

2. Runde
des
42. Bundeswettbewerbs Informatik

Aufgabe 3: Die Siedler

Bearbeitet
von
Florian Bange
Teilnahme-ID: 71639

15. April 2024

Inhaltsverzeichnis

1 Einleitung	2
2 Mathematische Grundlagen	3
2.1 Mengen	3
2.2 Punkte und Strecken in der Ebene	3
2.3 Kreise und Punkte in Kreisen	5
2.4 Polygone und ihre Eigenschaften	6
2.5 Punkte in Polygonen	8
2.6 Polygone mit Loechern	11
3 Definition des Problems und der Erweiterungen I - III	13
3.1 Originales Problem	13
3.2 Erweiterung I: Variable Abstaende	13
3.3 Erweiterung II: Polygone mit Loechern - Gewaesser	15
3.4 Erweiterung III: Beachtung des Flaecheninhalts	16
4 Komplexitaet des Problems	17
5 Loesungsvorschlaege zur Erweiterung I	17
5.1 Triangulierung	17
5.2 Pseudozufaellige ganzzahlige Punkte im Polygon	18
5.3 Loesungsvorschlag	18
5.4 Wahl der Punkte	19
5.5 Analyse der Zeitkomplexitaet der Loesungsvorschlaege	21
6 Erweiterung II: Polygone mit Loechern - Gewaesser	23
7 Erweiterung III: Beachtung des Flaecheninhalts	23
7.1 Kreise in Polygonen	23
7.2 Kreise in Kreisen	24
7.3 Loesungsvorschlag	25
7.4 Bestimmung der Radien	25
7.5 Analyse der Zeitkomplexitaet der Loesung	27
8 Implementierung	28
9 Beispiele	29
9.1 Die Beispiele der BwInf-Website	29
9.2 Erweiterung III	42
10 Quellcode	46
10.1 Erweiterung I	46
10.2 Erweiterung III	51
11 Literatur	55

1. Einleitung

Dieses Dokument beinhaltet meine Dokumentation der dritten Aufgabe¹ der zweiten Runde des 42. Bundeswettbewerbs Informatik² in den Jahren 2023 und 2024.

In dieser Aufgabe ist ein Polygon mit $n \in \mathbb{N}$ Punkten im zwei-dimensionalen Raum gegeben. Nun sollen in dem Gebiet des Polygons die Position eines Gesundheitszentrums, sowie moeglichst viele Positionen fuer Ortschaften gefunden werden, sodass die Weitergabe von Krankheiten zwischen den Ortschaften nicht moeglich ist. Dies ist der Fall, wenn zwei Ortschaften mindestens 20km voneinander entfernt sind. Auch sind Ortschaften mit einer Entfernung von maximal 85km zum Gesundheitszentrum vollstaendig geschuetzt. Weiter sollen alle Ortschaften je mindestens 10km voneinander entfernt sein.

Dieses Optimierungsproblem wird in Kapitel 2 und Kapitel 3 formalisiert. Dabei werden in Kapitel 2 Polygone und Kreise im zweidimensionalen Raum und verschiedene Eigenschaften (ueberschlagene, konvexe, konkave Polygone) definiert. Insbesondere wird dabei die Menge der Punkte innerhalb eines Kreises und innerhalb eines Polygons, sowie auf deren Raendern definiert. Ebenfalls werden Polygone mit Loechern definiert, wobei ebenfalls die Menge der Punkte innerhalb und auf diesen Polygonen definiert wird. D.h., Ortschaften und das Gesundheitszentrum werden durch Punkte innerhalb des Polygons dargestellt. Damit koennen anschlieszend in Kapitel 3 das originale Problem, sowie drei Erweiterungen definiert werden. Dabei werden in der Definition des Problems ausschlieslich ganzalige Punkte im zwei-dimensionalen Raum betrachtet. Dadurch wird eine Diskretisierung vorgenommen, die eine deutliche Vereinfachung der Darstellung der Punkte durch einen Computers darstellt. Um dabei eine gute Genauigkeit zu gewaehren werden die Punkte in der Einheit Millimeter definiert. Weiter wird sofort die erste Erweiterung eingefuehrt, welche variable Mindest-, bzw. Maximaldistanzen verwendet, statt die Festen (85km, 20km und 10km) verwendet. Daraufhin wird Erweiterung II definiert. Dabei werden Polygone mit Loechern behandelt. Diese koennen beispielsweise Gewaesser oder unbewohnbare Gebiete darstellen. Zuletzt wird Erweiterung III definiert, welche die Ortschaften durch Kreise darstellt, welche vollstaendig innerhalb des Polygons liegen sollen. Deren gemeinsamer Flaecheninhalt soll maximiert werden.

Die Formalisierung ermoeglicht zunaechst eine Betrachtung der Komplexitaet des Problems in Kapitel 4. Dabei laesst sich vermuten, dass das Problem NP-schwer ist. Dies laesst weiter vermuten, dass keine effiziente (d.h., polynomielle) Loesung des Problems existiert. Der Grund fuer diese Vermutung besteht in der Aehnlichkeit des Problems zu einigen als NP-vollstaendigen bzw. NP-schweren Problemen.

Anschlieszend werden in Kapitel 5 Loesungsvorschlaege fuer Erweiterung I in Form von Approximationen der Loesung vorgestellt. Dazu werden regelmaessig und dicht verteilte Punkte innerhalb des Polygons zufaellig erzeugt. Um diese Punkte zu erzeugen, wird das Polygon zunaechst trianguliert. Als Teilmenge dieser Punkte kann dann eine moeglichst gute Loesung gefunden werden. Dazu werden zwei verschiedene Verfahren vorgestellt. Diese Algorithmen werden anschlieszend hinsichtlich ihrer Zeitkomplexitaet analysiert.

In den darauffolgenden Kapiteln werden die bereits definierten Erweiterungen II und Erweiterung III des Problems behandelt. Dabei wird in Kapitel 6 Erweiterung II erwaeht, welche mit den Verfahren aus Kapitel 5 geloest werden kann, wenn man die Triangulierung an Polygone mit Loechern anpasst. Dies wurde leider nicht mehr behandelt, oder implementiert.

Weiter wird in Kapitel 7 Erweiterung III geloeost, indem zunaechst einige Verfahren zur Berechnung eines groeszt moeglichen Kreises mit festem Mittelpunkt in einem Polygon, bzw. in einem Polygon behandelt wird. Darauf aufbauend wird ein aehnliches Greedy-Verfahren verwendet, wie in Kapitel 5. Zu der letzten Erweiterungen wurde ebenfalls die Zeitkomplexitaet der Loesung diskutiert.

In Kapitel 10 wird kurz auf die konkreten Implementationsen der Loesungsvorschlaege eingegangen. Fuer diese wurden C++ und Python genutzt. Auf die wichtigsten Teile des dazugehoerige Quellcodes wird in Kapitel 12 eingegangen. Weiter werden in Kapitel 11 Beispiele zu den vorgestellten Problemen vorgestellt, wobei die der BwInf-Website vollstaendig geloeost werden. Diese werden auch fuer Erweiterung III verwendet.

¹Siehe <https://bwinfo.de/fileadmin/bundeswettbewerb/42/aufgaben422.pdf>

²Siehe <https://bwinfo.de/bundeswettbewerb/42/2/>

2. Mathematische Grundlagen

In diesem Kapitel sollen die Grundlagen der folgenden Definition des Problems und dessen Erweiterungen gegeben werden. Dabei werden verschiedene Objekte der analytischen Geometrie eingefuehrt. Darunter Punkte, Strecken, Strahlen, Geraden, Kreise und Polygone im der zweidimensionale Ebene \mathbb{R}^2 . Dabei wird insbesondere die Teilmenge der ganzzahligen Punkte $\mathbb{Z}^2 \subset \mathbb{R}^2$ betrachtet, da in den Loesungsvorschlaegen ebendiese verwendet werden. Fuer Kreisen und Polygonen wird eine moeglichst gute Definition dafuer angestrebt, dass ein Punkte (reell oder ganzzahlig) innerhalb oder auf dem Rand eines Kreises oder Polygons liegt.

2.1. Mengen

Definition 1: Mengenoperationen

Es seien A, B Mengen nach Cantor. Dann wird fuer den mengentheoretische Schnitt $A \cap B$, fuer die Vereinigung $A \cup B$ und fuer die Differenz $A - B$ geschrieben. Fuer die Vereinigung oder den Schnitt einer endlichen Anzahl an Mengen A_1, A_2, \dots, A_n mit $n \in \mathbb{N}$ wird im folgenden

$$\bigcup_{i=1}^n A_i \quad \text{und} \quad \bigcap_{i=1}^n A_i$$

verwendet. Auszerdem wird die Maechtigkeit einer Menge M durch $\#(M)$ angegeben.

2.2. Punkte und Strecken in der Ebene

Definition 2: Reelle und ganzaligen zweidimensionale Punkte

Es seien $x, y \in \mathbb{R}$ reelle Zahlen. Dann ist (x, y) ein 2-Tupel, wobei x das erste und y das zweite Element ist. Weiter ist

$$\mathbb{R}^2 := \{(x, y) : x, y \in \mathbb{R}\}$$

die Menge aller zweidimensionalen Punkte mit rellen Koordinaten. Fuer \mathbb{R}^2 sind die Addition, Subtraktion, und Skalarmultiplikation wie bekannt definiert. Beschräenk man sich in der Ebene auf gannzahlige Zahlen, erhaelt man die Menge

$$\mathbb{Z}^2 := \{(x, y) : x, y \in \mathbb{Z}\} \subset \mathbb{R}^2$$

der zweidimensionalen Punkte mit gannzahligen Koordinaten.

Definition 3: Euklidische Distanz zwischen Punkten

Es seien $A, B \in \mathbb{R}^2$ mit $A = (x_1, y_1)$ und $B = (x_2, y_2)$ gegeben. Dann ist die euklidische Distanz zwischen diesen Punkten gegeben durch

$$d(A, B) := \|A - B\| := \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \in \mathbb{R}$$

Diese Definition der Distanz stimmt mit dem anschaulichen Abstand ueberein und entspricht dem Satz des Pythagoras.

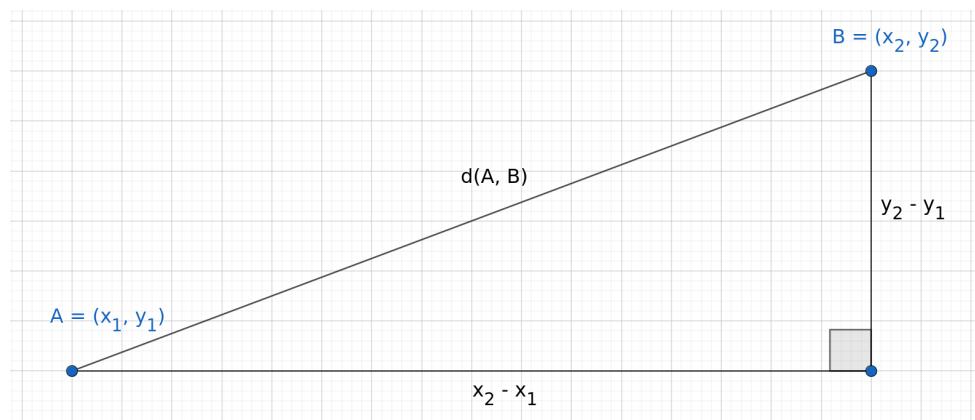


Abbildung 1: Distanz zwischen den Punkten A und B nach Pythagoras, wobei $x_2 > x_1$ und $y_2 > y_1$

Definition 4: Strecke zwischen zwei Punkten

Es seien $A, B \in \mathbb{R}^2$. Dann ist die Strecke $[AB]$ definiert als die Menge aller Punkte X mit

$$X = A + t \cdot (B - A)$$

fuer $t \in [0, 1] = \{x \in \mathbb{R} : 0 \leq x \leq 1\}$:

$$[AB] := \left\{ A + t \cdot (B - A) : t \in [0, 1] \right\}$$

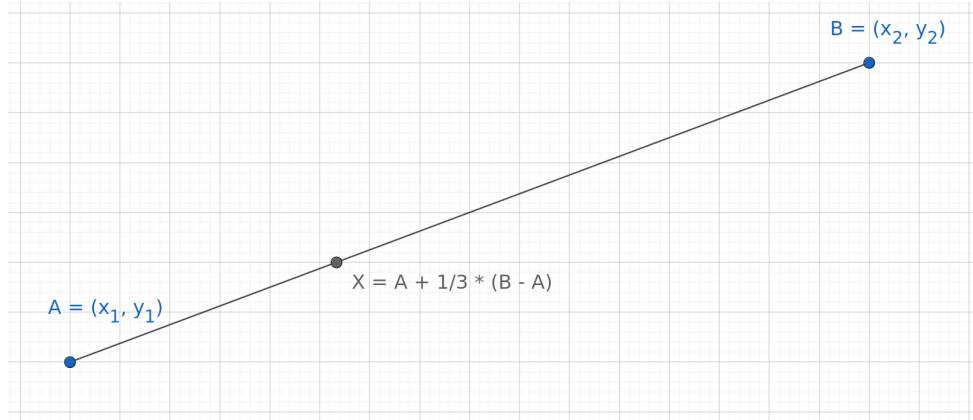


Abbildung 2: Punkt X auf der Strecke $[AB]$ mit $X = A + \frac{1}{3} \cdot (B - A)$

Definition 5: Gerade

Es seien $A, B \in \mathbb{R}^2$. Dann ist die Gerade AB durch die Punkte A durch B definiert als die Menge aller Punkte X mit

$$X = A + t \cdot (B - A)$$

fuer $t \in \mathbb{R}$:

$$AB := \left\{ A + t \cdot (B - A) : t \in \mathbb{R} \right\}$$

Definition 6: Strahlen

Es seien $A, B \in \mathbb{R}^2$. Dann ist der Strahl $[AB$ ausgehend von A durch B definiert als die Menge aller Punkte X mit

$$X = A + t \cdot (B - A)$$

fuer $t \in \mathbb{R}$ mit $t \geq 0$:

$$[AB] := \left\{ A + t \cdot (B - A) : t \in \mathbb{R} \text{ mit } t \geq 0 \right\}$$

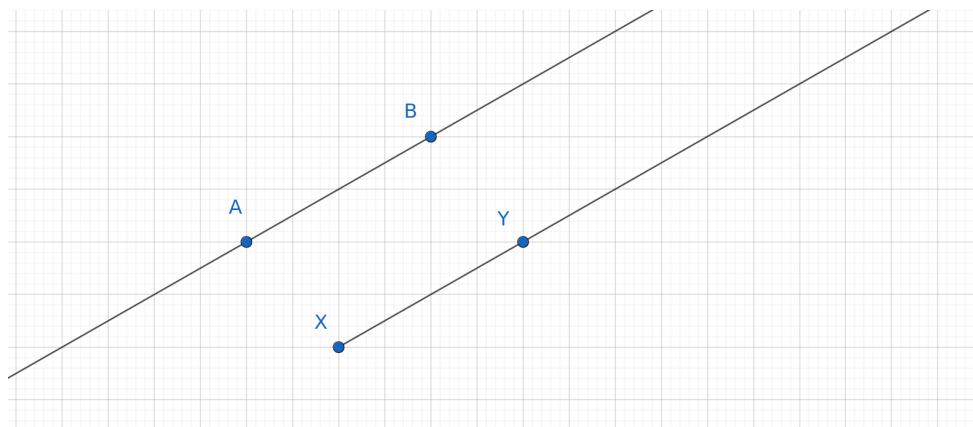


Abbildung 3: Grade AB und Strahl $[XY$

Bemerkung 7: Geraden, Strahlen und Strecken

Es ist zu bemerken, dass fuer zwei Punkte $A, B \in \mathbb{R}^2$ die Definition der Geraden, eines Strahl und einer Strecke sehr aehnlich und sie besitzen folgende Eigenschaft:

$$[AB] \subset [AB \subset AB]$$

Definition 8: Winkel zwischen Geraden

Es seien AB und XY zwei Geraden im Raum, die nicht parallel sind. In der euklidischen Geometrie (und somit auch in der analytischen Geometrie) schneiden sich diese Geraden. Es sei $Z = (x, y)$ dieser Schnittpunkt. Dann ist der Winkel α zwischen AB und XY definiert als die Laenge des Kreisbogens des Kreises $K = (Z, 1)$ (Mittelpunkt Z und Radius 1), der von AB in mathematisch positiver Richtung (gegen den Uhrzeigersinn) bis zu XY geht. Dabei gilt stets $0 < \alpha < \pi$. Weiter ist diese Definition sinnvoll, da dieser Wert auf beiden Seiten der Grade XY nach dem Nebenwinkelsatz derselbe ist.

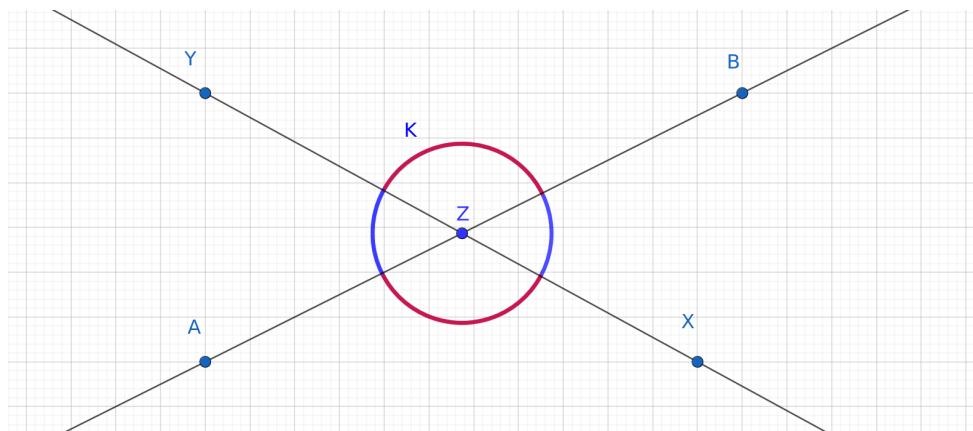


Abbildung 4: Winkel zwischen AB und XY mit Winkel zwischen AB und XY (rot) und Winkel zwischen XY und AB (blau)

2.3. Kreise und Punkte in Kreisen

Definition 9: Kreise in der Ebene

Es sei $M \in \mathbb{R}^2$ und $r \in \mathbb{R}$. Dann ist $K = (M, r)$ ein Kreis in der zweidimensionalen Ebene.

Definition 10: Menge der Punkte innerhalb eines Kreises

Es sei $K = (M, r)$ ein Kreis mit $M \in \mathbb{R}^2$ und $r \in \mathbb{R}$. Dann ist

$$\delta(K) := \left\{ A \in \mathbb{R}^2 : d(A, M) < r \right\}$$

die Menge aller Punkte innerhalb des Kreises K . Diese Definition laesst sich einfach auf ganzzahlige Punkte einschraenken:

$$\delta_{\mathbb{Z}}(K) := \left\{ A \in \mathbb{Z}^2 : d(A, M) < r \right\} \subset \delta(K)$$

Definition 11: Menge der Punkte auf einem Kreis

Es sei $K = (M, r)$ ein Kreis mit $M \in \mathbb{R}^2$ und $r \in \mathbb{R}$. Dann ist

$$\beta(K) := \left\{ A \in \mathbb{R}^2 : d(A, M) = r \right\}$$

die Menge aller Punkte auf dem Rand des Kreis K . Auch diese Definition laesst sich auf ganzzahlige Punkte einschraenken:

$$\beta_{\mathbb{Z}}(K) := \left\{ A \in \mathbb{Z}^2 : d(A, M) = r \right\} \subset \beta(K)$$

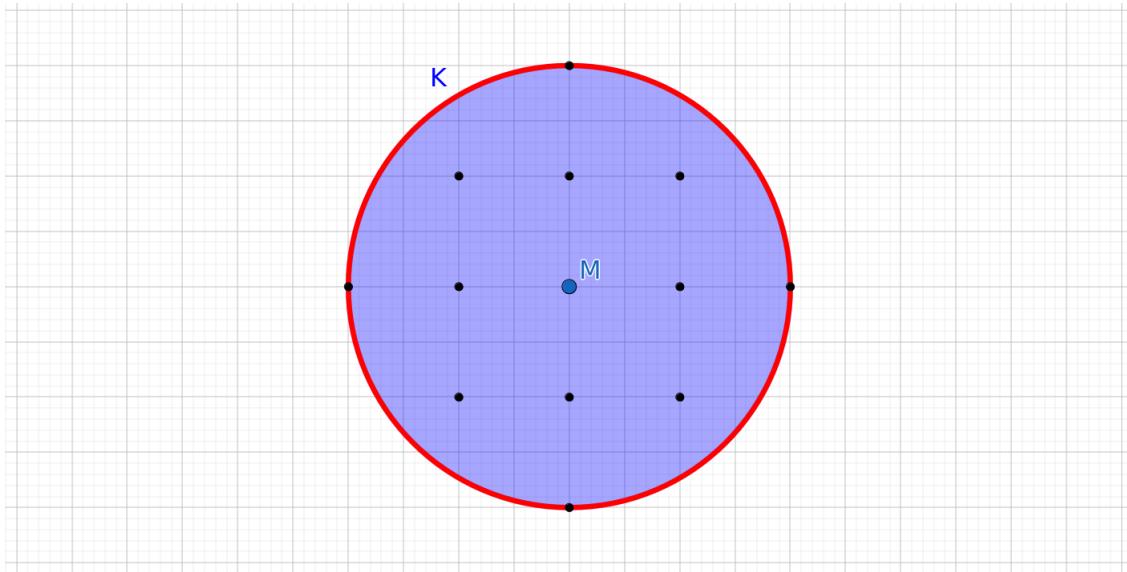


Abbildung 5: Kreis $K = (M, 2)$ mit $\delta(M)$ (blau), $\delta_{\mathbb{Z}}(M)$ (schwarz auf blau), $\beta(K)$ (rot) und $\beta_{\mathbb{Z}}(K)$ (schwarz auf rot)

2.4. Polygone und ihre Eigenschaften

Definition 12: Polygon

Es sei $n \in \mathbb{N}$ mit $n \geq 3$ und $P_1, P_2, \dots, P_n \in \mathbb{R}^2$ paarweise verschiedene, zweidimensionale Punkte. Wobei je drei aneinanderliegende Punkte (das sind P_i, P_{i+1}, P_{i+2} fuer $i \in \{1, 2, \dots, n-1\}$, sowie P_{n-2}, P_{n-1}, P_1 und P_{n-1}, P_1, P_2) nicht auf einer Geraden liegen, sodass Winkel zwischen drei aneinanderliegenden Punkten nie π sind. Dann ist ein Polygon P definiert durch ein n -Tupel seiner Punkte. D.h.,

$$P := (P_1, P_2, \dots, P_n)$$

Dabei werden die Punkte P_i auch Eckpunkte genannt, und die Strecken $[P_i P_{i+1}]$ fuer $i \in \{1, 2, \dots, n-1\}$ und $[P_n P_1]$ werden die Seiten oder die Kanten des Polygons genannt. Alle weiteren Strecken zwischen zwei Punkten sind Diagonalen.

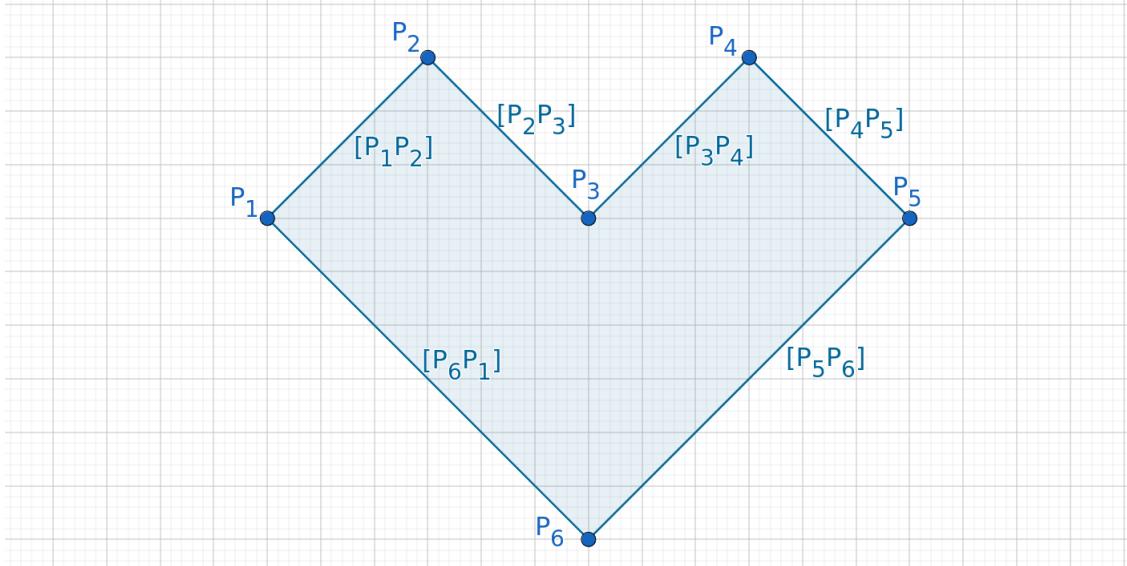


Abbildung 6: Polygon mit Eckpunkten und Seiten

Definition 13: Ganzzahlige Polygone

Es sei $P := (P_1, P_2, \dots, P_n)$ ein Polygon mit ganzzahligen Punkten P_i . Also $P_i \in \mathbb{Z}^2$ fuer alle $i \in \{1, 2, \dots, n\}$. Dann heiszt P ganzzahliges Polygon.

Definition 14: Einfache Polygone

Ein Polygon $P := (P_1, P_2, \dots, P_n)$ heiszt einfach, wenn je zwei verschiedene Seiten des Polygons maximal einen Schnittpunkt haben und sich genau dann treffen, wenn sie einen gemeinsamen Start- oder Endpunkt haben. In diesem Fall ist dieser gemeinsame Start- oder Endpunkt der Strecken der Schnittpunkt.

Definition 15: Ueberschlagene Polygone

Ein Polygon $P := (P_1, P_2, \dots, P_n)$ heiszt ueberschlagen, fall es nicht einfach ist. D.h., es existiert mindestens ein Paar an verschiedenen Kanten, die weitere Schnittpunkte als die Eckpunkte des Polygons haben.

Definition 16: Konvexe Polygone

Ein Polygon $P := (P_1, P_2, \dots, P_n)$ heiszt konvex, falls es einfach (nicht-ueberschlagen) ist und alle Innenwinkel kleiner als π sind.

Dies ist genau dann der Fall, wenn alle Diagonalen des Polygons vollstaendig in (bzw. in den Endpunkten auf dem Polygon) liegen.

Definition 17: Konkave Polygone

Ein Polygon $P := (P_1, P_2, \dots, P_n)$ heiszt konkav, falls es einfach und nicht konvex ist. Dies ist genau dann der Fall, wenn es einfach ist und mindestens ein Innenwinkel groeszer als π ist.

Dies ist genau dann der Fall, wenn P einfach ist und mindestens eine Diagonale nicht vollstaendig in und auf dem Polygon liegt.

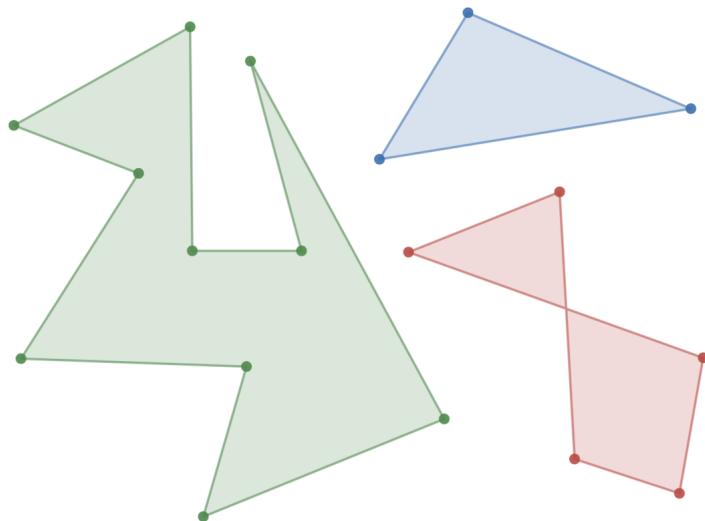


Abbildung 7: Verschiedene Polygone.

Einfach und Konkav (gruen); Einfach und konvex (blaue); nicht einfach bzw. ueberschlagen (rot). Quelle: https://en.wikipedia.org/wiki/File:Two_simple_polygons_and_a_self-intersecting_polygon.png

2.5. Punkte in Polygonen

Der Jordansche Kurvensatz³ besagt, dass jede geschlossene Jordan-Kurve $C \subset \mathbb{R}^2$ ihr Komplement $\mathbb{R}^2 \setminus C$ in zwei disjunkte Gebiete zerlegt. Diese Erkenntniss ist im Allgemein deutlich schwerer zu erkennen als Beispielsweise bei Kreisen. Ein weiterer Sonderfall dafuer sind einfache Polygone, wie sie soeben eingefuehrt wurden. Nun soll praezise definiert werden, welche Punkte sich innerhalb eines einfachen Polygons befinden.

Der Jordansche Kurvensatz fuer einfache Polygone wird beispielsweise in [3] bewiesen. Dazu sei P ein einfaches Polygon. Dann werden zwei Mengen A und B definiert, fuer welche anschlieszend gezeigt wird, dass sie genau dem Aeußereren (A) und dem Inneren (B) des Polygons entsprechen. Genauer wird gezeigt, dass «die nicht auf P gelegenen Punkte der Ebene in [...] A und B zerfallen, derart, dasz, je zwei Punkte derselben [Menge] durch einen Streckenzug verbunden werden koennen, der P nicht schneidet, waehrend jeder Streckenzug, der einen Punkt von A mit einem Punkt von B verbindet, P schneiden [muss].»

Dabei werden A und B wie folgt definiert: Zunaechst wird eine Richtung in der Ebene gewaehlt, welche zu keiner der Seiten von P parallel ist. Ein Punkt $X \in \mathbb{R}^2$ gehoert nun zu A , wenn der Strahl, welcher von X in die gewaehlte Richtung geht, das Polygon in einer geraden Anzahl von Punkten schneidet. Sonst gehoert der Punkt X zu B . D.h., X gehoert zu B , wenn dieser Strahl das Polygon in einer ungeraden Anzahl von Punkten schneidet. Dabei werden Schnittpunkte der Strahlen mit Ecken von P nur gezaehlt, wenn die beiden Seiten der Ecke auf unterschiedlichen Seiten der Geraden des Strahls liegen. Die zuvor genannten Ergebnisse lassen sich nun einfach zeigen. Dadurch besteht nun eine sehr einfache Charakterierung der Menge der Punkte innerhalb eines Polygons. Zunaechst soll dafuer die Menge der Punkte auf dem Rand eines Polygons definiert werden, um diese Punkte in der darauffolgenden Definition der Punkte innerhalb eines Polygons ausschlieszen zu koennen:

Definition 18: Menge der Punkte auf einem Polygon

Weiter stellt sich die Frage welche Punkte aus \mathbb{Z}^2 genau auf einem Polygon $P := (P_1, P_2, \dots, P_n)$ liegen. Dies ist genau die Vereinigung der Strecken $[P_1P_2]$, $[P_2P_3]$, \dots $[P_{n-1}P_n]$ und $[P_nP_1]$:

$$\beta(P) := P_nP_1 \cup \bigcup_{i=1}^{n-1} P_iP_{i+1}$$

Weiter ist die Menge der ganzzahligen Punkte auf dem Rand von P gegeben durch

$$\beta_{\mathbb{Z}}(P) := \left\{ X \in \mathbb{Z}^2 : X \in \beta(P) \right\}.$$

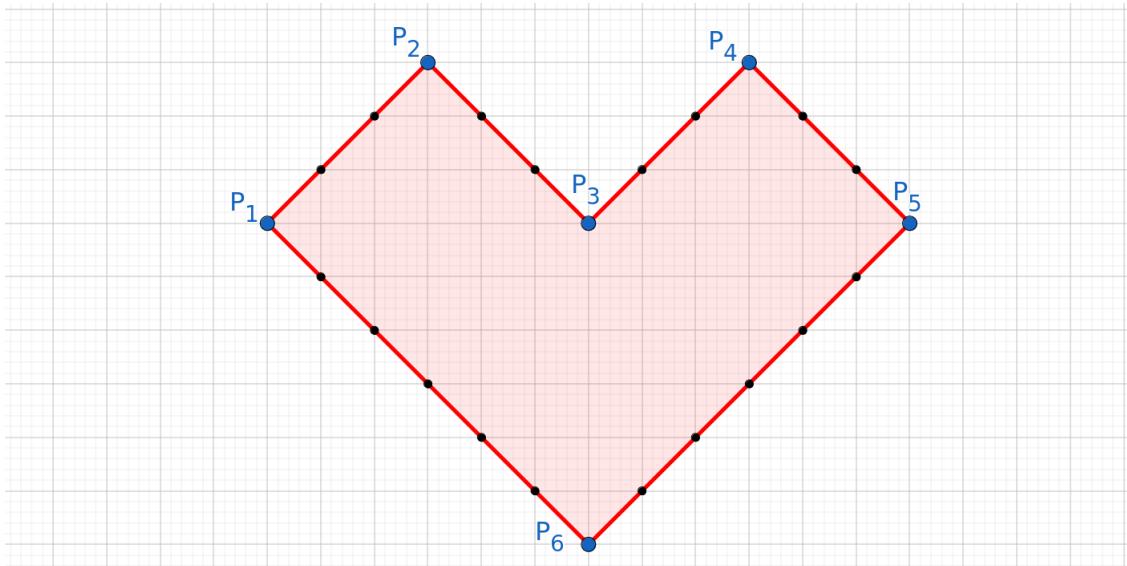


Abbildung 8: Punkte auf dem Rand des Polygons P (dunkel rot) und ganzzahlige Punkte auf dem Rand (schwarz) und Eckpunkte (blau)

³Siehe auch https://de.wikipedia.org/wiki/Jordanscher_Kurvensatz

Definition 19: Menge der Punkte in einem Polygon

Es sei $P := (P_1, P_2, \dots, P_n)$ ein einfaches Polygon mit $P_i \in \mathbb{R}^2$. Dann lässt sich eine Richtung bestimmen, welche zu keiner der Seiten des Polygons parallel ist. Wie oben sei B nun die Menge der Punkte in der Ebene ohne die Menge auf dem Rand des Polygons $Q \in \mathbb{R}^2 - \beta(P)$, sodass der Strahl ausgehend von Q in die zuvor bestimmte Richtung das Polygon in einer ungeraden Anzahl an Punkten schneidet. Wir bezeichnen die soeben definierte Menge B nun durch $\delta(P)$:

$$\delta(P) := B.$$

Weiter ist die Menge der ganzzahligen Punkte in P gegeben durch

$$\delta_{\mathbb{Z}}(P) := \left\{ X \in \mathbb{Z}^2 : X \in \delta(P) \right\}.$$

Beispiel 20: Punkte in Polygonen

In Abbildung 9 wird obige Definition auf vier Punkte angewandt:

1. Der Punkt K erzeugt Strahl i , welcher die Eckpunkte P_2 und P_4 trifft. Da die anliegenden Seiten dieser Eckpunkte je auf derselben Seite der zum Strahl gehörenden Geraden liegen, hat i insgesamt 0 Schnittpunkte, sodass K ausserhalb des Polygons liegt (gerade Anzahl).
2. Der Punkt G erzeugt den Strahl g , welcher die Eckpunkte P_3 und P_5 trifft. Dabei liegen die anliegenden Seiten des Punkts P_3 auf derselben Seite der Geraden und die anliegenden Seiten des Punkts P_5 auf verschiedenen Seiten. Somit hat g einen Schnittpunkt mit dem Polygon, sodass G innerhalb des Polygons liegt.
3. Der Punkt H schneidet die Strecke $[P_5P_6]$. Also besteht genau ein Schnittpunkt, sodass auch H innerhalb des Polygons liegt.
4. Der Punkt M schneidet die Strecken $[P_6P_1]$ und $[P_5P_6]$. Also bestehen zwei Schnittpunkte, sodass M ausserhalb des Polygons liegt.

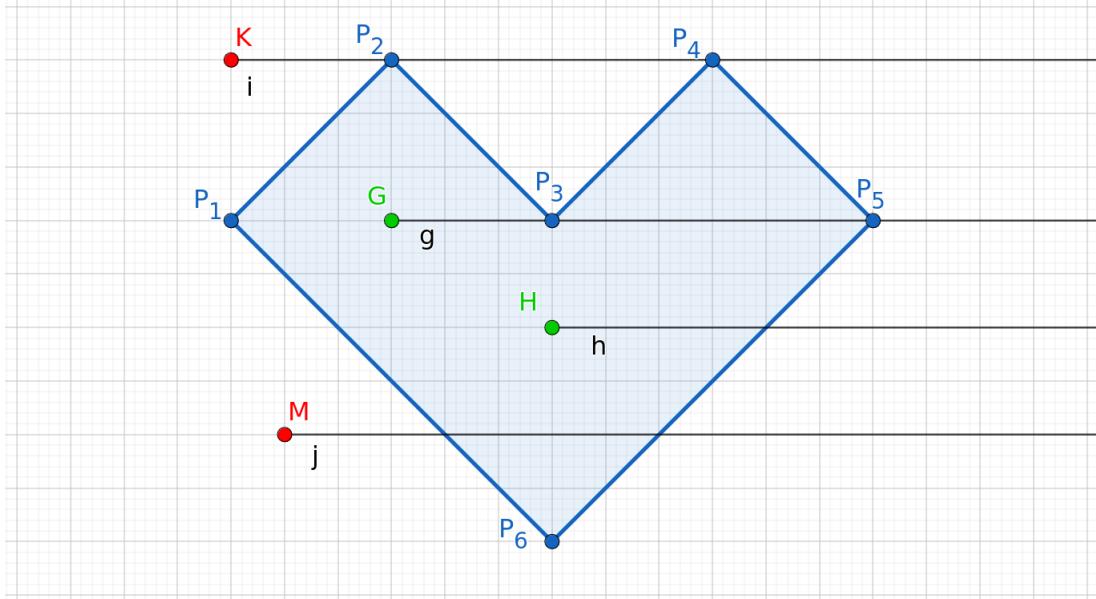


Abbildung 9: Punkte innerhalb (grün) und ausserhalb (rot) des Polygons P

Bemerkung 21: Strahl und Richtung

In der soeben gegebenen Definition bleiben evtl. mehrere Fragen offen. Zunächst soll formalisiert werden, wie Richtung eines Strahls, bzw. die Richtung einer Strecke zu verstehen sind.

Dazu betrachte man eine Strecke $[AB]$. Fuer die Gerade AB durch die Punkte $A = (x, y)$ und $B = (a, b)$ und die Gerade $g := AC$ mit $C = (x + 1, y)$ (parallel zur x -Achse durch A) laesst sich der Winkel α zwischen g und AB nach Definition 9 bestimmen. Genau dieser Winkel bestimmt die Richtung der Strecke. Dabei ist zu bemerken, dass es nach dem Stufenwinkelsatz keine Rolle spielt, ob g durch A oder durch B laeuft.

Hat man andersherum einen Punkt C und einen Winkel α , so ist die Gerade, bzw. der Strahl durch C mit diesem Winkel gegeben durch CD , bzw. $[CD$, wobei $D = C + (\cos(\alpha), \sin(\alpha))$.

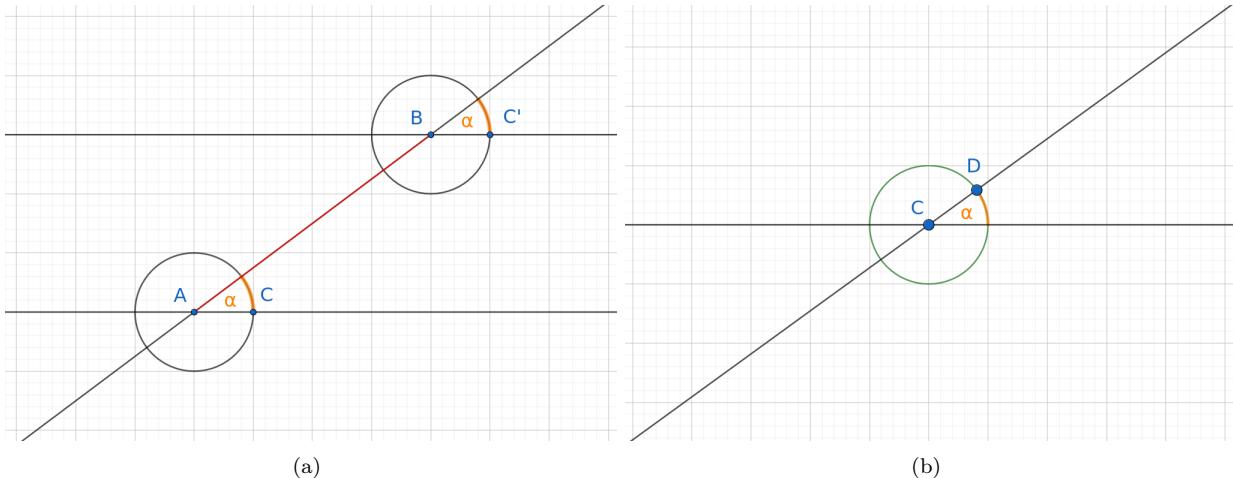


Abbildung 10: (a) Richtung α der Strecke AB ; (b) Gerade durch Punkt mit Richtung α

Bemerkung 22: Schnittpunkte eines Strahls mit dem Polygon

Als weitere Ergaenzung zu Definition 19, soll nun konkretisiert werden, wie die Anzahl der Schnittpunkte eines Strahls mit dem Polygon definiert ist. Dazu sei ein Strahl

$$[XY] := \{X + t \cdot (Y - X) : t \in [0, \infty)\}$$

gegeben, welcher nicht parallel zu einer der Seiten des Polygons ist. Dadurch wird verhindert, dass es unendlich Schnittpunkte gibt. Die Anzahl der Schnittpunkte dieses Strahls mit einem Polygon $P := (P_1, P_2, \dots, P_n)$ nach obiger Definition ergibt sich wie folgt: Die Anzahl k_0 der Seiten $[AB]$ des Polygons, sodass die offene Strecke $(AB) := [AB] - \{A, B\}$ den Strahl schneidet, sowie der Anzahl k_1 der Eckpunkten des Polygons P_i , die der Strahl mit den oben genannten Bedingungen schneidet.

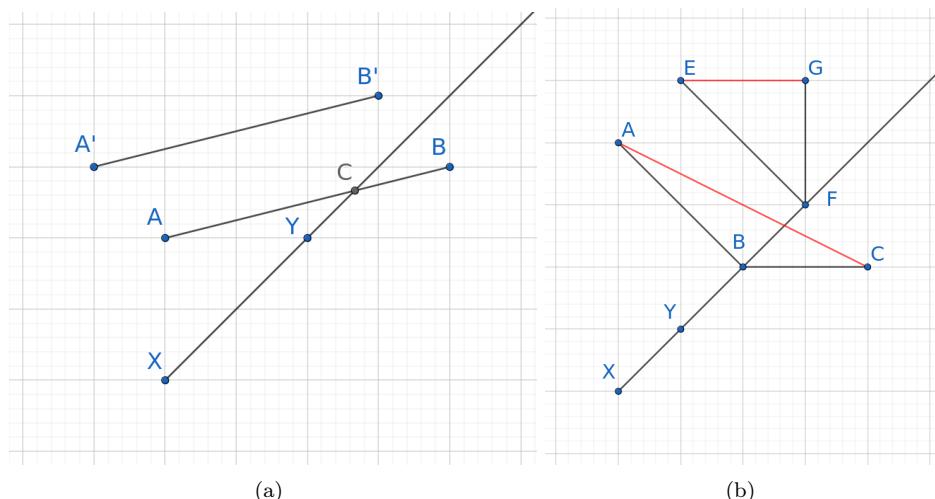


Abbildung 11: (a) Punkt 1: Schnittpunkt von Strahl mit Strecke;
(b) Punkt 2: Punkt auf Strahl (mit Seiten!)

1. Es sei $[AB]$ und man betrachte die offene Strecke $(AB) := [AB] - \{A, B\}$ (also die Strecke $[AB]$ ohne ihre Endpunkte). Dann gilt nach Definition 5:

$$[AB] - \{A, B\} = \left\{ A + t \cdot (B - A) : t \in [0, 1] \right\} - \{A, B\} = \left\{ A + t \cdot (B - A) : t \in (0, 1) \right\}$$

Nun ist gefragt, ob ein Schnittpunkt von (AB) und $[XY]$ existiert. D.h., ob

$$(AB) \cap [XY] \neq \emptyset$$

gilt. Also ob $t \in (0, 1)$ und $\lambda \in [0, \infty)$ existieren mit

$$A + t \cdot (B - A) = X + \lambda \cdot (Y - X)$$

Diese Gleichung laesst sich tatsaechlich simpel nach t und λ aufloesen, sodass anschlieszend nur die Nebenbedingungen geprueft werden muessen.

2. Es sei P ein Eckpunkt des Polygons mit anliegenden Seiten $[AP]$ und $[BP]$ des Polygons. Dann ist gefragt, ob P auf dem Strahl $[XY]$ liegt und die Seiten $[AP]$ und $[BP]$ auf unterschiedlichen Seiten der Geraden XY liegen.

- (a) Erstere Bedingung fragt, ob $P \in [XY]$ gilt, bzw. ob $\lambda \in [0, \infty)$ existiert, sodass

$$P = X + \lambda \cdot (Y - X)$$

- (b) Die Zeite Bedingung ist aquivalent dazu, dass die Strecke $[AB]$ den Strahl $[XY]$ schneidet und funktioniert analog zu Punkt 1.

2.6. Polygone mit Loechern

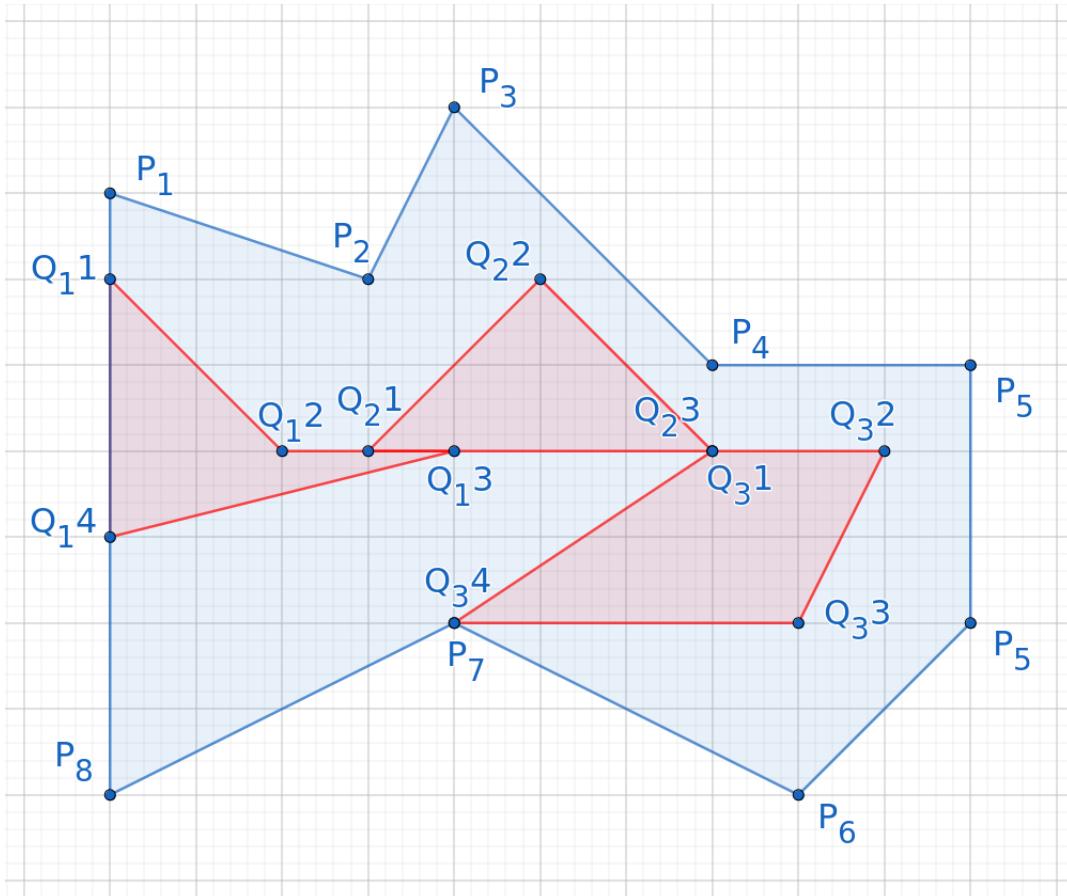


Abbildung 12: Polygon mit Löchern (rot)

Definition 23: Polygon mit Loechern

Es sei $P := (P_1, P_2, \dots, P_n)$ ein einfaches Polygon mit $P_i \in \mathbb{Z}^2$. Weiter seien $k \in \mathbb{N}$ und $n_1, n_2, \dots, n_k \in \mathbb{N}$, sowie einfache Polygone Q_1, Q_2, \dots, Q_k mit $Q_i = (Q_i^{(1)}, Q_i^{(2)}, \dots, Q_i^{(n_i)})$ fuer alle $i \in \{1, 2, \dots, k\}$, wobei weiter fuer alle $i \in \{1, 2, \dots, k\}$ gilt, dass

1. Alle der Polygone Q_i vollstaendig in und auf P liegen:

$$\delta(Q_i) \cup \beta(Q_i) \subset \delta(P) \cup \beta(P)$$

fuer alle $i \in \{1, 2, \dots, k\}$, und

2. Die Menge aller Punkte innerhalb (nicht auf) der Polygone Q_i paarweise diskunkt sind:

$$\delta(Q_i) \cap \delta(Q_j) = \emptyset$$

fuer alle $i, j \in \{1, 2, \dots, k\}$ mit $i \neq j$.

Dann ist $(P, Q_1, Q_2, \dots, Q_k)$ ein Polygon mit Loechern.

Definition 24: Menge der Punkte in einem Polygon mit Loechern

Es sei $T := (P, Q_1, Q_2, \dots, Q_k)$ ein Polygon mit Loechern nach Definition 23. Dann ist die Menge $\delta(T)$ der Punkte innerhalb von T definiert als die Menge $\delta(P)$ aller Punkte in P , ohne die Vereinigung aller Mengen an Punkten in und auf den Polygonen Q_i :

$$\delta(T) := \delta(P) - \bigcup_{i=1}^k \delta(Q_i) \cup \beta(Q_i)$$

Die Menge der ganzzahligen Punkte in T ist dann

$$\delta_{\mathbb{Z}}(T) := \left\{ X \in \mathbb{Z}^2 : X \in \delta(T) \right\}.$$

Definition 25: Menge der Punkte auf einem Polygon mit Loechern

Es sei $T := (P, Q_1, Q_2, \dots, Q_k)$ ein Polygon mit Loechern nach Definition 23. Dann ist die Menge $\beta(T)$ der Punkte auf dem Rand von T definiert als die Vereinigung der Menge $\beta(P)$ der Punkte auf dem Rand von P und der Vereinigung der Mengen der Punkte auf den Raendern der Polygone Q_i :

$$\beta(T) := \beta(P) \cup \bigcup_{i=1}^k \beta(Q_i)$$

Die Menge der gannzahligen Punkte auf T ist dann

$$\beta_{\mathbb{Z}}(T) := \left\{ X \in \mathbb{Z}^2 : X \in \beta(T) \right\}.$$

3. Definition des Problems und der Erweiterungen I - III

3.1. Originales Problem

Definition 26: Problem 0

Gegeben sei ein einfaches konkaves oder konvexes, gannzahliges Polygon $P := (P_1, P_2, \dots, P_n)$ (also $P_i \in \mathbb{Z}^2$). Dann werden $k \in \mathbb{N}$ und die Punkte $X \in \mathbb{Z}^2$ und $X_1, \dots, X_k \in \mathbb{Z}^2$ gesucht, welche folgende Bedingungen erfüllen:

- (B0) Die Punkte X_1, \dots, X_k liegen innerhalb von P . Also $X_i \in \delta(P)$ fuer alle $i \in \{1, \dots, k\}$.
- (B1) Fuer alle $i, j \in \{1, \dots, k\}$ mit $i \neq j$ gilt:

$$\|X_i - X_j\| \geq 10 \cdot 10^6$$

- (B2) Fuer alle $i, j \in \{1, \dots, k\}$ mit $i \neq j$ gilt:

$$\left(\|X_i - X_j\| \geq 20 \cdot 10^6 \right) \text{ oder } \left(\|X - X_i\| \leq 85 \cdot 10^6 \right) \text{ oder } \left(\|X - X_j\| \leq 85 \cdot 10^6 \right)$$

Dabei ist die Anzahl k der Punkte zu maximieren.

Bemerkung 27:

Das Polygon in der Eingabe des Problems sollte offensichtlich eine sinnvolle Groeszenordnung im Vergleich zu den Abstaenden haben, wenngleich dies fuer die Loesung einer Eingabe keine Rolle spielt. Dabei ist mit Betrachtung der Aufgabenstellung die Einheit Millimeter (mm), wobei 10^6 mm = 1 km gilt. Somit muss man die Eingaben der BwInf-Website mit 10^6 multipliziert werden, da sie in km angegeben sind.

Weiter stellen in der Definition des Problems X die Position des Gesundheitszentrums und X_1, \dots, X_k die Positionen der Ortschaften dar. Dabei besagt Bedingung (B0), dass diese Punkte innerhalb des Polygons liegen müssen. Dies schlieszt insbesondere Punkte (Ortschaften) auf dem Rand des Polygons (also den Grenzen des Gebiets) aus. Weiter besagt die Bedingung (B1), dass alle zwei verschiedenen Ortschaften einen Abstand von mindestens 10 Kilometern ($10 \cdot 10^6$ Millimetern) haben sollen. Darauf folgt die Bedingung (B2), die sicherstellt, dass sich Krankheiten nicht zwischen verschiedenen Ortschaften X_i und X_j uebertragen: Dafuer gibt es drei Moeglichkeiten. (1) Die Ortschaften haben einen Abstand von mindestens 20 km; (2) die Ortschaft X_i ist durch das Gesundheitszentrum geschuetzt. D.h., X_i hat einen Abstand von maximal 85km zum Gesundheitszentrum; und (3) analog zu (2) ist die Ortschaft X_j durch das Gesundheitszentrum geschuetzt.

3.2. Erweiterung I: Variable Abstaende

Das soeben definierte Problem entspricht genau dem der Aufgabenstellung. Allerdings werden die gegebenen Distanzen (10km, 20km und 85km) keinen sehr groszen Einfluss auf die folgenden Loesungsvorschlaege haben. Dies motiviert eine Erweiterung des Problems, bei welcher diese Distanzen Teil der Eingabe sind.

In dieser Erweiterung sollen die in der Aufgabenstellung genannten Abstaende (zwischen den Ortschaften und zum Gesundheitszentrum) als Eingaben fuer das Problem ergaenzt werden. Dabei sei $A \in \mathbb{R}$ der allgemeine Mindestabstand zwischen den Ortschaften, $B \in \mathbb{R}$ der Mindestabstand zwischen Ortschaften fuer den Schutz vor Uebertragung zwischen den Ortschaften, und $R \in \mathbb{R}$ der Schutzzradius des Gesundheitszentrums.

Dabei gilt (i) $A, B, R \geq 0$ und (ii) $B > A$. Denn (i) ist eine negative Distanz nicht moeglich, und (ii) waere $B \leq A$, so waeren alle Ortschaften bereits durch den Mindestabstand vor Uebertragungen geschuetzt, sodass das Gesundheitszentrum keinen Zweck haette.

Definition 28: Problem I

Gegeben sei ein einfaches konkaves oder konvexes, gannzahliges Polygon $P := (P_1, P_2, \dots, P_n)$ (also $P_i \in \mathbb{Z}^2$). Weiter seien $A, B, R \in \mathbb{Z}$ mit $A, B, R > 0$ und $B > A$ gegeben. Dann werden $k \in \mathbb{N}$ und Punkte $X \in \mathbb{Z}^2$ und $X_1, \dots, X_k \in \mathbb{Z}^2$ gesucht. Dabei ist die Anzahl k der Punkte zu maximieren, sodass die folgenden Eigenschaften erfüllt werden:

- (B0) Die Punkte X_1, \dots, X_k liegen innerhalb von P . Also $X_i \in \delta(P)$ fuer alle $i \in \{1, \dots, k\}$.
- (B1) Fuer alle $i, j \in \{1, \dots, k\}$ mit $i \neq j$ gilt:

$$\|X_i - X_j\| \geq A$$

(B2) Fuer alle $i, j \in \{1, \dots, k\}$ mit $i \neq j$ gilt:

$$\left(\|X_i - X_j\| \geq B \right) \text{ oder } \left(\|X - X_i\| \leq R \right) \text{ oder } \left(\|X - X_j\| \leq R \right)$$

Beispiel 29:

Es sei $P = (P_1, P_2, P_3, P_4, P_5, P_6)$ ein Polygon mit $P_i \in \mathbb{Z}^2$ (siehe Abbildung). Weiter sei $A = 1$, $B = 2$ und $R = 3$.

Nun sei $M \in \mathbb{Z}^2$ der Punkt (Standort) des Gesundheitszentrums und die Punkte X_1, X_2, X_3, X_4, X_5 Ortschaften innerhalb des Polygons P , sodass X_1 und X_2 innerhalb des Kreises (M, R) liegen.

Dann muessen je alle Punkte eine Distanz von mindestens $A = 1$ zu den anderen Punkten haben (Bedingung (B1)). Dies wird beispielsweise durch die gruenen Kreise um X_1 und X_2 dargestellt (es duerfen keine weiteren Punkte innerhalb dieser Kreise liegen). Die Bedingung (B2) fuer die Punkte X_3, X_4, X_5 untereinander reduziert sich zu einer Mindestdistanz von $B = 2$ von diesen Punkten untereinander, da sie alle ausserhalb Kreises (M, R) liegen. TODO

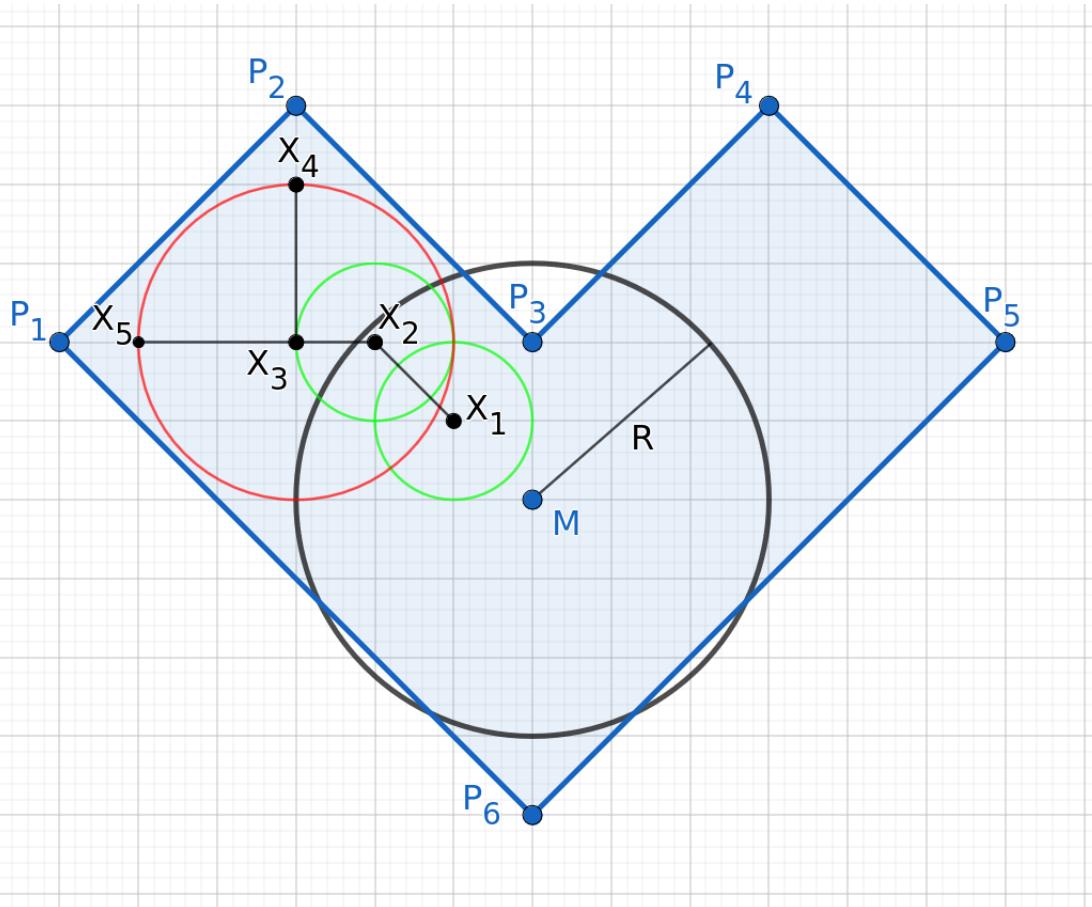


Abbildung 13: Beispiel Loesung fuer die Beispielprobleminstanz

3.3. Erweiterung II: Polygone mit Loechern - Gewaesser

Hier werden die Ortschaften und das Gesundheitszentrum weiter als zweidimensionale Punkte betrachtet werden. Nun soll allerdings ein Polygon mit Loechern betrachtet werden. Die Bedingungen bleiben dieselben.

Definition 30: Problem II

Es sei $T := (P, Q_1, Q_2, \dots, Q_k)$ ein Polygon mit Loechern nach Definition 17, wobei P, Q_1, \dots, Q_k ganzzahlige Polygone sind (ihre Eckpunkte sind ganzalig). Weiter seien $A, B, R \in \mathbb{Z}$ mit $A, B, R > 0$ und $B > A$ gegeben.

Dann werden $k \in \mathbb{N}$ und Punkte $X \in \mathbb{Z}^2$ und $X_1, \dots, X_k \in \mathbb{Z}^2$ gesucht, sodass

(B0) Die Punkte X_1, \dots, X_k liegen innerhalb von T . Also $X_i \in \delta(T)$ fuer alle $i \in \{1, \dots, k\}$.

(B1) Fuer alle $i, j \in \{1, \dots, k\}$ mit $i \neq j$ gilt:

$$\|X_i - X_j\| \geq A$$

(B2) Fuer alle $i, j \in \{1, \dots, k\}$ mit $i \neq j$ gilt:

$$\left(\|X_i - X_j\| \geq B \right) \text{ oder } \left(\|X - X_i\| \leq R \right) \text{ oder } \left(\|X - X_j\| \leq R \right)$$

Dabei ist die Anzahl k der Punkte zu maximieren.

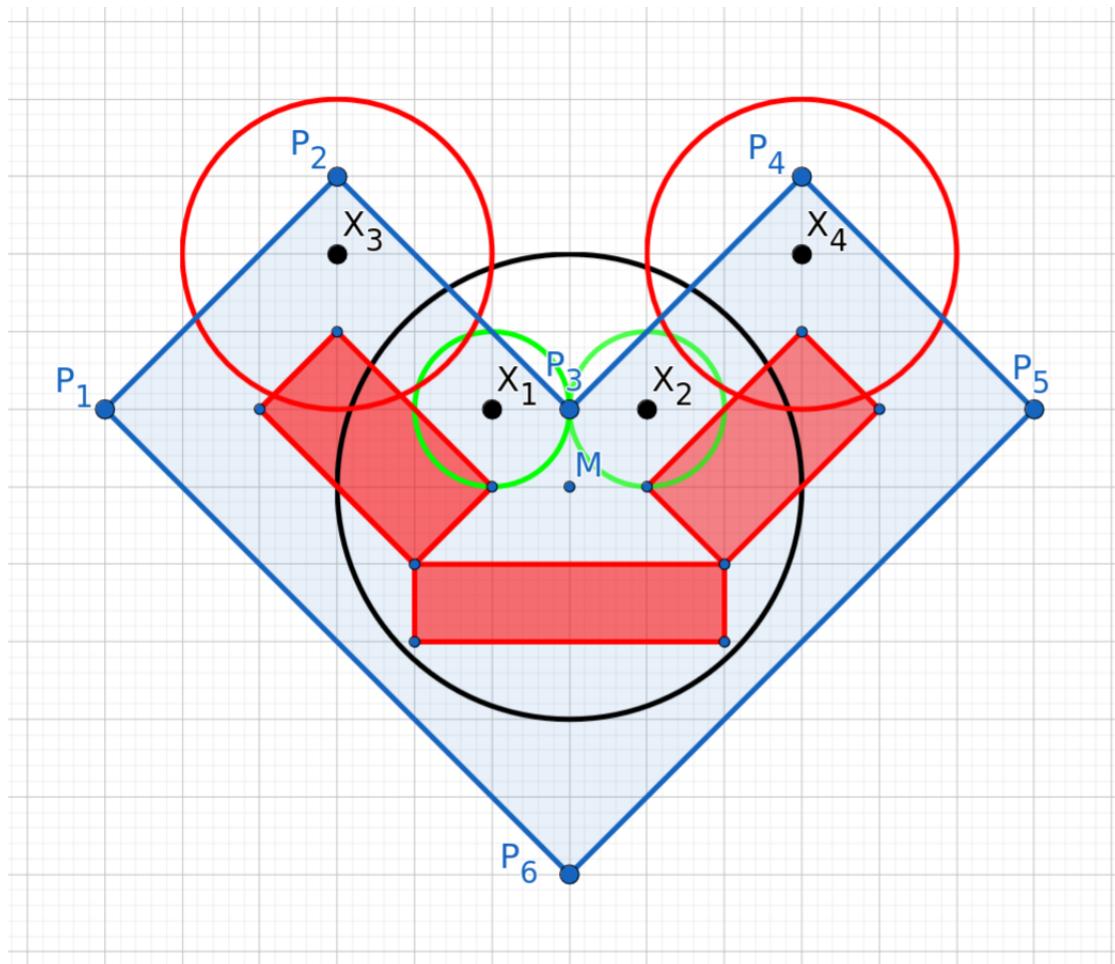


Abbildung 14: Polygon mit Loechern

Beispiel 31:

3.4. Erweiterung III: Beachtung des Flaecheninhalts

In dieser Erweiterung soll statt der Anzahl der Ortschaften die Flaeche ebendieser maximiert werden. Erneut sind ein Polygon im 2-dimensionalen Raum und die Distanzen A , B und R gegeben. Nun sollen die Ortschaften allerdings als Kreise modelliert werden, sodass die Gesamtflaeche der Kreise maximal ist. Dabei sollen alle Kreise vollstaendig im Polygon und die Distanzbestimmungen eingehalten sein.

Definition 32: Problem III

Gegeben sei ein einfaches, konkaves oder konvexas, gannzahliges Polygon $P := (P_1, P_2, \dots, P_n)$ (also $P_i \in \mathbb{Z}^2$). Weiter seien $A, B, R \in \mathbb{Z}$ mit $A, B, R > 0$ und $B > A$ gegeben. Weiter sei $R_0 \in \mathbb{R}$ der minimale Radius der Kreise und $R_1 \in \mathbb{R}$ der maximale Raidus (wobei auch $R_1 = \infty$ moeglich ist). Nun werden $k \in \mathbb{N}$, sowie $X \in \mathbb{Z}^2$ und Kreise K_1, \dots, K_k mit $K_i = (X_i, r_i)$ gesucht, wobei $X_i \in \mathbb{Z}^2$ der Mittelpunkt des Kreises K_i und $r_i \in \mathbb{Z}$ dessen Radius ist, wobei $R_0 \leq r_i \leq R_1$ gelten soll. Dabei sollen folgende Eigenschaften gelten:

(B0) Alle Kreise liegen vollstaendig innerhalb des Polygons P : $\delta(K_i) \subset \delta(P)$ fuer alle $i \in \{1, \dots, k\}$.

(B1) Fuer alle $i, j \in \{1, \dots, k\}$ mit $i \neq j$ gilt:

$$\|X_i - X_j\| - (r_i + r_j) \geq A$$

(B2) Fuer alle $i, j \in \{1, \dots, k\}$ mit $i \neq j$ gilt:

$$\left(\|X_i - X_j\| - (r_i + r_j) \geq B \right) \text{ oder } \left(\|X - X_i\| + r_i \leq R \right) \text{ oder } \left(\|X - X_j\| + r_j \leq R \right)$$

Dabei soll die Gesamtflaeche der Kreise (Ortschaften)

$$F := \sum_{i=1}^k \pi \cdot r_i^2$$

maximiert werden.

Beispiel 33:

Beispiel zu Erweiterung III:

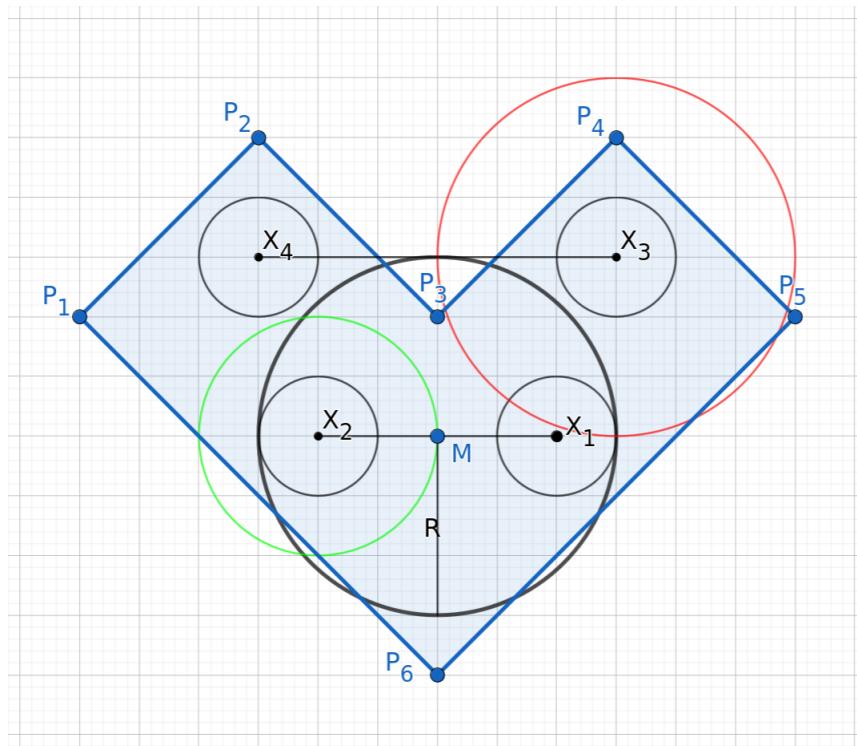


Abbildung 15

4. Komplexitaet des Problems

Nachdem die mathematischen Definitionen des Problems gegeben wurden, werden nun Ueberlegungen zur Komplexitaet des Problems angestellt. Dabei wird im Folgenden angenommen, dass das Problem NP-schwer ist. Diese Annahme, zusammen mit der von den meisten Wissenschaftlern angenommen Vermutung, dass $P \neq NP$, hat zur Folge, dass es fuer die soeben definierten Probleme keine effizienten Loesungsverfahren gibt. Das bedeutet konkret, dass kein Algorithmus mit einer polynomiellen Laufzeitkomplexitaet existiert, der das Problem exakt loesen kann.

Dementspraechend werden im Folgenden Algorithmen mit polynomieller Laufzeitkomplexitaet angegeben, die das Problem nur moeglichst gut aber mitnichten⁴ perfekt loesen. Genaueres zu diesen Loesungen folgt im naechsten Kapitel.

Der Grund fuer die Annahme der NP-schwere ist, die Naehe der Probleme zu der Menge der Packungsprobleme, insbesondere der Kreispackungsprobleme. Es konnte allerdings kein mathematischer Beweis in Form einer Reduktion gefunden werden, der zeigt, dass eines der vorgestellten Probleme NP-schwer ist.

5. Loesungsvorschlaege zur Erweiterung I

In diesem Kapitel sollen Loesungsvorschlaege fuer Erweiterung I behandelt werden. Dadurch wird das originale Problem implizit geloeost.

Dafuer wird ein randomisiertes Verfahren verwendet, welches zunaechst zwei Teilmengen T und G der Punkte $\delta_Z(P)$ aus zufaellig berechnet und anschlieszend den Punkt des Gesundheitszentrums aus der Menge G und die Punkte der Ortschaften aus der Menge T auswaehlt.

5.1. Triangulierung

Um eine Teilmenge der ganzzahligen Punkte innerhalb des Polygons P zu berechnen, soll zunaechst die sogenannte Triangulierung behandelt werden. Dabei wird ein gegebenes Polygon in Dreiecke unterteilt⁵.

Dabei gilt, dass fuer jedes einfache Polygon eine Triangulierung exisiternt und jede Triangulierung eines Polygons mit n Eckpunkten aus $n - 3$ Diagonalen und $n - 2$ Dreiecken besteht [2].

Um eine solche Triangulierung zu finden existieren verschiedene Verfahren [1, 2, 4] mit verschiedenen Laufzeitkomplexitaeten. Ein einfaches Verfahren wird in [4] vorgestellt und basiert auf dem «Two ears theorem»⁶ [5]. Neben diesem Verfahren, das nun vorgestellt werden soll, exisiternt Verfahren mit besseren Laufzeitkomplexitaeten ($\mathcal{O}(n \log n)$).

Das «Two ears theorem» besagt, dass jedes einfache Polygon mit mehr als 3 Eckpunkten mindestens zwei «ears» bestizt, die sich nicht ueberschneiden. Ein ear besteht aus drei aufeinanderfolgenden Eckpunkten Q_1, Q_2, Q_3 des Polygons, wobei der Winkel $\angle Q_1 Q_2 Q_3$ bei Q_2 kleiner ist π und die Diagonale $[Q_1 Q_3]$ vollstaendig in P liegt. Mit Hilfe der ears funktionieren die sogenannten «Ear Clipping» - Verfahren, welche im Polygon P ein ear $\triangle Q_1 Q_2 Q_3$ sucht, den Punkt Q_2 aus dem Polygon entfernt und diese Schritte wiederholt, bis das Polygon nur noch aus 2 Eckpunktken besteht. Ein naives Verfahren zum Finden eines ears in einem Polygon funktioniert wie folgt:

Fuer jedes der n Tripel $Q_1 Q_2 Q_3$ aufeinanderfolgender Eckpunkte des Polygons wird geprueft, ob
(i) $\angle Q_1 Q_2 Q_3 < \pi$ und, (ii) ob kein anderer Eckpunkt X von P innerhalb oder auf dem Rand des Dreiecks liegt. Dies ist aquivalent zur zuvor beschriebenen Defintion der ears. Ersteres laesst sich pruefen, indem geprueft wird, ob Q_2 auf rechten Seite der Strecke $[Q_1 Q_3]$ liegt, sofern das Polygon in mathematisch positiver Richtung (d.h., gegen den Uhrzeigersinn angegeben wird); um fuer ein Dreieck zu pruefen, ob ein Punkt X auf bzw. innerhalb dieses liegt, werden zwei Faelle geprueft:

1. X liegt innerhalb des Dreiecks.

Dies ist der Fall, wenn X auf derselben Seite aller Seiten des Dreiecks liegt.

2. X liegt auf einer der Seiten des Dreiecks.

Dies laesst sich durch die Streckengleichung $X = A + t \cdot (B - A)$ pruefen. Diese laesst sich zu zwei Werten

⁴Wie wenn wenn Kollegah mit seiner Verwandtschaft kommt.

⁵Siehe auch https://en.wikipedia.org/wiki/Polygon_triangulation

⁶Siehe auch https://en.wikipedia.org/wiki/Two_ears_theorem

von t auflösen:

$$t = \frac{x - a}{c - a}$$

und

$$t = \frac{y - b}{d - b}$$

wobei $X = (x, y)$, $A = (a, b)$ und $B = (c, d)$ gilt. Dann ist X auf der Strecke $[AB]$, falls

$$\frac{x - a}{c - a} = \frac{y - b}{d - b} \in [0, 1]$$

gilt. Die Fälle $c = a$ und $d = b$ stellen dabei die Fälle dar, in denen A und B dieselbe x - bzw. y -Koordinate haben. Diese Fälle lassen sich einfach prüfen, indem man prüft, ob X die passende x - bzw. y -Koordinate haben und die jeweils andere Koordinate im richtigen Intervall liegt.

Um für eine Strecke $[PQ]$ und einen Punkt X zu prüfen, auf welcher Seite von $[PQ]$ X liegt, wenn man von P nach Q läuft, wird das Vorzeichen von $s := (Q - P) \times (X - P)$ betrachtet. Dabei ist das Kreuzprodukt definiert durch $(a, b) \times (c, d) = a \cdot d - b \cdot c$. Dann gilt: (i) ist $s < 0$, so liegt X rechts von $[PQ]$, (ii) ist $s > 0$, so liegt X links von $[PQ]$. Im Fall $s = 0$ sind die Punkte kollinear.

Das «Ear Clipping» - Verfahren, das in [4] vorgestellt wird, ermöglicht eine bessere Laufzeit von $\mathcal{O}(n^2)$.

5.2. Pseudozufällige ganzzahlige Punkte im Polygon

Mit Hilfe der Triangulierung kann nun ein Verfahren vorgestellt werden, um Punkte innerhalb Polygons zu bestimmen. Dazu wird eine gegebene Triangulierung des Polygons genutzt. Für ein Dreieck PQR der Triangulierung lässt sich dann die Menge der Punkte innerhalb des Dreiecks angeben durch

$$I = \left\{ Q + \alpha \cdot (P - Q) + \beta \cdot (R - Q) : \alpha, \beta \in [0, 1] \text{ mit } \alpha + \beta \leq 1 \right\}$$

Um nun einen zufälligen Punkt $X \in I$ innerhalb des Dreiecks (und somit auch innerhalb des Polygons) zu bestimmen, können pseudozufällige Werte $\alpha, \beta \in [0, 1]$ mit $\alpha + \beta \leq 1$ bestimmt werden. Dazu werden zunächst α und β im Intervall $[0, 1]$ pseudozufällig generiert. Für den Fall, dass $\alpha + \beta > 1$ gilt, wird dann α' auf den Wert $1 - \alpha$ und β' auf den Wert $1 - \beta$ gesetzt. Dann gilt $\alpha', \beta' \in [0, 1]$ und

$$\alpha' + \beta' = 2 - (\alpha + \beta) < 1$$

Dann ist

$$X = \alpha \cdot (P - Q) + \beta \cdot (R - Q).$$

Da X ganzzahlig sein soll, werden die Koordinaten hier jeweils gerundet. Dies kann theoretisch zu Problemen führen, wie in Kapitel 8 erläutert wird.

5.3. Lösungsvorschlag

Nun soll basierend auf dem vorgestellten Verfahren zur Berechnung von Punkten innerhalb des Polygons ein Lösungsalgorithmus vorgestellt werden. Dieser berechnet zunächst zwei Mengen an Punkten innerhalb des Polygons: $G \subseteq \delta_{\mathbb{Z}}(P)$ und $T \subseteq \delta_{\mathbb{Z}}(P)$. Im Folgenden werden die Punkte der Menge G als potenzielle Positionen des Gesundheitszentrums betrachtet und die Menge T als potenzielle Punkte der Ortschaften. Dabei ist können diese Mengen unabhängig sein, oder beispielsweise $G = T$ als auch $G \subset T$ gelten.

Nun soll für jedes $Z \in G$ eine Teilmenge $T_Z \subset T$ gefunden werden, die möglichst groß ist und die Bedingungen bezüglich der Abstände einhält. Die Lösung besteht dann in der größten Menge T_Z und Z (siehe Algorithmus 1).

Algorithmus 1 : Loesungsvorschlag

Input : Polygon P , Menge $G \subseteq \delta_Z(P)$ und Menge $T \subseteq \delta_Z(P)$
Output : Punkt $Z \in G$, sowie $T_Z \subseteq T$

```

1  $T_{\text{best}} \leftarrow \emptyset$ 
2  $Z_{\text{best}} \leftarrow \text{NULL}$ 
3 foreach  $Z \in G$  do
4    $T_Z \leftarrow \text{generate}(P, T, Z)$ 
5   if  $\#(T_Z) > \#(T_{\text{best}})$  then
6      $T_{\text{best}} \leftarrow T_Z$ 
7      $Z_{\text{best}} \leftarrow Z$ 
8   end if
9 end foreach
10 return  $Z_{\text{best}}, T_{\text{best}}$ 
```

Die Groesze der Mengen G und T sollte so gewaehlt werden, dass das Polygon anschaulichen gut ueberdeckt ist. Dazu kann man beispielsweise die Anzahl der Punkte, die man in einem bestimmten Dreieck der Triangulierung erzeugt, abhaenig vom Flaecheninhalt des Dreiecks und der Distanz A setzen. In der Implementierung wurde dafuer ein Algorithmus verwendet, welcher fuer eine bestimmte Anzahl an Punkten n in jedem Dreieck der Triangulierung nach obigen Verfahren $n \cdot \frac{A_\Delta}{F}$ Punkte erstellt, wobei A_Δ die Flaeche des Dreiecks und F die Flaeche des Polygons ist.

Dabei kann der Flaecheninhalt eines Dreiecks bestimmt werden durch $A_\Delta = \sqrt{s(s-a)(s-b)(s-c)}$ mit $s = \frac{1}{2} \cdot (a+b+c)$, wobei a , b , und c die Laenge der Seiten des Dreiecks sind (Satz von Heron).

Um die Menge n sinnvoll zu waeren koennte beispielsweise auch Dichte d festgelegt werden und $n = d \cdot \frac{F}{\pi \cdot A^2}$ gewaehlt werden.

5.4. Wahl der Punkte

Somit wurde das Problem darauf reduziert, fuer ein gegebenes Gesundheitszentrum Z eine Teilmenge T_Z der Menge $T \subset \delta_Z(P)$ an Punkten innerhalb des Polygon zu finden, sodass die bekannten Bedinungen eingehalten werden:

Fuer die Mindestabstaende $A, B, R \in \mathbb{Z}$ mit $A, B, R > 0$ und $B > A$, wobei A dem allgemeinen Mindestabstand zwischen Ortschaften, B dem Abstand zwischen Ortschaft, falls beide Ortschaften ausserhalb des Radius des Gesundheitszentrums liegen und R ebendiesem Radius des Gesundheitszentrums entsprechen, soll fuer die gesuchte Menge $T_Z \subseteq T$ gelten, dass

(B1) Fuer alle $X, Y \in T_Z$ mit $X \neq Y$ gilt:

$$\|X - Y\| \geq A$$

(B2) Fuer alle $X, Y \in T_Z$ mit $X \neq Y$ gilt:

$$\left(\|X - Y\| \geq B \right) \text{ oder } \left(\|Z - X\| \leq R \right) \text{ oder } \left(\|Z - Y\| \leq R \right)$$

Dabei ist bereits gegeben, dass alle Punkte innerhalb des Polygons liegen.

Im Folgenden werden zwei Verfahren vorgestellt, um diese Teilmenge zu berechnen.

ILP Programm

Das erste Verfahren soll eine optimale Loesung finden, indem ein ganzzahliges lineares Programm (engl. integer linear programm, ILP) verwendet wird. Dabei ist diese Loesung bei gegebener Position Z des Gesundheitszentrums optimal fuer die gegebene Menge T , nicht aber fuer das gesamte Polygon P , ausser wenn $T = \delta_Z(P)$.

Dazu wird fuer jeden Punkt $Y \in T$ eine boolesche Variable x_Y eingefuehrt. D.h., $x_Y \in \{0, 1\}$. Dann wird $x_Y = 1$ bedeuten, dass Y Teil der Menge T_Z ist. Weiter wird als Zielfunktion die Summe aller Variablen, d.h. die Anzahl der gewaehlten Punkte, gewaehlt. Diese soll maximiert werden. Zuletzt werden die linearen Bedingung definiert. Dazu werden fuer alle $X, Y \in T_Z$ mit $X \neq Y$ zwei Faelle unterschieden:

1. Es gilt $\|Z - X\| \leq R$ oder $\|Z - Y\| \leq R$.

Das heiszt, mindestens einer der beiden Punkte liegt innerhalb des Radius des Gesundheitszentrums. In diesem Fall sollen die Punkte mindestens den Abstand A haben. Fuer den Fall, dass $\|X - Y\| < A$, wird die Bedinung

$$x_X + x_Y \leq 1$$

zum ILP hinzugefuegt, da die Punkte X und Y in diesem Fall nicht beide Teil der Menge T_Z sein duerfen. Gilt allerdings $\|X - Y\| \geq A$, so wird keine Bedinung hinzugefuegt, da die Punkte beide gleichzeitig Teil der Menge T_Z sein duerfen.

2. Es gilt $\|Z - X\| > R$ und $\|Z - Y\| > R$.

Das heiszt beide Punkte liegen ausserhalb des Radius des Gesundheitszentrums. Dann muessen sie mindestens einen Abstand von B haben. Ist dies nicht der Fall, also $\|X - Y\| < B$, so wird analog zum vorherigen Fall die Bedingung

$$x_X + x_Y \leq 1$$

zum ILP hinzugefuegt. Gilt wiederum $\|X - Y\| \geq B$, wird keine Bedinung hinzugefuegt.

Formaler ist das ILP wie folgt definiert:

1. Variablen:

Fuer alle $Y \in T$ die boolesche (bzw. bianere) Variable x_Y .

2. Zielfunktion:

Es soll die Funktion

$$f := \sum_{Y \in T} x_Y$$

maximiert werden.

3. Bedinungen:

Fuer alle $X, Y \in T$ mit $X \neq Y$ die Bedinung

$$x_X + x_Y \leq 1,$$

falls eine der Bedinungen gilt: (1) ($\|Z - X\| \leq R$ oder $\|Z - Y\| \leq R$) und $\|X - Y\| < A$;
(2) ($\|Z - X\| > R$ und $\|Z - Y\| > R$) und $\|X - Y\| < B$.

Satz 34:

Es sei eine Loesung des ILPs gegeben durch eine Belegung der Variablen x_Y fuer alle $Y \in T$. Dann erfuellt die Menge

$$T_Z := \{Y \in T : x_Y = 1\}$$

die Eigenschaften (B1) und (B2).

(ohne Beweis)

Bemerkung 35: Optimierung

Eine moegliche Optimierung fuer das vorgestellt Loesungsverfahren besteht in der Aufteilung des Polygons in einzelne Polygone, die zusammen das Polygon P ergeben. Dann wird die Menge T fuer jedes der Teilstücke berechnet. Fuer einen gegebenen Punkt Z wird dann fuer jede der Mengen T nacheinander ein ILP gelöst. Dieses muss zu den bereits vorgestellten Bedinungen auch die bereits ausgewaehlt Punkte in den anderen Polygone beachten.

Dieses Verfahren ermoeglich es insgesamt mehr Punkte als potenzielle Ortschaften zu verwenden.

Greedy-Verfahren

Das zweite Verfahren wird keine optimale Loesung finden, aber eine bessere Laufzeit aufweisen. Dabei handelt es sich um ein Greedy-Verfahren, welches die Menge T_Z iterativ erzeugt. Dabei wird mit einem beliebigen Punkt $Y \in T$ begonnen. D.h., T_Z wird als $\{Y\}$ initialisiert. Dann wird fuer eine bisher erstellte Menge T_Z die Menge M aller Punkte $Y \in T$ bestimmt, sodass Y gemaesz Bedinungen (B1) und (B2) einen ausreichenden Abstand zu allen Punkten der Menge T_Z hat:

$$M = \left\{ Y \in T : \forall X \in T_Z : \left((\|Z - X\| \leq R \vee \|Z - Y\| \leq R) \wedge \|X - Y\| \geq A \right) \text{ oder} \right. \\ \left. \left((\|Z - X\| > R \wedge \|Z - Y\| > R) \wedge \|X - Y\| \geq B \right) \right\}$$

Aus dieser Menge wird ein Punkt ausgewählt und zu T_Z hinzugefügt. Dies wird so lange wiederholt, bis die Menge M leer ist:

Algorithmus 2 : Greedy-Verfahren zur Berechnung von T_Z

Input : Polygon P , Punkt Z und Menge $T \subseteq \delta_{\mathbb{Z}}(P)$
Output : $T_Z \subseteq T$

- 1 $Y \leftarrow$ some element in T
- 2 $T_Z \leftarrow \{Y\}$
- 3 $M \leftarrow \left\{ Y \in T : \forall X \in T_Z : \left((\|Z - X\| \leq R \vee \|Z - Y\| \leq R) \wedge \|X - Y\| \geq A \right) \text{ oder} \right. \\ \left. \left((\|Z - X\| > R \wedge \|Z - Y\| > R) \wedge \|X - Y\| \geq B \right) \right\}$
- 4 **while** $M \neq \emptyset$ **do**
- 5 $X \leftarrow$ some element in M
- 6 $T_Z \leftarrow T_Z \cup \{X\}$
- 7 $M \leftarrow \left\{ Y \in T : \forall X \in T_Z : \left((\|Z - X\| \leq R \vee \|Z - Y\| \leq R) \wedge \|X - Y\| \geq A \right) \text{ oder} \right. \\ \left. \left((\|Z - X\| > R \wedge \|Z - Y\| > R) \wedge \|X - Y\| \geq B \right) \right\}$
- 8 **end while**
- 9 **return** T_Z

Bemerkung 36: Berechnung der Menge M

Die Menge M kann in der Praxis einfach berechnet werden, indem die angegebene Bedingung für alle möglichen Punkte $Y \in T$ und $X \in T_Z$ geprüft wird.

Dies lässt sich allerdings optimieren, indem die neue Menge M basierend auf der vorherigen berechnet wird. Dazu muss berichtet werden, dass die einzigen Veränderungen für die neue Menge M ist, dass für ein $Y \in T$ die bekannte Bedingung nun auch für den neu hinzugefügten Punkt $X \in M$ gelten muss. Diese Tatsache kann verwendet werden, um die neue Menge M effizienter zu berechnen: Dazu werden alle Punkte der alten Menge M zur neuen Menge hinzugefügt, die die Bedingung für den zu T_Z hinzugefügten Punkt X erfüllen.

5.5. Analyse der Zeitkomplexität der Lösungsvorschläge

Triangulierung

Wie bereits erwähnt besitzt das vorgestellte Verfahren eine Laufzeit von $\mathcal{O}(n^3)$, während auch Verfahren mit Laufzeit $\mathcal{O}(n^2)$ und $\mathcal{O}(n \log(n))$ existieren.

Die Laufzeit $\mathcal{O}(n^3)$ ergibt sich wie folgt:

Es sei n die Anzahl der Eckpunkte des Polygons. Dann müssen insgesamt $n - 2$ Ears gefunden werden. Um ein Ear zu finden werden $\mathcal{O}(n)$ viele Tripel probiert. Für jedes dieser Tripel muss geprüft werden, ob alle $\mathcal{O}(n)$ anderen Punkte außerhalb des Dreiecks liegen. Da jede dieser Prüfungen in konstanter Zeit $\mathcal{O}(1)$ stattfindet ergibt sich insgesamt diese Laufzeit:

$$(n - 2) \cdot \mathcal{O}(n) \cdot \mathcal{O}(n) \cdot \mathcal{O}(1) = \mathcal{O}(n^3)$$

Pseudozufällige Punkte

Bei gegebenem Dreieck ist die Erstellung eines einzelnen pseudozufälligen Punktes innerhalb des Dreiecks offensichtlich konstant, also $\mathcal{O}(1)$. Dementsprechend ist die Komplexität, um n Punkte zu erstellen $\mathcal{O}(n)$.

Lösungsvorschlag

Für die Größe $\#(T)$ der Menge T und die Größe $\#(G)$ der Menge G ist die Komplexität des Algorithmus 1

$$\mathcal{O}(\#(G) \cdot f(P, T, Z)),$$

wobei f die Komplexität des Algorithmus *generate* in Abhängigkeit von P , T und Z angibt. Diese soll nun für die beiden Lösungsvarianten analysiert werden.

ILP-Verfahren

Im allgemeinen ist die worst-case Komplexitaet des Loesens eines ILP's exponentiell. Da die tatsaechliche Laufzeit auf Grund der enormen Optimierung der ILP-Solver nur schlecht theoretisch erfasst werden kann, soll nun nur die Anzahl der Variablen und Bedingungen erfasst werden. Offensichtlich sind genau $\#(T) = \mathcal{O}(\#(T))$ Variablen vorhanden. Weiter sind $\mathcal{O}(\#(T)^2)$ Bedingungen gegeben. Die Anzahl der Bedingungen laesst sich etwas genauer betrachten: Die Anzahl der Punkte mit Abstand kleiner als A bzw. kleiner als B um einen gegebenen Punkt sollte fuer alle Punkte etwa gleich sein, sofern die Punkte gleichmaesig verteilt sind. Dies entspricht aber jeweils einem Faktor $c \in (0, 1)$, sodass um einen Punkt $\mathcal{O}(c \cdot \#(T)) = \mathcal{O}(\#(T))$ Punkte gegeben sind, sodass man erneut die quadratische Anzahl findet.

Greedy-Verfahren

Bei diesem Verfahren laesst sich die Laufzeit besser behandeln. Die Initialisierung und Neusetzung der Menge M benoetigt stets eine Zeit von $\mathcal{O}(\#(T))$ mit Hilfe der erwaehten Optimierung. Denn es reicht fuer die $\mathcal{O}(\#(T))$ Punkte in der Menge M die angegebenen Bedingung zu pruefen, was stets in konstanter Zeit moeglich. Nun soll eine obere Schranke fuer die Anzahl der Durchlaeufe der While-Schleife gefunden werden. Diese ist ebenfalls $\mathcal{O}(\#(T))$, da die Menge M initial die Groesze $\#(T)$ und dann in jedem Durchgang der Schleife mindestens um 1 verkleinert wird. Daraus folgt die Laufzeit $\mathcal{O}(\#(T)^2)$. Insgesamt folgt dann diese Laufzeit:

$$\mathcal{O}(\#(G) \cdot f(P, T, Z)) = \mathcal{O}(\#(G) \cdot \#(T^2)),$$

6. Erweiterung II: Polygone mit Loechern - Gewaesser

Hier werden die Ortschaften und das Gesundheitszentrum weiter als zweidimensionale Punkte betrachtet. Nun soll allerdings ein Polygon mit Loechern betrachtet werden. Die Bedingungen bleiben dieselben.

Das Loesungsverfahren fuer diese Erweiterung ist im Grunde dasselbe wie zuvor. Allerdings ist nun bei der zufaelligen Berechnung der Menge T ein Polygon mit Loechern gegeben. Dafuer muss ausschlieszlich die Triangulierung grundlegend veraendert werden. Ein Verfahren dafuer ist ebenfalls in [4] angegeben. Leider hat die Zeit nicht mehr gereicht, um dieses Verfahren zu erklaeren, und zu implementieren. Dementspraechend ist die Loesung des Problems dem Leser als Uebungsaufgabe ueberlassen. Allerdings folgt nun eine weitere Erweiterung dessen Bearbeitung gluecklicherweise bereits durchgefuehrt wurde.

7. Erweiterung III: Beachtung des Flaecheninhalts

In dieser Erweiterung soll statt der Anzahl der Ortschaften die Flaeche ebendieser maximiert werden. Erneut sind ein Polygon (ohne Loecher) im 2-dimensionalen Raum und die Distanzen A , B , sowie Radien R und R_0 gegeben. Nun sollen die Ortschaften als Kreise modelliert, und die Gesamtflaeche der Kreise maximiert werden. Dabei sollen alle Kreise vollstaendig im Polygon und die bekannten Distanzbestimmungen eingehalten werden.

7.1. Kreise in Polygonen

In diesem Kapitel soll behandelt werden, wie sich fuer ein gegebenes Polygon P und einen gegebenen Punkt X innerhalb des Polygons der groeszt moegliche Radius r bestimmen laesst, sodass $K = (X, r)$ vollstaendig innerhalb des Polygons liegt.

Satz 37:

Es sei ein Polygon $P = (P_1, P_2, \dots, P_n)$ und ein Punkt $X \in \delta(P)$ gegeben. Dann ist der groeszte Radius $r \in \mathbb{R}$, sodass $K = (X, r)$ vollstaendig innerhalb des Polygons liegt gegeben durch

$$r = \min \left\{ \|X - Q\| : Q \in \beta(P) \right\}$$

Dabei ist $\beta(P)$ die Menge der Punkte auf dem Rand des Polygons, sodass r der kleinste Abstand von X zu einem Punkt Q auf dem Rand von P ist.

Beweis:

1. Teil: Es gilt $\delta((X, r)) \subset \delta(P)$.

Es sei $Y \in \delta((X, r))$. Es gilt zu zeigen, dass $Y \in \delta(P)$.

Wegen $Y \in \delta((X, r))$ gilt $\|X - Y\| < r$. Per Definition von r hat die Strecke $[XY]$ keinen Schnittpunkt mit dem Polygon P . Denn wuerde ein solcher Schnittpunkt Q' existieren, dann haette dieser einen Abstand zu X kleiner als r . Im Widerspruch zur Wahl von r .

Da $X \in \delta(P)$, existiert ein Strahl ausgehend von X , der eine ungrade Anzahl an Schnittpunkten mit P hat. Da $[XY]$ keinen Schnittpunkt mit P hat, existiert ein solcher Strahl auch fuer Y .

2. Teil: r ist der groeszte Wert mit $\delta((X, r)) \subset \delta(P)$.

Angenommen, es gaebe einen Radius $r_0 \in \mathbb{R}$ mit $r_0 > r$, sodass $\delta((X, r_0)) \subset \delta(P)$.

Per Definition von r existiert ein Punkt Q auf dem Rand von P , sodass $\|X - Q\| = r$. Wegen $r_0 > r$, existiert ein Punkt Q' innerhalb des Kreises $K' := (X, r_0)$ auf dem Strahl $[XQ$ mit $\|X - Q'\| > r$. Somit hat die Strecke $[XQ']$ den Punkt Q als Schnittpunkt mit P . Da X innerhalb des Polygons P liegt, folgt daraus, dass Q' ausserhalb liegt, sodass K' nicht vollstaendig innerhalb des Polygons liegt.

Bemerkung 38:

Praktischer Weise laesst sich der Wert r aus obigem Satz einfach bestimmen. Dazu muss fuer jede Kante $[YZ]$ des Polygons der Punkt $Q \in [YZ]$ bestimmt werden, der den minimalen Abstand $\|X - Q\|$ hat. Es gelte $Y = (a, b)$, $Z = (c, d)$ und $X = (x, y)$. Dann ist die Strecke $[YZ]$ gegeben durch

$$[YZ] = \left\{ Y + t \cdot (Z - A) : t \in [0, 1] \right\}.$$

Dann ist fuer die zur Strecke gehoerenden Grade

$$YZ = \{Y + t \cdot (Z - A) : t \in \mathbb{R}\}.$$

der Punkt $Q \in YZ$ mit dem groessten Abstand zu X in Abhaengigkeit von t gegeben durch

$$t = \frac{x \cdot (c-a) + y \cdot (d-b) - a \cdot (c-a) - b \cdot (d-b)}{(c-a)^2 + (d-b)^2}.$$

Dieser Wert laesst sich durch das Nullsetzen der Ableitung der Funktion $t \rightarrow \|X - Q\|$ bestimmen. Nun sind drei Faelle zu unterscheiden:

1. $t \in (0, 1)$. Dann liegt Q auf $[YZ]$ und es handelt sich um den Punkt mit dem kleinsten Abstand zu X .
2. $t \geq 1$. Dann ist Z der Punkt mit dem kleinsten Abstand zu X .
3. $t \leq 0$. Dann ist Y der Punkt mit dem kleinsten Abstand zu X .

7.2. Kreise in Kreisen

Es sei (Z, R) ein Kreis mit Mittelpunkt $Z \in \mathbb{R}^2$ und $R \in \mathbb{R}$. Weiter sei ein Punkt $X \in \mathbb{R}^2$ mit $\|Z - X\| < R$ und $X \neq Z$ gegeben. In diesem Kapitel soll untersucht werden, was der groeszt moegliche Radius r ist, sodass der Kreis (X, r) vollstaendig innerhalb von (Z, R) liegt. Dabei wird ein aehnliches Ergebnis gegeben wie im vorherigen Kapitel:

Satz 39:

Es sei $K = (Z, R)$ ein Kreis und X ein Punkt innerhalb des Kreises. Dann ist der groeszte Radius r , sodass (X, r) vollstaendig in (Z, R) enthalten ist, gegeben durch

$$r = \min \{ \|X - Q\| : Q \in \beta(K)\}.$$

Das ist der kleinste Abstand von X zu einem Punkt Q dem Rand des Kreises.

Beweis: Trivial.

Bemerkung 40:

Erneut laesst sich der Wert r nach obigen Satz einfach bestimmen:

Wir suchen den Punkt Q auf dem Rand des Kreises K , der auf dem Strahl $[ZX$ liegt. Das heiszt, wenn $Z = (a, b)$, $X = (x, y)$ und $Q = (c, d)$, ist bekannt, dass

1. $Q = Z + t \cdot (X - Z)$ fuer $t > 1$, und
2. $(c - a)^2 + (d - b)^2 = R^2$.

Daraus folgt:

$$(c, d) = (a + t \cdot (x - a), b + t \cdot (y - b))$$

Dann

$$t^2 \cdot (x - a)^2 + t^2 \cdot (y - b)^2 = R^2$$

sodass weiter

$$t = \frac{R}{\sqrt{(x - a)^2 + (y - b)^2}}$$

Dann gilt

$$Q = Z + \frac{R}{\sqrt{(x - a)^2 + (y - b)^2}} \cdot (X - Z)$$

und

$$r = \|X - Q\|.$$

7.3. Loesungsvorschlag

Wie zuvor sollen zunaechst mit den bereits vorgestellten Methoden zwei Mengen an Punkten innerhalb des Polygons berechnet werden: $G \subseteq \delta_{\mathbb{Z}}(P)$ und $T \subseteq \delta_{\mathbb{Z}}(P)$, wobei G die Menge der potenziellen Positionen des Gesundheitszentrums und T die Menge der potenziellen Punkte der Ortschaften ist. Dabei koennen diese Menge wieder unabhaenig sein, oder beispielsweise $G = T$ oder $G \subset T$ gelten.

Nun soll ebenfalls jedes $Z \in G$ als Gesundheitszentrum getestet werden. Fuer ein festes Z soll dann eine Menge T_Z der Form

$$T_Z = \left\{ (Y, r_Y) : Y \in T \text{ und } r_y \in \mathbb{Z} \text{ mit } r_y > 0 \right\}$$

an Kreisen mit Mittelpunkten in T erzeugt werden, sodass die Bedinungen bezueglich der Abstaende A , B und R eingehalten werden. Die Loesung besteht dann in dem Punkt Z und der dazugehoerigen Menge T_Z der Kreise, sodass die Summe $\sum_{(Y, r_Y) \in T_Z} \pi r_Y^2$ maximal ist.

Algorithmus 3 : Loesungsvorschlag

```

Input : Polygon  $P$ , Menge  $G \subseteq \delta_{\mathbb{Z}}(P)$  und Menge  $T \subseteq \delta_{\mathbb{Z}}(P)$ 
Output : Punkt  $Z \in G$ , sowie Menge der Kreise  $T_Z$ 
1  $T_{\text{best}} \leftarrow \emptyset$ 
2  $Z_{\text{best}} \leftarrow \text{NULL}$ 
3 foreach  $Z \in G$  do
4    $T_Z \leftarrow \text{generate}(P, T, Z)$  // Menge an Kreisen mit Mittelpunkten in  $T$ 
5   if  $\sum_{(Y, r_Y) \in T_Z} \pi r_Y^2 > \sum_{(Y, r_Y) \in T_{\text{best}}} \pi r_Y^2$  then
6      $T_{\text{best}} \leftarrow T_Z$ 
7      $Z_{\text{best}} \leftarrow Z$ 
8   end if
9 end foreach
10 return  $Z_{\text{best}}, T_{\text{best}}$ 
```

7.4. Bestimmung der Radien

Zur Bestimmung dieser Radien fuer ein gegebenes Gesundheitszentrum am Punkt Z sollen nun ein Loesungsverfahren vorgestellt werden. Dabei soll fuer die Menge T_Z der Kreise $K = (Y, r_Y)$ $R_1 \geq r_y \geq R_0$ und die folgenden Bedingungen gelten:

(B0) Alle Kreise liegen vollstaendig innerhalb des Polygons P : $\delta(K) \subset \delta(P)$ fuer alle $K = (Y, r_Y) \in T_Z$. Dies ist genau dann der Fall, wenn r_Y kleiner oder gleich dem Wert aus Kapitel 7.1 ist.

(B1) Fuer alle $K = (X, r_X), K' = (Y, r_Y) \in T_Z$ mit $X \neq Y$ gilt:

$$\|X - Y\| - (r_X + r_Y) \geq A$$

(B2) Fuer alle $K = (X, r_X), K' = (Y, r_Y) \in T_Z$ mit $X \neq Y$ gilt:

$$\left(\|X - Y\| - (r_X + r_Y) \geq B \right) \text{ oder } \left(\|Z - X\| + r_X \leq R \right) \text{ oder } \left(\|Z - Y\| + r_Y \leq R \right)$$

Greedy-Verfahren

Die Bestimmung der Radien fuer die Punkte der Menge T soll durch ein Greedy-Verfahren geschehen. Dabei ist fuer $Y \in T$ R_Y der maximale Radius des Punktes Y , sodass der Kreis $K = (Y, R_Y)$ vollstaendig innerhalb des Polygons P liegt (siehe Kapitel 7.1) und R'_Y der maximale Radius, sodass $K = (Y, R'_Y)$ vollstaendig innerhalb des Kreises (Z, R) liegt.

Es sollen nun itereativ Kreise (Y, r_y) mit Mittelpunkten in T und Radien $r_Y \leq R_Y$ bestimmt werden, sodass die Bedinungen (B1) und (B2) erfuellt sind. Dazu sollen als erstes moeglichst viele Kreise (Ortschaften) vollstaendig im Kreis (Z, R) des Gesundheitszentrums generiert werden und als zweites die restlichen Kreise.

Fuer den ersten Teil wird wie folgt vorgegangen:

Es wird die Menge M_0 der Kreise (Y, r_Y) mit Mittelpunkten $Y \in T$ betrachtet, mit dem groeszt moeglichen

Radius groeszer 0, sodass der Mindestabstand A zu allen bisher gewählten Kreisen gemäß Bedingung (B1) eingehalten wird. Das ist genau die Menge

$$M_0 = \left\{ (Y, r_Y) : Y \in T \text{ mit } Y \in \delta((Z, R)) \text{ und } \exists r_Y = \min\{\|X - Y\| - A - r_x : (X, r_X) \in T_Z\} > R_0 \right\}$$

Dabei ergibt sich der Radius

$$r_Y = \min\{\|X - Y\| - A - r_x : (X, r_X) \in T_Z\}$$

durch eine Umformung der Bedingung (B1). Fuer den Sonderfall, dass die Menge T_Z (noch) leer ist, soll der Wert ∞ benutzt werden.

Aus dieser Menge wird der Kreis mit dem groessten Radius $\min\{r_Y, R_Y, R'_Y, R_1\}$ ausgewählt (sodass der Kreis die Mindestabstaende einhaelt, im Kreis (Z, R) und in P liegt und den Maximalradius R_0 gewaehrleistet) und zur Menge der Kreise T_Z hinzugefuegt. Dieser Vorgang wird wiederholt bis keine Punkte mehr gefunden werden (d.h. die Menge der betrachteten Punkte leer ist).

Fuer den zweiten Teil wird die Menge M_1 der Kreise mit Mittelpunkt in T ausserhalb des Kreises (Z, R) betrachtet, die die Mindestabstaende A bzw. B zu allen bisher gewählten Punkten gemäß Bedingungen (B1) und (B2) einhalten. Dies ist genau die Menge

$$M_1 = \left\{ (Y, r_Y) : Y \in T \text{ mit } Y \notin \delta((Z, R)) \text{ und } \exists r_Y = \min\{r'_Y, r''_Y\} > R_0 \right\}$$

Der groeszt mögliche Radius des Punktes Y ist dabei gegeben durch das Minimum $\min\{r'_Y, r''_Y\}$, wobei r'_Y und r''_Y wie folgt definiert werden:

$$\min\{\|X - Y\| - A - r_x : (X, r_X) \in T_Z \text{ mit } (X, r_X) \in \delta((Z, R))\}$$

und

$$\min\{\|X - Y\| - B - r_x : (X, r_X) \in T_Z \text{ mit } (X, r_X) \notin \delta((Z, R))\}$$

D.h., sie entsprechen dem groesstmöglichen Wert der Sich aus bedinung (B1) bzw. (B2) ergibt.

Aus der Menge M_1 wird dann erneut der Kreis mit dem groessten Radius $\min\{r_Y, R_Y, R'_Y, R_1\}$ gewählt. Auch dieser Vorgang wird wiederholt bis keine Punkte mehr gefunden werden (d.h. die Menge der betrachteten Punkte leer ist).

Algorithmus 4 : Greedy-Verfahren zur Berechnung von T_Z

```

Input : Polygon  $P$ , Punkt  $Z$  und Menge  $T \subseteq \delta_{\mathbb{Z}}(P)$ 
Output : Menge an Kreisen  $T_Z$ 

1 // Erster Teil
2  $T_Z \leftarrow \emptyset$ 
3  $M_0 \leftarrow \left\{ (Y, r_Y) : Y \in T \text{ mit } Y \in \delta((Z, R)) \text{ und } \exists r_Y = \min\{\|X - Y\| - A - r_x : (X, r_X) \in T_Z\} > R_0 \right\}$ 
4 while  $M_0 \neq \emptyset$  do
5    $(Y, r_Y) \leftarrow$  Circle with biggest value  $\min\{r_Y, R_Y, R'_Y, R_1\}$  in set  $M_0$ 
6    $T_Z \leftarrow T_Z \cup \{(Y, \min\{r_Y, R_Y, R'_Y, R_1\})\}$ 
7    $M_0 \leftarrow \left\{ (Y, r_Y) : Y \in T \text{ mit } Y \in \delta((Z, R)) \text{ und } \exists r_Y = \min\{\|X - Y\| - A - r_x : (X, r_X) \in T_Z\} > R_0 \right\}$ 
8 end while
9 // Zweiter Teil
10  $M_1 \leftarrow \left\{ (Y, r_Y) : Y \in T \text{ mit } Y \notin \delta((Z, R)) \text{ und } \exists r_Y = \min\{r'_Y, r''_Y\} > R_0 \right\}$ 
11 while  $M_1 \neq \emptyset$  do
12    $(Y, r_Y) \leftarrow$  Circle with biggest value  $\min\{r_Y, R_Y, R_1\}$  in set  $M_1$ 
13    $T_Z \leftarrow T_Z \cup \{(Y, \min\{r_Y, R_Y, R_1\})\}$ 
14    $M_1 \leftarrow \left\{ (Y, r_Y) : Y \in T \text{ mit } Y \notin \delta((Z, R)) \text{ und } \exists r_Y = \min\{r'_Y, r''_Y\} > R_0 \right\}$ 
15 end while
16 return  $T_Z$ 

```

7.5. Analyse der Zeitkomplexitaet der Loesung

Analog zum letzten Kapitel setzt sich die Laufzeit des Greedy-Loesungsverfahren aus dem Algorithmus 3 und der *generate*-Funktion zusammen. Es seien wieder $\#(T)$ die Groesze der Menge T und $\#(G)$ die Groesze der Menge G .

Zunaechst soll die Zeitkomplexitaet der *generate*-Funktion analysiert werden. Wieder ist es moeglich die Mengen M_0 und M_1 in Zeit $\mathcal{O}(\#(T))$ zu initialisiert und neuzusetzen, wobei R_Y und R'_Y . Da die While-Schleifen ebenfalls jeweils maximal $\mathcal{O}(\#(T))$ mal ausgefuehrt werde, ergibt sich erneut die Zeitkomplexitaet $\mathcal{O}(\#(T)^2)$. Nun laesst sich die finale Zeitkomplexitaet des Algorihtmus 3 angeben. Es folgt die Zeitkomplexitaet

$$\mathcal{O}(\#(G) \cdot \#(T)^2),$$

denn die *generate*-Funktion wird genau $\#(G)$ mal ausgefuehrt und die Berechnung der Flaeche und der Radianen R_Y und R'_Y funktioniert jeweils in $\mathcal{O}(\#(T))$.

8. Implementierung

Die vorgestellten Loesungen wurde in C++ und Python implementiert. Dabei wurden die geometrischen Algorithmen ohne die Benutzung von Bibliotheken in C++ implementiert. Darunter die Algorithmen zur Triangulierung und zufaellige Berechnung von Punkten innerhalb eines Polygons. Dabei wurden Punkte durch pairs an long long's umgesetzt. Die Loesungsvorschlaege, die ein Greedy-Verfahren nutzen, wurden ebenfalls in C++ implementiert.

Nur der Loesungsvorschlag, der ein ILP verwendet, wurden in Python implementiert. Dabei wurden die Punkte, die in C++ berechnet wurden, in eine Datei geschrieben und in Python wieder ausgelesen. Fuer das ILP wurde das Python Paket »PuLP« zur Modellierung und Loesung verwendet. Genauer wurde intern »CBC« (»COIN-OR branch and cut«) als Solver verwendet.⁷ Dieses Python Paket ermoeglicht es Variablen, Bedingungen (in Form der Linearen Ungleichungen) und die Zielfunktion, die maximiert oder minimiert wird, zu definieren und anschlieszend mit einem bestimmtem Solver (hier CBC) das Ganzzahlige Optimierungsproblem perfekt zu loesen. Fuer sehr grosze Instanzen kann es allerdings sinnvoll sein, ein Zeitlimit festzulegen, sodass das Programm nach einer gegebenen Zeit beendet wird und ggf. die bis dahin beste Loesung ausgegeben wird. Dabei besteht auch die Moeglichkeit das in der vorgegebenen Zeit keine Loesung gefunden werden kann.

Detailliert wurden diese vorgestellten Loesungsvorschlaege implementiert:

1. Erweiterung I

- (a) Triangulierung und generierung zufaelliger Punkte (C++)
- (b) Loesungsvorschlag mit Greedy-Verfahren (C++)
- (c) Loesungsvorschlag mit ILP (Python)

2. Erweiterung III

- (a) Loesungsvorschlag mit Greedy-Verfahren (C++)

Die Programme lesen Dateien ein, welche die Punkte als Dezimalzahlen in der Einheit Kilometern beinhalten, so wie die Eingaben der BwInf-Website. Ausserdem ist vorrausgesetzt, dass die Polygone in mathematisch positiver Richtung (Orientierung gegen den Uhrzeigersinn) gegeben sind, damit die Triangulierung funktioniert. Beim Einlesen werden die Dezimalzahlen dann durch Multiplikation mit 10^6 in ganze Zahlen umgewandelt (siehe Kapitel 3).

Die einzige Stelle, an der dies zu einem potenziellen Problem wird, ist die Erzeugung der pseudozufaelligen Punkte innerhalb des Polygons, die auf die Erzeugung pseudozufaelliger Punkte in Dreiecken zurueckgefuehrt wurde. Den hier muss gerundet werden, sodass es sich um ganzzahlige Punkte handelt. Ist $\alpha + \beta$ (sehr) nah an 1, so kann es durch das Runden moeglicherweise passieren, dass Punkte ausserhalb des Dreiecks (und somit teilweise ausserhalb des Polygons) liegen. Um dies zu verhindern, wurde in der Implementierung ein kleineres Intervall als $[0, 1]$ verwendet.

⁷Bei beiden handelt es sich um Open-Source Projekte der »Computational Infrastructure for Operations Research«

9. Beispiele

9.1. Die Beispiele der BwInf-Website

Fuer diese Beispiele soll jeweils die Triangulierung, die Menge der zufaelligen Punkte innerhalb des Polygons und die Ergebnisse der zwei Loesungsverfahren gezeigt und erlaeutert werden. Die Loesung des Greedy-Verfahrens nutzt jeweils etwa 1000 gleichmaesig verteilte, pseudozufaellige Punkte fuer die Mengen G und T .

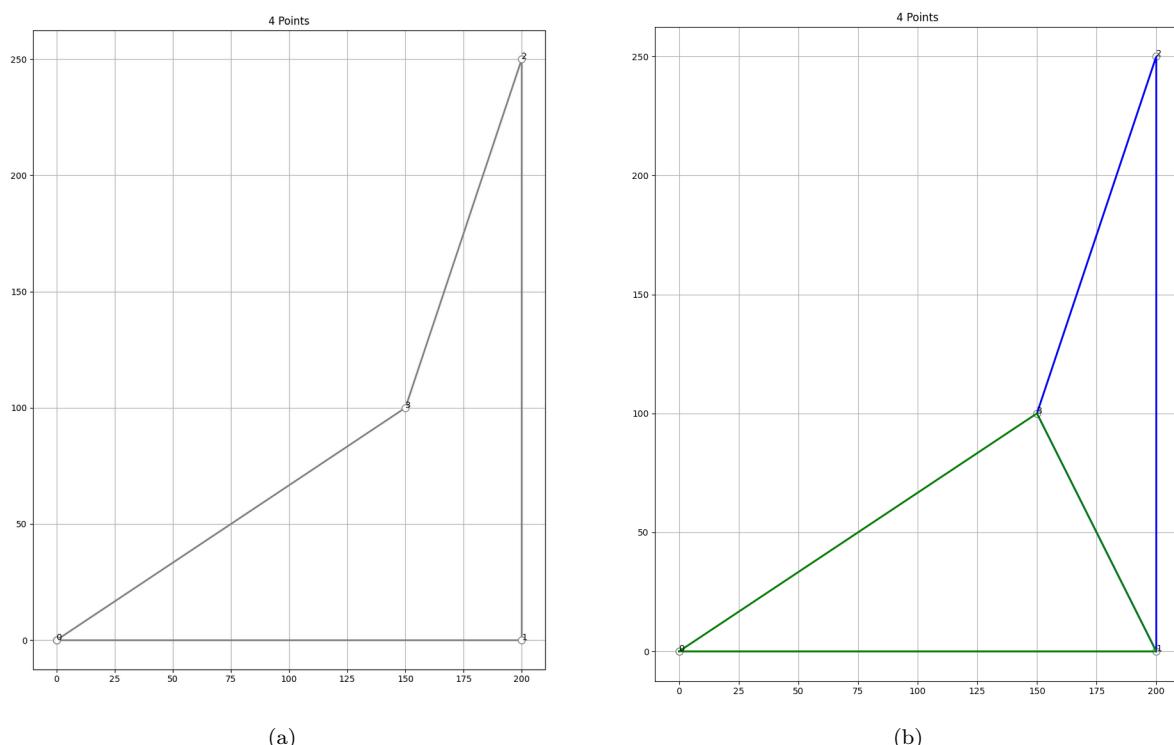
Die Loesungen des ILP-Verfahrens wurden nicht aufgefuehrt, da diese in der Praxis kaum bessere Loesungen ergeben. Konkret sollten fuer das ILP-Verfahren Mengen T und G mit um die 300-400 Punkten verwendet werden. Merkbar bessere Ergebnisse erhaelt man allerdings erst, wenn man ebenfalls etwa 1000 Punkte verwendet. Dies fuehrt allerdings zu einer deutlich zu hohen Laufzeit.

Die Abbildungen wurden durch Python-Programme mit der Hilfe der Python-Pakete Numpy und Matplotlib erstellt.

1. Beispiel: sieder1.txt

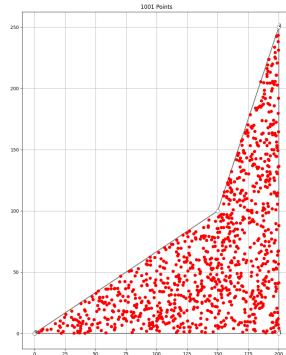
Dieses Beispiel ist in mathematisch positiver Richtung (d.h., gegen den Uhrzeigersinn) gegeben. Das Polygon, und eine Triangulierung sehen wie folgt aus: Die Triangulierung besteht aus zwei Dreiecken: $((200.0, 0.0), (200.0, 250.0), (150.0, 100.0))$ und $((0.0, 0.0), (200.0, 0.0), (150.0, 100.0))$.

Abbildung 16: Polygon und Triangulierung sieder1.txt



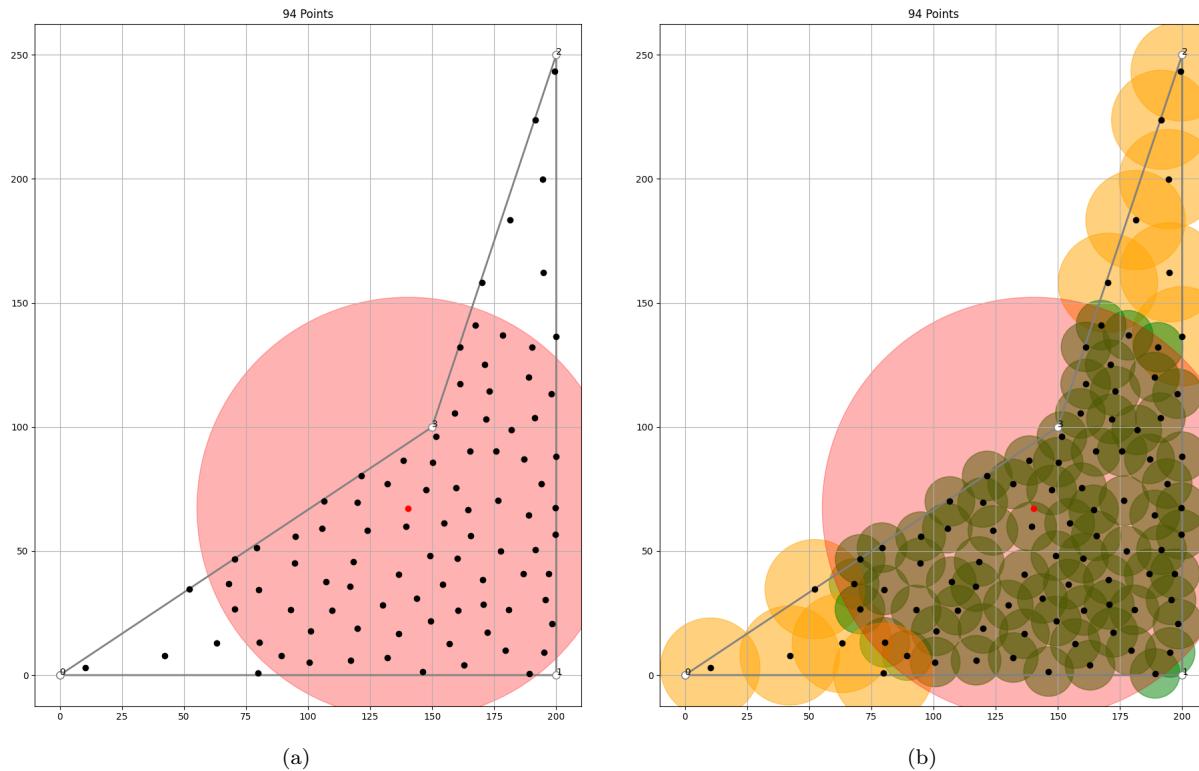
Eine beispielhafte, pseudozufaellige gleichmaesige Verteilung von ca. 1000 Punkten in dem Polygon des ersten Beispiels sieht wie folgt aus:

Abbildung 17: Pseudozufaellige Punkte fuer siedler1.txt

**Loesung des Greedy-Verfahrens:**

Nutzt man diese Punkte fuer die Mengen G und T im Greedy-Verfahren, so erhaelt man innerhalb von 5011 Millisekunden eine Loesung mit 93 Ortschaften, exklusive Gesundheitszentrum:

Abbildung 18: 1. Loesung siedler1.txt



In den Abbildungen ist das Gesundheitszentrum durch einen roten Punkt dargestellt und dessen Radius durch einen roten Kreis dargestellt. Die Ortschaften wiederum sind durch schwarze Punkte angegeben.

Die rechte Abbildung weisst zusaetlich fuer alle Ortschaften einen weiteren Kreis auf, der den Abstandsbedingungen entspricht: Falls die Ortschaft innerhalb des 85km-Radius des Gesundheitszentrums liegt, wird um sie ein gruener Kreis mit Radius 10km dargestellt. Fall sie allerdings ausserhalb des 85km-Radius des Gesundheitszentrums liegt, wird um sie ein orangener Kreis mit Radius 20km dargestellt.

Die Abstandsbedingungen bedeutet dann, dass jeweils ein gruener Kreis nicht den Mittelpunkt eines anderen gruenen Kreises enthaelten darf, und dass ein orangener Kreis nicht den Mittelpunkt eines anderen orangenen Kreis enthaelten darf.

Abbildung 19: Polygon siedler2.txt

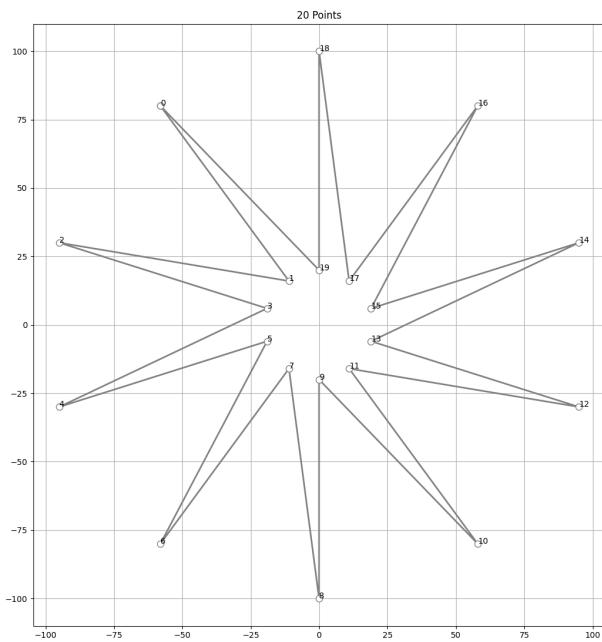
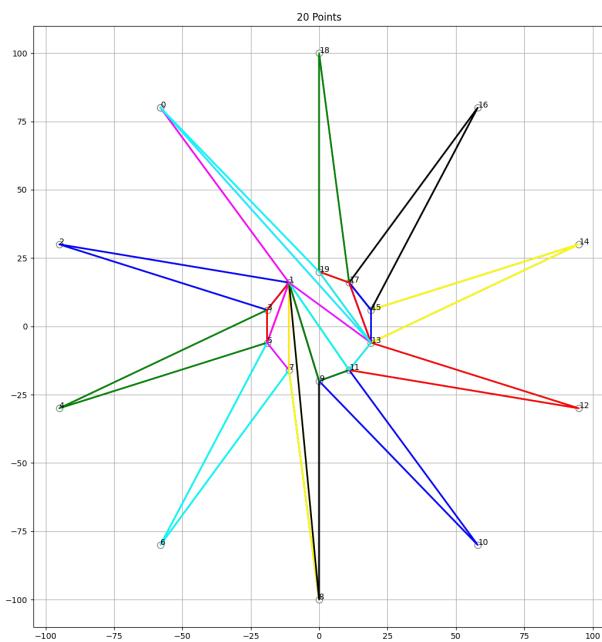


Abbildung 20: Triangulierung siedler2.txt



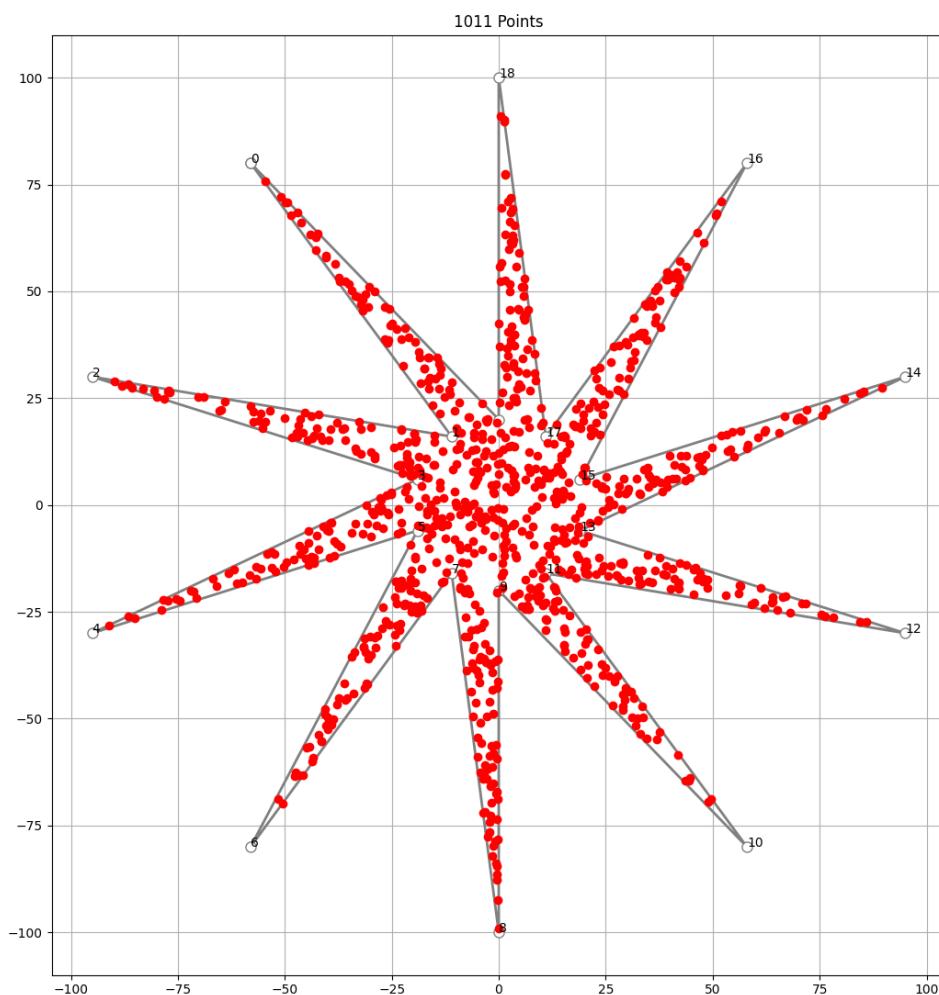
2. Beispiel: sieder2.txt

In der BwInf-Eingabe war dieses Beispiel in mathematisch negativer Richtung angegeben. Deswegen wurde die Reihenfolge der Punkte umgekehrt. Das Polygon, und eine Triangulierung sehen wie folgt aus (s. oben). Dabei besteht die Triangulierung aus diesen 18 Dreiecken:

((-11.0, 16.0), (-95.0, 30.0), (-19.0, 6.0)), ((-19.0, 6.0), (-95.0, -30.0), (-19.0, -6.0)),
 ((-11.0, 16.0), (-19.0, 6.0), (-19.0, -6.0)), ((-19.0, -6.0), (-58.0, -80.0), (-11.0, -16.0)),
 ((-11.0, 16.0), (-19.0, -6.0), (-11.0, -16.0)), ((-11.0, 16.0), (-11.0, -16.0), (0.0, -100.0)),
 ((-11.0, 16.0), (0.0, -100.0), (0.0, -20.0)), ((0.0, -20.0), (58.0, -80.0), (11.0, -16.0)),
 ((-11.0, 16.0), (0.0, -20.0), (11.0, -16.0)), ((11.0, -16.0), (95.0, -30.0), (19.0, -6.0)),
 ((-11.0, 16.0), (11.0, -16.0), (19.0, -6.0)), ((-58.0, 80.0), (-11.0, 16.0), (19.0, -6.0)),
 ((19.0, -6.0), (95.0, 30.0), (19.0, 6.0)), ((19.0, 6.0), (58.0, 80.0), (11.0, 16.0)), ((19.0, -6.0), (19.0, 6.0), (11.0, 16.0)),
 ((11.0, 16.0), (0.0, 100.0), (0.0, 20.0)), ((19.0, -6.0), (11.0, 16.0), (0.0, 20.0)), und
 ((-58.0, 80.0), (19.0, -6.0), (0.0, 20.0)).

Etwa 1000 pseudozufaellig veteilte Punkte im Polygon sehen dann so aus:

Abbildung 21: Pseudozufaellige Punkte fuer sieder2.txt



Loesung des Greedy-Verfahrens:

Verwendet man das Greedy-Verfahren mit zufaellig generierten Punkten fuer die Mengen G und T wie oben beschrieben, erhaelt man nach 4598 Millisekunden eine Loesung mit 68 Ortschaften:

Abbildung 22: 1. Loesung siedler2.txt

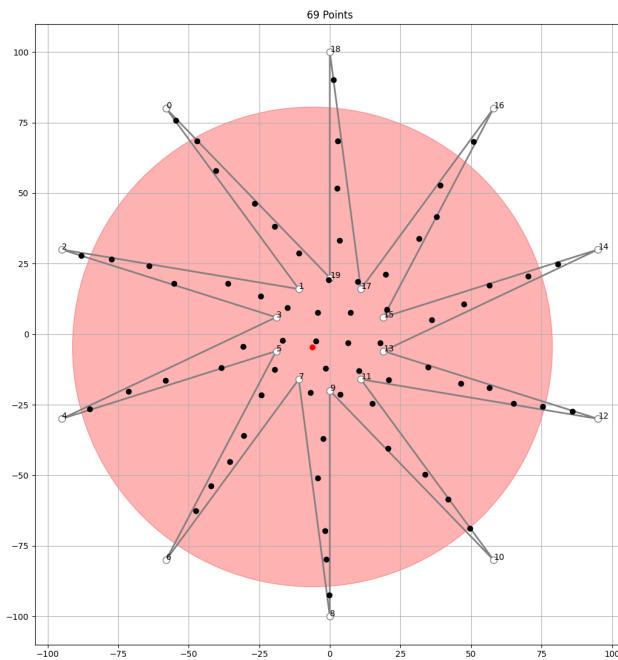
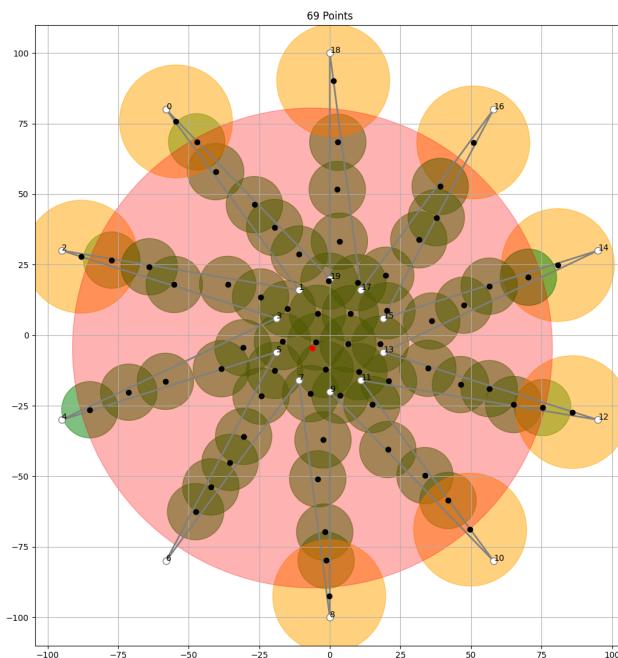


Abbildung 23: 1. Loesung siedler2.txt



3. Beispiel: sieder3.txt

Das Polygon des dritten Beispiels der BwInf-Website ist wieder in mathematisch positiver Richtung angegeben und seine Triangulierung besteht ebenfalls aus zwei Dreiecken:

$((0.0, 0.0), (150.0, 0.0), (150.0, 150.0))$ und $((0.0, 0.0), (150.0, 150.0), (0.0, 150.0)).$

Eine Verteilung von ca. 1000 Punkten fuer dieses Beispiel sieht wie folgt aus:

Abbildung 24: Polygon und Triangulierung sieder3.txt

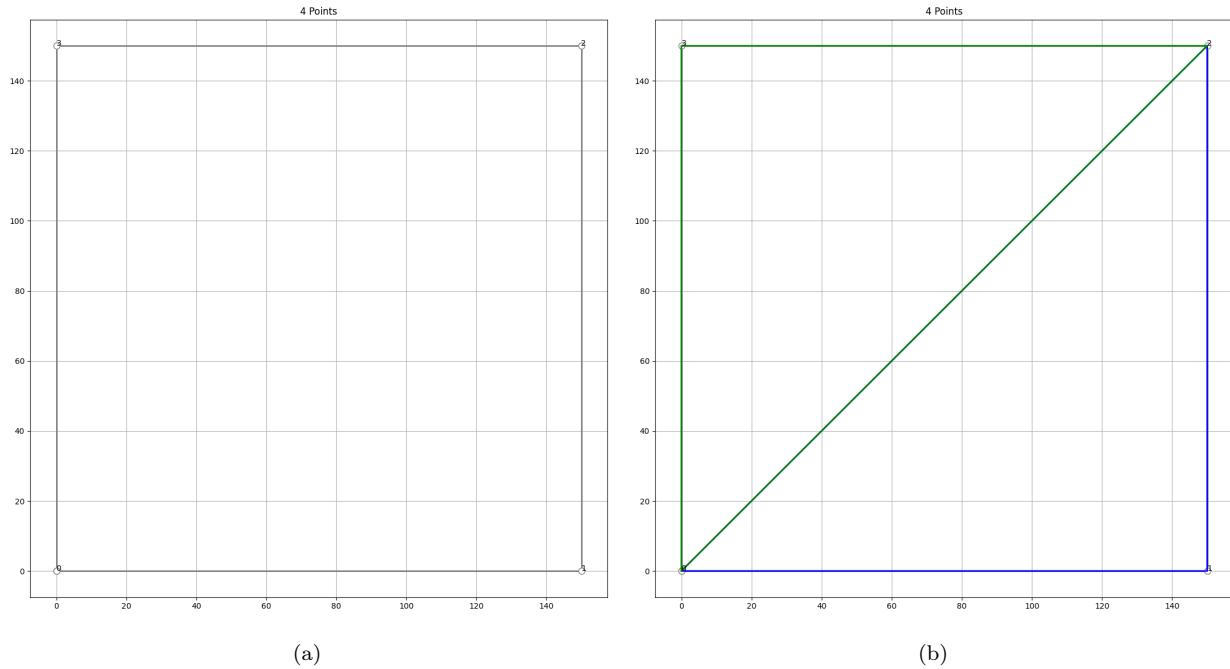
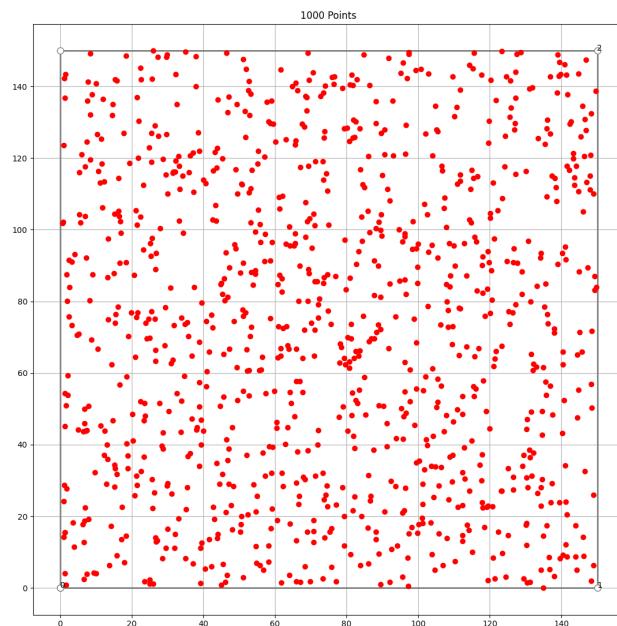


Abbildung 25: Pseudozufaellige Punkte fuer sieder3.txt



Loesung des Greedy-Verfahrens:

Verwendet man das Greedy-Verfahren mit zufaellig generierten Punkten fuer die Mengen G und T wie oben beschrieben, erhaelt man nach 6148 Millisekunden eine Loesung mit 121 Ortschaften:

Abbildung 26: 1. Loesung siedler3.txt

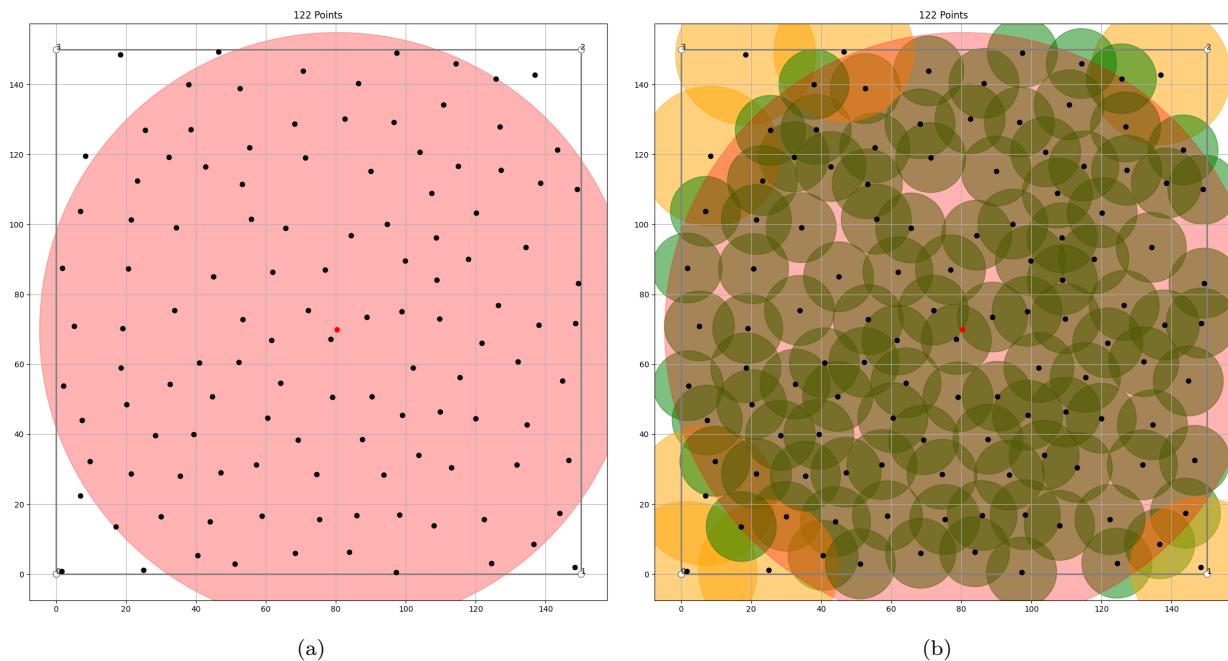


Abbildung 27: Polygon siedler4.txt

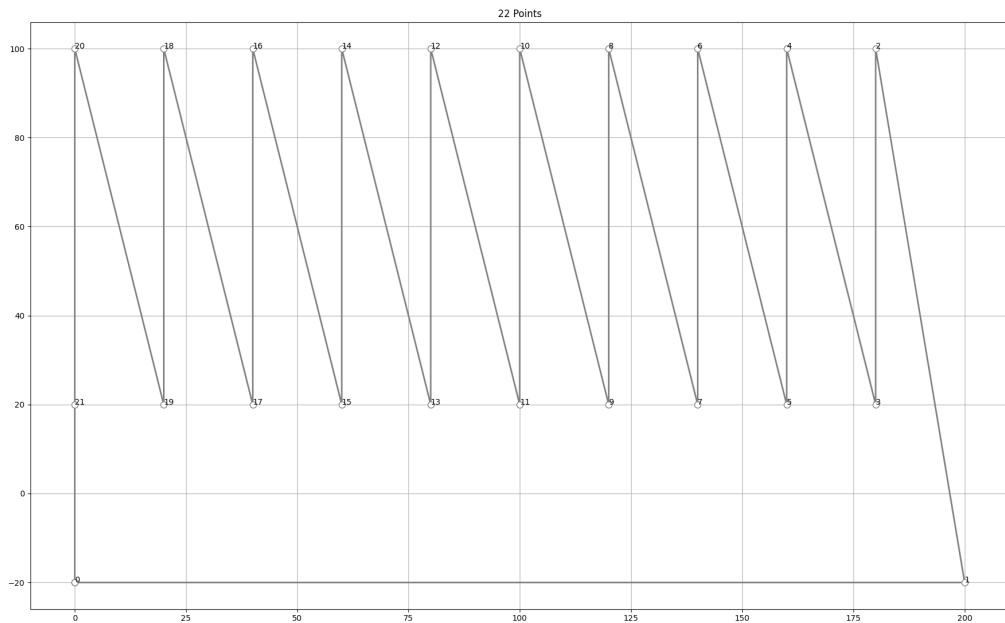
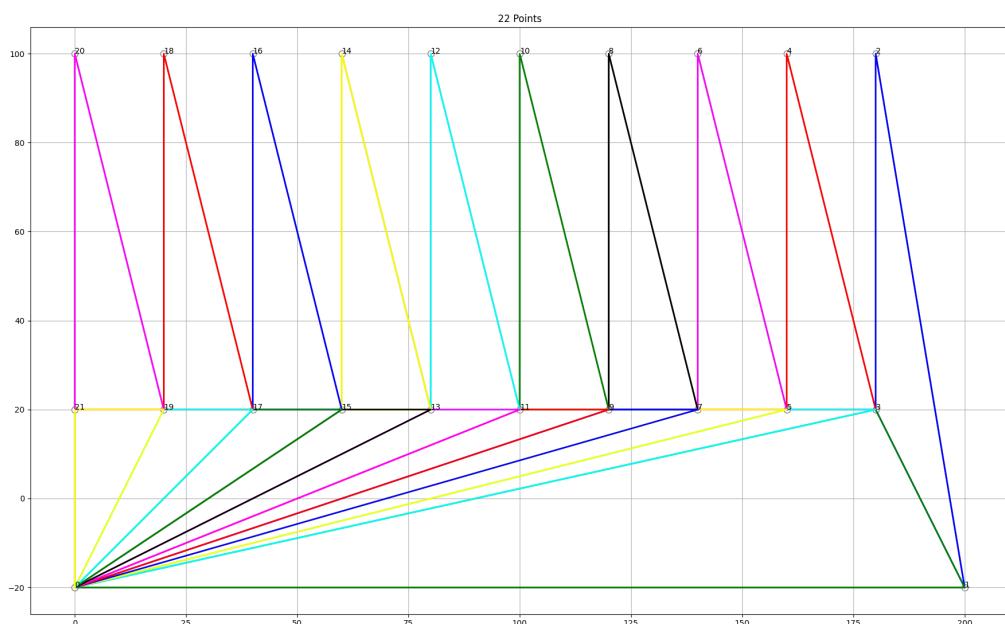


Abbildung 28: Triangulierung siedler4.txt



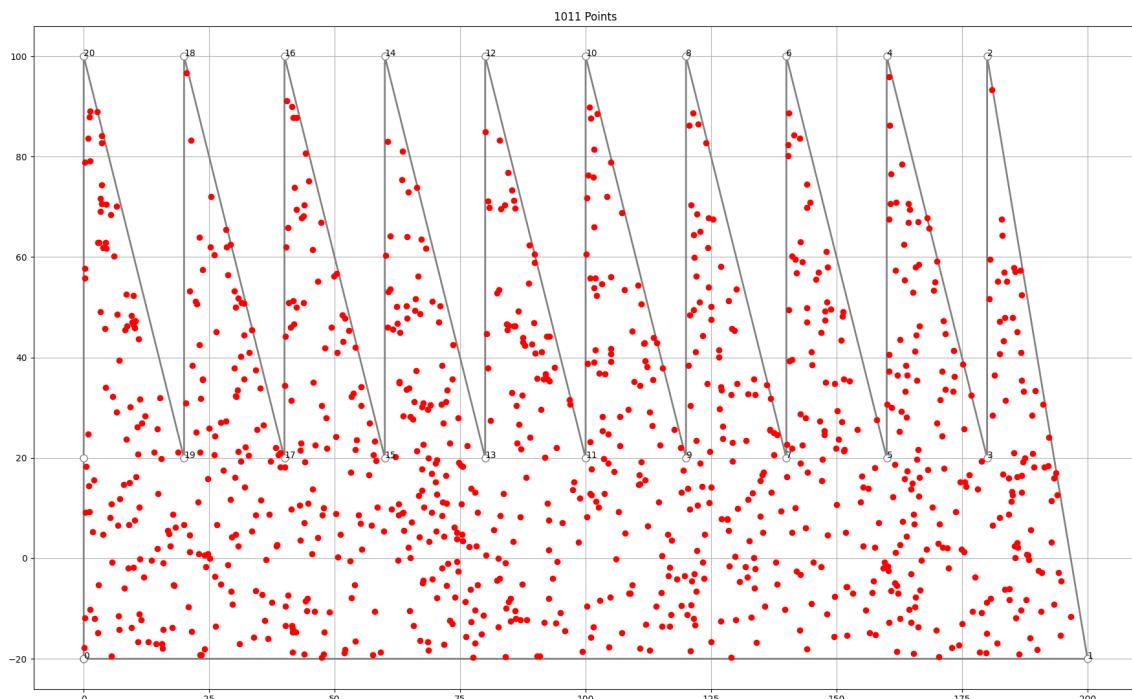
4. Beispiel: sieder4.txt

Das Polygon im vierten Beispiel der BwInf-Website ist erneut in mathematisch negativer Richtung gegeben. Deswegen wurde die Reihenfolge der Punkte umgekehrt. Die dazugehörige Triangulierung besteht aus diesen 20 Dreiecken und sieht wie folgt aus (s. oben):

((200.0, -20.0), (180.0, 100.0), (180.0, 20.0)), ((0.0, -20.0), (200.0, -20.0), (180.0, 20.0)),
((180.0, 20.0), (160.0, 100.0), (160.0, 20.0)), ((0.0, -20.0), (180.0, 20.0), (160.0, 20.0)),
((160.0, 20.0), (140.0, 100.0), (140.0, 20.0)), ((0.0, -20.0), (160.0, 20.0), (140.0, 20.0)),
((140.0, 20.0), (120.0, 100.0), (120.0, 20.0)), ((0.0, -20.0), (140.0, 20.0), (120.0, 20.0)),
((120.0, 20.0), (100.0, 100.0), (100.0, 20.0)), ((0.0, -20.0), (120.0, 20.0), (100.0, 20.0)),
((100.0, 20.0), (80.0, 100.0), (80.0, 20.0)), ((0.0, -20.0), (100.0, 20.0), (80.0, 20.0)),
((80.0, 20.0), (60.0, 100.0), (60.0, 20.0)), ((0.0, -20.0), (80.0, 20.0), (60.0, 20.0)),
((60.0, 20.0), (40.0, 100.0), (40.0, 20.0)), ((0.0, -20.0), (60.0, 20.0), (40.0, 20.0)),
((40.0, 20.0), (20.0, 100.0), (20.0, 20.0)), ((0.0, -20.0), (40.0, 20.0), (20.0, 20.0)),
((20.0, 20.0), (0.0, 100.0), (0.0, 20.0)), und ((0.0, -20.0), (20.0, 20.0), (0.0, 20.0)).

Eine beispielhafte Verteilung 1000 pseudozufälliger Punkte sieht dann wie folgt aus:

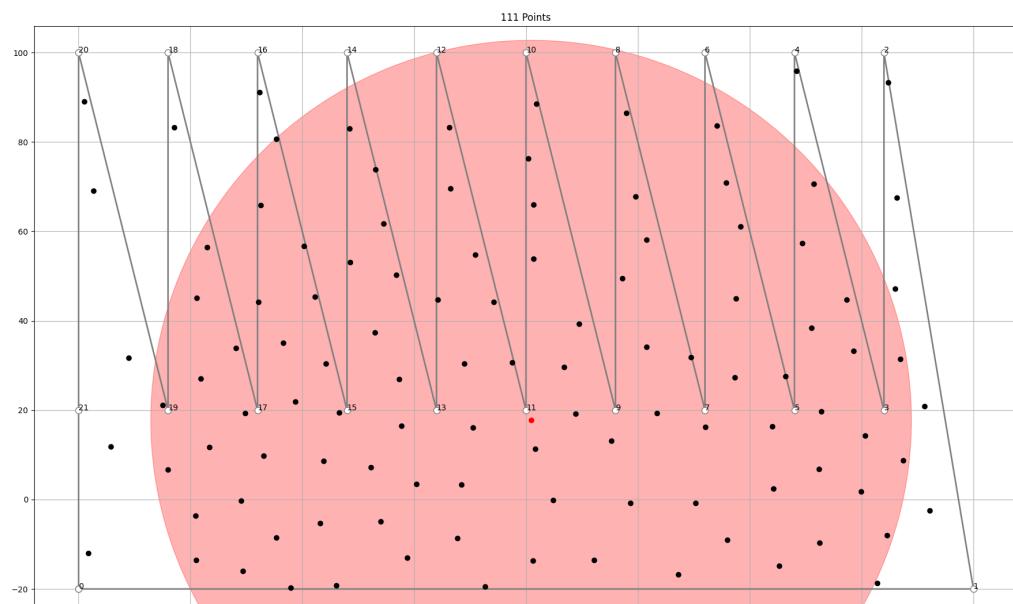
Abbildung 29: Pseudozufällige Punkte für sieder4.txt



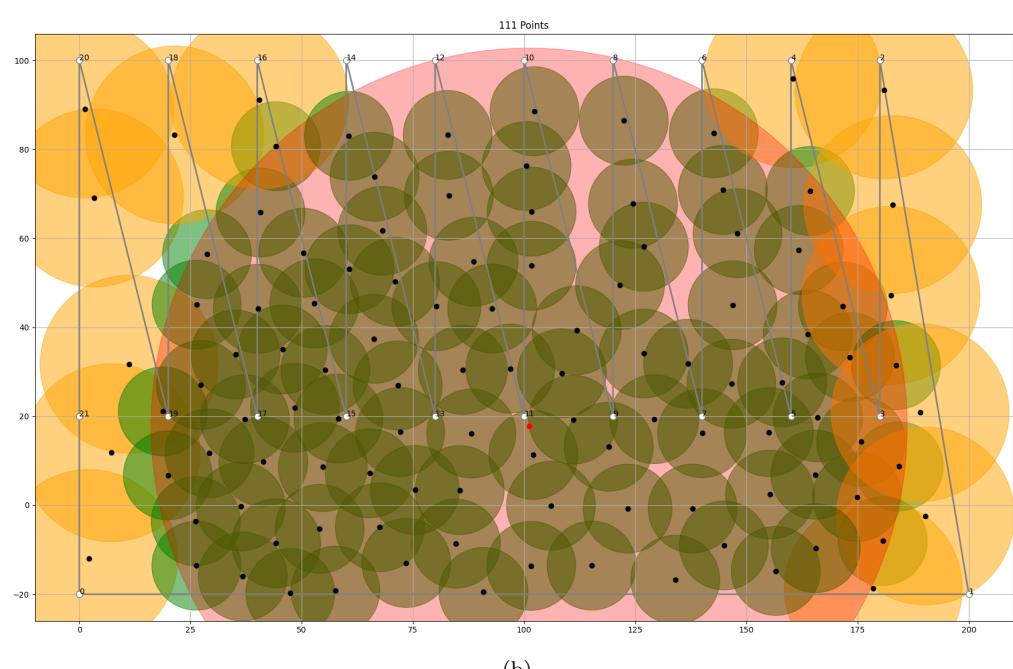
Loesung des Greedy-Verfahrens:

Verwendet man das Greedy-Verfahren mit zufaellig generierten Punkten fuer die Mengen G und T wie oben beschrieben, erhaelt man nach 7054 Millisekunden eine Loesung mit 110 Ortschaften:

Abbildung 30: 1. Loesung siedler4.txt



(a)



(b)

Abbildung 31: Polygon siedler5.txt

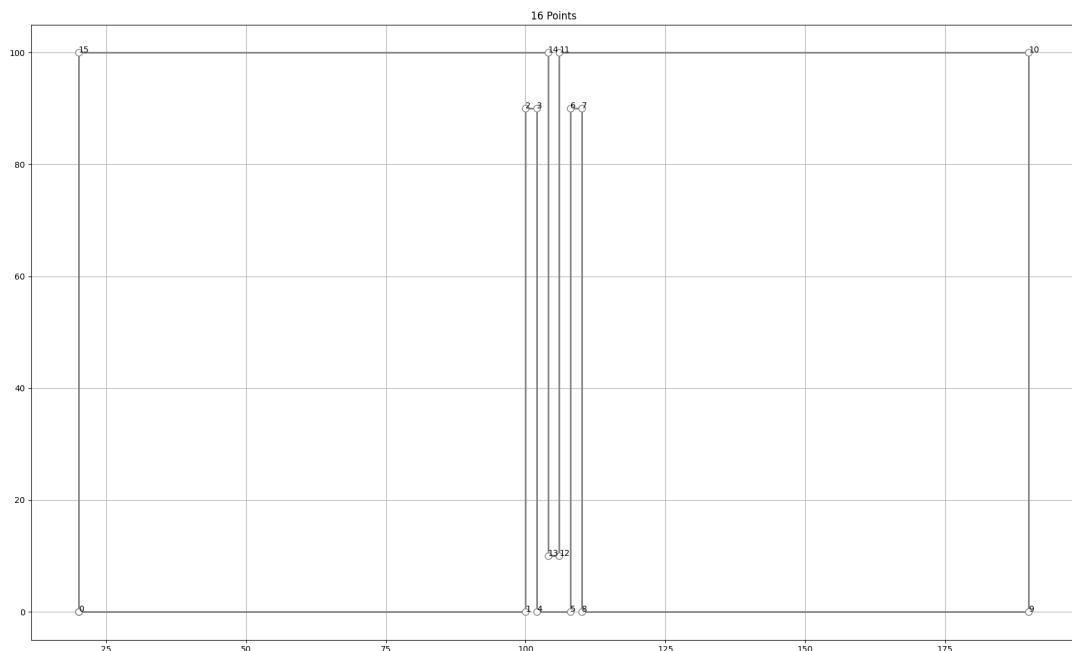


Abbildung 32: Triangulierung siedler5.txt

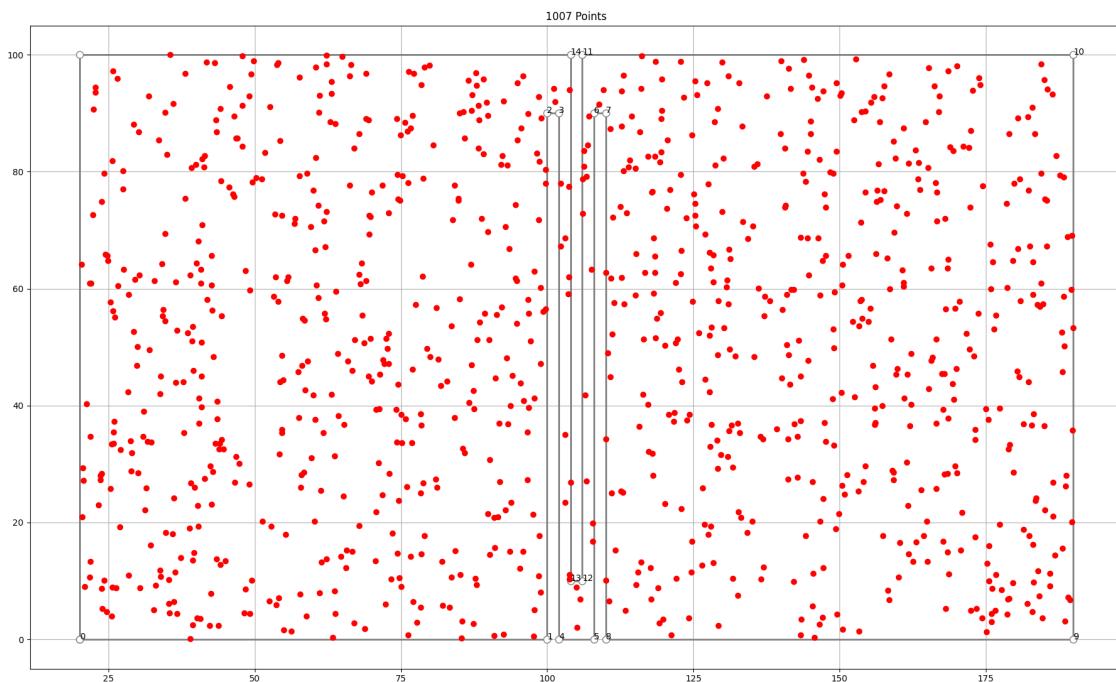


5. Beispiel: sieder5.txt

Das letzte Beispiel der BwInf-Website ist wieder in mathematisch positiver Richtung gegeben, sodass die Reihenfolge der Punkte beibehalten wird⁸. Das Polygon und die dazugehörige Triangulierung sehen wie folgt aus (siehe oben). Dabei besteht die Triangulierung aus diesen 14 Dreiecken:

((20.0, 0.0), (100.0, 0.0), (100.0, 90.0)), ((110.0, 90.0), (110.0, 0.0), (190.0, 0.0)),
 ((110.0, 90.0), (190.0, 0.0), (190.0, 100.0)), ((108.0, 90.0), (110.0, 90.0), (190.0, 100.0)),
 ((108.0, 90.0), (190.0, 100.0), (106.0, 100.0)), ((108.0, 0.0), (108.0, 90.0), (106.0, 100.0)),
 ((108.0, 0.0), (106.0, 100.0), (106.0, 10.0)), ((102.0, 0.0), (108.0, 0.0), (106.0, 10.0)),
 ((102.0, 0.0), (106.0, 10.0), (104.0, 10.0)), ((102.0, 90.0), (102.0, 0.0), (104.0, 10.0)),
 ((102.0, 90.0), (104.0, 10.0), (104.0, 100.0)), ((100.0, 90.0), (102.0, 90.0), (104.0, 100.0)),
 ((20.0, 0.0), (100.0, 90.0), (104.0, 100.0)) und ((20.0, 0.0), (104.0, 100.0), (20.0, 100.0)).

Abbildung 33: Pseudozufällige Punkte für sieder5.txt

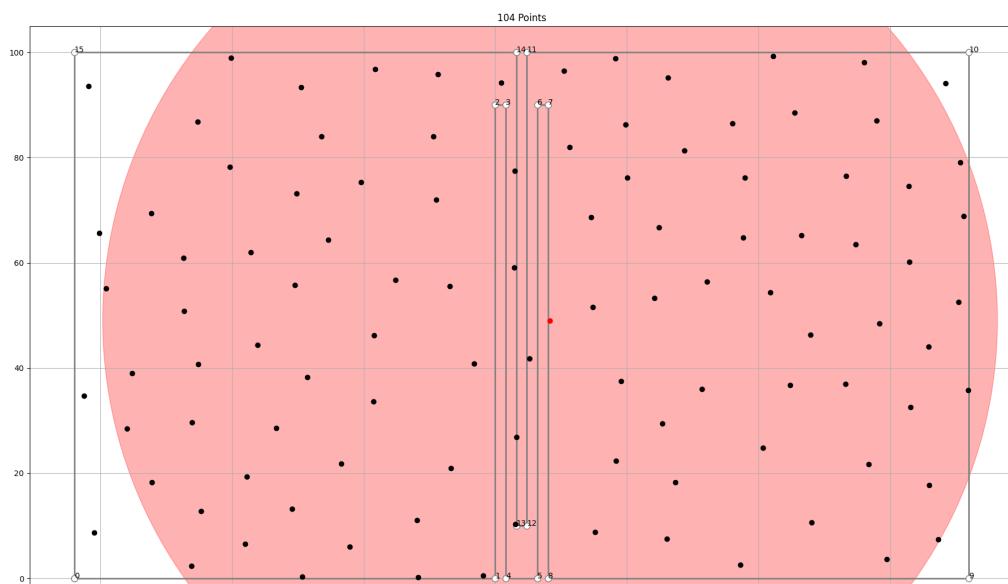


⁸An dieser Stelle möchte ich nicht offiziell über die Tatsache beschweren, dass dies nicht einheitlich vorgenommen wurde.

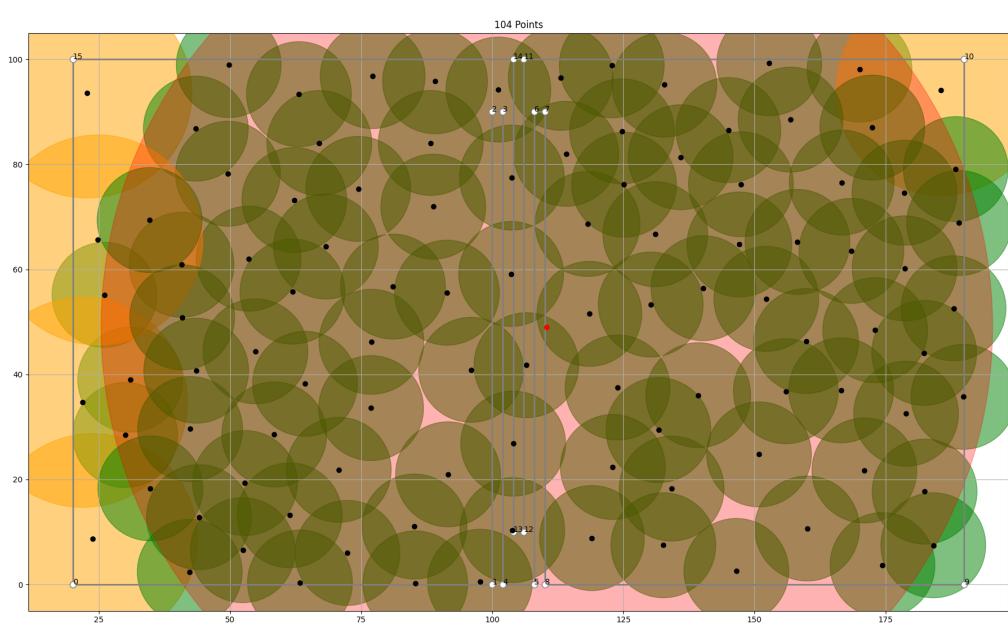
Loesung des Greedy-Verfahrens:

Verwendet man das Greedy-Verfahren mit zufaellig generierten Punkten fuer die Mengen G und T wie oben beschrieben, erhaelt man nach 5797 Millisekunden eine Loesung mit 103 Ortschaften:

Abbildung 34: 1. Loesung siedler5.txt



(a)



(b)

9.2. Erweiterung III

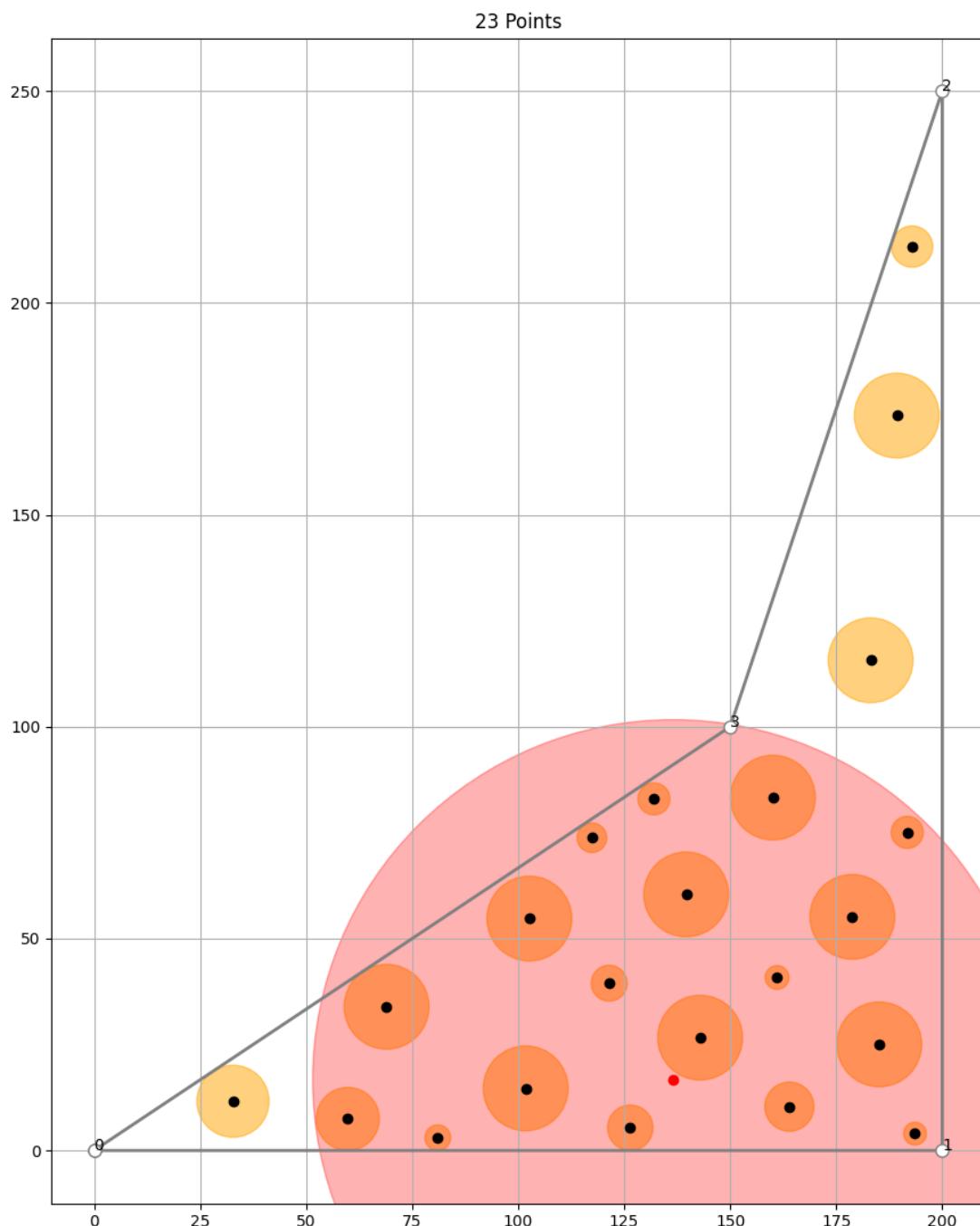
In diesem Kapitel sollen Beispiele zur Erweiterung III angegeben werden. Dazu wurden die Beispiele der BwInf-Website mit den originalen Distanzen verwendet. Weiter wurde der Mindestradius der Ortschaften als 2km und der Maximalradius der Ortschaften als 10km gewahelt. Zur Loesung wurden fuer T und G die selben Mengen an zufaelligen Punkten benutzt wie zuvor.

In den folgenden Beispielen wird jeweils eine Abbildung angegeben, die die Loesung des Beispiels darstellt. Die Liste der gewaelhten Punkten mit Radien ist in den Dateien zu finden.

1. Beispiel: siedler1.txt

Laufzeit: 1607ms
Flaeche: 4071.74 km²

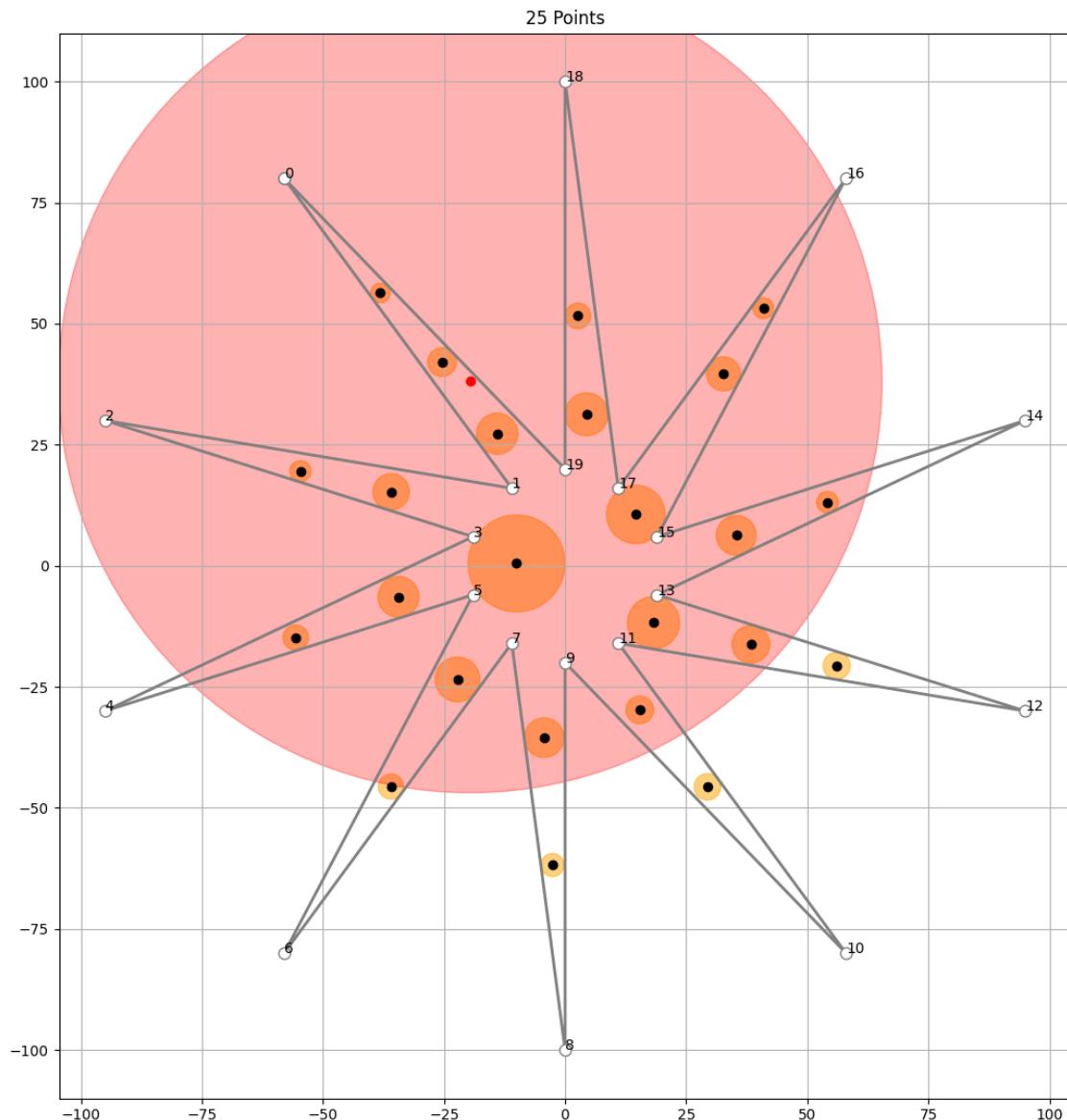
Abbildung 35: siedler1.txt



2. Beispiel: siedler2.txt

Laufzeit: 1085ms
Flaeche: 1236.24 km²

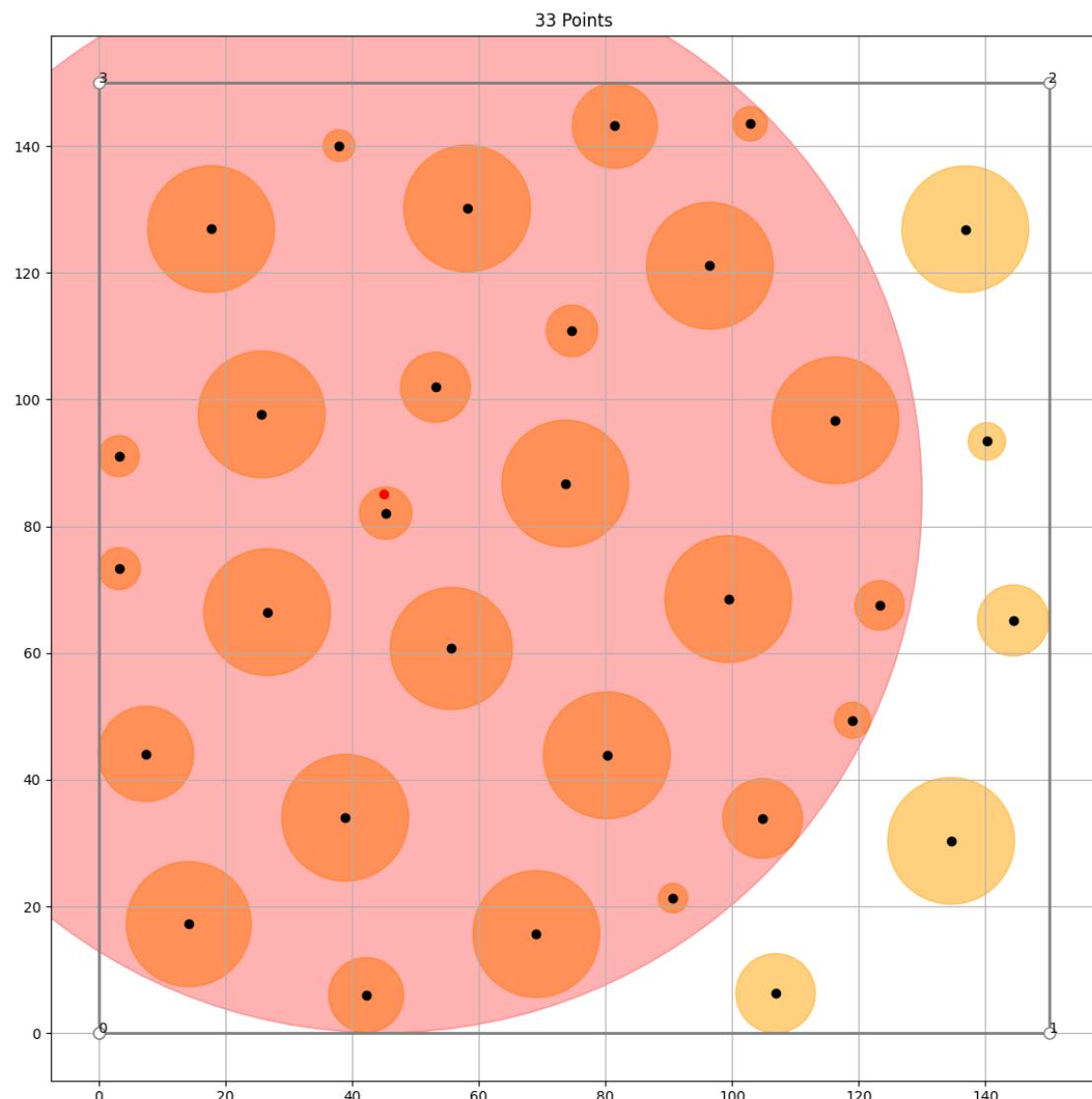
Abbildung 36: siedler2.txt



3. Beispiel: siedler3.txt

Laufzeit: 2073ms
Flaeche: 5882.85 km²

Abbildung 37: siedler3.txt

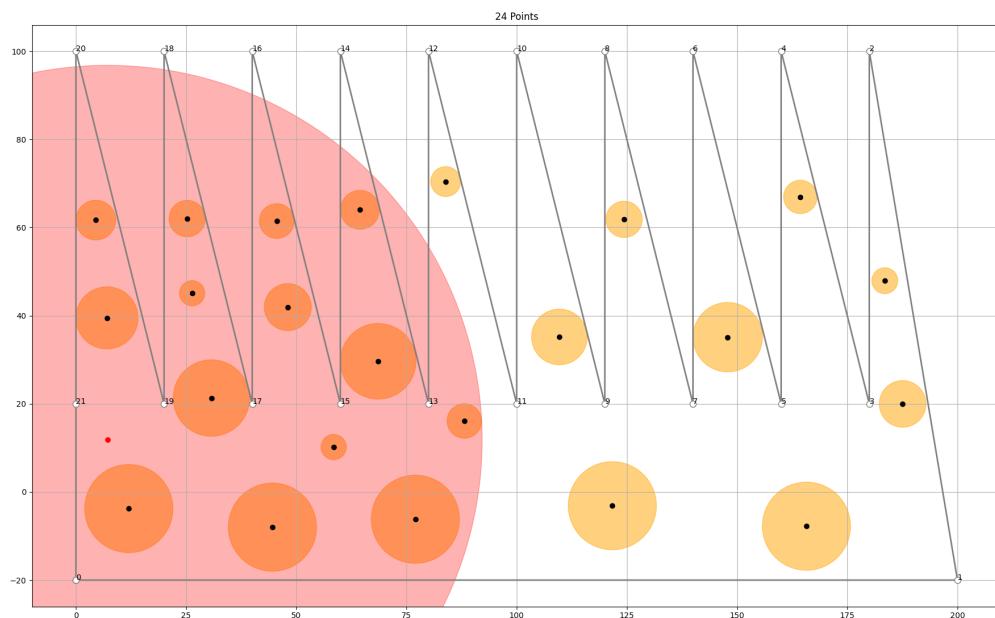


4. Beispiel: siedler4.txt

Laufzeit: 1845ms

Flaeche: 3180.97 km²

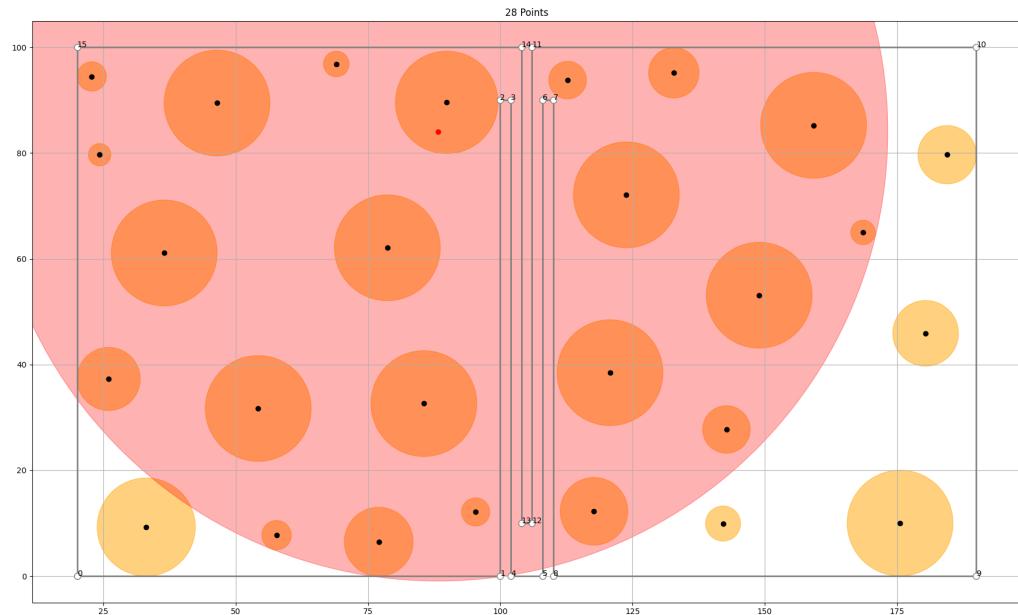
Abbildung 38: siedler4.txt

**5. Beispiel: siedler5.txt**

Laufzeit: 1781ms

Flaeche: 4624.63 km²

Abbildung 39: siedler5.txt



10. Quellcode

Nun folgt der Quellcode der wichtigsten Teile der Loesung in C++. Dabei wurde *cord* als pair an zwei long long's (wiederum definiert als *ll*) und *ctrip* als 3-tuple an *cord* definiert.

10.1. Erweiterung I

Triangulierung

Die Implementierung der Triangulierung eines Polygons wurde in C++ vorgenommen und benutzt, wie in Kapitel 5 erklärt, die naive Ear-Clipping-Methode. Der Hauptteil des Algorithmus besteht in den zwei Funktionen *triangulation* und *find ear*:

```

1 // Given a polygon with 3 or more vertices, return a ear
2     ctrip find_ear(vector<cord> &vertices) {
3         int n = vertices.size();
4         ctrip ear;
5
6         // Find ear (v1, v2, v3) <-> v2 convex && no other point of vertices lies in or on the triangle
7         for (int i = 0; i < n; i++) {
8             int a = i, b = (i + 1) % n, c = (i + 2) % n;
9             bool valid = true;
10
11             if (!is_convex(vertices[a], vertices[b], vertices[c])) {
12                 continue;
13             }
14
15             for (int j = 0; j < n; j++) {
16                 if (j == a || j == b || j == c) {
17                     continue;
18                 }
19
20                 // Check if X lies in the current triangle or on one of its sides
21                 cord X = vertices[j];
22                 if (point_in_or_on_triangle(vertices[a], vertices[b], vertices[c], X)) {
23                     valid = false;
24                     break;
25                 }
26             }
27
28             if (valid == true) {
29                 ear = {vertices[a], vertices[b], vertices[c]};
30                 break;
31             }
32         }
33
34         return ear;
35     }
36
37 // Triangulate a given polygon by naive ear-clipping
38 vector<ctrip> triangulation(vector<cord> vertices) {
39     int n = vertices.size();
40     vector<ctrip> triangles;
41
42     for (int i = 0; i < n - 2; i++) {
43         // Find ear
44         ctrip ear = find_ear(vertices);
45         triangles.push_back(ear);
46
47         // Remove the ear tip (middle vertex)
48         cord tip = get<1>(ear);
49         auto tip_it = find(vertices.begin(), vertices.end(), tip);
50         vertices.erase(tip_it);
51     }
52
53     return triangles;
54 }
```

Die Funktion *find ear* benutzt dabei einige geometrische Hilfsfunktionen:

```

// Operations on vectors: Cross Product
2  ll cross_product(cord &A, cord &B) {
    return (A.first * B.second) - (A.second * B.first);
4 }

6 // Check if X lies left (1) or right (-1) when going from P to Q;
// returning 0 if P, Q, X are co linear
8 int get_left_right(cord &P, cord &Q, cord &X) {
    cord v = subtract(Q, P);
10   cord u = subtract(X, P);
11   ll s = cross_product(u, v);

12   if (s > 0) {
14       return 1;
15   } else if (s < 0) {
16       return -1;
17   } else {
18       return 0;
19   }
20 }

22 // Check if there is a convex angle at B when going from A to B to C.
23 bool is_convex(cord &A, cord &B, cord &C) {
24     int s = get_left_right(A, C, B);

26     if (s == 1) {
27         return true;
28     } else {
29         return false;
30     }
31 }

32 // Check whether a given point X is on line AB
33 bool point_on_line(cord &A, cord &B, cord &X) {
34     ll a = A.first, b = A.second;
35     ll c = B.first, d = B.second;
36     ll x = X.first, y = X.second;

38     // 1. Case: c == a
39     if (c == a) {
40         ll mi = min(b, d);
41         ll ma = max(b, d);
42         return x == a && mi <= y && ma >= y;
43     }

46     // 2. Case: b == d
47     if (b == d) {
48         ll mi = min(a, c);
49         ll ma = max(a, c);
50         return y == b && mi <= x && ma >= x;
51     }

52     // 3. Case c != a and b != d
53     long double t = ((long double) (x-a)) / (((long double) (c-a)));
54     return (d-b) * (x-a) == (y-b) * (c-a) && t >= 0 && t <= 1;
55 }

58 // Return wheater X lies in or on the triangle ABC
59 bool point_in_or_on_triangle(cord &A, cord &B, cord &C, cord &X) {
60     int s0 = get_left_right(A, B, X);
61     int s1 = get_left_right(B, C, X);
62     int s2 = get_left_right(C, A, X);

64     if (s0 == s1 && s1 == s2) {
65         return true;
66     }

68     if (point_on_line(A, B, C) || point_on_line(B, C, X) || point_on_line(C, A, X)) {
69         return true;
70     }

72     return false;
73 }
74 }
```

Pseudozufällige Punkte im Polygon

Die folgende Funktion erzeugt eine bestimmte Anzahl an Punkten in einem Dreieck:

```
2 // Given a triangle and some amount of points, return a vector with
3 // that amount of points inside of the triangle
4 vector<cord> create_points(ctrip &triangle, int amount) {
5     vector<cord> points;
6
7     cord A = get<0>(triangle);
8     cord B = get<1>(triangle);
9     cord C = get<2>(triangle);
10
11    a1 = A.first, a2 = A.second;
12    b1 = B.first, b2 = B.second;
13    c1 = C.first, c2 = C.second;
14
15    for (int i = 0; i < amount; i++) {
16        // Create pseudo random point contained in the triangle
17        MY_MIN = 0.005 * RAND_MAX;
18        MY_MAX = 0.995 * RAND_MAX;
19
20        rand1 = (MY_MIN + rand()) % MY_MAX;
21        rand2 = (MY_MIN + rand()) % MY_MAX;
22
23        long double alpha = (long double) rand1 / (long double) RAND_MAX;
24        long double beta = (long double) rand2 / (long double) RAND_MAX;
25
26        if (alpha + beta > 1) {
27            alpha = 1 - alpha;
28            beta = 1 - beta;
29        }
30
31        x = a1 + alpha*(b1 - a1) + beta*(c1 - a1);
32        y = a2 + alpha*(b2 - a2) + beta*(c2 - a2);
33
34        points.push_back({x, y});
35    }
36
37    return points;
38}
```

Greedy-Verfahren

Der folgende Quellcode entspricht der Implementierung des Greedy-Verfahrens zur Loesung der Erweiterung I, wobei die Menge M mit Hilfe der erwähnten Optimierung effizient erneuert wird.

```
1   vector<cord> generate_greedy(vector<cord> &P, vector<cord> &T, cord Z) {
2     cord Y = T.back();
3     vector<cord> result = {Y};
4
5     // Create Set M
6     vector<cord> M;
7
8     for (cord C : T) {
9       if (
10         ((dist(Z, Y) <= R || dist(Z, C) <= R) && dist(C, Y) >= A) ||
11         (dist(Z, Y) > R && dist(Z, C) > R && dist(C, Y) >= B)
12       ) {
13         M.push_back(C);
14       }
15     }
16
17     while (M.size() > 0) {
18       cord X = M.back();
19       result.push_back(X);
20
21       // Update M
22       vector<cord> M_next;
23
24       for (cord C : M) {
25         if (
26           ((dist(Z, X) <= R || dist(Z, C) <= R) && dist(C, X) >= A) ||
27           (dist(Z, X) > R && dist(Z, C) > R && dist(C, X) >= B)
28         ) {
29           M_next.push_back(C);
30         }
31       }
32
33       M = M_next;
34     }
35
36     return result;
37   }
38
39   pair<cord, vector<cord>> solve(vector<cord> &P, vector<cord> &G, vector<cord> &T) {
40     vector<cord> result;
41     cord best;
42
43     for (cord Z : G) {
44       vector<cord> current = generate_greedy(P, T, Z);
45
46       if (current.size() > result.size()) {
47         result = current;
48         best = Z;
49       }
50     }
51
52     return {best, result};
53   }
54 }
```

ILP-Verfahren

Nun folgt die Implementierung des ILP-Verfahrens in Python:

```

1 def ilp(Z, points, message=0, time=60):
2     # --- Init ILP Solver ---
3     # 1) Create Variables
4     variables = []
5
6     for p in points:
7         name = "Point:" + str(p[0]) + ":" + str(p[1])
8         x = LpVariable(name, lowBound=0, cat="Binary")
9         variables.append(x)
10
11    # 2) Define Problem and objective
12    PROBLEM = LpProblem("Problem-1", LpMaximize)
13    objective = lpSum(variables)
14    PROBLEM += objective
15
16    # 3) Add constraints
17    n = len(points)
18
19    for i in range(0, n):
20        for j in range(i+1, n):
21            if (((points[i][0] - Z[0])**2 + (points[i][1] - Z[1])**2)**2 <= R**2 or
22                ((points[j][0] - Z[0])**2 + (points[j][1] - Z[1])**2)**2 <= R**2) and
23                ((points[i][0] - points[j][0])**2 + (points[i][1] - points[j][1])**2)**2 < A**2:
24                    PROBLEM += ((variables[i] + variables[j]) <= 1)
25            elif (((points[i][0] - Z[0])**2 + (points[i][1] - Z[1])**2)**2 > R**2 and
26                  ((points[j][0] - Z[0])**2 + (points[j][1] - Z[1])**2)**2 > R**2) and
27                  ((points[i][0] - points[j][0])**2 + (points[i][1] - points[j][1])**2)**2 < B**2:
28                    PROBLEM += ((variables[i] + variables[j]) <= 1)
29
30    solver = pulp.PULP_CBC_CMD(msg=message, timeLimit=time)
31    PROBLEM.setSolver(solver)
32    PROBLEM.solve()
33
34    result = []
35    for i in range(n):
36        if variables[i].value() == 1:
37            result.append(points[i])
38
39    return result
40
41 def solve(G, T):
42     best_set = []
43     best_Z = None
44
45     for Z in G:
46         current_set = ilp(Z, G)
47
48         if len(current_set) > len(best_set):
49             best_set = current_set
50             best_Z = Z
51
52     return (best_Z, best_set)

```

10.2. Erweiterung III

In diesem Kapitel wird der Quellcode zum Loesungsverfahren fuer Erweiterung III aus Kapitel 7 aufgefuehrt.

Greedy-Verfahren

Als erstes der Hauptteil der Loesung bestehend aus den Funktionen *solve* und *generate greedy*:

```

1   pair<cord, vector<pair<cord, ll>>> solve(vector<cord> &P, vector<cord> &G, vector<cord> &T) {
2     // Precompute values of R_Y: Max. radius of points Y to be completely contained in polygon P
3     vector<ll> radius0(T.size(), 0);
4
5     for (int i = 0; i < T.size(); i++) {
6       cord X = T[i];
7       radius0[i] = max_radius_polygon(P, X);
8     }
9
10    vector<pair<cord, ll>> result;
11    cord best;
12    ll best_value = 0;
13
14    for (cord Z : G) {
15      // Precompute values of Rprime_Y: Max. radius
16      // for point Y to be completely contained inside of Circle (Z, R)
17      vector<ll> radius1(T.size(), 0);
18
19      for (int i = 0; i < T.size(); i++) {
20        cord X = T[i];
21        if (X.first == Z.first && X.second == Z.second) {
22          radius1[i] = R;
23        } else {
24          radius1[i] = max_radius_circle(X, Z, R);
25        }
26      }
27
28      vector<pair<cord, ll>> current = generate_greedy(radius0, radius1, P, T, Z);
29      ll current_value = 0;
30      for (pair<cord, ll> p : current) {
31        current_value += p.second*p.second;
32      }
33
34      if (current_value > best_value) {
35        result = current;
36        best = Z;
37      }
38    }
39
40    return {best, result};
41  }

```

```
1  vector<pair<cord, ll>> generate_greedy( vector<ll> &radius0, vector<ll> &radius1, vector<cord> &P,
2    vector<cord> &T, cord Z) {
3      // 1. Part - Generate points in circle (Z, R)
4      vector<pair<cord, ll>> result;
5      vector<pair<int, ll>> M0; // Contains circles, the coord is saved as index in T
6
7      // Create M0
8      for (int i = 0; i < T.size(); i++) {
9        if (dist(T[i], Z) <= R) {
10          ll r = min(radius0[i], min(radius1[i], R1));
11
12          if (r >= R0) {
13            M0.push_back({i, r});
14          }
15        }
16      }
17
18      while (M0.size() > 0) {
19        // Get circle with biggest possible radius in M0
20        pair<cord, ll> K;
21        ll radius = -1;
22
23        for (pair<int, ll> circle : M0) {
24          ll current = circle.second;
25
26          if (radius == -1 || current > radius) {
27            K = {T[circle.first], current};
28            radius = current;
29          }
30        }
31
32        result.push_back(K);
33
34        // Update M0
35        vector<pair<int, ll>> M0_next;
36
37        for (pair<int, ll> circle : M0) {
38          // Check distance to circle K
39          int i = circle.first;
40          ll r = min(circle.second, (ll) dist(T[i], K.first) - A - K.second);
41          r = min(r, min(radius0[i], min(radius1[i], R1)));
42
43          if (r >= R0) {
44            M0_next.push_back({circle.first, r});
45          }
46        }
47
48        M0 = M0_next;
49      }
50
51      // 2. Part
52      vector<pair<int, ll>> M1;
53
54      // Create M1
55      for (int i = 0; i < T.size(); i++) {
56        if (dist(T[i], Z) > R) {
57          ll r = -1;
58
59          for (pair<cord, ll> p : result) {
60            ll current;
61
62            if (dist(p.first, Z) <= R) {
63              current = dist(p.first, T[i]) - A - p.second;
64            } else {
65              current = dist(p.first, T[i]) - B - p.second;
66            }
67
68            if (r == -1 || current < r) {
69              r = current;
70            }
71          }
72
73          r = min(r, min(radius0[i], R1));
74
75          if (r >= R0) {
76            M1.push_back({i, r});
77          }
78        }
79      }
80
81      P = M1;
82    }
83
84    return result;
85  }
```

```
77         }
78     }
79 }

81     while (M1.size() > 0) {
82         // Get circle with biggest possible radius in M1
83         pair<cord, ll> K;
84         ll radius = -1;
85
86         for (pair<int, ll> circle : M1) {
87             ll current = circle.second;
88
89             if (radius == -1 || current > radius) {
90                 K = {T[circle.first], current};
91                 radius = current;
92             }
93         }
94
95         result.push_back(K);
96
97         // Update M1
98         vector<pair<int, ll>> M1_next;
99
100        for (pair<int, ll> circle : M1) {
101            // Check distance to circle K
102            int i = circle.first;
103            ll r = min(circle.second, (ll) dist(T[i], K.first) - B - K.second);
104            r = min(r, min(radius0[i], R1));
105
106            if (r >= R0) {
107                M1_next.push_back({circle.first, r});
108            }
109        }
110
111        M1 = M1_next;
112    }
113
114    return result;
115 }
```

Weiter wurden die Berechnungen fuer den groeszt moeglichen Radius eines Punktes X , sodass der entstehende Kreis vollstaendig in einem Polygon, bzw. in einem Kreis liegt, wie folgt implementiert:

```
1 // Compute the maximal integer radius r for some point X, such that the circle (X, r)
2 // is inside of some given Polygon P
3 ll max_radius_polygon(vector<cord> &P, cord &X) {
4     ll radius = -1;
5
6     long double x = X.first;
7     long double y = X.second;
8
9     // For each edge of P: Calculate the smallest distance from X to some point on the edge
10    for (int i = 0; i < P.size(); i++) {
11        cord Y = P[i];
12        cord Z = P[(i+1) % P.size()];
13
14        long double a = Y.first;
15        long double b = Y.second;
16        long double c = Z.first;
17        long double d = Z.second;
18
19        long double t = (x*(c-a) + y*(d-b) - a*(c-a) - b*(d-b)) / ((c-a)*(c-a) + (d-b)*(d-b));
20        cord Q;
21
22        if (t <= 0) {
23            Q = Y;
24        } else if (t >= 1) {
25            Q = Z;
26        } else {
27            cord v = subtract(Z, Y);
28            cord u = multiply(v, t);
29            Q = add(Y, u);
30        }
31
32        ll cur_radius = dist(X, Q);
33
34        if (radius == -1 || cur_radius < radius) {
35            radius = cur_radius;
36        }
37    }
38
39    return radius;
40}
41
42 // Compute the maximal integer radius r for some point X, such that the circle (X, r)
43 // is inside of some given Circle (Z, R)
44 ll max_radius_circle(cord &X, cord &Z, ll R) {
45     long double x = X.first;
46     long double y = X.second;
47     long double a = Z.first;
48     long double b = Z.second;
49
50     long double t = R / ((x-a)*(x-a) + (y-b)*(y-b));
51
52     cord v = subtract(X, Z);
53     cord u = multiply(v, t);
54     cord Q = add(Z, u);
55
56     return dist(X, Q);
57}
```

11. Literatur

- [1] O'Rourke, Joseph; Suri, Subhash; Tóth, Csaba D. (2017), «Handbook of Discrete and Computational Geometry», Chapman and Hall/CRC - 3rd Edition, Chapter 29 - Triangulations and Mesh Generation
- [2] O'Rourke, Joseph; Suri, Subhash; Tóth, Csaba D. (2017), «Handbook of Discrete and Computational Geometry», Chapman and Hall/CRC - 3rd Edition, Chapter 30 - Polygons
- [3] Courant, Richard; Robbins, Herbert (1992), «Was ist Mathematik?», Springer-Verlag, 4. Ausgabe, Kap. 5, S. 202-204, doi: 10.1007/978-3-642-88688-1
- [4] Eberly, David (2002), «Triangulation by Ear Clipping», Geometric Tools, <https://www.geometrictools.com/Documentation/TriangulationByEarClipping.pdf>
- [5] Meisters, G. H. (1975), «Polygons Have Ears», The American Mathematical Monthly, Vol. 82 (6): p. 648-651, doi: 10.1080/00029890.1975.11993898