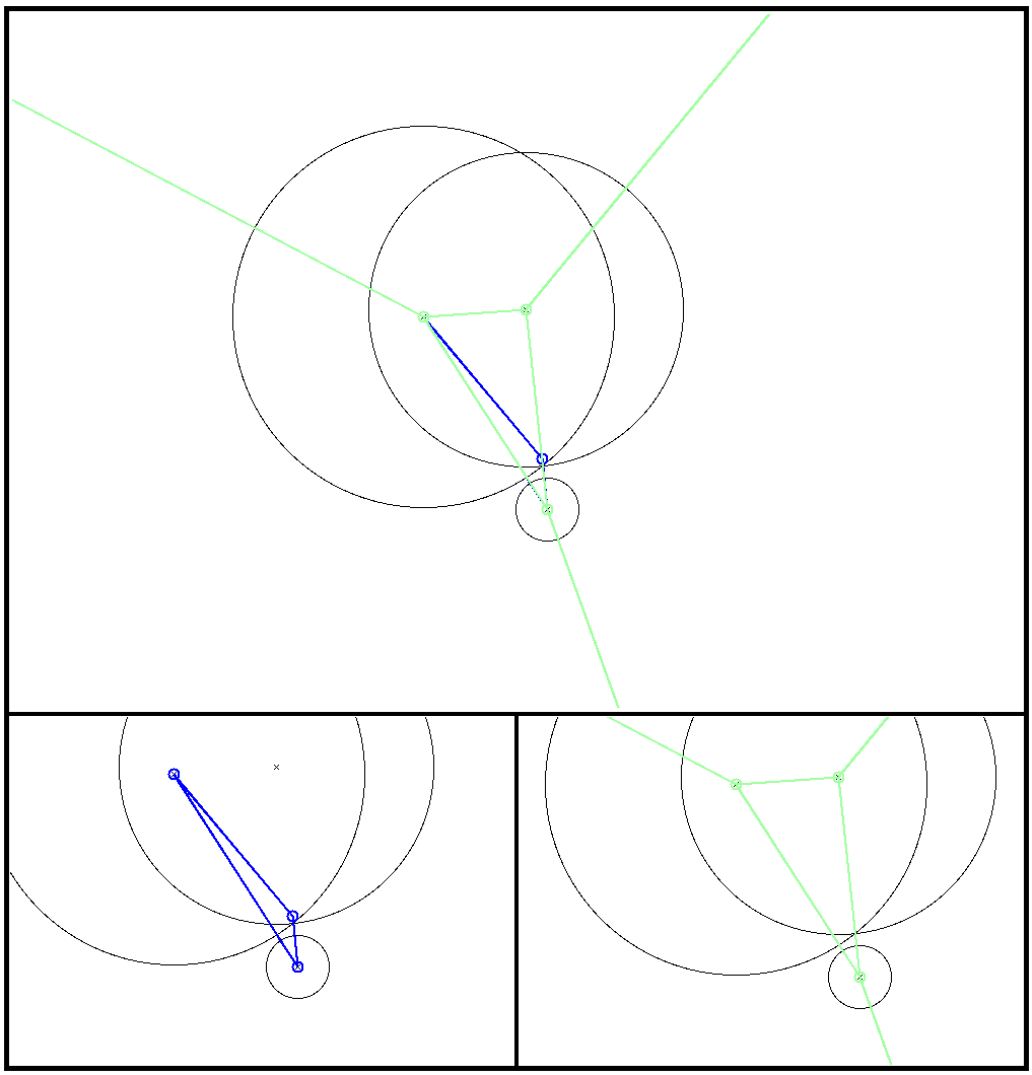
Abbildung 4.3: Eine Darstellung der Flächeninhalte und der Anzahl der Knoten der Teilbereiche des Max-Diagrammes einer Diskmenge sowie des Min-Diagrammes der dualen Diskmenge.

Kapitel 5

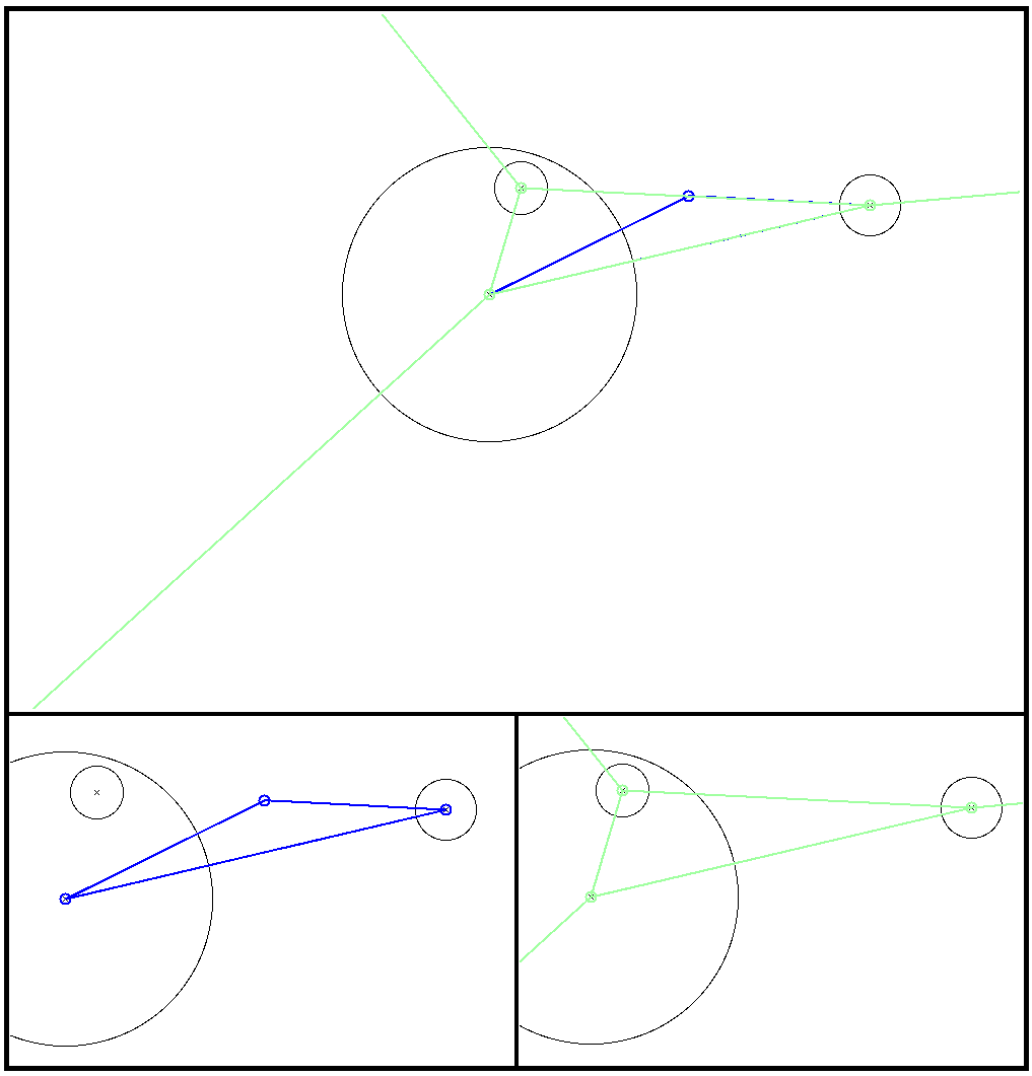
Experimente

Wie schon erwähnt entsprechen die Max-Regionen einer Menge von Disks, die die strenge Dualitätsbedingung erfüllen, genau den Min-Regionen der dazu dualen Diskmenge. Die Theorie sagt aber nichts darüber aus, wie sehr sich die Max-Regionen einer Menge von Disks und die Min-Regionen der dazu dualen Diskmenge ähnlich sind, wenn die ursprüngliche Diskmenge die strenge Dualitätsbedingung nicht erfüllt. Wenn man wüsste, dass sich die Max-Regionen der ursprünglichen Diskmenge, auch für solche allgemeinen Fälle, von den Min-Regionen der dualen Diskmenge nicht alzu sehr unterscheiden, dann könnte man die Min-Regionen der dualen Diskmenge, die sich einfacher und stabiler berechnen lassen als Max-Regionen, als Approximation für die Max-Regionen der ursprünglichen Diskmenge verwenden.
Um ein Gefühl für diese Unterschiede zu bekommen, habe ich einige Experimente durchgeführt.

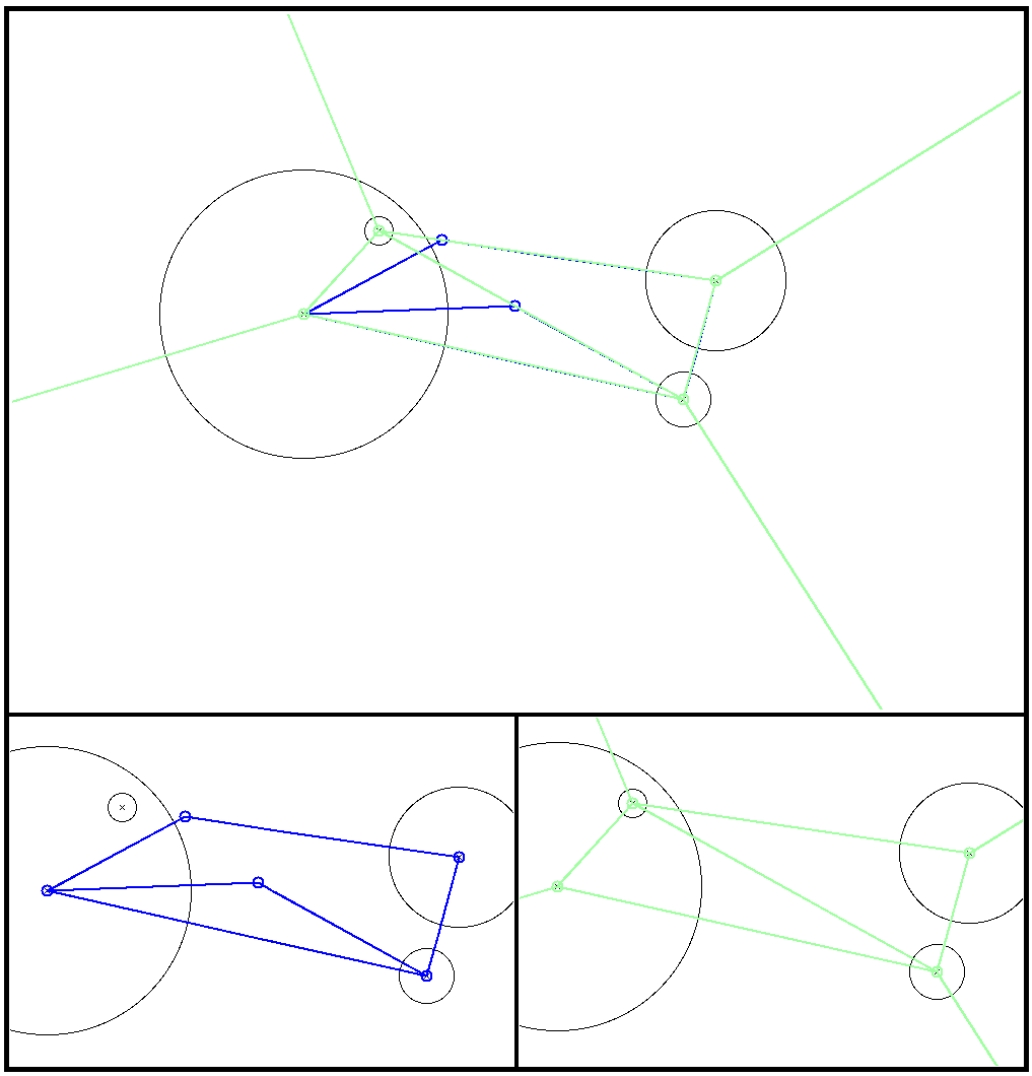
Die erste Diskmenge besteht lediglich aus zwei grossen und einer kleineren Disk. Die Diskmenge verletzt die strenge Dualitätsbedingung. Die Max-Region der ursprünglichen Diskmenge (blau) ist deutlich kleiner als die Min-Region der dualen Diskmenge (grün).



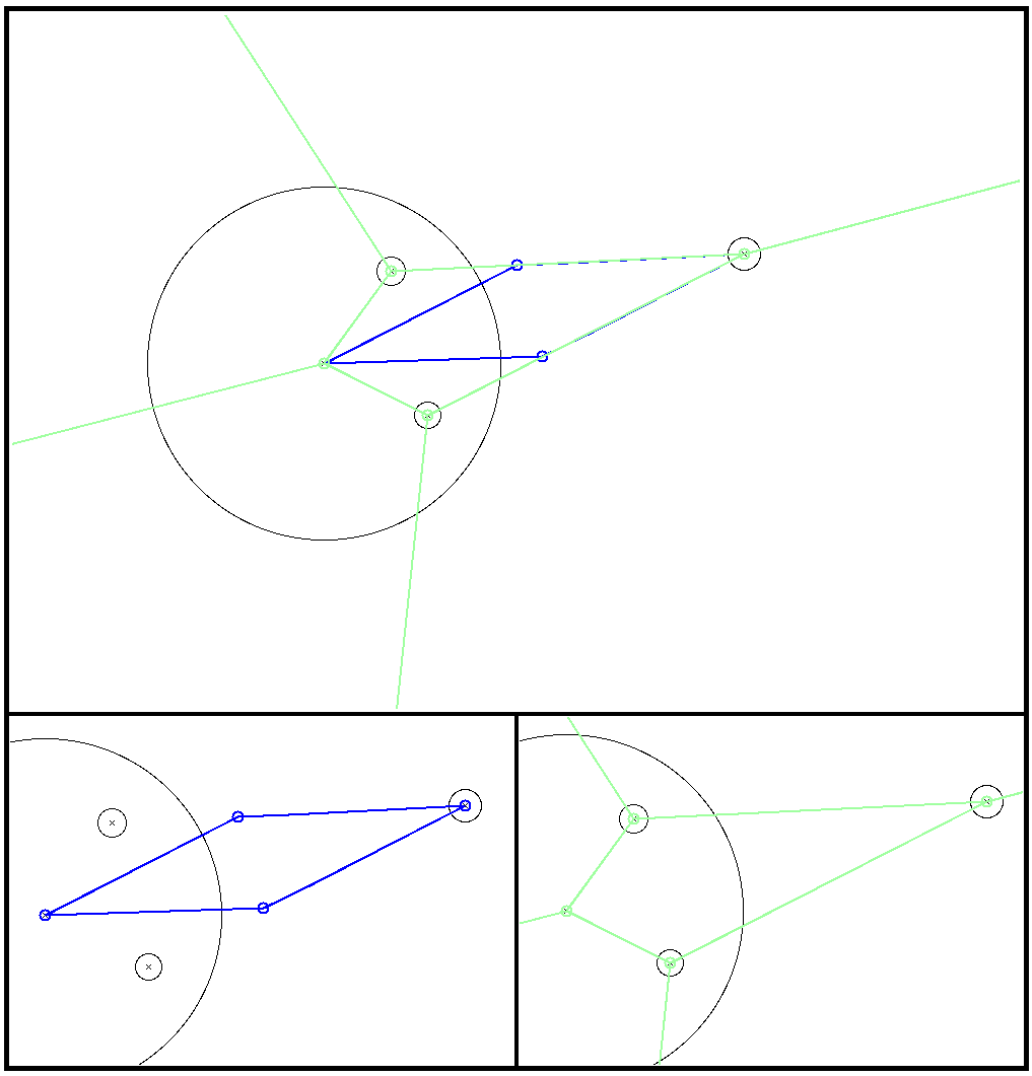
Die nächste Diskmenge besteht aus einer grossen und zwei kleineren Disks. Auch hier ist die Max-Region der ursprünglichen Diskmenge (blau) deutlich kleiner als die Min-Region der dualen Diskmenge (grün).



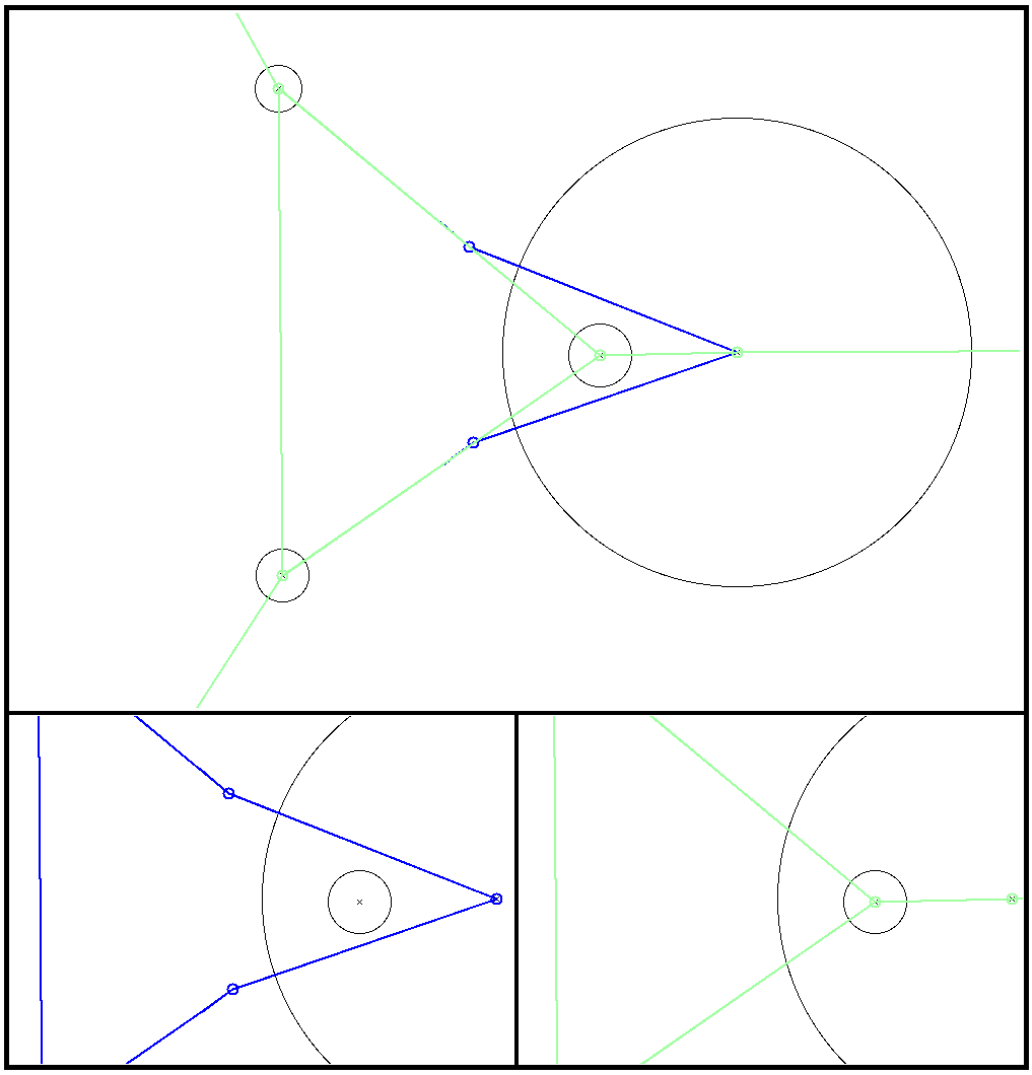
Für diese Diskmenge besitzt das Max-Diagram der ursprünglichen Diskmenge (blau) zwei endliche Regionen genau wie das Min-Diagramm der dualen Diskmenge (grün). Interessant ist jeweils die stark unterschiedliche Form, Grösse und Knotenzahl der einander entsprechenden Regionen.



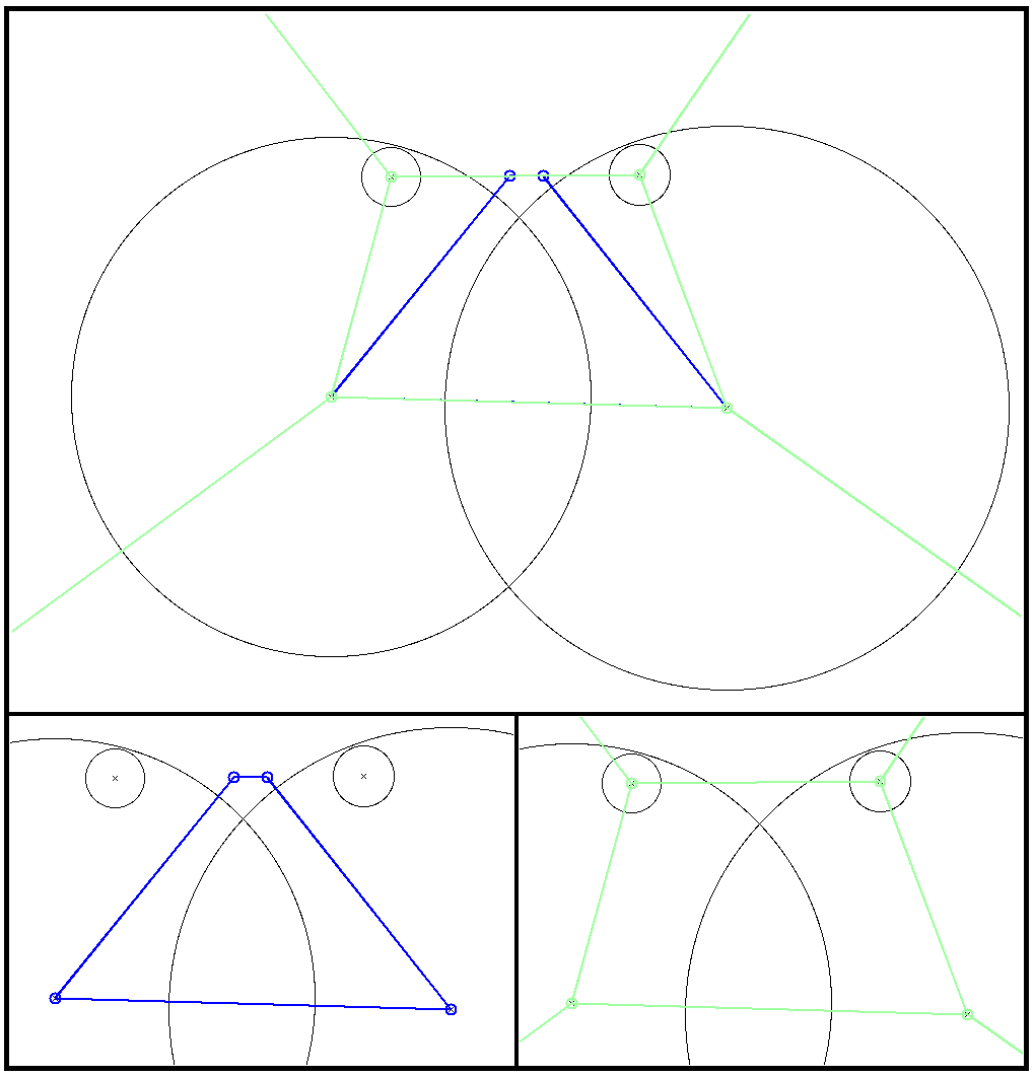
Diese Diskmenge besteht aus einer grossen und drei kleineren Disks. Die Max-Region der ursprünglichen Diskmenge (blau) ist deutlich kleiner als die Min-Region der dualen Diskmenge (grün).



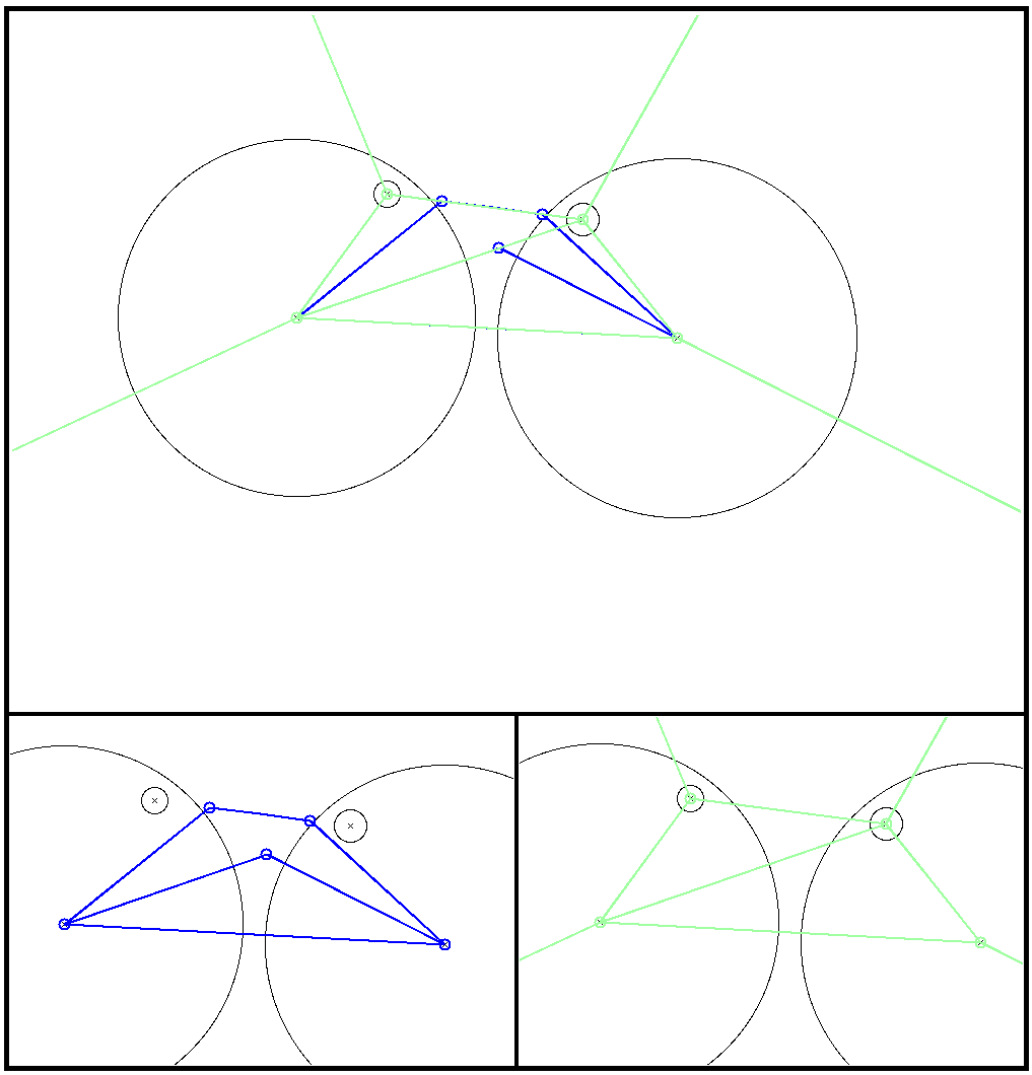
Auch diese Diskmenge besteht aus einer grossen und drei kleineren Disks. Für diese Diskmenge ist die Max-Region der ursprünglichen Diskmenge (blau) grösser als die Min-Region der dualen Diskmenge (grün). Die Max-Region der ursprünglichen Diskmenge besitzt 5 Knoten, während die Min-Region der dualen Diskmenge nur 3 hat.



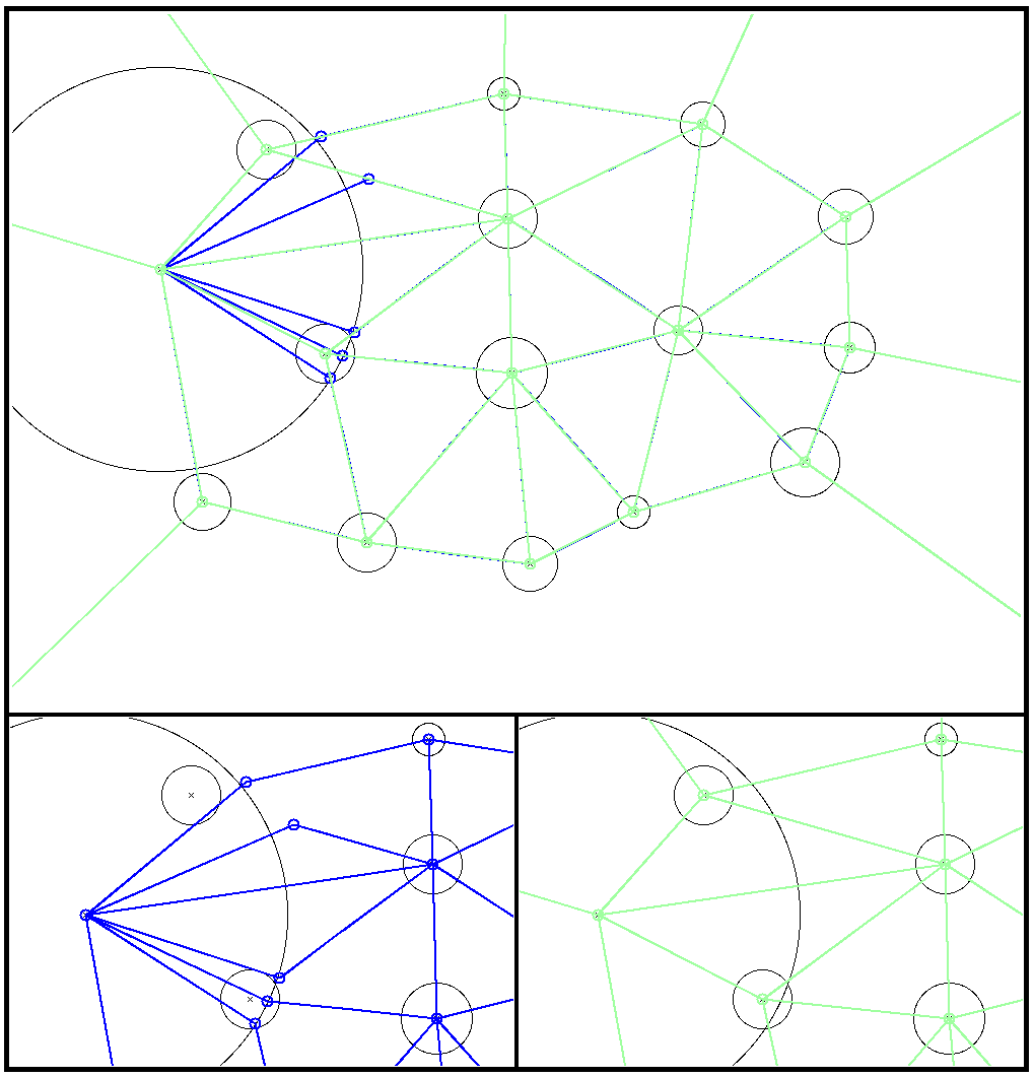
Für diese Diskmenge aus zwei grossen und zwei kleineren Disks ist die Max-Region der ursprünglichen Diskmenge (blau) deutlich kleiner als die Min-Region der dualen Diskmenge (grün).



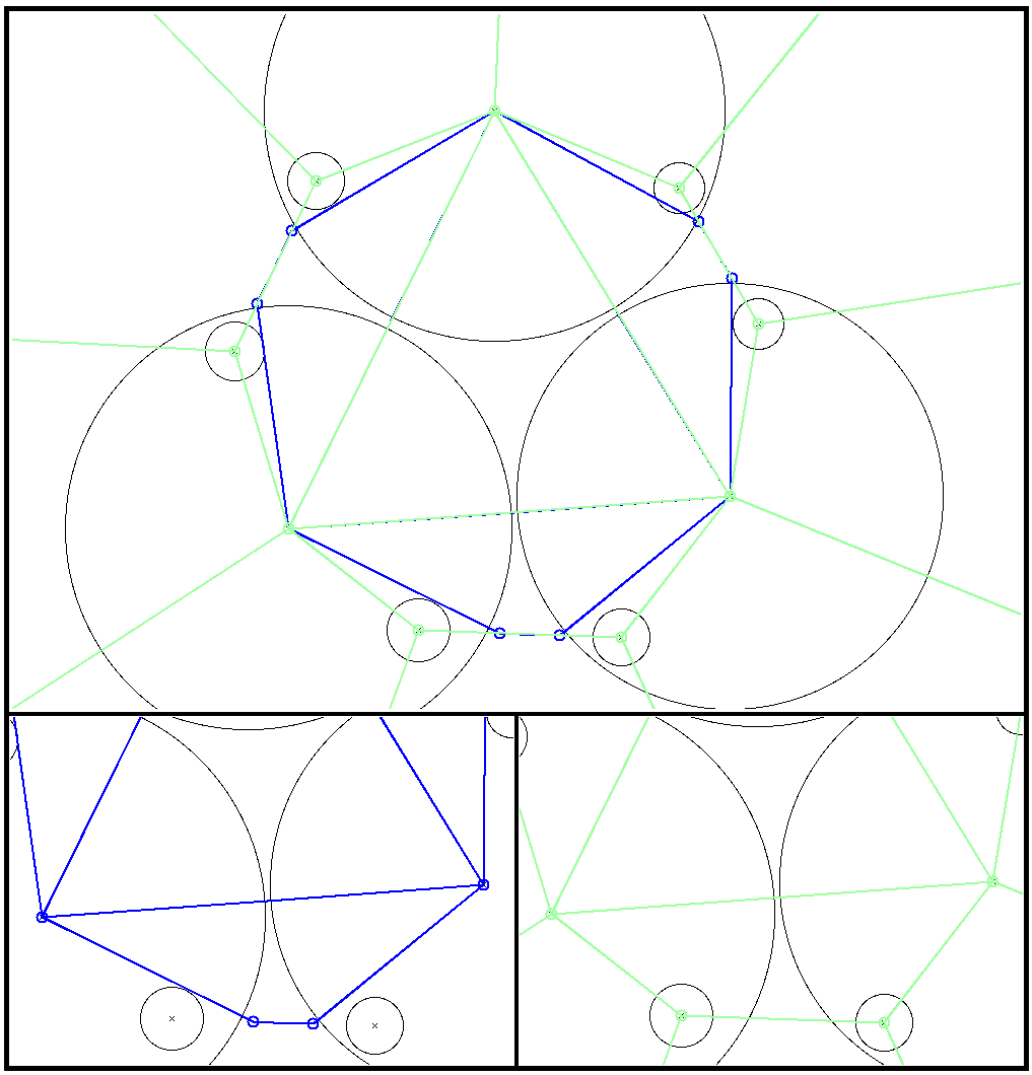
Das Auffälligste bei dieser Diskmenge aus zwei grossen und zwei kleineren Disks sind die sehr unterschiedlichen Formen der Max-Regionen der ursprünglichen Diskmenge (blau) und der entsprechenden Min-Regionen der dualen Diskmenge (grün).



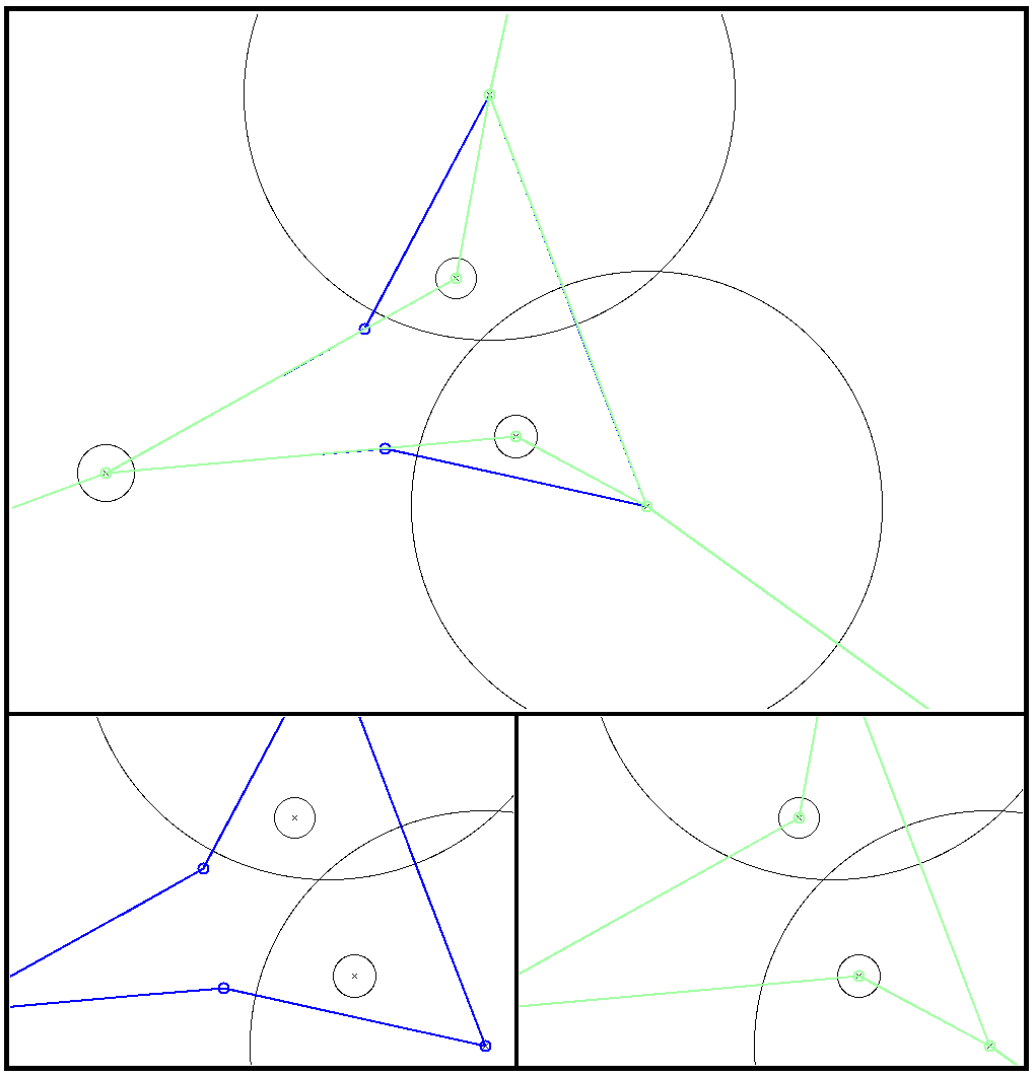
Hier habe ich für eine etwas grössere Menge von Disks das Max-Diagramm (blau) mit dem Min-Diagramm der dualen Diskmenge (grün) verglichen. Hier sieht man deutlich, dass Abweichungen zwischen den beiden Diagrammen auftreten, wenn die strenge Dualitätsbedingung verletzt wird. Die Abweichungen zwischen den Diagrammen treten aber nur lokal auf und zwar dort, wo die strenge Dualitätsbedingung verletzt wurde. Die strenge Dualitätsbedingung wird verletzt, wenn benachbarte Disks stark unterschiedliche Radien aufweisen.



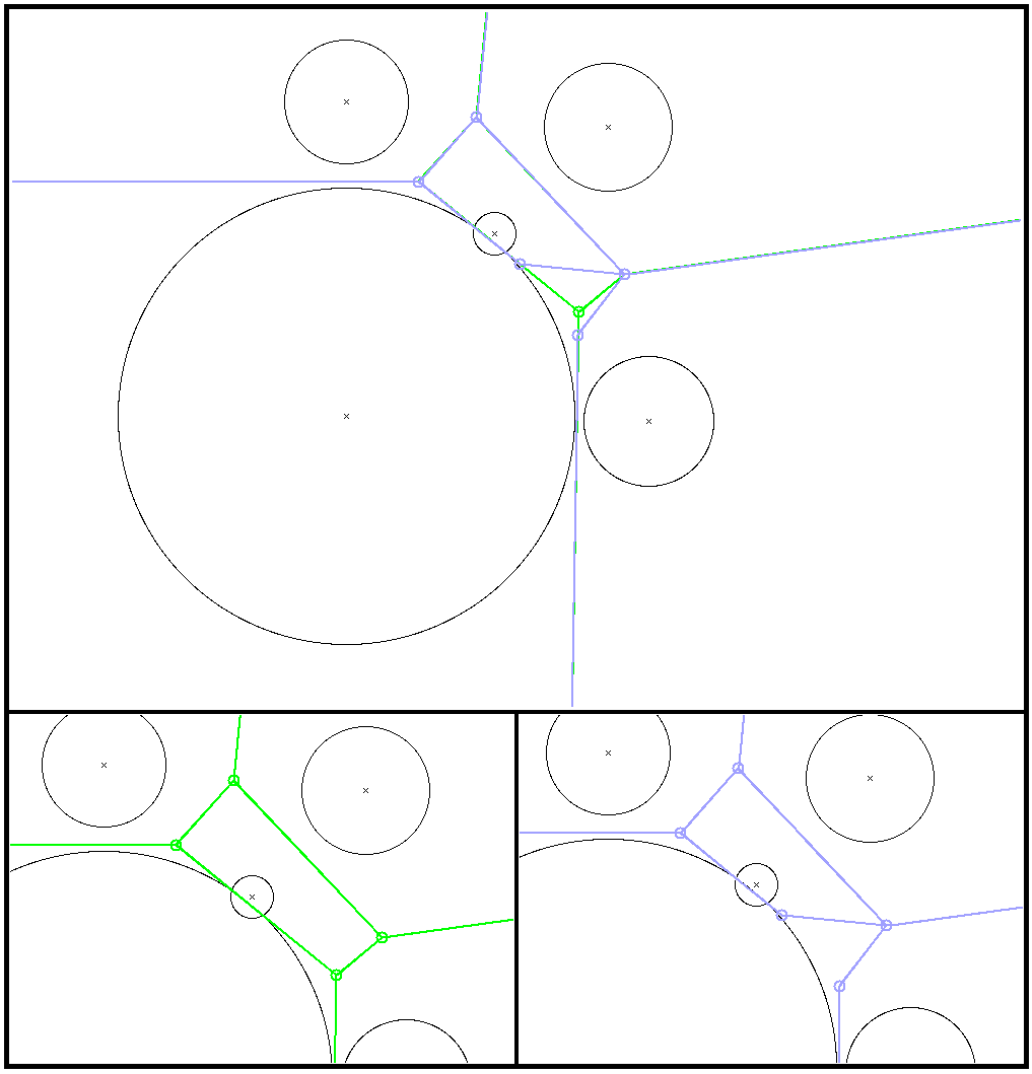
Dieses Beispiel zeigt, dass sich auch grössere Mengen von Disks konstruieren lassen, für die der gesamte Flächeninhalt aller Max-Regionen (blau) eindeutig kleiner ist als der gesamte Flächeninhalt aller Min-Regionen der dualen Diskmenge (grün).



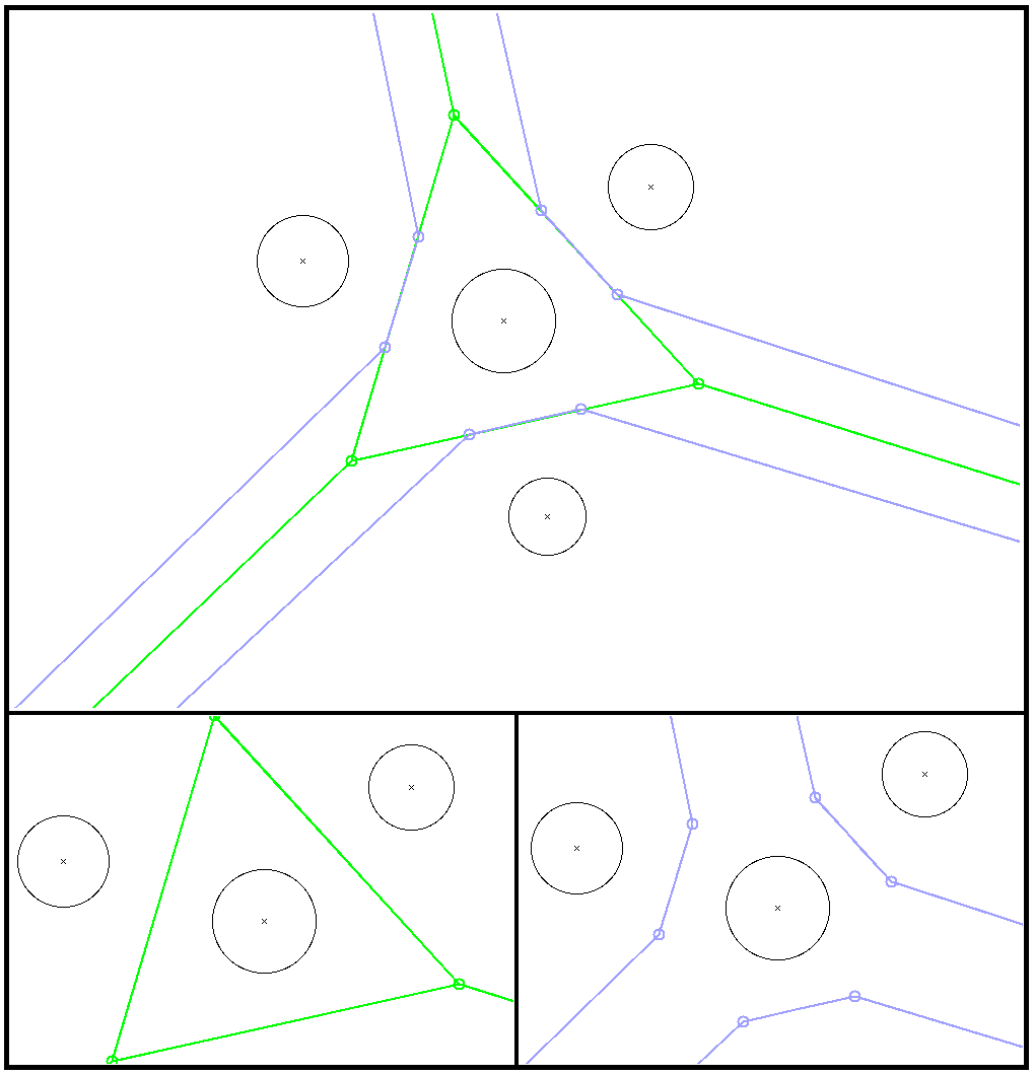
Für diese Diskmenge ist, im Gegensatz zur letzten, der gesamte Flächeninhalt aller Max-Regionen (blau) grösser als der gesamte Flächeninhalt aller Min-Regionen der dualen Diskmenge (grün).



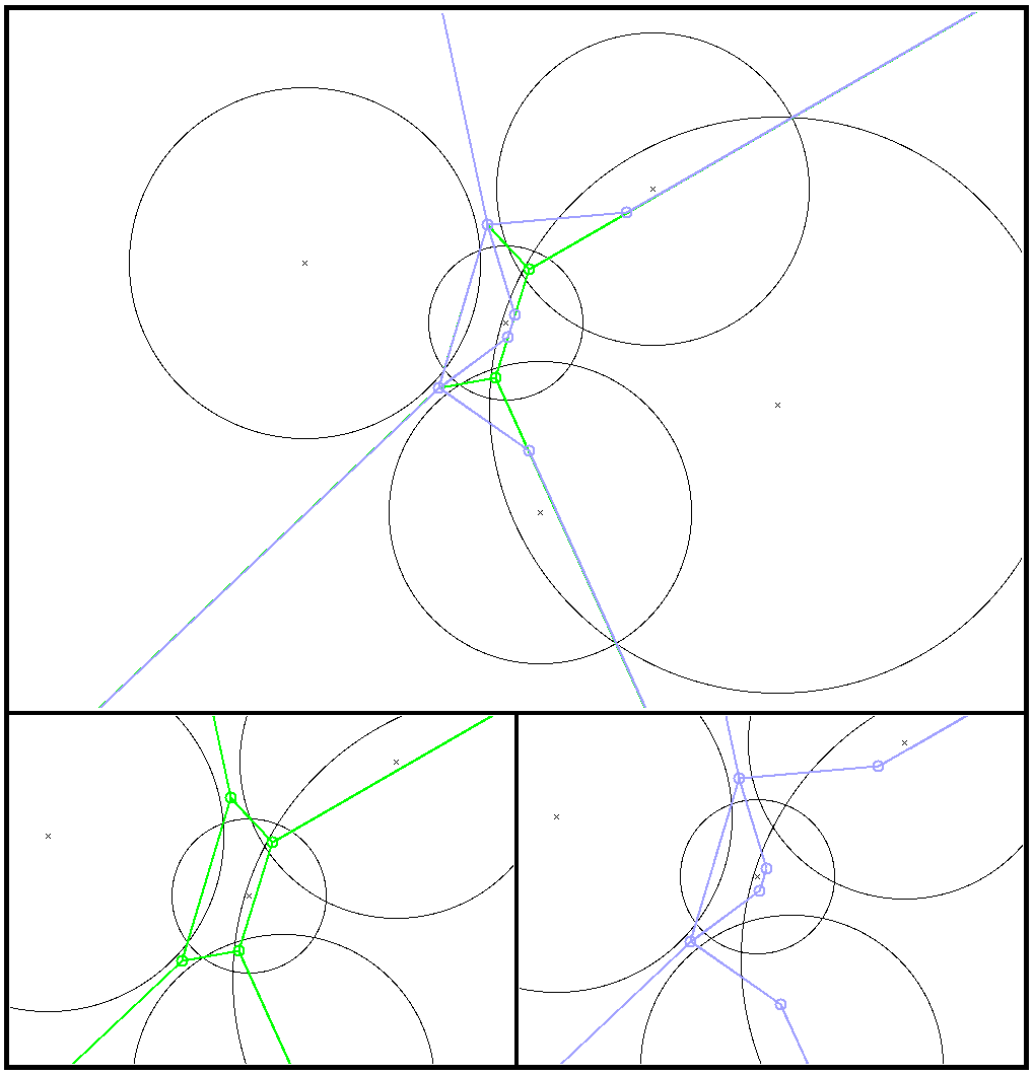
Für diese Diskmenge habe ich die Min-Regionen mit den Max-Regionen der dazu dualen Diskmenge verglichen. Um endliche Min-Regionen zu erhalten braucht man übrigens jeweils mindestens vier Knoten. Die Min-Region der ursprünglichen Diskmenge (grün) ist kleiner als die Max-Region der dualen Diskmenge (blau).



Für diese Diskmenge existiert zwar eine endliche Min-Region (grün), doch die zugehörige Max-Region der dualen Diskmenge (blau) ist unbegrenzt.



Dies ist noch einmal eine Diskmenge, für die die Min-Region (grün) deutlich kleiner ist als die Max-Region der dualen Diskmenge (blau).



Wie die Beispiele gezeigt haben, lassen sich ohne weiteres Diskmengen konstruieren, die die strenge Dualitätsbedingung verletzen und für die die Max-Regionen sich sehr stark von den entsprechenden Min-Regionen der dualen Diskmenge unterscheiden - sowohl in deren Flächeninhalt als auch in der Anzahl der Knoten sowie in der Form. Diese Beispiele weisen an Stellen, wo sich die Max-Regionen der ursprünglichen Diskmenge von den Min-Regionen der dualen Diskmenge unterscheiden, jeweils sehr stark unterschiedliche Radien von benachbarten Disks auf. Die Frage die sich nun stellt ist, ob solch starke Unterschiede bei den Radien benachbarter Disks auch bei Eingabedaten mit realem Bezug, wie z.B bei der Repräsentation eines Makromoleküls, auftreten würden. Dazu wäre eine genauere Betrachtung der Verhältnisse der Atomradien benachbarter Atome in solchen Molekülen notwendig.

Literaturverzeichnis

[1] Joachim Giesen, Matthias John
Dynamical Systems from Disks in the Plane
ETH Zürich, CH-8092 Zürich, Switzerland

[2] Joachim Giesen, Matthias John
A new Diagram from Disks in the Plane
ETH Zürich, CH-8092 Zürich, Switzerland

[3] S.W. Cheng, H. Edelsbrunner, P. Fu and K.P. Lam
Design and Analysis of Planar Shape Deformation
Proc. 14th Symp. Comp. Geom. pp. 29–38, (1998)

[4] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf
Computational Geometry - Algorithms and Applications
2nd Edition
Springer (2001)

[5] Bjarne Stroustrup
Die C++ Programmiersprache
ADDISON-WESLEY

[6] CGAL Homepage:
`http://www.cgal.org/`

[7] LEDA Homepage:
`http://www.algorithmic-solutions.com/as_html/products/leda/products_leda.html`