Praktikumsbericht

Implementierung von Algorithmen für das Minimum Convex Partition Problem

David Fischer

Abgabedatum:	30. März 2020
Betreuer:	Prof. Dr. Alexander Wolff
	Dr. Thomas van Dijk

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

Abstract

Im Rahmen des CG:SHOP2020-Wettbewerbs werden in diesem Bericht bekannte approxmative und exakte Algorithmen für das Minimum Convex Partition Problem vorgestellt, sowie eigene, neue Lösungsansätze, darunter eine Heuristik und ein Integer Linear Program (ILP), für das Minimum Convex Partition Problem auf kollinearen Punktemengen entwickelt. Des weiteren wird über die Implementierung ausgewählter Ansätze berichtet; diese werden mit Hilfe von Testdaten bezüglich Lösungsgüte und Laufzeit analysiert.

1 Einleitung

Dieses Praktikum beschäftigt sich mit der Implementierung von Approximationsalgorithmen für das *Minimum Convex Partition Problem* (MCP) im Rahmen des Wettbewerbs „CG Challenge 2020“ [CG2], Teil des „36th International Symposium on Computational Geometry “ [SoC]. Das Ziel dieses Wettbewerbs besteht darin, gegebene Instanzen des MCP möglichst gut zu lösen, also für jede dieser Instanz eine gültige Lösung zurückzugeben, die möglichst nah an der optimalen Lösung ist.

Denn obwohl Algorithmen existieren, die das Problem für alle Instanzen exakt lösen können, ist deren Laufzeit so groß, dass die Lösung für große Instanzen mit heutiger Rechenleistung nicht in realistischer Zeit möglich ist.

Das Problem ist aber nicht nur theoretischer Natur, und dessen Lösung auch nicht nur für die CG Challenge 2020 relevant. Wenn zum Beispiel die Punkte des Eingabesets Knoten in einem Netzwerk darstellen, kann Routing in diesem Netzwerk etwa dadurch erfolgen, dass simple Routing-Algorithmen auf eine konvexe Partition der Eingabepunktsets angewandt werden. Eine minimale konvexe Partition minimiert dann auch die vom Netzwerk benötigten Verbindungen, um die Routing-Algorithmen darauf anwenden zu können; daher erscheint es sinnvoll, ein Netzwerk gemäß einer minimalen, konvexen Partition der Knoten zu erstellen [BBC+02].

Diese Arbeit stützt sich in großen Teilen auf die Arbeit von Knauer und Spillner [KS06]. In ihrer Arbeit stellen die Autoren mehrere Approximationsalgorithmen für das Minimum Convex Partition Problem für Punktemengen, die keine kollinearen Punkte enthalten, vor. Einer dieser Algorithmen, ein 3-Approximationsalgorithmus, soll für kollineare Punkte angepasst werden, und dessen Ergebnisse durch Vorverarbeiten der Punktemenge bezüglich Kollinearitäten weiter verbessert werden. Dieser neue Algorithmus bietet leider keinen beweisbaren Approximationsfaktor, aber er liefert eine Heuristik, deren Güte anhand von Beispielen erkläutert werden soll.

Dabei ist diese Arbeit wie folgt aufgebaut. In Kapitel 2 sollen bereits bekannte Algorithmen für das MCP, wie zum Beispiel die bereits erwähnte Arbeit von Knauer und Spillner, zusammengefasst vorgestellt werden, um dem Leser eine Übersicht über den aktuellen Stand der Forschung zu verschaffen. In Kapitel 3 werden Lösungsansatze für das MCP auf kollinearen Punktemengen vorgestellt; dies enthält den im vorherigen Absatz erwähnten Algorithmus und bildet den Hauptteil der Arbeit. In Kapitel 4 wird über die Implementierung von ausgewählten Ergebnissen aus Kapitel 3 berichtet, die im Rahmen der CG Challenge 2020 erfolgt ist. In Kapitel 5 wird schließlich ein Fazit über die gesamte Arbeit gezogen.

Zunächst sollen aber zum Verständnis der Arbeit relevante Definitionen erläutert werden.

Sei P eine Punktmenge der Größe n in der zweidimensionalen Ebene. Wir nennen e eine Kante von P, wenn e eine gerade Strecke zwischen zwei Punkten p_1, p_2 aus P bildet. Die konvexe Hülle von P bezeichnen wir als $\mathrm{CH}(P)$.

Eine *konvexe Partition* von P ist eine Kantenmenge E, sodass die Kanten aus E sich nicht gegenseitig schneiden, und E die konvexe Hülle $\mathrm{CH}(P)$ in eine Menge $R(E)$ von leeren Facetten partitioniert; E enthält die Kanten von $CH(P)$ also nicht. Dabei ist eine

Facette genau dann leer, wenn sie keinen Punkt aus P in ihrem Inneren enthält, und die begrenzenden Kanten der Facette keinen Punkt in ihrem relativ Inneren enthalten.

Hierbei sei angemerkt, dass diese Definition, im Gegensatz zur Definition von Knauer und Spillner [KS06], kollineare Punkte erlaubt. Dies ergibt sich aus dem Grund, dass bei der CG Challenge 2020 [CG2] kollineare Punkte in den zu lösenden Instanzen ebenfalls erlaubt sind.

Mit diesen Definitionen kann nun das *Minimum Convex Partition Problem* (MCP) aufgestellt werden: Beim Minimum Convex Partition Problem wird eine Kantenmenge E von P gesucht, sodass E die konvexe Hülle $\mathrm{CH}(P)$ in eine *minimale* Anzahl von Regionen $R(E)$ partitioniert.

Für eine einfachere Notation sollen an dieser Stelle noch einige Begriffe definiert werden. Sei P eine Punktemenge mit konvexer Hülle $\mathrm{CH}(P)$. Wir bezeichnen mit k die Anzahl an inneren Punkten von P, also die Anzahl an Punkten, die nicht auf der konvexen Hülle, sondern im Inneren dieser liegen. Damit liegen auf $\mathrm{CH}(P)$ genau $n - k$ viele Punkte.

Sei $\mathrm{CH}_{\mathrm{in}}(P)$ die konvexe Hülle der inneren Punkte von P. Sei v ein Knoten auf $\mathrm{CH}_{\mathrm{in}}(P)$ und seien u, w seine Nachbarn auf $\mathrm{CH}_{\mathrm{in}}$. Sei H_w die Halbebene mit begrenzender Gerade durch u, v, die w nicht enthält, und H_u die Halbebene mit begrenzender Gerade durch v, w, die u nicht enthält. Wir nennen einen solchen Punkt v Knoten vom Typ (a) genau dann, wenn der Schnitt von H_u mit H_w mindestens einen Punkt aus P, der auf $\mathrm{CH}(P)$ liegt, enthält. Ist dies nicht der Fall, so nennen wir v Knoten vom Typ (b). Sei schließlich a die Anzahl der Knoten vom Typ (a), b die Anzahl der Knoten vom Typ (b) auf der inneren konvexen Hülle von P. Diese Definitionen sind für das Verständnis der Arbeit von Knauer und Spillner [KS06] wichtig.

2 Bekannte Ergebnisse

Die aktuell besten Approximationsalgorithmen für das Problem liefern die beiden Autoren Knauer und Spillner [KS06]. Die Ergebnisse dieser Arbeit sollen hier in zusammengefasster Form vorgestellt werden, da sie die Basis für die weitere Arbeit bilden. Hier ist noch einmal wichtig anzumerken, dass ihre Ergebnisse für die untere Schranke des Problems und die Approximationsfaktoren ihrer Algorithmen nur für *nicht-kollineare* Punktemengen P halten; bei Punktemengen, in denen Kollinearitäten erlaubt sind, kann der Approximationsfaktor ihrer Algorithmen nicht mehr nach oben hin abgeschätzt werden. Im folgenden betrachten wir also erst einmal nur solche, nicht-kollineare Punktemengen. Wir nennen das entsprechende Problem *non-collinear Minimum Convex Partition Problem* oder auch $ncMCP$.

2.1 Eine Untere Schranke

Knauer und Spillner können für alle konvexen Partition E von beliebigen, nicht-kollinearen Punktemengen P die Menge an konvexen Regionen $R(E)$ bei einer Partitionierung wie folgt von unten beschränken:

$$\frac{k}{2} + b + \frac{a}{2} + 1 \leq |R(E)| \tag{1}$$

Diese Abschätzung begründen sie, indem sie der Summe der inzidenten Kanten über alle Punkte berechnen. Dabei implizieren die Punkte auf $\mathrm{CH}(P)$ insgesamt mindestens $2(n-k)+a+2b$ inzidente Kanten, und die inneren Punkte von P insgesamt mindestens $3k$ inzidente Kanten; diese Implikation ergibt sich aus der Eigenschafft, dass keine der k Punkte kollinear zueinander sind. Durch Berechnung der Kantenanzahl hieraus und anschließende Abschätzung von $|R(E)|$ mit Hilfe der eulerschen Relation $1 = |P| - |E| + |R(E)|$ ergibt sich die vorliegende Schranke.

2.2 Ein Faktor-3-Approximationsalgorithmus

Knauer und Spillner geben einen einfachen 3-Approximationsalgorithmus an, bei dem sie induktiv zeigen, dass für jede Punktemenge P eine konvexe Partition E existiert, sodass:

$$|R(E)| \leq \frac{3}{2}k + \frac{3}{2} \tag{2}$$

Zusammen mit der unteren Schranke aus Abschnitt 2.1 ergibt diese Abschätzung eine 3-Approximation für $ncMCP$. Die grobe Idee der Induktion ist dabei folgende:
Induktionsanfang: Zeige, dass die Annahme für $k = 0$ und $k = 1$ erfüllt ist.
Induktionsschritt: Nehme nun an, dass $k \geq 2$ und die Induktionsvoraussetzung für alle Mengen mit echt weniger als k Punkten im Inneren gilt. Betrachte die konvexe Hülle der inneren Punkte $\mathrm{CH}_{\mathrm{in}}(P)$. Suche die beiden Schnittkanten der Geraden durch die unterste Kante e von $\mathrm{CH}_{\mathrm{in}}(P)$ mit $\mathrm{CH}(P)$. Verbinde die geometrisch über den beiden Schnittpunkten liegenden Endpunkte der Schnittkanten mit den ersten darunter liegenden Punkten von $\mathrm{CH}_{\mathrm{in}}(P)$. Dadurch ergibt sich eine konvexe Kette C von einem Punkt

aus $\mathrm{CH}(P)$ zu einem anderen Punkt aus $\mathrm{CH}(P)$, auf der l innere Punkte liegen, mit $l \geq 2$.

Der Bereich unter der konvexen Kette C kann konvex partitioniert werden, indem alle Punkte links des linken Endpunktes von e sowie der linke Endpunkt von e mit dem unteren Endpunkt der Schnittkante mit $\mathrm{CH}(P)$ verbunden werden, für die rechte Seite analog; dabei entstehen höchstens l konvexe Regionen.

Die konvexe Kette C bildet zusammen dem Teil der konvexen Hülle $\mathrm{CH}(P)$ über C ein konvexes Polygon Q mit $k-l < k$ inneren Punkten. Nach (IV) kann dieses mit höchstens $\frac{3}{2}(k-l)+\frac{3}{2}$ konvexen Regionen partitioniert werden. Damit gilt die Induktion.

Die Laufzeit von $O(n \log n)$ ergibt sich daraus, dass zuerst alle verschachtelten konvexen Hüllen von P mit einem Algorithmus von Chazelle [Cha85] berechnet werden, und dann die Rekursion in $O(n \log n)$ ausgeführt wird.

2.3 Ein Faktor-30/11-Approximationsalgorithmus

Knauer und Spillner können des Weiteren zeigen, dass ein $\frac{30}{11}$-Approximationsalgorithmus existiert, indem sie zeigen, dass für alle Punktemengen P eine konvexe Partition E existiert, sodass gilt:

$$|R(E)| \leq a + 2b + \frac{15}{11}k - \frac{24}{11} \tag{3}$$

Die Approximationsgüte ergibt sich durch Vergleich mit der unteren Schranke in Abschnitt 2.1.

Zuerst zeigen sie dabei, dass, hat man eine gute konvexe Partition für die inneren Punkte gefunden, man den Bereich zwischen $\mathrm{CH}(P)$ und $\mathrm{CH}_{\mathrm{in}}(P)$ mit maximal $a+2b$ konvexen Regionen partitionieren kann; nach Definition der Knotentypen (a) und (b).

Nun bleibt zu zeigen, dass die k inneren Knoten mit höchstens $\frac{15}{11}k - \frac{24}{11}$ konvexen Regionen partitioniert werden können. Die grobe Vorgehensweise dabei ist, eine obere Schranke der Form $\alpha n - \beta$ mit $\alpha \geq 1$, $\beta \geq 0$ zu zeigen, und dann α zu minimieren. Dies passiert induktiv: Den Induktionsanfang für $3 \leq n \leq 8$ liefert die obere Schranke von Neumann-Lara für die Anzahl an konvexen Regionen bei Partitionierung einer Menge mit n Punkten: $|R(E)| \leq \frac{10}{7}n - \frac{18}{7}$. [NRU04] Dadurch werden 6 Ungleichungen aufgestellt, die α und β erfüllen müssen.

Nun sei $n \geq 9$. Hier müssen 2 Fälle unterschieden werden:

1. Fall: Es liegen 4 oder mehr Punkte auf der konvexen Hülle von P. Dann kann die konvexe Hülle durch eine Diagonale in 2 konvexe Teilpolygone mit jeweils echt weniger als n Punkten unterteilt werden, welche nach Induktionsvorraussetzung lösbar sind. Für die gesamte Partition müssen diese beiden Unterpartitionen wieder zusammengesetzt werden, was eine weitere Ungleichung für das Gleichungssystem liefert, das α und β erfüllen müssen.

2. Fall: Es liegen genau 3 Punkte auf der konvexen Hülle von P. Da hier keine Diagonale gezogen werden kann, muss anders vorgegangen werden. Die Idee hier ist, sich die untersten ℓ Punkte für eine Konstante ℓ (im späteren Algorithmus gilt $\ell = 6$) zu nehmen, und mit den Endpunkten v_1, v_2 der untersten Kante der konvexen Hülle von

P die konvexe Hülle H dieses Sets zu bilden, was ein konvexes Polygon Q liefert. Des weiteren werden alle j Punkte von H mit dem letzten Punkt v_3 von CH(P) verbunden; damit erhält man Dreiecke $D_1, \ldots, D_j$. Nun wird Q abhängig von der Anzahl der Punkte in seinem Inneren direkt gelöst, die j Dreiecke werden per Induktionsvoraussetzung partitioniert. Diese Schritte liefern weitere Ungleichungen für α und β.

Setzt man alle so erhaltenen Ungleichungen für α und β als Nebenbedingungen für ein lineares Programm, das α minimiert, so erhält man als optimale Lösung für die beiden Werte: $\alpha = \frac{15}{11}$ und $\beta = \frac{24}{11}$. Damit gilt die Induktion für diese Zahlenwerte und die Behauptung ist bewiesen.

Da der Beweis konstruktiv ist, kann aus diesem direkt ein Algorithmus abgeleitet werden. Dieser besitzt eine Laufzeit von $O(n^2)$.

2.4 Exakte Algorithmen für MCP

Es gibt auch einige exakte Algorithmen für das nicht kollineare MCP, die eine optimale Lösung berechnen. Diese sind jedoch entweder exponentiell in der Größe der Eingabe n, oder Fixed-Parameter-Tractable für die Anzahl k der Punkte des Punktesets der Eingabe P, und exponentiell in dieser. Da für große n davon ausgegangen werden kann, dass k gegen n geht, bleibt auch hier die Laufzeit sehr schlecht. Des weiteren sind in der CG:SHOP2020-Competition Instanzen mit bis zu $n = 1.000.000$ Punkten gegeben, die gelöst werden müssen; dies bedeutet, dass kein Rechner der Welt aktuell solche Instanzen mit einem der exakten Algorithmen in der Zeit bis zum Ende der Competition optimal lösen kann.

Da jedoch auch kleine Instanzen Teil der Competition sind (z.B. $n = 10, 100, ...$), könnte ein exakter Algorithmus zur Lösung dieser Instanzen verwendet werden, um eine optimale Lösung und somit einen optimalen Score für diese Instanzen zu bekommen, was den Gesamtscore verbessert. Daher soll hier ein exakter FPT-Algorithmus von Spillner [Spi05] kurz beschrieben werden.

Die Idee des Algorithmus ist, schrittweise Kanten aus der optimalen Lösung zu raten, und diese in die zurückzugebende Kantenmenge E einzufügen. Der Algorithmus geht dabei so vor, dass er beginnt, Kanten an Punkten der konvexen Hülle von P einzufügen. Durch dieses Einfügen entsteht immer ein Subproblem, nämlich das Problem, wie wir die nächste Kante einfügen, um zu einer gültigen und optimalen Lösung zu gelangen. Spillner zeigt, dass es 7 Typen von Subproblemen gibt, die bei einem solchen Einfügen entstehen können, und gibt für jeden Typ von Subproblem an, wie dieses gelöst werden kann. Des weiteren beweist er, dass bei jeder Lösung eines Subproblems nur neue Subprobleme nach Art eines der 7 Typen entstehen können.

Der Algorithmus fügt also immer neue Kanten ein, löst die entstehenden Subprobleme und fügt weiter Kanten ein, bis das Punkteset P durch diese Kantenmenge konvex partitioniert wird. Die Laufzeit von $O(2^k k^3 n^3 + n \log n)$ ergibt sich durch die Anzahl an möglichen Subproblemen, die behandelt werden müssen, sowie die Zeit zu deren jeweiligen Lösung. Da im Algorithmus dynamische Programmierung verwendet wird, ergibt sich ein Speicherverbrauch von $O(2^k k^2 n^2 + n^3)$.

3 Lösungsansätze für MCP auf kollinearen Punktemengen

In diesem Kapitel sollen Lösungsansätze für das MCP speziell auf kollinearen Punktemengen erarbeitet und vorgestellt werden. Diese dienen als theoretische Grundlage für den Programmcode, mit dessen Hilfe die bei der Competition [CG2] gegebenen Instanzen gelöst werden sollen.

3.1 Beschreibung einer Heuristik

Die Idee des für die Competition verwendeten Algorithmus ist es, das gegebene Punkteset P, das kollineare Punkte enthalten darf, durch Entfernen kollinearer Punkte so vorzuarbeiten, dass ein neues Punkteset P' entsteht, in dem deutlich weniger Kollinearitäten existieren als in der ursprünglichen Punktemenge. Auf diesem Punkteset P' soll eine modfizierte Version des 3-Approximations-Algorithmus von Knauer und Spillner angewendet werden, die P' konvex partitioniert, und, falls sich Punkte aus $P \setminus P'$ im Inneren von so berechneten, konvexen Polygonen befinden, den Algorithmus rekursiv auf diesen Polygonen mit den darin befindlichen inneren Punkten aus $P \setminus P'$ aufruft.

Für die Punktemenge P gilt die in Abschnitt 2.1 gefundene untere Schranke für MCP aufgrund möglicher Kollinearitäten nicht mehr. Dadurch ist die Güte der Lösung des hier beschriebenen Algorithmus für allgemeine, kollineare P nicht durch einen festgelegten Approximationsfaktor abschätzbar. Jedoch wird im späteren Teil dieses Kapitels anhand von Beispielen erklärt, warum sich der Algorithmus intuitiv zumindest deutlich besser als die simple Anwendung des Knauer und Spillner Algorithmus auf P verhält, und dass er deshalb eine gute Heuristik liefert, mit der ein angemessener Erfolg bei der Competition erzielt werden könnte. Zuerst folgt aber die Beschreibung des Algorithmus selbst:

Wir sagen Punkte $p_1, \cdots, p_k \in P$, mit $k \geq 3$, liegen auf einer *kollinearen Kette* C, wenn die Punkte $p_1, \cdots, p_k$ kollinear zueinander sind, d.h. sie alle auf einer Geraden liegen. Seien $C_1, \cdots, C_l$ alle kollinearen Ketten, die in P vorkommen; diese können leicht berechnet werden. Ein Punkt p kann dabei auf beliebig vielen, verschiedenen kollinearen Ketten liegen. Zwei solche Ketten C_i, C_j, mit $C_i = (p_{i_1}, \cdots, p_{i_k})$ und $C_j = (p_{j_1}, \cdots, p_{j_m})$ schneiden sich genau dann, wenn die Kanten $(p_{i_1} p_{i_k})$ und $(p_{j_1} p_{j_m})$ sich schneiden.

Wir wollen später im Algorithmus möglichst alle benachbarten Punkte auf einer kollinearen Kette durch Kanten miteinander verbinden, und diese Kanten zur Lösung hinzufügen. Da in unserer Lösung für das MCP nur sich nicht schneidende Kanten vorkommen dürfen, wollen wir die Schnitte zwischen kollinearen Ketten bei der Vorverarbeitung auflösen, damit sich später keine dadurch eingefügten Kanten schneiden. Wir betrachten dafür jeden Schnittpunkt x von beliebig vielen kollinearen Ketten in diesem Punkt, und verändern die daran beteiligten Ketten so, dass sich keine kollinearen Ketten mehr in x schneiden. So gehen wir dann für jedes solches x vor, und haben am Ende eine schnittfreie Menge $\mathcal{C}$ an kollinearen Ketten aus P. Seien $C_1, \cdots, C_k$ die kollinearen Ketten, die sich in x schneiden. Für einen solchen Schnittpunkt x müssen wir folgende zwei Fälle betrachten:

1. Fall: $x \in P$: Diese Art von Schnitt muss nicht aufgelöst werden, da sich später Kanten, die zwischen benachbarten Punkten von kollinearen Ketten verlaufen, sich in x

nicht schneiden, sondern in x zueinander inzident sind; dies ist gültig für unsere Lösung. Ein Beispiel für diesen Fall ist in Abbildung 1 gegeben.

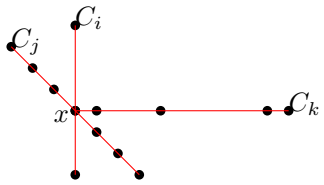


Abb. 1: Beispiel, wie das Preprocessing kollineare Ketten mit Schnittpunkt $x \in P$ aussieht

2. Fall: $x \notin P$. In diesem Fall würden sich die Kanten zwischen den zwei zu x nächsten Punkten jeder kollinearen Kette in x schneiden. Wir lösen den Schnittpunkt x auf, indem wir die kollineare Kette durch x, die die meisten Punkte aus P enthält, behalten, und jede andere kollineare Kette, die am Schnitt in x beteiligt ist, in zwei kollineare Ketten zerlegen: Eine Kette, die alle Punkte links von x, und eine Kette, die alle Punkte rechts von x enthält. Hat eine dieser neuen Ketten weniger als drei (kollineare) Punkte, so löschen wir diese komplett aus der Menge $\mathcal{C}$ der kollinearen Ketten. Dieser Fall ist beispielhaft in Abbildung 2 grafisch dargestellt.

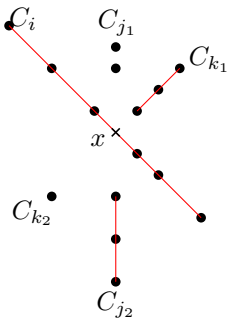


Abb. 2: Beispiel, wie das Preprocessing kollineare Ketten mit Schnittpunkt $x \notin P$ aussieht

Wir erhalten dadurch also eine Menge an schnittfreien, kollinearen Ketten $\mathcal{C}$. Wir erhalten außerdem ein Punkteset P' aus P, indem wir alle Punkte aus P entfernen, die im Inneren einer kollinearen Kette aus $\mathcal{C}$ liegen, d.h. die auf mindestens einer kollinearen Kette aus $\mathcal{C}$ liegen, aber kein Endpunkt einer kollinearen Kette aus $\mathcal{C}$ sind. Auf dieser neuen Punktemenge P' führen wir nun den Knauer und Spillner 3-Approximationsalgorithmus aus. Dabei priorisiert der Algorithmus für den Anfang der konvexen Kette Kanten von $\mathrm{CH}_{\mathrm{in}}$, deren Endpunkte auch die Endpunkte der längsten, auf der aktuellen inneren konvexe Hülle liegenden kollinearen Kette sind.

P' wird damit korrekt konvex partitioniert. Da alle Punkte aus $P \setminus P'$ im Inneren von kollinearen Ketten liegen, gibt es für jeden dieser Punkte $v \in P \setminus P'$ genau zwei Möglichkeiten:

1. Der Punkt v liegt auf einer Kante der konvexen Partitionierung von P'. Damit müssen wir diesen Punkt nicht weiter betrachten, da die berechnete konvexe Partitionierung

von P' somit auch gültig für $P' \cup v$ ist.

2. Der Punkt v liegt im inneren einer durch die konvexe Partitionierung von P' erstellten Facette. Für diesen Fall wenden wir für alle Facetten, in deren Inneren Punkte aus $P \setminus P'$ liegen, den hier beschriebenen Algorithmus rekursiv auf die Facette zusammen mit allen, darin enthaltenen Punkten an. Da jeder Punkt im Inneren von höchstens einer Facette ist, und die Facetten durch die konvexe Partitionierung des Algorithmus alle konvex sind, partitioniert die rekursive Anwendung des Algorithmus die gesamte Punktemenge P konvex.

Die Idee des Algorithmus ist, dass die hier beschriebene, erste Möglichkeit für Punkte aus $P \setminus P'$ in jedem rekursiven Schritt, durch Priorisieren von kollinearen Ketten in der Knauer und Spillner Unterroutine, mit deutlich größerer Wahrscheinlichkeit auftritt als die zweite Möglichkeit, und der Algorithmus somit nicht nur effizient bleibt, sondern auch eine konvexe Partitionierung mit möglichst wenig Facetten berechnet. Denn für Punkte, die die erste Eigenschaft erfüllen, entstehen bei deren Abarbeitung keine neuen Facetten.

Das Berechnen aller kollinearen Ketten für ein Punktset P mit n Punkten erfolgt mit Hilfe eines Sortieralgorithmus in $O(n^2 \log n)$ [Way05], die Auflösung der Schnittpunkte aller Ketten würde $O(n^4)$ Zeit kosten, da es bis zu $O(n^2)$ viele kollineare Ketten geben kann. Da Tests aber gezeigt haben, dass es sowohl für Laufzeit als auch für Güte der Lösung eher wenige, aber dafür große Ketten zu verwenden, werden in der Heuristik höchstens die n größten Ketten benutzt, wodurch sich eine Auflösung der Schnittpunkte in $O(n^2)$ durchführen lässt; zu bestimmten, welche Ketten am größten sind, lässt sich mit einem einfache Sortieralgorithmus in $O(n^2 \log^2 n)$ Zeit lösen. Die darauffolgende Ausführung des Knauer und Spillner 3-Approximationsalgorithmus kostet $O(n \log n)$ Zeit. Der Algorithmus wird rekursiv höchstens für jeden Punkt, also höchstens n mal, aufgerufen; zu Bestimmen, welche Punkte im Inneren welcher Facette liegen, ist in $O(n^2)$ möglich. Daraus ergibt sich eine Gesamtlaufzeit des oben beschriebenen Algorithmus von $O(n^3 \log^2 n)$.

Zur Intuition, warum der hier angegebene Algorithmus auf Punktemengen mit vielen kollinearen Punkten besser funktioniert als der simple 3-Approximationsalgorithmus von Knauer und Spillner, ist mit Abbildung 3 ein einfaches Beispiel gegeben. Hier ist ein mit verschiedenen Ansätzen konvex partitioniertes Punkteset P dargestellt. Die erste Abbildung zeigt die Partitionierung mit dem 3-Approximationsalgorithmus von Knauer und Spillner, die Zweite den in diesem Kapitel beschriebenen Algorithmus, und die dritte Abbildung die optimale konvexe Partitionierung von P. Dabei sind die schwarzen Linien die Liniensegmente der äusseren konvexen Hülle der Punktesets, die blauen Linien die vom jeweiligen Algorithmus eingefügten Liniensegmente, und die rot gestrichelten Linien in der zweiten Abbildung die von unserem Algorithmus vorberechneten, kollinearen Ketten.

Dabei wird deutlich, dass für dieses Beispiel mit k inneren Punkten der simple Knauer und Spillner Algorithmus eine konvexe Partitionierung berechnet, die P in $O(k)$ konvexe Regionen aufteilt, während sowohl unser Algorithmus, als auch das Optimum, eine konvexe Partitionierung in konstant viele konvexe Regionen erreichen.

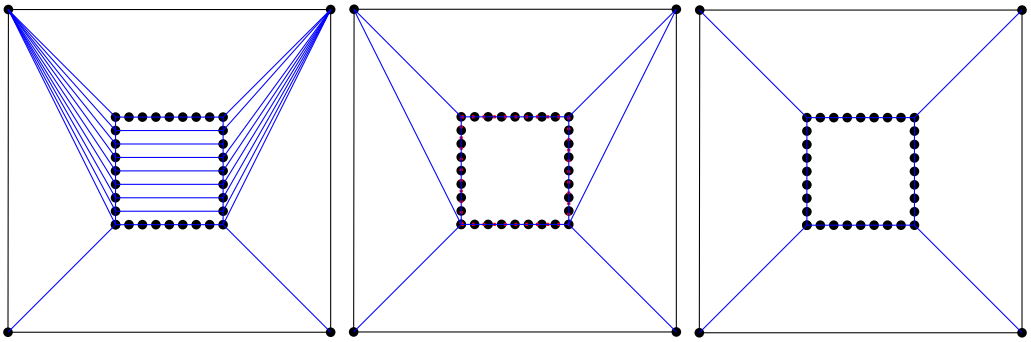


Abb. 3: Beispiel für konvexe Partitionierung eines Punktesets P mit $|P| = 4 + k$, wobei k die Anzahl innerer Punkte angibt und die inneren Punkte ein Quadrat bilden mit $k/4$ Punkten pro Seite, k beliebig.

3.2 Das Problem als ILP

In diesem Abschnitt soll das MCP für kollineare Punktemengen als *Integer Linear Program* (ILP) formuliert werden. Dieses ILP soll später bei der Lösung für Teilstrukturen der approximativen Lösung des Problems verwendet werden, da das ILP das Problem exakt lösen kann, dabei jedoch exponentielle Laufzeit in der Instanzgröße des Punktesets P braucht. Können jedoch Lösungsteile der approximativen Lösung erkannt werden, die besonders schlecht approximiert werden, aber nur einen kleinen Teil von P enthalten, können diese in realistischer Zeit mit Hilfe des ILP optimal gelöst werden, um so die Gesamtlösung deutlich zu verbessern.

Sei $V(\mathrm{CH}(P))$ die Menge der Punkte auf der konvexen Hülle des Punktesets P, und sei $Conc(v, a, b)$ die Menge aller Punkte aus P im konkaven Bereich des Winkels $\angle avb$ über den beiden, in v inzidenten Liniensegmenten va und vb, mit $v, a, b \in P$. Dann lässt sich MCP wie folgt als ILP formulieren:

$$\text{Minimiere} \quad \sum_{\forall a,b \in P} x_{ab}$$

$$\begin{aligned}
\text{so, dass} \quad & \sum_{\forall b \in P} x_{ab} && \geq 2, && \forall a \in P \\
& x_{ab} + x_{cd} && \leq 1, && \forall a, b, c, d \in P, \forall s \in \mathbb{R}^2 : s \in ab \wedge s \in cd \\
& \sum_{\forall c \in Conc(v,a,b)} x_{vc} && \geq x_{va} + x_{vb} - 1, && \forall v, a, b \in P \setminus V(\mathrm{CH}(P)), v \neq a \neq b \\
& x_{ab} && = 1, && \forall a, b \in V(\mathrm{CH}(P)) \\
& x_{ab} && = x_{ba}, && \forall a, b \in P \\
& x_{ab} && \in \{0, 1\}, && \forall a, b \in P
\end{aligned}$$

Der Nachteil dieses ILPs ist, dass die Zeile im ILP, die die $Conc(v, a, b)$-Funktion verwendet, kubisch viele Nebenbedinungen in der Anzahl der Punkte aus P generiert; diese Menge kann schnell sehr groß werden, und bei der Implementierung zu Problemen mit Platz im Arbeitsspeicher führen.

Ein speichersparenderer Ansatz für ein ILP wird von Barbosa et al. in [BdSdR19] formuliert. Wir können dieses ILP von Barbosa et al. [BdSdR19] mit einer kleiner Änderung an MCP für kollineare Punktemengen anpassen, und verwenden. Der Vorteil dieses ILPs ist, dass es nur zwei mal quadratisch viele Constraints in der Anzahl der Punkte benötigt, im Gegensatz zu unserem ILP, das kubisch viele solcher Constraints braucht. Barbosa et al. definieren dabei $CCW(ab)$ für $a, b \in P$ und die **gerichtete** Kante ab wie folgt:

$$CCW(ab) = \{c \in P, 0^\circ < \angle abc < 180^\circ\}$$

Damit fordern sie nun als Nebenbedinung, dass für alle Paare a, b, mit o.B.d.A a Punkt im Inneren der durch die konvexe Hülle implizierte Facette, mindestens eine Kante von a zu einem Punkt aus $CCW(ab)$ existieren muss. Dadurch, dass dies für alle Paare und beide Richtungen der Kante jedes Paares gelten muss, existieren also in der Lösung keine Winkel Instanzgrößer oder gleich 180°, da so alle in der Lösung vorkommenden Winkel in den Constraints abgedeckt werden. Da wir aber, im Gegensatz zu den Autoren von [BdSdR19], kollineare Punktemengen betrachten, können bei uns in einer korrekten Lösung auch Winkel mit Instanzgröße genau 180° auftreten. Verändern wir also die Definition von $CCW(ab)$ zu $CCW'(ab)$ mit:

$$CCW'(ab) = \{c \in P, 0^\circ < \angle abc \leq 180^\circ\}$$

und übernehmen den Rest des ILPs, berechnet das neue ILP eine korrekte, optimale Lösung für Punktesets P, die kollineare Punkte enthalten dürfen. Wir erstellen dabei für jedes Punktepaar $a, b \in P$ maximal zwei solcher Constraints, nämlich einen Constraint für jede Richtung der Kante (ab), falls beide Punkte im inneren der konvexen Hülle von P liegen. Damit haben wir also insgesamt nur $2 \cdot |P|^2$ solcher Constraints.

4 Implementierung

In diesem Kapitel wird über die Implementierung der im Rahmen der CG Challenge 2020 [CG2] verwendeten Algorithmen gesprochen, so wie deren Laufzeit und die Güte der von ihnen produzierten Lösungen analysiert.

Die Implementierung der Algorithmen erfolgte in Java mit Hilfe der IDE „Eclipse Neon“ ; Einlesen der Instanzen im JSON-Format, sowie Ausgabe der Lösung als Kanten ebenfalls im von den Veranstaltern der CG:SHOP2020 spezifizierten JSON-Format erfolgte mit Hilfe der „json-simple“ Bibliothek [jso].

4.1 Verwendete Algorithmen und Bibliotheken

Zur Lösung der von der CG Challenge 2020 gestellten Instanzen wurden zwei Algorithmen implementiert:

Der 3-Approximationsalgorithmus von Knauer und Spillner [KS06], der in dieser Arbeit in Kapitel 2.2 beschrieben wurde, und ebendieser Algorithmus in Verbindung mit Preprocessing von kollinearen Ketten, wie in Kapitel 3.1 beschrieben. Der erste Algorithmus wird im Folgenden „KnSSimple “, der zweite „KnSCollChains “ gennant.

Die Verwendung von mehreren verschiedenen Algorithmen ergibt sich dabei aus folgenden Gründen:

1. Da KnSSimple als Unterroutine von KnSCollChains kollinearen Ketten verwendet wird, musste KnSSimple sowieso implementiert werden.

2. Da KnSSimple um einiges effizienter ist als KnSCollChains ist, konnte KnSSimple zur Lösung von mehr Instanzen bzw. zur Lösung von Instanzen mit größerer Eingabemenge herangezogen werden.

Als Computational Geometry Bibliothek für die Implementierung wurde die sog. „JTS Topolgy Suite“ [jts] verwendet. Dabei wurden verschiedene Datenstrukturen, wie z.B. Implementierungen für "Punkt", "LinienSegment", und "Polygon", und Algorithmen, wie z.B. eine Implementierung des Graham-Scan-Algorithmus für die Berechnung von konvexen Hüllen, oder Schnittpunktberechnung von zwei Liniensegmenten, aus dieser Bibliothek in beiden Algorithmen eingesetzt.

Dabei wurde bei der Implementierung von KnSCollChains kollinearen Ketten der vorher implementierte KnS-Algorithmus als Unterroutine verwendet. Dabei ist zu beachten, dass die Implementierung von KnSSimple nicht, wie theoretisch möglich und im Paper von Knauer und Spillner [KS06] beschrieben, eine Laufzeit von $O(n \log n)$, sondern eine Laufzeit von $O(n^2 \log n)$ besitzt. Dies ergibt sich daraus, dass der im Paper von Chazelle [Cha85] als Unterroutine für KnSSimple beschriebene Algorithmus für verschachtelte konvexe Hüllen mit $O(n \log n)$ Laufzeit nicht mit vertretbarem Aufwand implementiert werden konnte; stattdessen wird in jedem Schleifendurchlauf von KnSSimple der von der JTS-Bibliothek bereitgestellte Graham-Scan in $O(n \log n)$ auf der Menge der inneren Punkte durchgeführt.

Dabei ist ebenfalls anzumerken, dass der in der JTS Topology Suite implementierte Graham-Scan-Algorithmus kollineare Punkte der Punktemenge auf der konvexen Hülle dieser nicht mit zurückgibt; deswegen werden in jedem Schritt der Ausführung von

KnSSimple für eine neue, innere konvexe Hülle alle Punkte, die auf der neuen inneren konvexen Hülle, und gleichzeitig aber aufgrund von Kollinearitäten auf der aktuellen äußeren konvexen Hülle liegen, gefunden und behandelt; danach musste die innere konvexe Hülle neu berechnet werden. Dies addiert einen Faktor von $O(n^3 \log n)$ zur Laufzeit der Implementierung beider Algorithmen, kommt aber praktisch nicht vor. Insgesamt ergibt sich damit eine worst-case-Gesamtlaufzeit von $O(n^3 \log n)$ für die Implementierung von KnSSimple, und eine worst-case-Gesamtlaufzeit von weiterhin $O(n^3 \log^2 n)$ von KnSCollChains. Obwohl die worst-case-Laufzeiten beider Algorithmen sehr ähnlich sind, ist, wie später auch anhand einer praktischen Analyse gezeigt werden kann, die tatsächliche Laufzeit von KnSSimple deutlich geringer als von KnSCollChains, da der oben erwähnte worst-case zumindest in den bei der Competition gegebenen Instanzen nie auftritt.

Die Implementierung und Verwendung des in Kapitel 3.2 erwähnten ILPs für die optimale Lösung von Teilstrukturen der von KnSSimple bzw. KnSCollChains kollinearen Ketten verwendeten Lösung war aus Zeitgründen leider nicht möglich.

4.2 Güte der Lösungen und Laufzeitanalyse

Für die Competition waren 247 verschiedene Instanzen gegeben, also 247 verschiedene Punktmengen, die es konvex zu partitionieren galt. Dabei waren vier verschiedene Typen von Instanzen gegeben, die wie folgt auf der Website der Competition [CG2] definiert sind:

- uniform: uniformly at random from a square
- edge: randomly generated according to the distribution of the rate of change (the „edges“) of an image
- illumination: randomly generated according to the distribution of brightness of an image (such as an illumination map)
- orthogonally collinear points: randomly generated on an integral grid to have a lot of collinear points (similar to PCBs and distorted blueprints)

Jede dieser Instanzen besitzt zwischen 10 und 100.000 Punkten, wobei zusätzlich eine besonders große Instanz mit 1.000.000 Punkten gegeben war. Die Instanzen wurden dann von den Veranstaltern in weitere vier Gruppen aufgeteilt und in folgenden Kategorien zum Download angeboten:

- uniform: Instanzen der Kategorie *uniform*
- images: Instanzen der Kategorie *edge* und *illumination*
- ortho_rect: ein Teil der Instanzen der Kategorie *orthogonally collinear points*
- rand_ortho: der zweite Teil der Instanzen der Kategorie *orthogonally collinear points*

Zu dieser Aufteilung gibt es keine offizielle Angabe auf der Website, doch sie ergibt sich anscheinend aus der Anzahl an kollinearen Punkten in den Punktesets (low to high); da KnSCollChains speziell auf solche Punktemengen spezialisiert ist, wird für die nachfolgende Analyse auch genau diese zweite Aufteilung benutzt.

Eingereichte Lösungen für jede Instanz werden dabei wie folgt mit einem Score zwischen 0 und 1 bewertet. Ein Score von 0 bedeutet, dass nur die Triangulierung als Lösung für die bewertete Instanz abgegeben wurde; eine Triangulierung ist die trivialste Form einer konvexen Partitionierung einer Punktemenge. Ein Score von 1 bedeutet, dass eine Lösung gefunden wurde, die nur eine Facette enthält; dies zu erreichen ist für keine Instanz möglich, aber dient als „upper bound“ für den Score. Das Ziel der Competition war es, die Summe der Scores über alle Instanzen zu maximieren, das Team mit dem höchsten Gesamtscore (max. 247) gewann die Competition [CG2].

Im Folgenden sollen die von beiden verwendeten Algorithmen (KnSSimple und KnSCollChains) produzierten Lösungen für ausgewählte Instanzen anhand ihres Scores und der Laufzeit der Algorithmen analysiert und miteinander verglichen werden. Dabei wurde ein Rechner mit folgenden Spezifikationen für die Ausführung der Programme verwendet:

- Prozessor: Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz, 3401 MHz
- Arbeitsspeicher: 12,0 GB RAM
- Betriebssystem: Microsoft Windows 10 Education

Für die Analyse wurden für KnSCollChains und KnSSimple Instanzen mit bis zu 10.000 Punkten verwendet. Für die Competition konnten mit KnSSimple Instanzen mit bis zu 50.000 Punkten gelöst werden, jedoch konnten diese nicht mit KnSCollChains gelöst werden und sind deswegen in der vergleichenden Analyse nicht enthalten. In den Abbildungen 4, 5, 6, und 7 sind jeweils die Ergebnisse von KnSSimple blau, die von KnSCollChains rot eingefärbt.

In Abbildung 4 und Abbildung 5 werden Score und Laufzeit von KnSSimple und KnSCollChains für jeweils die Kategorien „uniform“ und „images“ verglichen. Dabei gibt es, wie zu erwarten, kaum Unterschiede in der Güte der Lösungen zwischen den zwei Algorithmen, da die Punktesets jeweils keine oder nur sehr wenige kollineare Punkte enthalten. Diese geringen Unterschiede in der Güte ergeben sich aus der zufälligen Wahl des KnSSimple-Algorithmus, welches Liniensegment der inneren konvexen Hülle zur konvexen Partitionierung benutzt wird. Auch die Laufzeit der Algorithmen ist ähnlich, da KnSCollChains ohne Kollinearitäten nicht in eine Rekursion kommt; die etwas höhere Laufzeit von KnSCollChains ergibt sich aus der Vorberechnung der kollinearen Ketten in $O(n^2 \log n)$.

Anders sieht es aber bei Abbildung 6 und Abbildung 7 aus, wo die beiden Algorithmen jeweils für die Kategorien „ortho_rect“ und „rand_ortho“ verglichen werden. Da diese Punktesets viele kollineare Punkte enthalten, unterscheidet sich hier auch die Güte der Lösungen beider Algorithmen deutlich voneinander. Der KnSCollChains-Algorithmus produziert hier deutlich bessere Lösungen als KnSSimple. Gleichzeitig hat KnSCollChains hier aber auch eine deutlich höhere Laufzeit als KnSSimple, was sich sowohl durch die

große Rekursionstiefe erklären lässt, die besonders bei Punktesets mit vielen, sich gegenseitig schneidenden kollinearen Ketten vorkommt, als auch die große Anzahl der kollinearen Ketten (bis zu $O(n^2)$) insgesamt, die nach Größe geordnet werden müssen, um die größten n Ketten für den Algorithmus zu benutzen. Gerade bei der Kategorie „rand_ortho“ ist die Anzahl an kollinearen Ketten hoch. Dafür konnte in der Kategorie „rand_ortho“ aber auch der höchste Score unter allen Instanzen mit dem KnSCollChain-Algorithmus erreicht werden; dieser beträgt 0.39.

Allgemein ist aber der Score bei den Kategorien „ortho_rect“ und „rand_ortho“ für beide Algorithmen durchschnittlich höher als für die anderen beiden Kategorien „uniform“ und „images“. Dies lässt sich ebenfalls durch die in den ersten beiden genannten Kategorien erklären, da diese dadurch allgemein mit weniger Facetten in der Anzahl der Punkte konvex partitioniert werden können, als die letzten beiden.

Aus der Analyse ergibt sich schließlich auch, warum mit KnSSimple Instanzen bis zu 50.000 Punkten, mit KnSCollChains jedoch nur Instanzen bis zu 10.000 Punkten gelöst werden konnten: Für große Instanzen mit vielen kollinearen Punkten ergibt sich für KnSCollChains eine hohe Rekursionstiefe und darausfolgend eine hohe Laufzeit; die Laufzeit wurde für Instanzen mit mehr als 10.000 Punkten zu groß, um sie mit den gegebenen Ressourcen in realistischer Zeit lösen zu können.

Abb. 4: Analyse der Kategorie „uniform“

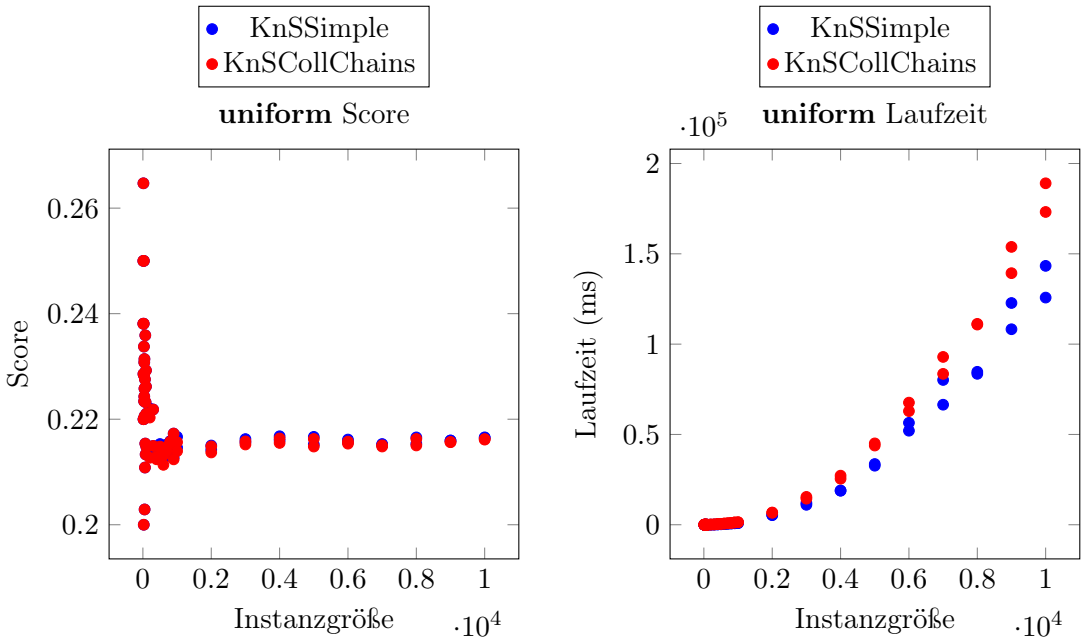


Abb. 5: Analyse der Kategorie „images“

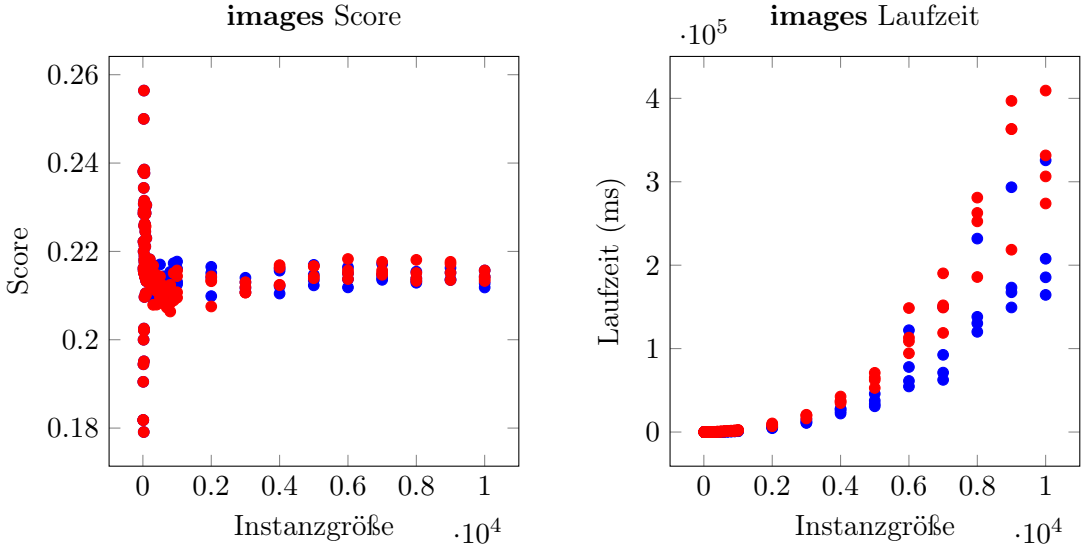


Abb. 6: Analyse der Kategorie „ortho_rect“

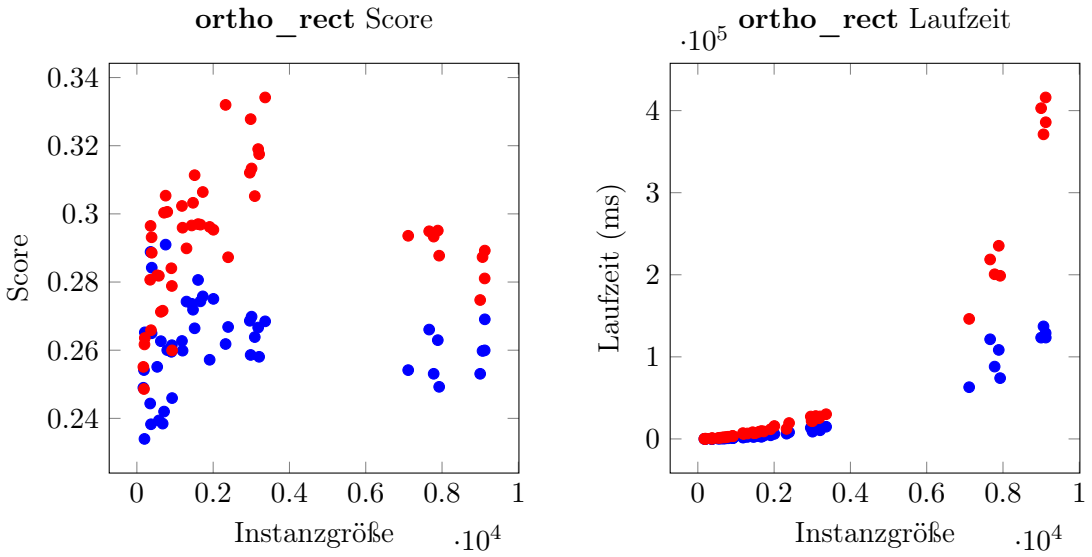
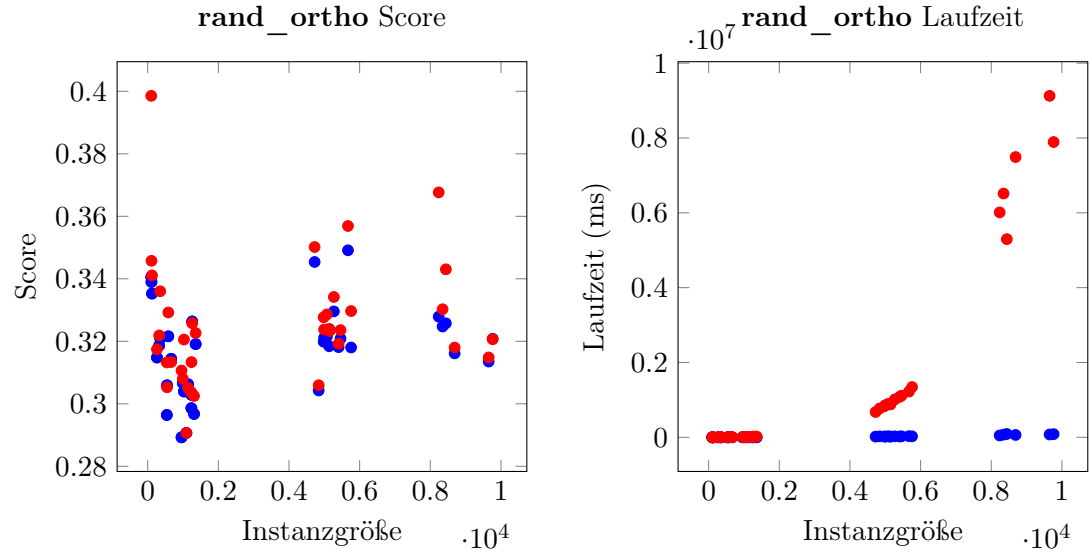


Abb. 7: Analyse der Kategorie „rand_ortho“



5 Fazit

Insgesamt bietet die beschriebene Heuristik eine zwar langsamere, aber immer noch effiziente Alternative zu bestehenden Algorithmen für das Minimum Convex Partition Problem, die besonders gut auf Punktemengen funktioniert, die viele kollineare Punkte enthalten. Gerade eine Implementierung dieser Heuristik, die, wie von Knauer und Spillner vorgeschlagen [KS06], den Chazelle-Algorithmus für verschachtelte konvexe Hüllen [Cha85] mit Berücksichtigung kollinearer Punkte auf den konvexen Hüllen verwendet, könnte noch viel an Effizienz dazugewinnen.

Des weiteren könnte die Idee des Preprocessings von kollinearen Ketten und die darauffolgende Anwendung eines bekannten Algorithmus für das MCP nicht nur für den 3-Approximationsalgorithmus von Knauer und Spillner, sondern auch für andere Algorithmen, z.B. die $\frac{30}{11}$-Approximation der beiden Autoren [KS06], zum Einsatz kommen.

Am Besten wäre es jedoch, nicht nur eine Heuristik, sondern einen Algorithmus mit beweisbarem Approxmationsfaktor zu finden, der diesen Faktor auch für Punktemengen mit kollinearen Punkten hält; dies war im Rahmen dieser Arbeit leider nicht möglich.

Literatur

[BBC+02] Prosenjit Bose, Andrej Brodnik, Svante Carlsson, Erik D. Demaine, Rudolf Fleischer, Alejandro López-Ortiz, Pat Morin und J. Ian Munro: Online Routing in Convex Subdivisions. *Int. J. Comput. Geometry Appl.*, 12(4):283–296, 2002.

[BdSdR19] Allan S. Barboza, Cid C. de Souza und Pedro J. de Rezende: Minimum Convex Partition of Point Sets. In: Pinar Heggernes (Herausgeber): *Proc. 11th Int. Conf. Algorithms & Complexity (CIAC)*, Band 11485 der Reihe *Lect. Notes Comp. Sci.*, Seiten 25–37. Springer Cham, 2019. https://doi.org/10.1007/978-3-030-17402-6_3.

[CG2] CG Challenge 2020. https://cgshop.ibr.cs.tu-bs.de/competition/cg-shop-2020/. Accessed: 2019-10-31.

[Cha85] B. Chazelle: On the convex layers of a planar set. *IEEE Transactions on Information Theory*, 31(4):509–517, 1985. https://doi.org/10.1109/TIT.1985.1057060.

[jso] JSON Simple. https://github.com/fangyidong/json-simple. Accessed: 2020-02-18.

[jts] JTS Topology Suite. https://locationtech.github.io/jts/. Accessed: 2020-02-18.

[KS06] Christian Knauer und Andreas Spillner: Approximation Algorithms for the Minimum Convex Partition Problem. In: Lars Arge und Rusins Freivalds (Herausgeber): *Proc. 10th Skandinavian Workshop Algorithm Theory (SWAT)*, Band 4059 der Reihe *Lect. Notes Comp. Sci.*, Seiten 232–241. Springer Berlin Heidelberg, 2006. https://doi.org/10.1007/11785293_23.

[NRU04] Victor Neumann-Lara, Eduardo Rivera-Campo und Jorge Urrutia: A Note on Convex Decompositions of a Set of Points in the Plane. *Graphs and Combinatorics*, 20(2):223–231, 2004. https://doi.org/10.1007/s00373-004-0555-2.

[SoC] 36th International Symposium on Computational Geometry. https://socg20.inf.ethz.ch/. Accessed: 2020-02-18.

[Spi05] Andreas Spillner: Optimal convex partitions of point sets with few inner points. In: *Proc. 17th Canad. Conf. Comput. Geom.*, Seiten 39–42, 2005. https://www.cccg.ca/proceedings/2005/43.pdf.

[Way05] Kevin Wayne: COS 226 Programming Assignment. https://www.cs.princeton.edu/courses/archive/spring03/cs226/assignments/lines.html, 2005. Accessed: 2020-02-20.