Institut für Theoretische Informatik
Lehrstuhl für Algorithmik I

Seminarband

Algorithmentechnik

Wintersemester 2013/2014

Lehrstuhl für Algorithmik I
Institut für Theoretische Informatik
Fakultät für Informatik
Karlsruher Institut für Technologie

KIT – Universität des Landes Baden-Württemberg und
nationales Forschungszentrum in der Helmholtz-Gemeinschaft

www.kit.edu

Vorwort

Im Rahmen des Seminars *Algorithmentechnik* werden ausgewählte aktuelle Forschungsergebnisse aus der Algorithmik behandelt. Die einzelnen Themen stammen insbesondere aus den Bereichen Graphenalgorithmen, geometrische Algorithmen, Algorithmen für Sensornetze und Algorithmen zum Graphenzeichnen. Damit vertieft das Seminar einzelne Themen aus dem Spektrum der übrigen am Lehrstuhl angebotenen Vertiefungsvorlesungen.

Das Seminar ist Bestandteil der Vertiefungsfächer Algorithmentechnik und Theoretische Grundlagen im Master-Studium. Ziel des Seminars ist es, dass die Teilnehmer lernen sich in wissenschaftliche Originalarbeiten einzuarbeiten und Literaturrecherche zu betreiben. Insbesondere müssen die Teilnehmer anhand eines Fachvortrages und einer Seminarausarbeitung die von Ihnen bearbeiteten Themen in ansprechender Form präsentieren. Damit werden Fähigkeiten erworben und erweitert, die auch zum Verfassen einer Masterarbeit in der Algorithmik erforderlich sind.

Das Seminar fand im Wintersemester 2013/2014 mit 8 Teilnehmern statt. Der vorliegende Seminarband umfasst die schriftlichen Ausarbeitungen der Teilnehmer.

Teilnehmer	**Betreuer**
Andreas Bauer	Fabian Fuchs
Daniel Feist	Andreas Gemsa
Vitali Henne	Martin Nöllenburg
Fabian Klute	Tamara Mchedlidze
Lea Köckert	Tamara Mchedlidze
Tobias Maier	Tanja Hartmann
Yassine Marakchi	Benjamin Niedermann
Lothar Weichert	Thomas Bläsius

1 Inhaltsverzeichnis

Verteilte Berechnung einer Knotenfärbung im SINR Model

Einleitung . . . 4
Stand der Forschung . . . 4
Modelle und Definitionen . . . 5
Algorithmus . . . 6
TDMA im SINR Modell . . . 18
Zusammenfassung . . . 19

Dynamic Point Labeling

Motivation and Introduction . . . 20
Complexity of Dynamic Point Labeling . . . 22
Predefinions for a Heuristic Algorithm for Dynamic Point Labeling . . . 27
Preliminary considerations on label speed, behindness, and freeness . . . 28
A Heuristic Algorithm for Dynamic Point Labeling . . . 28
Experimental Results . . . 33
Future Work . . . 34

Flussbasiertes Zählen von Triangulierungen

Einleitung . . . 36
Grundlagen . . . 38
Der Algorithmus . . . 43
Implementierungsdetails und Ausblick . . . 48
Fazit . . . 50

Superpatterns and Universal Point Sets for Graphs with Bounded Pathwidth

Introduction . . . 53
Definitions . . . 54
From Superpatterns to universal point sets . . . 55
Chessboard representation . . . 59
Subsequence majorization . . . 61
Strahler Number and Applications . . . 62
Graphs with bounded Pathwidth . . . 65
Summary . . . 68

Flips in Triangulations

Einleitung . . . 69
Ähnliche Themen . . . 70
Maximale Anzahl benötigter Flips für eine Transformation . . . 71

Untere Schranken 81
Zusammenfassung 83

Schnittapproximation durch Graphdekomposition in j-Bäume

Einleitung 85
Theoretische Grundlagen des Verfahrens 86
Existenz einer $(\alpha, \mathcal{J})$-Dekomposition 89
Weiterentwicklungen des Ergebnisses 99
Zusammenfassung 100

Approximation Schemes for Maximum Weight Independent Set of Rectangles

Introduction 102
Problem definition 103
Algorithm 103
Bounding the approximation ratio 105
Conclusion 114

Segmentation of Trajectories on Non-Monotone Criteria

Introduction 115
Trajectory Segmentation 115
The Start-Stop Diagram 116
Discrete Segmentation 117
Complexity of Trajectory Segmentation 119
Compute the Minimal Staircase 120
Computing the Start-Stop Diagram 127
Conclusion 128

2 Verteilte Berechnung einer Knotenfärbung im SINR Model

Andreas Bauer

Zusammenfassung

In verteilten, drahtlosen Sensornetzwerken müssen die einzelnen Sensoren untereinander kommunizieren können. Zu Beginn gibt es jedoch kein Protokoll, dass verhindert, dass sich Sensoren gegenseitig stören. Dies kann durch ein TDMA MAC-Protokoll basierend auf einer Knotenfärbung gelöst werden. Allen Knoten, die sich gegenseitig stören können, werden dabei unterschiedliche Farben zugewiesen. In jedem Zeitslot dürfen dann nur Knoten jeweils einer Farbe senden, um Störungen zu verhindern. In dieser Seminararbeit wird ein Algorithmus vorgestellt, der ohne funktionierendes MAC-Layer im strengen Signal-to-Interference-plus-Noise-Ratio-Modell ($SINR$-Modell) eine $O(\Delta)$ Knotenfärbung in $O(\Delta \log n)$ Zeitslots berechnet, wobei Δ der Maximalgrad des Graphen ist. Außerdem wird gezeigt, dass mithilfe dieser Knotenfärbung ein TDMA MAC-Protokoll aufgebaut werden kann, das Störungen unter den Sensoren vermeidet

2.1 Einleitung

In verteilten, drahtlosen Sensornetzwerken arbeiten eine Vielzahl von Sensoren zusammen. Dazu wird ein Mechanismus benötigt, der eine kollisionsfreie Kommunikation zwischen diesen ermöglicht. Dies ist vor allem dann eine schwierige Aufgabe, wenn die Sensoren zu Beginn keine Informationen über die Gesamttopologie haben.

Der Aufbau eines MAC-Layers (Medium Access Control), welcher eine solche kollisionsfreie Kommunikation ermöglicht, kann beispielsweise durch ein TDMA-Protokoll (Time Division Multiple Access) realisiert werden, bei denen Sensoren nur zu gewissen Zeitpunkten senden, aber jederzeit Nachrichten empfangen können. Es muss sichergestellt werden, dass nur solche Knoten zur selben Zeit senden, die sich nicht gegenseitig behindern. Dies kann man auch als ein Knotenfärbungsproblem ansehen, bei dem alle Knoten, die gleichzeitig senden, dieselbe Farbe haben, und in denen keine Knoten, die sich gegenseitig stören könnten, dieselbe Farbe haben. Wie man eine solche in einem realistischen Modell berechnen kann, wenn zu Beginn keine kollisionsfreie Kommunikation zwischen Knoten gewährleistet ist, ist Gegenstand dieser Seminararbeit.

In Abschnitt 2.2 werden zunächst bisherige Forschungsergebnisse zusammengefasst. In Abschnitt 2.3 wird das Protokoll Modell und das Signal-to-Interference-plus-Noise-Ratio-Modell ($SINR$ Modell) beschrieben, welches die Störungen zwischen den Sensoren modelliert. Anschließend wird in Abschnitt 2.4 der Algorithmus vorgestellt, der die Knotenfärbung berechnet. In Abschnitt 2.5 wird noch einmal verdeutlicht, wie man diese Knotenfärbung verwenden kann, um ein MAC-Layer aufzubauen, bevor in Abschnitt 2.6 die Seminararbeit zusammengefasst wird.

2.2 Stand der Forschung

Baenboim und Elkin beschreiben in [1] einen Algorithmus, der für verteilte Netzwerke in $O(\Delta) + 1/2 \log^* n$ eine $\Delta + 1$ Knotenfärbung berechnet. Dabei ist Δ der Maximalgrad des Graphen. Dies ist relativ nahe an der theoretischen unteren Schranke von $1/2 \log^* n$ [5]. Allerdings wird hierbei bereits eine kollisionsfreie Kommunikation zwischen den Knoten

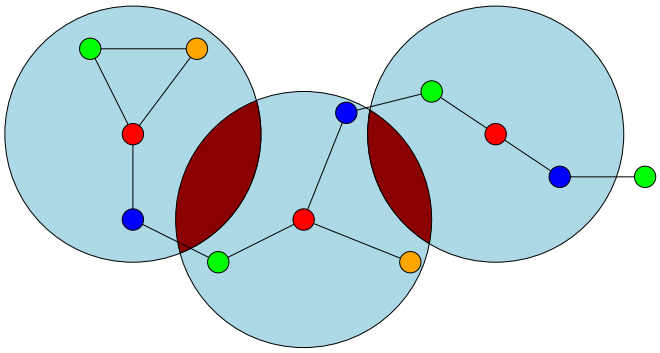


Abbildung 1 Wenn in einer Distanz-2 Knotenfärbung alle Knoten einer Farbe senden, kommt es im Protokoll Modell zu keiner Störung an einem der Knoten

angenommen. Moscibroda und Wattenhofer haben für den Fall, dass dies zu Beginn nicht gegeben ist, einen Algorithmus entwickelt, der in $O(\Delta \log n)$ Zeit eine Färbung mit $O(\Delta)$ Farben berechnet [6]. Dies wurde von Schneider und Wattenhofer auf eine Zeit von $O(\Delta + \log \Delta \log n)$ und $\Delta+1$ Farben verbessert [7]. In beiden Fällen wird allerdings nur das Protokoll Modell und nicht das $SINR$ Modell verwendet. Das $SINR$ Modell, und wie Nachrichten in diesem Modell erfolgreich übertragen werden können, wird von Goussevskaia u.a. sowie Halldórsson und Mitra in [3, 4] betrachtet. Derbel und Talibi [2] passen den Algorithmus von Moscribroda und Wattenhofer an, so dass dieser auch im $SINR$ Modell funktioniert. Dieser Algorithmus wird in dieser Seminararbeit betrachtet.

2.3 Modelle und Definitionen

Im Protokoll Modell für Sensornetzwerke wird davon ausgegangen, dass die Sendeleistung eines jeden Sensors in einem Radius R stark genug ist, um empfangen zu werden. Allerdings sind sie in diesem Radius auch stark genug, um andere Signale zu stören. Ein Signal wird am Ziel genau dann empfangen, wenn genau ein Sensor im Umkreis mit Radius R sendet. Es wird angenommen, dass alle Sensoren in einer zweidimensionalen Ebene liegen. Jeder Sensor wird als Knoten modelliert. Zwischen zwei Knoten ist eine Kante, wenn die dazugehörigen Knoten in Sendereichweite liegen. Diese Modellierung führt zum sogenannten *Unit Disk Graph*. In diesem Modell kann durch ein TDMA MAC Layer basierend auf einer Distanz-2 Knotenfärbung eine kollisionsfreie Kommunikation sichergestellt werden, da keine zwei Knoten u, v in der Nachbarschaft eines dritten Knotens w gleichzeitig senden und somit keine Störungen vorliegen (siehe Abb. 1).

Dieses Modell spiegelt die Realität jedoch nur unzureichend wieder. So sinkt die Sendeleistung mit Abstand zum Sender kontinuierlich, verschwindet aber nicht abrupt wie im Protokoll Modell. Dies führt dazu, dass selbst weit entfernte Sensoren noch einen Beitrag zur Störung liefern. So können viele Sensoren, die einzeln keine große Störung verursachen würden, zusammen trotzdem eine signifikante Störung erzeugen (siehe Abb. 2). Deswegen wird im folgenden das $SINR$-Modell verwendet. Ein Signal von Knoten v wird am Ziel u genau dann empfangen, wenn folgende Bedingung erfüllt ist

$$\frac{\frac{P}{\delta(u,v)^\alpha}}{N + \sum_{w \in V \setminus \{v\}} \frac{P}{\delta(w,u)^\alpha}} \geq \beta$$

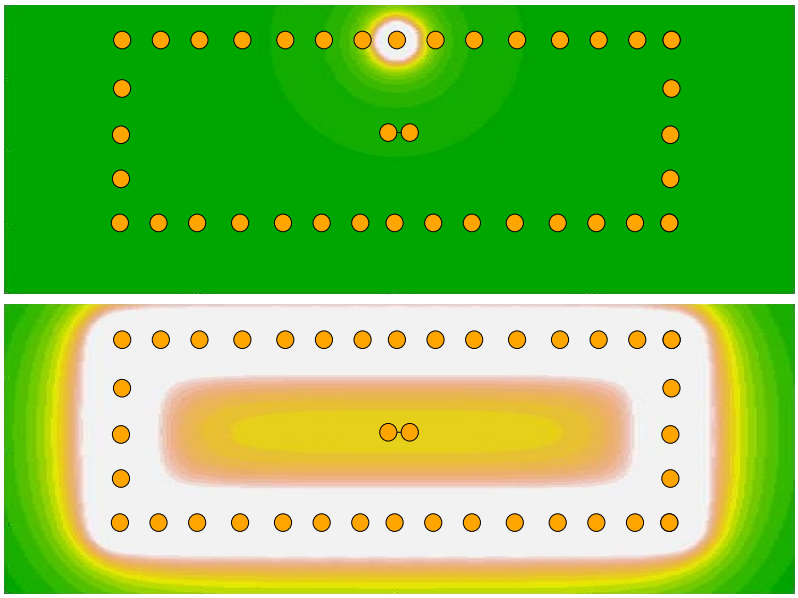


Abbildung 2 Ein einzelner Knoten auf dem Ring kann die Knoten in der Mitte nicht stören (oben). Bei vielen Knoten addiert sich jedoch genug Interferenz, um die Mitte zu stören (unten)

Hier bezeichnet P die Sendeleistung, $\delta(v, u)$ die Distanz zwischen zwei Knoten v und u. Die Konstante $\alpha > 2$ beschreibt, wie schnell die Sendeleistung mit der Distanz abfällt. N ist das konstante Hintergrundrauschen, welches selbst vorhanden ist, wenn gerade keine Signale gesendet werden. Das Signal-zu-Rausch-Verhältnis β gibt an, wie stark das Signal von v an u (Term im Zähler) im Verhältnis zum Hintergrundrauschen und allen anderen Signalen am Zielknoten u (Term im Nenner) mindestens sein muss, damit das Signal empfangen werden kann. In der Praxis ist $\beta > 1$ üblich.

Auch im $SINR$ Modell können die Sensoren durch einen Graphen modelliert werden. Wenn kein anderer Knoten sendet, kann ein Signal maximal in einem Radius $R_{max} = (\frac{P}{N\beta})^{1/\alpha}$ empfangen werden. Die *Transmission Range* ist eine etwas konservativere Abschätzung, definiert durch $R_T = (\frac{P}{2N\beta})^{1/\alpha}$. Ein Knoten u ist in der Nachbarschaft B_v vom Knoten v, wenn u nicht weiter als R_T von v entfernt ist. Zwischen allen Knoten, die benachbart sind, existiert eine Kante.

2.4 Algorithmus

Im folgenden wird der Algorithmus beschrieben, der mit hoher Wahrscheinlichkeit verteilt eine gültige Distanz-1 Knotenfärbung berechnet, auch wenn zu Beginn die kollisionsfreie Kommunikation nicht sichergestellt ist.

In Abschnitt 2.4.1 wird zunächst ein Überblick über den Algorithmus gegeben. Anschließend werden die einzelnen Teile ausführlicher behandelt. In Abschnit 2.4.2 werden notwendige Konstanten definiert. In Abschnitt 2.4.3 wird erläutert, wie Nachrichten im $SINR$ Modell übertragen werden können. Die Konkurrenz von Knoten um eine Farbe wird in Abschnitt 2.4.4 erklärt. In Abschnitt 2.4.5 und Abschnitt 2.4.6 wird die initiale bzw. endgültige Knotenfärbung behandelt. Zusammenfassend wird in Abschnitt 2.4.7 das Gesamtverhalten des Algorithmus analysiert.

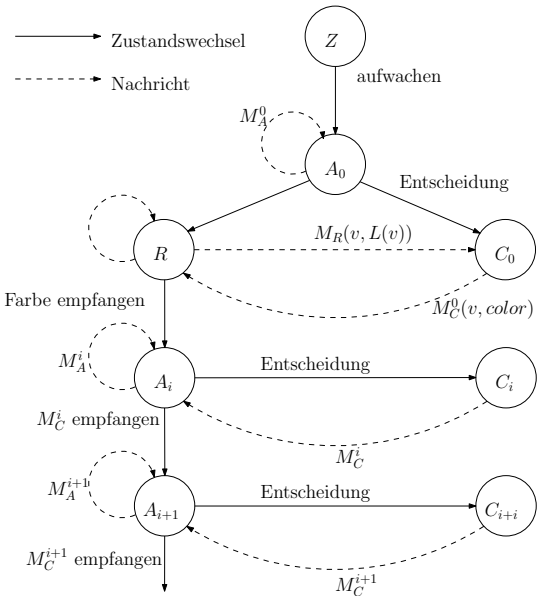


Abbildung 3 Nach dem Aufwachen konkurrieren Knoten um Farbe 0. Ist ein Knoten erfolgreich, geht er in Zustand C_0 über und wird ein lokaler Anführer. Sonst geht er in Zustand R und erwartet vom lokalen Anführer die initiale Färbung, woraufhin er um die finale Farbe i in Zuständen A_i konkurriert.

2.4.1 Überblick

Nachdem ein Knoten angeschalten wird, durchläuft er im Laufe des Algorithmus verschiedene Zustände (siehe Abb. 3). Dabei gibt es drei Kategorien von Zuständen:

- In Zuständen A_i bemüht sich der Knoten darum, die Farbe i als erster in der Nachbarschaft anzunehmen. Dieser Versuch kann gelingen oder misslingen
- In Zuständen C_i hat der Knoten endgültig die Farbe i angenommen. Knoten im Zustand C_0 werden darüber hinaus als *lokale Anführer* bezeichnet
- Der Zustand R wird von Knoten eingenommen, wenn der Versuch die Farbe 0 anzunehmen fehlgeschlagen ist. Einer der Knoten in ihrer Nachbarschaft, der die Farbe 0 eher angenommen hat, wird ihr lokaler Anführer. Diese Anführer weisen den Knoten eine initiale Farbe zu

Jeder Knoten startet nach der Aktivierung im Zustand A_0 und konkurriert mit Knoten in der Nachbarschaft um die Farbe 0 (siehe Abschnitt 2.4.4). Jeder erfolgreiche Knoten wird lokaler Anführer in seiner jeweiligen Nachbarschaft. Er bildet zusammen mit anderen Knoten, die wegen ihm die Farbe 0 nicht annehmen konnten, einen Cluster. Der Anführer teilt jedem Knoten im Cluster eine für den Cluster eindeutige Farbe zu (siehe Abschnitt 2.4.5). Es kann damit nur noch Konflikte zwischen Knoten aus verschiedenen Clustern geben. In Abschnitt 2.4.6 wird gezeigt, dass es pro Knoten nach der initialen Färbung nur eine begrenzte Anzahl k an Knoten in deren Nachbarschaft geben kann, die dieselbe Intraclusterfarbe haben. Mit diesen wird dann um die endgültige Farbe konkurriert, was nach spätestens k Versuchen für jeden Knoten erfolgreich ist.

2.4.2 Konstanten

Für den Algorithmus werden eine Reihe Konstanten benötigt, die hier definiert werden.

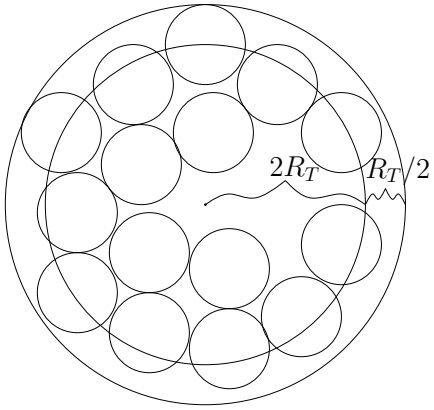


Abbildung 4 In einem Kreis von Radius R kann es nur endlich viele Knoten selber Farbe geben, da die Kreise mit Radius $R_T/2$ um die Knoten disjunkt sind. Diese Kreise sind alle in einem Kreis mit Radius $R + R_T/2$ enthalten.

Zuerst wird der Interferenzradius definiert. Wir werden später zeigen, dass unter gewissen Umständen die erwartete Interferenz außerhalb dieses Radius durch eine Konstante abgeschätzt werden kann.

$$R_I = 2R_T(96\rho\beta\tfrac{\alpha-1}{\alpha-2})^{1/(\alpha-2)}$$

Dabei ist ρ so zu wählen, dass gilt: $R_I \geq 2R_T$ (gültig für $\rho > 1$).

Weitere Konstanten ergeben sich aus der Tatsache, dass es in einem bestimmten Bereich nur eine begrenzte Anzahl an Knoten mit gleicher Farbe geben kann.

▶ **Lemma 1.** *Sei $\Phi(R)$ die Anzahl von Knoten selber Farbe in einem Kreis mit Radius R, dann ist $\Phi(R)$ beschränkt durch $\frac{\pi(R+R_T/2)^2}{\pi(R_T/2)^2}$*

Beweis. Knoten, die dieselbe Farbe haben, müssen bei einer gültigen Färbung mindestens R_T voneinander entfernt sein. Kreise um Knoten derselben Farbe mit Radius $R_T/2$ sind damit disjunkt. Alle diese Kreise von Knoten derselben Farbe im Umkreis R sind vollständig in einem Kreis mit Radius $R + R_T/2$ enthalten (Siehe Abb. 4). Damit kann man die Anzahl Knoten derselben Farbe im Umkreis mit Radius R abschätzen durch

$$\frac{\text{Fläche}(\text{Kreis}(R + R_T/2))}{\text{Fläche}(\text{Kreis}(R_T/2))} = \frac{\pi(R + R_T/2)^2}{\pi(R_T/2)^2} = \text{const.}$$

◀

Folgende Werte von $\Phi(R)$ werden im folgenden häufiger verwendet:

$$\Phi(2R_T) = k,\ \Phi(R_I + R_T) = k_{I+T},\ \Phi(R_I) = k_I,\ \Phi(R_T) = k_T$$

Folgende Konstanten werden außerdem verwendet:

Δ : Maximalgrad des Knoten

$$\lambda = \frac{1-1/\rho}{e^{k_I/k_{I+T}}} \cdot \left(1 - \frac{k_I}{k_{I+T}^2 \Delta}\right) \cdot \left(1 - \frac{1}{k_{I+T}}\right)^{k_I}$$

$$\lambda' = \frac{1-1/\rho}{e^{k_{I+T}}} \cdot \left(1 - \frac{1}{k_{I+T}\Delta}\right) \cdot \left(1 - \frac{1}{k_{I+T}}\right)^{k_{I+T}}$$

$$\sigma = \frac{2c}{\lambda'}, \gamma = \frac{ck_{I+T}}{\lambda}, c \geq 5$$

$$q_\ell = \frac{1}{k_{I+T}}, q_s = \frac{1}{k_{I+T}\Delta}$$

Dabei ist q_l die Sendewahrscheinlichkeit für lokale Anführer und q_s die für andere Knoten. Es gilt $\sigma > 2\gamma$. Des Weiteren seien die Konstanten ν, μ beliebig mit $\nu \geq 2\gamma k + \sigma + 1$ und $\mu \geq \gamma$

2.4.3 Nachrichten im SINR Modell

Während des gesamten Algorithmus ist eine kollisionsfreie Kommunikation zwischen den Knoten nicht sichergestellt. Im folgenden wird daher erläutert, wie trotzdem mit hoher Wahrscheinlichkeit erfolgreich Nachrichten zwischen den Knoten übertragen werden können. Dies wird dadurch erreicht, dass jeder Knoten v in jedem Zeitslot nur mit einer Wahrscheinlichkeit p_v eine Nachricht aussendet. In diesem Abschnitt wird zuerst gezeigt, dass bei geeigneter Wahl der Sendewahrscheinlichkeiten die erwartete Interferenz außerhalb des Interferenzradius um die Knoten konstant ist. Dies wird anschließend verwendet, um die Wahrscheinlichkeit eines erfolgreichen Sendevorgangs in einem Zeitslot, und zuletzt in $\gamma\zeta \log n$ aufeinanderfolgenden Zeitslots zu berechnen. Dabei ist $\zeta = 1$ falls u ein lokaler Anführer ist und $\zeta = \Delta$ sonst.

Als Sendewahrscheinlichkeiten wird für lokale Anführer q_l und für alle anderen Knoten q_s gewählt (siehe Abschnitt 2.4.2) . Damit gilt für jeden Knoten v:

$$\sum_{w \in B_v} p_w = \sum_{w \in B_v \cap C_0} q_l + \sum_{w \in B_v \setminus C_0} q_s \leq \sum_{w \in B_v \cap C_0} \frac{1}{k_{I+T}} + \sum_{w \in B_v \setminus C_0} \frac{1}{k_{I+T}\Delta} \leq 2$$

Die letzte Ungleichung gilt, da es maximal k_T Knoten mit gleicher Farbe in der Nachbarschaft von v geben kann (Lemma 1) und $k_T \leq k_{I+T}$, und es in der Nachbarschaft von v nur maximal Δ Knoten geben kann.

Die große Herausforderung des $SINR$ Modells ist, dass sämtliche Knoten einen Beitrag zum Rauschen liefern. Die Knoten agieren jedoch alle lokal. Um das Verhalten von Knoten lokal analysieren zu können, ist folgendes Lemma hilfreich, in dem die erwartete Interferenz $\Psi_u^{v \notin R_I}$ an einem Knoten u durch Knoten außheralb des Interferenzradius beschränkt wird

▶ **Lemma 2.** *Unter der Annahme, dass alle Knoten in Zustand C_0 ein* independent-set *I bilden (d.h. es gibt keine Kante zwischen zwei Knoten v, w, wenn beide Bestandteil von I sind), dann gilt für die erwartete Interferenz $\Psi_u^{v \notin R_I}$ von Knoten außerhalb des Interferenzradius um u: $\Psi_u^{v \notin R_I} \leq \frac{P}{2\rho\beta R_T^\alpha}$.*

Beweis. Zuerst berechnen wir die erwartete Interferenz auf einem Ring H_ℓ mit Durchmesser R_I, der $\ell * R_I$ vom Ziel entfernt ist, d.h. für alle Knoten v, für die gilt: $\ell * R_I \leq d(u, v) <$

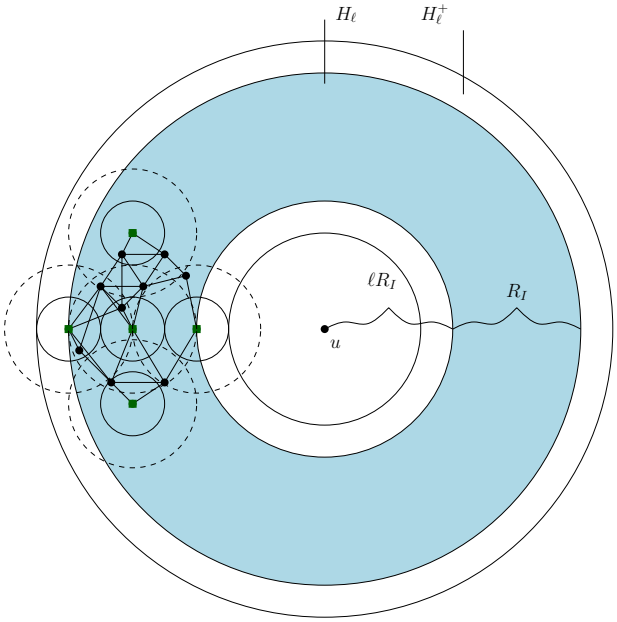


Abbildung 5 Betrachtet man in einem Ring H_ℓ der Breite R_I ein maximales independent-set, so sind Kreise mit Radius $R_T/2$ disjunkt. Außerdem sind alle Knoten in der Nachbarschaft eines Knotens aus dem independent-set (gestrichelte Kreise). Macht man den Ring an beiden Seiten um $R_T/2$ breiter (H_ℓ^+), so sind alle durchgezogenen Kreise in diesem erweiterten Ring enthalten

$((\ell+1) * R_I)$ (siehe Abb. 5). Anschließend berechnen wir die erwartete Interferenz über alle Ringe außerhalb des Interferenzradius.

Wir betrachten ein maximales independent-set I in H_ℓ. Zwischen zwei Knoten v, w besteht genau dann eine Kante, wenn sie gegenseitig in ihrer Nachbarschaft liegen, d.h. $d(w,v) \leq R_T$. Damit gilt, dass alle Kreise mit Radius $R_T/2$ um Knoten aus I disjunkt sind. Wir erweitern nun den Ring H_ℓ zu H_ℓ^+, so dass alle diese Kreise vollständig im Ring enthalten sind, d.h. $x \in H_\ell^+$ für $\ell R_I - R_T/2 \leq d(u,x) < \ell R_I + R_T/2$. Dann kann man die Anzahl der Knoten in I Abschätzen durch: $\frac{\text{Fläche}(H_\ell^+)}{\text{Fläche}(\text{Kreis}(R_T/2))}$. Außerdem sind aufgrund der Maximalität alle Knoten in der Nachbarschaft mindestens eines Knotens aus I. Damit gilt für die erwartete Interferenz an u durch Knoten in H_ℓ, $\Psi_u^{H_\ell}$

$$\begin{aligned}
\Psi_u^{H_\ell} &= \sum_{v \in H_\ell} \Psi_u^v \leq P \sum_{v \in I} \sum_{w \in B_v} \frac{p_w}{(\ell R_I)^\alpha} \leq \frac{\text{Fläche}(H_\ell^+)}{\text{Fläche}(\text{Kreis}(R_T/2))} \frac{2P}{(\ell R_I)^\alpha} \sum_{w \in B_v} p_w \\
&\leq \frac{\pi(((\ell+1)R_I + R_T/2)^2 - (\ell R_I - R_T/2)^2)}{\pi R_T/2} \frac{P}{\ell^\alpha R_I^\alpha} \\
&= \frac{4(2\ell+1)(R_I^2 + R_I R_T)}{R_T^2} \frac{2P}{\ell^\alpha R_I^\alpha} \leq \frac{1}{\ell^{\alpha-1}} \frac{48 P R_I^2}{R_T^2 R_I^\alpha}
\end{aligned}$$

Damit gilt für die gesamte erwartete Interferenz außerhalb des Interferenzradius $\Psi_u^{v \notin R_I}$:

$$\Psi_u^{v \notin R_I} = \sum_{\ell=1}^{\infty} \Psi_u^{H_\ell} = \frac{48 P R_I^2}{R_T^2 R_I^\alpha} \sum_{\ell=1}^{\infty} \frac{1}{\ell^{\alpha-1}} \leq P \frac{48 R_I^{2-\alpha}}{R_T^2} \frac{\alpha - 1}{\alpha - 2} \leq \frac{P}{2\rho\beta R_T^\alpha}$$

◀

Mithilfe von Lemma 2 ist es jetzt möglich, das Sendeverhalten lokal zu analysieren. Zum einem zeigen wir in Lemma 3, dass eine Nachricht in begrenzter Zeit zu einem beliebigen Knoten gesendet werden kann. In Lemma 4 zeigen wir, dass in jeder Nachbarschaft eines Knotens mindestens ein Knoten nach einer begrenzten Zeit erfolgreich an alle seine Nachbarn gleichzeitig eine Nachricht verschickt hat.

▶ **Lemma 3.** *Es gelte, dass alle Knoten in Zustand C_0 ein independent-set bilden. Seien v und u Nachbarn. Dann kann u innerhalb von $\gamma \log n$ Zeitslots erfolgreich eine Nachricht an v mit Wahrscheinlichkeit größer $1 - n^{-c}$ senden, wenn u ein lokaler Anführer ist, oder in $\gamma\Delta \log n$ Zeitslots, wenn u kein lokaler Anführer ist.*

Beweis. Wenn u der einzige Knoten ist, der im Interferenzradius von v sendet, so gibt es dort keine Interferenz. Außerhalb ist die erwartete Interferenz $\frac{P}{2\rho\beta R_T^\alpha}$. Die Wahrscheinlichkeit, dass diese Interferenz nicht um mehr als den Faktor ρ überstiegen wird, beträgt laut Markov-Ungleichung $1 - 1/\rho$. Dann empfängt v das Signal, da die $SINR$ Bedingung erfüllt ist:

$$\frac{\frac{P}{\delta(u,v)^\alpha}}{\rho\Psi_u^{v \notin I_v} + N} \geq \frac{\frac{P}{\delta(u,v)^\alpha}}{\frac{P}{2\beta R_T^\alpha} + \frac{P}{2\beta R_T^\alpha}} \geq \beta$$

Die Wahrscheinlichkeit, dass u als einziger Knoten in einem Bereich I_v mit Radius R_I um v sendet, ist $p_u \cdot \prod_{w \in I_v \setminus \{u\}}(1 - p_w)$. Damit beträgt die Gesamtwahrscheinlichkeit, dass Knoten u erfolgreich an v sendet:

$$\begin{aligned}
&(1 - 1/\rho) \cdot p_u \cdot \prod_{w \in I_v \setminus \{u\}} (1 - p_w) = (1 - 1/\rho) \cdot p_u \cdot \prod_{w \in \{I_v \setminus \{u\}\} \cap C_0} (1 - q_\ell) \cdot \prod_{w \in \{I_v \setminus \{u\}\} \setminus C_0} (1 - q_s) \\
&\geq (1 - 1/\rho) \cdot p_u \cdot \left(1 - \frac{1}{k_{T+I}}\right)^{k_I} \cdot \left(1 - \frac{1}{k_{T+I}\Delta}\right)^{k_I\Delta} \\
&\geq (1 - 1/\rho) \cdot p_u \cdot \left(1 - \frac{1}{k_{T+I}}\right)^{k_I} \cdot \left(1 - \frac{k_I}{k_{T+I}^2\Delta}\right) e^{-k_I/k_{I+T}} \\
&= \lambda p_u
\end{aligned}$$

Damit beträgt die Wahrscheinlichkeit $\mathbb{P}_{\text{Fehlschlag}}$, dass u $\gamma\zeta \log n$ mal hintereinander nicht erfolgreich sendet:

$$\mathbb{P}_{\text{Fehlschlag}} = (1 - \lambda p_u)^{\gamma\zeta \log n} = \left(1 - \frac{\lambda}{k_{I+T}\zeta}\right)^{\frac{ck_{I+T}}{\lambda}\zeta \log n} \leq \left(\left(\frac{1}{e}\right)^{\log n}\right)^c = n^{-c}$$

◀

▶ **Lemma 4.** *Es gelte, dass alle Knoten in Zustand C_0 ein independent-set bilden. Sei Knoten v in Zustand A_i für i beliebig. Innerhalb von $\frac{\sigma}{2}\Delta \log n$ Zeitslots kann ein Nachbar w von v an alle Knoten in seiner eigenen Nachbarschaft erfolgreich innerhalb eines einzigen Zeitslots mit Wahrscheinlichkeit größer $1 - n^{-c}$ erfolgreich senden.*

Beweis. Sei w ein Nachbar von v. Sei B_w die Menge aller Knoten in der Nachbarschaft von w, I_u der Kreis mit Radius R_I um einen Knoten u. Dann sei I_{B_w} die Vereinigung aller I_u mit $u \in B_w$. Die Wahrscheinlichkeit, dass w der einzige Knoten in I_{B_w} ist, der sendet, ist $p_w \cdot \prod_{r \in I_{B_w}}(1 - p_r)$. Die erwartete Interferenz für jeden Knoten u außerhalb von I_{B_w} ist laut Lemma 2 $\frac{P}{2\rho\beta R_T^\alpha}$. Laut Markov Ungleichung ist die Wahrscheinlichkeit, dass diese erwartete

Interferenz um einen Faktor ρ übertroffen wird $1/\rho$. Analog zum vorherigen Beweis wird in diesem Fall die $SINR$ Bedingung erfüllt und jeder Knoten u erhält die Nachricht in einem Zeitslot mit folgender Wahrscheinlichkeit:

$$(1-1/\rho)\cdot p_w \cdot \prod_{r\in I_{B_w}}(1-p_r) = (1-1/\rho)\cdot p_u \cdot \prod_{r\in I_{B_w}\setminus\{w\}\cap C_0}(1-q_\ell)\cdot \prod_{r\in I_{B_w}\setminus\{u\}\}\setminus C_0}(1-q_s)$$
$$\geq (1-1/\rho)\cdot p_w \cdot \left(1-\frac{1}{k_{T+I}}\right)^{k_{I+T}} \cdot \left(1-\frac{1}{k_{T+I}\Delta}\right)^{k_{I+T}\Delta} = \lambda^{'}/\Delta$$

Die Wahrscheinlichkeit, dass dies $\frac{\sigma}{2}\Delta\log n$ mal hintereinander nicht funktioniert, beträgt

$$\left(1-\frac{\lambda^{'}}{\Delta}\right)^{\frac{c}{\lambda^{'}}\Delta\log n} \leq \left(\left(\frac{1}{e}\right)^{\log n}\right)^{c} = n^{-c}$$

◀

2.4.4 Konkurrenz um eine Farbe

In diesem Abschnitt wird erläutert, wie die Konkurrenz um eine Farbe abläuft. Dies ist ein elementarer Bestandteil des Algorithmus, der verwendet wird, um zu Beginn die lokalen Anführer zu bestimmen (siehe Abschnitt 2.4.5). Außerdem wird dieser Mechanismus benötigt, um die endgültige Knotenfärbung zu berechnen (siehe Abschnitt2.4.6). Der entsprechende Pseudocode ist in Algorithmus 1 dargestellt. Am Ende des Abschnittes wird gezeigt, dass

dabei mit hoher Wahrscheinlichkeit eine gültige Knotenfärbung berechnet wird.

1 $P_v := \emptyset; \zeta_i := \begin{cases} 1 \text{ if } i = 0 \\ \Delta \text{ if } i > 0 \end{cases}$; $A_{succ} := \begin{cases} R \text{ if } i = 0 \\ A_{i+1} \text{ if } i > 0 \end{cases}$;
2 **for** $\lceil \eta \Delta \log n \rceil$ *Zeitslots* w **do**
3 | **for** *each* $w \in P_v$ **do** $d_v(w) := d_v(w) + 1$;
4 | ;
5 | **if** $M_A^i(w, c_w)$ *empfangen* **then** $P_v := P_v \cup \{w\}$; $d_v(w) := c_w$;
6 | ;
7 | **if** $M_C^i(w)$ *empfangen* **then** state $:= A_{suc}$; $L(v) := w$;
8 | ;
9 $c_v := \chi(P_v)$, wobei $\chi(P_v)$ der maximale Wert ist, bei dem gilt: $\chi(P_v) \leq 0$ und $\chi(P_v) \notin \{d_v(w) - \lceil \gamma\zeta \log n \rceil, ..., d_v(w) + \lceil \gamma\zeta \log n \rceil\}$ für jedes $w \in P_v$;
10 **while** *state* $= A_i$ **do**
11 | $c_v := c_v + 1$;
12 | **for** *jedes* $w \in P_v$ **do** $d_v(w) := d_v(w) + 1$;
13 | ;
14 | **if** $c_v \geq \lceil \sigma \Delta \log n \rceil$ **then** state $:= C_i$;
15 | ;
16 | sende $M_A^i(v, c_v)$ mit Wahrscheinlichkeit q_s;
17 | **if** $M_C^i(w)$ **then** state $:= A_{suc}$; $L(v) := w$;
18 | ;
19 | **if** $M_A^i(w, C_w)$ *empfangen* **then**
20 | | $P_v := P_v \cup \{w\}$; $d_v(w) := c_w$;
21 | | **if** $|c_v - c_w| \leq \lceil \gamma\zeta_i \Delta \log n \rceil$ **then** $c_v := \chi(P_v)$;
22 | | ;

Algorithmus 1 : Code für Knoten v in Zustand A_i

Wird der Zustand A_i von einem Knoten v betreten, beginnt eine $\eta\Delta \log n$ lange Horchphase. In dieser werden Nachrichten von allen Knoten empfangen, die um dieselbe Farbe konkurrieren (Zeile 2-5) . Anschließend wird in jedem Zeitslot ein Zähler hochgezählt (Zeile 8). Erreicht dieser $\sigma\Delta \log n$, nimmt der Knoten die Farbe i endgültig an. Wird während des Hochzählens eine Nachricht eines Knotens u im Zustand C_i empfangen, wird in den Folgezustand übergegangen, so dass keine ungültige Knotenfärbung entsteht. Dies funktioniert nur dann, wenn u erfolgreich eine Nachricht an v sendet. Dies ist mit Wahrscheinlichkeit $1 - n^{-c}$ in $\gamma \log n$ (Anführer) bzw. $\gamma\Delta \log n$ (andere Knoten) Zeitslots möglich. Diese Zeitspanne nennen wir den **kritischen Bereich**. Während des Hochzählens sendet jeder Knoten in jedem Zeitslot mit Wahrscheinlichkeit q_s seinen Zustand, Nummer und Zähler (Zeile 11). Empfängt ein Knoten eine solche Nachricht von einem Knoten im selben Zustand, wird dessen Zähler lokal gespeichert und in jedem Zeitslot erhöht (Zeile 3-4,9,14). Außerdem wird beim Empfangen der eigene Zähler soweit erniedrigt, dass der Knoten nicht im kritischem Bereich eines gespeicherten Zählers und höchstens 0 ist (Zeile 6,15).

▶ **Theorem 5.** *Für alle $i \geq 0$ bilden alle Knoten, die in Zustand C_i sind, mit einer Wahrscheinlichkeit von mindestens $1 - (O(n^{2-c}))$ ein independent-set.*

Beweis. Zu Beginn ist kein Knoten in einem Zustand C_i, sodass die Behauptung zu Beginn erfüllt ist. Wir zeigen nun für eine Klasse C_i, dass sie am Ende mit Wahrscheinlichkeit $1 - O(n^{1-c})$ ein independent-set bilden.

Sei $v \in A_i$ der Knoten, der zum Zeitpunkt t_v^* zuerst die Unabhängigkeit der Knoten in C_i verletzt. Bis zu diesem Zeitpunkt gilt Lemma 3 . Sei w ein Nachbar von v, der zu einem Zeitpunkt $t_w^* \leq t_v^*$ in den Zustand C_i übergegangen ist. Es gibt nun zwei Fälle.

Fall 1: Falls $t_w^* < t_v^* - \gamma\zeta_i \log n$. Wir zeigen, dass dieser Fall unwahrscheinlich ist. Nach Lemma 3 ist genügend Zeit vergangen, dass w an v mit Wahrscheinlichkeit $1 - n^{-c}$ eine Nachricht geschickt hat. Sollte v zum Zeitpunkt t_w^* noch nicht in Zustand A_i sein, so dauert es mindestens $(\nu + \sigma)\Delta \log n > \gamma\zeta_i \log n$ Zeitslots, bis v in A_i übergehen kann, und erhält ebenfalls mit Wahrscheinlichkeit mindestens $1 - n^{-c}$ eine Nachricht von w. Der Knoten v verlässt daraufhin den Zustand A_i und geht nicht in C_i über (Zeile 5,11).

Fall 2: Falls $t_w^* < t_v^* - \gamma\zeta_i \log n$. Es muss gelten, dass $c_w > \sigma\Delta \log n$ (Zeile 11). Da $\sigma\Delta \log n > 2\gamma\Delta \log n$, kann w in einem Zeitraum J_w der Länge $\gamma\zeta_i \log n$ vor t_w^* nicht zurückgesetzt worden sein, da er sonst in diesem Zeitraum nicht in den Zustand C_i hätte übergehen können. In diesem Zeitraum hätte w mit Wahrscheinlichkeit $1-n^{-c}$ eine erfolgreiche Nachricht an v senden können. Da der Zähler von w im kritischen Bereich von v ist, hätte dessen Zähler dann zu einem Wert kleiner oder gleich 0 zurückgesetzt werden müssen, wodurch v zum Zeitpunkt t_w^* nicht in den Zustand C_i hätte übergehen können.

Die Wahrscheinlichkeit, dass keiner der beiden Fälle eintritt, beträgt damit $1 - 2n^{-c}$. Für alle Knoten in C_i gilt dies mit Wahrscheinlichkeit $1 - O(n^{1-c})$. Da es maximal n verschiedene Zustände C_i geben kann, ist die Wahrscheinlichkeit, dass C_i für ein $i \geq 0$ kein independent-set darstellt $1 - O(n^{2-c})$. ◀

```
1  Farbe_v := i;
2  if i > 0 then
3      repeat sende M_C^i(v) mit Wahrscheinlichkeit q_s until Protokoll beendet;
4  else if i = 0 then
5      tc := 0; ϱ := ∅;
6      repeat
7          if M_R(w, v) empfangen und w ∉ ϱ then ϱ := ϱ ∪ w;
8          ;
9          if ϱ is empty then
10             sende M_C^0(v) mit Wahrscheinlichkeit q_l;
11         else
12             tc := tc + 1;
13             Sei w das erste Element in ϱ;
14             for ⌈μ log n⌉ Zeitslots do sende M_C^0(v, w, tc) mit Wahrscheinlichkeit q_l;
15             ;
16             Entferne w von ϱ;
17     until Protokoll beendet;
```

Algorithmus 2 : Code für Knoten v in Zustand C_i

2.4.5 Initiale Knotenfärbung

```
1 while state = R do
2 |   sende M_R(v, L(v)) mit Wahrscheinlichkeit q_s;
3 |   if M_C^0(L(v), v, tc_v) erhalten then
4 |   |   state := A_{tc_v k+1};
```

Algorithmus 3 : Code für Knoten v in Zustand R

Nach dem Einschalten ist jeder Knoten zunächst im Zustand A_0. Es soll nun eine initiale Knotenfärbung berechnet werden, die bereits annähernd, wenn auch noch nicht komplett gültig ist.

Wenn ein Knoten $v \in A_0$ eine Nachricht eines Knotens $u \in C_0$ erhält, geht er in den Zustand R über und wählt u als seinen lokalen Anführer. Alle Knoten, die u als lokalen Anführer wählen, bilden einen Cluster. Pseudocode für Knoten im Zustand C_i mit $i \geq 0$ wird in Algorithmus 2 und für Knoten in Zustand R in Algorithmus 3 dargestellt. Jeder Knoten v sendet nun in jedem Zeitslot mit Wahrscheinlichkeit q_s eine Anfrage an u. Erhält u eine solche Anfrage, setzt er diese an das Ende einer Liste. Sind alle vorhergehenden Anfragen in der Liste abgearbeitet, sendet u für $\mu \log n$ Zeitslots mit Wahrscheinlichkeit q_l eine Nachricht, die die Intraclusterfarbe für Knoten v spezifiziert. Wir schätzen nun die Zeit ab, die dieser Teil des Algorithmus benötigt.

▶ **Lemma 6.** *Alle Knoten verbringen mit einer Wahrscheinlichkeit von mindestens* $1-O(n^{2-c})$ *maximal* $O(kk_{I+T})\Delta \log n$ *Zeitslots im Zustand* R*.*

Beweis. Sei $v \in R$ und u der lokale Anführer von v. Laut Lemma 3 versendet v innerhalb von $\gamma\Delta \log n$ Zeitslots erfolgreich seine Anfrage an u mit Wahrscheinlichkeit $1 - n^{-c}$. Im Cluster von u gibt es maximal Δ Knoten, die eine Anfrage senden können. Damit beginnt der Algorithmus spätestens nach $(\Delta - 1)\mu \log n$ Zeitslots, an v zu senden. Da $\mu > \gamma$ und Lemma 3 gilt, erhält v dann innerhalb von $\mu \log n$ mit Wahrscheinlichkeit $1 - n^{-c}$ eine Antwort. Erhält er eine Antwort, verlässt er damit den Zustand R nach $\mu\Delta \log n$ Zeitslots. Lemma 3 gilt nur, wenn alle Knoten in C_0 ein independent-set bilden, was laut Lemma 5 mit Wahrscheinlichkeit $1 - O(n^{2-c})$ der Fall ist. Damit verlässt jeder Knoten v mit Wahrscheinlichkeit $1 - O(n^{2-c})$ nach spätestens $(\mu + \gamma)\Delta \log n$ Zeitslots den Zustand R. Einsetzen der Werte für γ und μ zeigt die Behauptung. ◀

2.4.6 Endgültige Knotenfärbung

Nach der initialen Knotenfärbung kann eine ungültige Knotenfärbung nur durch Paare u,v exisiteren, die in ihrer jeweiligen Nachbarschaft, aber in unterschiedlichen Clustern liegen. Wir zeigen, dass es für einen Knoten u nur konstant viele Knoten v gibt, die beide Voraussetzungen erfüllen. Anschließend wird im Beweis von Lemma 8 erläutert, wie diese Konflikte aufgelöst werden.

▶ **Lemma 7.** *Es gelte, dass alle Knoten in* C_0 *ein independent-set bilden. Für* $i > 0$ *gibt es dann für jeden Knoten maximal* k *Nachbarn, die im Zustand* A_i *sind.*

Beweis. Jeder Cluster ist auf einen Radius R_T um den lokalen Anführer beschränkt. Wenn sich die Nachbarschaft eines Knotens v mit dem Cluster des lokalen Anführers u schneidet, kann u nur maximal $2R_T$ von v entfernt sein. Nach Lemma 1 kann es im Radius $2R_T$ nur maximal k lokale Anführer und damit k Cluster geben, die sich mit der Nachbarschaft von v schneiden (inklusive des eigenen Clusters). Also kann v nur mit maximal $k - 1$ Knoten in Konflikt stehen. ◀

▶ **Lemma 8.** *Jeder Knoten durchläuft maximal $k+3$ Zustände.*

Beweis. Lokale Anführer durchlaufen nur die Zustände A_0 und C_0. Alle anderen durchlaufen zunächst A_0 und R, und bekommen anschließend eine Intraclusterfarbe zugewiesen. Es gibt nur Δ Intraclusterfarben t_c, die von 1 bis Δ nummeriert sind. Für jede dieser Intraclusterfarben werden nun k neue Farben eingeführt, die entsprechenden Zustände sind dann $A_{t_c k+1}$ bis $A_{(t_c+1)k}$. Jeder Knoten in R geht in den Zustand $A_{t_c k+1}$ über (Siehe Algorithmus 1 Zeile 4). Der Knoten nimmt dann entweder eine finale Farbe an, oder wechselt in den Zustand A_{i+1} (Algorithmus 1 Zeile 1). Nach spätestens k Zuständen gibt es keine Konkurrenten, da es in $A_{t_c k+1}$ nur $k-1$ Konkurrenten gab (Lemma 7) und diese bereits in den $k-1$ vorangegangen Zuständen eine finale Farbe angenommen haben. Damit nimmt v spätestens dann eine finale Farbe an. Damit hat er insgesamt $k+3$ Zustände durchlaufen. ◀

2.4.7 Analyse

In diesem Abschnitt wird die Laufzeit des Algorithmus untersucht. Zuerst wird gezeigt, dass beim Zurücksetzen eines Knotens der Zähler nicht beliebig klein werden kann. Anschließend wird gezeigt, dass jeder Knoten nur begrenzte Zeit in einem Zustand A_i verbringen kann. Hierzu schätzen wir zunächst den minimalen Wert des Zählers eines Knotens in Lemma 9 ab. Diese Abschätzung wird dann in Lemma 10 verwendet, um die Gesamtzeit eines Knoten in einem Zustand A_i abzuschätzen.

▶ **Lemma 9.** *Es gelte, dass alle Knoten in C_0 ein independent-set bilden. Sei c_v der Zähler von Knoten v. Es gilt: $c_v \geq -2\gamma\Delta \log n - 1$ wenn v im Zustand A_0 ist, sonst gilt $c_v \geq -2\gamma k\Delta \log n - 1$.*

Beweis. Der Zähler c_v wird nur beim Zurücksetzen auf einen negativen Wert gesetzt, ansonsten wird er immer erhöht (Algorithmus 1 Zeile 6,15). Beim Zurücksetzen wird der Zähler auf den maximalen negativen Wert gesetzt, so dass kein lokal gespeicherter Zähler eines anderen Knotens im kritischen Bereich liegt. Dieser Bereich ist $[c_v - \gamma \log n, ..., c_v + \gamma \log n]$ für Knoten in Zustand A_0 bzw. $[c_v - \gamma\Delta \log n, ..., c_v + \gamma\Delta \log n]$ für Knoten in Zustand A_i mit $i > 0$ und damit von der Größe $2\gamma \log n$ bzw. $2\gamma\Delta \log n$. Es kann nur Δ Knoten in der Nachbarschaft eines Knotens im Zustand A_0 bzw. k Knoten in der Nachbarschaft eines Knotens im Zustand A_i mit $i > 0$ geben (Lemma 8). Damit folgt die Behauptung. ◀

▶ **Lemma 10.** *Alle Knoten verbringen mit einer Wahrscheinlichkeit von mindestens $1 - O(n^{2-c})$ in einem Zustand A_i maximal $O(k_{I+T} k^2 \Delta \log n)$ Zeitslots.*

Beweis. Nach Theorem 5 bilden alle Knoten in Zustand C_0 ein independent-set und Lemma 3 und Lemma 4 gelten. Wir betrachten nun einen Knoten $v \in A_i$ und zeigen, dass dieser nach begrenzt vielen Zeitslots den Zustand A_i verlässt. Wenn v die Zeile 6 das erste mal ausführt (nach $\nu\Delta \log n$ Zeitslots), benötigt er noch mindestens $\sigma\Delta \log n$ Zeitslots, bis er in den Zustand C_i übergeht. Nach Lemma 4 gibt es einen Nachbarn w, der innerhalb von $\frac{\sigma}{2}\Delta \log n$ Zeitslots erfolgreich in einem Zeitslot an alle Knoten in seiner Nachbarschaft eine Nachricht schickt. Sei dieser Zeitpunkt t_w. Wir zeigen nun, dass w ab dem Zeitpunkt t_w nicht mehr zurückgesetzt wird.

Zum Zeitpunkt t_w haben alle Nachbarn u von w, die im Zustand A_i sind, einen korrekten Zähler c_w von w lokal gespeichert. Solange w nicht zurückgesetzt wird, haben alle Nachbarn weiterhin einen korrekten Wert von c_w gespeichert (Algorithmus 1 Zeile 3, 8). Nach Zeile 15 in Algorithmus 1 und weil jeder Nachbar u (aktuell) eine korrekte Kopie von c_w hat, wird der Zähler des Nachbarn bei Empfang einer Nachricht so gesetzt, dass er nicht im kritischen

Bereich von c_w liegt. Sendet ein Nachbar u mit Zähler c_u anschließend eine Nachricht an w, so ist c_w nicht im kritischen Bereich von c_u, und c_w wird nicht zurückgesetzt. Nach Lemma 9 ist $c_w > -2\gamma k\Delta \log n - 1$. Der Knoten w benötigt also maximal $(-2\gamma k + \sigma)\Delta \log n - 1$ Zeitslots um in C_i überzugehen. Der Knoten w kann damit auch nicht durch Knoten zurückgesetzt werden, die zum Zeitpunkt t_w nicht im Zustand A_i waren: Diese senden für die ersten $\nu\Delta \log n$ Zeitslots keine Nachricht (Algorithmus 1 Zeile 2-5), und $\nu > 2\gamma k + \sigma + 1$. Damit wird c_w nach dem Zeitpunkt t_w nicht mehr zurückgesetzt.

Es sind nun verschiedene Fälle zu unterscheiden:

Fall 1: *Der Knoten w geht nach spätestens $(2\gamma k + \sigma)\Delta \log n - 1$ Zeitslots in den Zustand C_i über.* Da v ein Nachbar von w ist, hat v einen korrekten Zähler c_w und der Zähler von v ist nicht im kritischen Bereich von c_w. Damit bekommt v mit Wahrscheinlichkeit $1 - n^{-c}$ in $\gamma\zeta_i \log n$ Zeitslots nachdem w in C_i übergegangen ist eine Nachricht von w und geht in den Nachfolgezustand über.

Fall 2: *Der Knoten v geht vor w in den Zustand C_i über.* Ab dem Zeitpunkt t_w ist v also weniger als $(2\gamma k + \sigma)\Delta \log n - 1$ Zeitslots in A_i.

Fall 3: *Ein Knoten u, der gemeinsamer Nachbar von v und u ist, geht vor w in den Zustand C_i über.* Der Knoten u geht ab dem Zeitpunkt in weniger als $(2\gamma k + \sigma)\Delta \log n - 1$ Zeitslots in den Zustand C_i und sendet mit Wahrscheinlichkeit $1 - n^{-c}$ in $\gamma\zeta_i \log n$ Zeitslots eine Nachricht an v, so dass v A_i verlässt.

Fall 4: *Ein Knoten u, der ein Nachbar von w, aber nicht von v ist, geht vor w in den Zustand C_i über.* Der Knoten u ist maximal $2R_T$ von v entfernt. In den nächsten $\frac{\sigma}{2}\Delta \log n$ Zeitslots sendet ein weiterer Nachbar w' mit Wahrscheinlichkeit $1-n^{-c}$ an alle seine Nachbarn in einem einzigen Zeitslot $t_{w'}$, und es tritt wieder Fall 1 - 4 ein. Jedes mal, wenn Fall 4 eintritt, geht ein Knoten im Umkreis mit Radius $2R_T$ von v in Zustand C_i über. Dies kann nur $k-1$ mal der Fall sein, da im Umkreis von $2R_T$ nach Lemma 1 maximal k Knoten dieselbe Farbe haben können. Fall 4 Kann also maximal $k-1$ mal mit eintreten (mit der Wahrscheinlichkeit, das ein Nachbar w erfolgreich sendet von jeweils $1 - n^{-c}$), und zwischen dem Eintreten von Fall 4 liegen weniger als $(2\gamma k + \sigma)\Delta \log n - 1 + \sigma\Delta \log n$ Zeitslots. Anschließend tritt Fall 1,2 oder 3 ein und v verlässt A_i.

Nun lässt sich die Laufzeit zusammenfassen: Ein Knoten v verbringt zunächst $\nu\Delta \log n$ Zeitslots in der Horchphase in der er nichts sendet. Im schlimmsten Fall ist er $k-1$ mal in Fall 4, und einmal in einem der übrigen Fälle. Zuletzt benötigt es noch $\gamma\zeta_i \log n$ Zeitslots um v eine Nachricht zu senden, falls ein Nachbar von v in den Zustand C_i übergeht. Die Gesamtzeit, die v in A_i verbringt ist damit maximal $k((2\gamma k + \sigma)\Delta \log n - 1 + \sigma\delta \log n) + \gamma\zeta_i \log n = O(k^2 k_{I+T}\Delta \log n)$. Die Wahrscheinlichkeit, dass Fall 1 - 4 erfolgreich ist, beträgt jeweils n^{-c}, die des Sendens der letzten Nachricht an v ebenfalls n^{-c} und die Wahrscheinlichkeit, dass C_0 ein independent-set darstellt ist $O(n^{2-c})$. Die Gesamtwahrscheinlichkeit, dass die Zeit in einem Zustand A_i eingehalten wird, ist also $1 - ((k+1)n^{-c} + O(n^{2-c}) > 1 - O(n^{2-c})$ für n oder c groß genug. ◀

Mit dem bisherigem lässt sich nun die Gesamtlaufzeit des Algorithmus abschätzen.

▶ **Theorem 11.** *Mit Wahrscheinlichkeit von mindestens $1 - O(n^{2-c})$ berechnet der Algorithmus eine gültige Distanz-1 Knotenfärbung mit $(k+1)\Delta$ Farben in $O(\Delta \log n)$ Zeitslots, nachdem die Knoten aufgewacht sind.*

Beweis. Jeder Knoten durchläuft maximal $k+1$ Zustände vom Typ A_i und maximal einmal den Zustand R (Lemma 8). In jedem dieser Zustände befindet sich jeder Knoten für $O(k_{I+T}k^2\Delta \log n)$ Zeitslots (Lemma 6, Lemma 10) mit Wahrscheinlichkeit $1 - O(n^{2-c})$. Damit ist die Gesamtlaufzeit mit Wahrscheinlichkeit $1 - O(n^{2-c})$ in $O(k_{I+T}k^3)\Delta \log n$. Es

gibt maximal Δ Intraclusterfarben. Für jede Intraclusterfarbe gibt es k zusätzliche Farben. Berücksichtigt man noch die Farbe C_0, so gibt es am Ende des Algorithmuses maximal $k\Delta+1$ Farben. ◀

2.5 TDMA im SINR Modell

In diesem Abschnitt wird gezeigt, wie durch eine gültige $d+1$-Distanz Knotenfärbung ein funktionierendes MAC-Layer aufgebaut werden kann. Anschließend wird noch erläutert, wie mit dem in dieser Seminararbeit vorgestellten Algorithmus eine $d+1$-Distanz Knotenfärbung berechnet werden kann.

▶ **Theorem 12.** *Für $d = (32\frac{\alpha-1}{\alpha-2}\beta)^{1/\alpha}$ ermöglicht eine Distanz-$(d+1)$ Knotenfärbung eine kollisionsfreie Kommunikation zwischen den Knoten, wenn jeder Knoten nur in dem für seine Farbe definierten Zeitslot sendet.*

Beweis. Wir betrachten einen Knoten v, der an einen Knoten u in seiner Nachbarschaft sendet, und zeigen, dass diese Nachricht immer empfangen wird. Wie im Beweis zu Lemma 2 betrachten wir zuerst einen Kreisring H_ℓ um den Knoten u, wobei für alle Knoten $w \in H_\ell$ gilt: $\ell R_I \leq d(u,w) < (\ell+1)R_I$ und berechnen die Interferenz anderer Knoten aus diesem Ring. Man beachte, dass für $\ell = 0$ H_ℓ der Interferenzbereich um u ist. In jedem dieser Ringe senden nur Knoten zur selben Zeit wie v, die die selbe Farbe wie v haben. Aufgrund der Tatsache, dass eine $d+1$-Distanz Knotenfärbung vorliegt, sind alle Knoten mit gleicher Farbe mindestens die Distanz dR_T voneinander entfernt. Analog zum Vorgehen in Lemma 2 kann man damit die Anzahl Knoten gleicher Farbe in einem Kreisring H_ℓ abschätzen durch

$$\frac{\text{Fläche}(H_\ell^+)}{\text{Fläche}(\text{Kreis}(dR_T/2))} = \frac{\pi((\ell+3/2)dR_T)^2 - \pi((\ell-1/2)dR_T)^2}{\pi d^2 R_T^2/4} \leq 16\ell$$

Der Bereich H_ℓ^+ ist dabei ebenfalls analog zu Lemma 2 definiert als alle Punkte x mit $(\ell-1/2)dR_I \leq d(x,u) < (\ell+1/2)R_I$. Damit gilt für die Gesamtinterferenz $\phi_{u\setminus\{v\}}$ am Knoten u ohne v

$$\phi_{u\setminus\{v\}} = \sum_{w\in V\setminus\{v\}} \frac{P}{\delta(w,u)^\alpha} = \sum_{\substack{w\in V\setminus\{v\} \\ w \text{ hat Farbe } c}} \frac{P}{\delta(w,u)^\alpha} = P\cdot\sum_{\ell=1}^{\infty} \sum_{\substack{w\in V\setminus\{v\} \\ w \text{ hat Farbe } c}} \frac{1}{(\ell dR_T)^\alpha}$$
$$\leq P\cdot\sum_{\ell=1}^{\infty}\frac{16\ell}{(\ell dR_T)^\alpha} = \frac{16P}{d^\alpha R_T^\alpha}\sum_{\ell=1}^{\infty}\frac{1}{\ell^{\alpha-1}} = \frac{16P}{d^\alpha R_T^\alpha}\frac{\alpha-1}{\alpha-2} \leq \frac{P}{2\beta R_T^\alpha}$$

Da v in der Nachbarschaft von u liegt, ist die $SINR$ Bedingung erfüllt:

$$\frac{\frac{P}{R_T^\alpha}}{\phi_{u\setminus v}+N} \geq \frac{\frac{P}{R_T^\alpha}}{\frac{P}{2\beta R_T^\alpha}+\frac{P}{2\beta R_T^\alpha}} \geq \beta$$

Damit ist ein TDMA-MAC Layer wie es in Abschnitt 2.3 vorgestellt wurde auch im $SINR$-Modell möglich. ◀

Der in dieser Seminararbeit vorgestellte Algorithmus berechnet eine 1-Distanz Knotenfärbung. Lässt sich die Sendeleistung der Sensoren einstellen, so kann man diese zu Beginn

der Berechnung der Färbung soweit erhöhen, dass für die neue *Transmission Range* $R_{T_{neu}}$ gilt: $R_{T_{neu}} = (d+1)R_T$. Somit sind nun alle Knoten in der alten $d+1$ Nachbarschaft jetzt direkte Nachbarn und erhalten nicht dieselbe Farbe. Ist der Algorithmus abgeschlossen, dreht man die Sendeleistung wieder zurück. Damit ist eine $d+1$-Distanz Knotenfärbung berechnet worden. Wählt man praxisübliche Parameter wie $\alpha = 3$ und $\beta = 1$, so wäre $d = 4$. Dies bedeutet, man müsste die Sendeleistung so weit erhöhen, dass sie den fünffachen Radius abdeckt. Diese bedeutet leider Einschränkungen für die Anwendung des Algorithmus in der Praxis.

Hinweis: Der hier vorgestellte Algorithmus berechnet eine $k\Delta + 1$ Knotenfärbung. Es existieren Algorithmen, die eine $\Delta + 1$ Knotenfärbung berechnen, aber eine funktionierende Kommunikation zwischen Nachbarn voraussetzen (z.B. [1]). Nachdem der hier vorgestellte Algorithmus eine $k\Delta+1$ Knotenfärbung berechnet hat, kann man diese Algorithmen einsetzen, um eine $\Delta + 1$ Knotenfärbung zu berechnen. Dadurch wird das MAC-Layer um etwa den Faktor k schneller.

2.6 Zusammenfassung

Es wurde ein Algorithmus vorgestellt, der im $SINR$ Modell in $O(\Delta \log n)$ Zeit mit Wahrscheinlichkeit $O(1 - n^{-O(1)})$ eine gültige Knotenfärbung berechnen kann. Desweiteren wurde gezeigt, wie dieser Algorithmus dazu verwendet werden kann, um ein MAC-Layer aufzubauen, um die kollisionsfreie Kommunikation zwischen Sensoren zu ermöglichen.

Eine offene Frage ist, ob man die Analyse rein lokal, d.h. ohne die Kenntnis von Δ und n durchführen kann, wie es im Protokoll Modell gelungen ist ([7]). Außerdem ist es für die Praxis fraglich, ob man die Sendeleistung an den Sensoren wirklich so hoch einstellen kann, dass eine $(d+1)$-Distanz Knotenfärbung berechnet werden kann.

Referenzen

1 Leonid Barenboim and Michael Elkin. Distributed $(\delta + 1)$-coloring in linear (in δ;) time. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 111–120, New York, NY, USA, 2009. ACM.

2 B. Derbel and E. Talbi. Distributed node coloring in the sinr model. In *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, pages 708–717, June 2010.

3 Olga Goussevskaia, Thomas Moscibroda, and Roger Wattenhofer. Local broadcasting in the physical interference model. In *Proceedings of the Fifth International Workshop on Foundations of Mobile Computing*, DIALM-POMC '08, pages 35–44, New York, NY, USA, 2008. ACM.

4 Magnús M. Halldórsson and Pradipta Mitra. Towards tight bounds for local broadcasting. In *Proceedings of the 8th International Workshop on Foundations of Mobile Computing*, FOMC '12, pages 2:1–2:9, New York, NY, USA, 2012. ACM.

5 N. Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

6 Thomas Moscibroda and Roger Wattenhofer. Coloring unstructured radio networks. *Distributed Computing*, 21(4):271–284, 2008.

7 Johannes Schneider and Roger Wattenhofer. Coloring unstructured wireless multi-hop networks. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing*, PODC '09, pages 210–219, New York, NY, USA, 2009. ACM.

3 Dynamic Point Labeling

Daniel Feist

Abstract. *This seminar paper deals with labeling a set of dynamic points which move along (continuous) trajectories. We prove that deciding whether an overlap-free labeling exists is strongly PSPACE-complete (in appropriate label models), making optimizations PSPACE-hard (Section 3.2). This is done by reducing from Non-deterministic Constraint Logic (NCL), a single-player game on a graph. To solve this problem in pratical time, we develop a heuristic algorithm considering several quality criteria to ensure comfortable, readable labelings for free-label maximization on dynamic point sets (Section 3.5). In this context we introduce a new label model called the α-trailing model (Section 3.4). Additionally, we apply an optimization technique called hourglass trimming on our heuristic algorithm (Subsection 3.5.3). A short experimental evaluation rounds off this work (Section 3.6).*

3.1 Motivation and Introduction

This work is based on Dirk Gerrits *"Pushing and Pulling - Computing push plans for disk-shaped robots, and dynamic labelings for moving points"* [9] and covers dynamic point labeling problems. If not marked with another reference, every algorithmic technique and theoretical results presented here came from this source.

Imagine the situation of having a set of points on a map (which for example represent cities) and we want to place textual labels next to these points (each of them represent the name of the particular city). This task is known as *static point labeling* in general. Now consider we want to label a map which can be panned, rotated and/or zoomed from time to time, causing points to move, appear and disappear from the view. Labeling sets of points which are not fixed but change over time which means that new points are added, others are removed and/or points moving continuously and may appear or disappear even without referece to a specific viewport is named (*general*) *dynamic point labeling*. To obtain an eye-friedly point labeling, we have the fundamental request on the label placements that each label has the corresponding point on its boundary (to have a clear link between labels and points). Secondly, we desire the label interiors to be pairwise disjoint, i.e., a smaller number of overlapping labels is better. How strict we are in terms of allowed overlapping depends on the maximization we want to achieve.

In addition, one may ask which label positions around the points are allowed. This is specified by the label model as shown in Figure 6. Fixed-position models have a finite number of label candidates as indicated by the gray-bordered rectangles. Slider models (introduced by [15]) are their infinite counterpart as depicted by the gray shaded rectangles. In section 3.3 we will introduce another label model called the α-trailing model which is a subset of the 4S model below. The most obvious labeling problem is stated in the following definition:

▶ **Definition 1** (Decision Problem)**.** The *Decision Problem* asks: exists a labeling $\mathcal{L}$ that assigns labels to all points in P without overlapping?

Maximizations of the decision problem under three different constraints are now determined as needed for a universal static/dynamic point labeling problem definition afterwards:

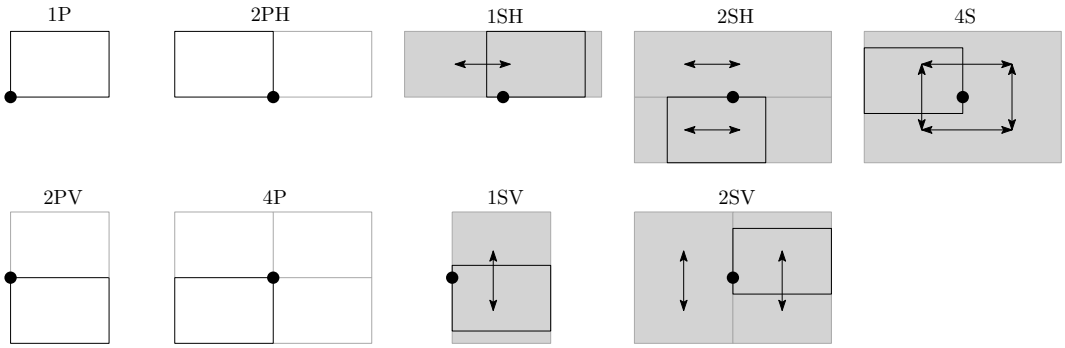


Figure 6 Fixed-position label models with allowed label positions indicated by gray frame(s) (left). Slider label models with allowed label positions depicted by gray shaded areas (right).

▶ **Definition 2** (Size Maximization Problem)**.** The *Size Maximization Problem* asks: which is the maximum selectable label scaling factor if we want to label all points without any overlappings?

▶ **Definition 3** (Number Maximization Problem)**.** The *Number Maximization Problem* asks: how many points can we label without any overlap at the maximum?

▶ **Definition 4** (Free-label Maximization Problem)**.** The *Free-label Maximization Problem* asks: if we allow overlapping, what is the maximum number of free labels, where free means without overlap?

▶ **Definition 5** (Static Point Labeling Problem)**.** Suppose a static set of points $P \subset \mathbb{R}^2$ and a given label model for positioning the labels. The *Static Point Labeling Problem* asks for a labeling $\mathcal{L}$ of P solving one of the problems denoted in Definition 1, Definition 2, Definition 3 or Definition 4.

▶ **Definition 6** ((General) Dynamic Point Labeling Problem)**.** Suppose a dynamic set of points $P \subset \mathbb{R}^2$ and a specified time interval called *lifespan* for every $p \in P$ so that $\mathit{life}(p) = [\mathit{birth}(p), \mathit{death}(p)]$ consisting of the point in time $\mathit{birth}(p)$ where p is added and the point in time $\mathit{death}(p)$ where p is removed from the view. Here, $p(t)$ denotes the position of point p at time t, $P(t) = \{p(t) \mid p \in P, t \in \mathit{life}(p)\}$ the respective position of all alive points at time t. We require a label model for positioning the labels. The *Dynamic Point Labeling Problem* asks for the existence of static labelings $\mathcal{L}(t)$ to $P(t)$ solving one of the problems denoted in definition 1, definition 2, definition 3 or definition 4 for all t.

Static point labeling was in focus of extensive research. Experimentally validated heuristics as well as algorithms with proven approximation ratios are known for all aforementioned problems but free-label maximization since its a new variant. The following list presents a short overview over the present state of the art.

- Decision Problem (definition 1) of static point labeling. *Complexity*: strongly NP-complete for {3P [6], 4P [7, 13], 4S [15]} label models with 1×1 labels.
- Size Maximization Problem (definition 2) of static point labeling. *Complexity*: strongly NP-hard for 4P [7] label models with 1×1 labels. *Approximation ratio*: Does not admit a polynomial-time $(2 - \varepsilon)$-approximation algorithm assuming $\varepsilon > 0$, P $\neq$ NP, 4P model and 1×1 labels [7].

- Number Maximization Problem (definition 3) of static point labeling. *Complexity*: strongly NP-hard for the 1P label model even with 1×1 labels [8, 11]. *Approximation ratios*: There exists a PTAS for all label models and 1×1 labels. The following holds for unweighted points only: [1] introduces the first PTAS and describes an $\mathcal{O}(n \log n)$-time 2-approximation algorithm for fixed-position 1×1 labels, [15] for slider models and [4] an $\mathcal{O}(\log \log n)$-time approximation algorithm for arbitrary fixed-position rectangles.

Static point labeling can be generalized to dynamic point sets as described in definition 6. A dynamic point labeling that optimizes size maximization has the problem that continuously scaling labels are hard to read. In contrast, number maximization makes labels appearing and disappearing whether no new points are added or existing points are removed. This might look confusing. As a consequence, the new variant called free-label maximization seems to be most promising. This hypothesis is supported by air-traffic controllers from which [2] found out that they evaluate fast label movement worse than overlapping.

3.2 Complexity of Dynamic Point Labeling

We already mentioned that previous results have shown that static point labeling is strongly NP-complete (and therefore NP-hard). In this chapter we prove strong PSPACE-completeness of the decision problem having a dynamic point set P and a 4P, 2S or 4S label model. This is even true if the dynamic nature of P is drastically restricted and all labels are of size 1×1. This proof entails that any dynamic generalization like {size/number/free-label}-maximization is strongly PSPACE-hard regarding these label models.

To prove PSPACE-hardness of dynamic point labeling, one needs a PSPACE-complete problem which can be reduced to this in polynomial time. We reduce from Non-deterministic Constraint Logic (NCL) by Hearn and Demaine [10], a single player game that is defined next (definition 7). They showed that deciding simple questions (definitions 8, 9 and 10) in this game are PSPACE-complete even for planar, 3-regular constraint graphs consisting only of AND vertices and protected OR vertices. The vertex types of *NCL* are depicted in Figure 7. We distinguish between three different types of vertices. An AND vertex has weight 2 and incident edges having weight 1, 1 and 2. An OR vertex with weight 2 and all incident edges having a weight of 2. An *protected* OR vertex since it has two edges which cannot directed inward simultaneously because it leads to an illegal constraint graph configuration.

▶ **Definition 7** (Non-deterministic Constraint Logic (NCL), [10])**.** The *NCL* is an abstract, single-player game on a *constraint graph* (which is an undirected graph with weights on both the vertices and the edges). A constraint graph's *configuration* describes the orientation of every edge and is called *legal* iff each vertex inflow is not higher than its own weight. In contrast, however, the vertex outflow is not restricted. Reversing a single edge in a legal configuration so that the configuration is still legal again is called a *move* in this game.

▶ **Definition 8** (Configuration-to-configuration NCL)**.** Given two legal configurations $\mathcal{C}_A$ and $\mathcal{C}_B$, is there a sequence of moves transforming $\mathcal{C}_A$ into $\mathcal{C}_B$?

▶ **Definition 9** (Configuration-to-edge NCL)**.** Given a legal configuration $\mathcal{C}_A$ and an orientation for a single edge e_B, is there a sequence of legal moves transforming $\mathcal{C}_A$ into a legal configuration $\mathcal{C}_B$ in which e_B has the specified orientation?

▶ **Definition 10** (Edge-to-edge NCL)**.** Given orientations for edges e_A and e_B, do there exist legal configurations $\mathcal{C}_A$ and $\mathcal{C}_B$, and a sequence of legal moves transforming the one into the other, such that e_A (e_B) has the specified orientation in $\mathcal{C}_A$ ($\mathcal{C}_B$)?

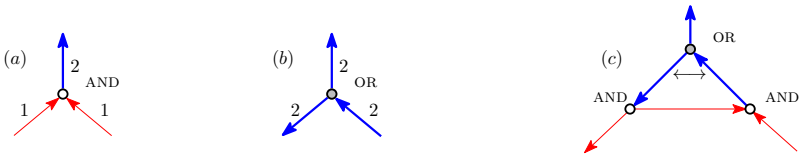


Figure 7 Different vertex types of *NCL*. (a) shows an AND vertex (white disc) with weight 2 and incident edges having weight 1, 1 and 2. (b) shows an OR vertex (gray disk) with weight 2 and all incident edges having a weight of 2. (c) depicts a *protected* OR vertex having two edges as indicated by $\longleftrightarrow$ which cannot directed inward simultaneously.

In the next step we describe how a dynamic point set P can be represented by a planar constraint graph G with n vertices. Hereby, the labeling of P correspond to configurations of G and label movements correspond to inverting edge orientations in G.

We map G on a regular grid turned by 45° relative to the coordinate axes. G's vertices are on the grid's vertices and G's edges are on grid's edges. As a result, the grid graph is planar since the grid lines form interior-disjoint paths. See Figure 8 (1) for an example. This embedding is computable in $\mathcal{O}(n)$ time by the algorithm of Biedl and Kant [3]. The grid's vertices and edges represent a particular type of point set, each a subset of P, represented by gadgets as illustrated in Figure 8 (2) – (4).

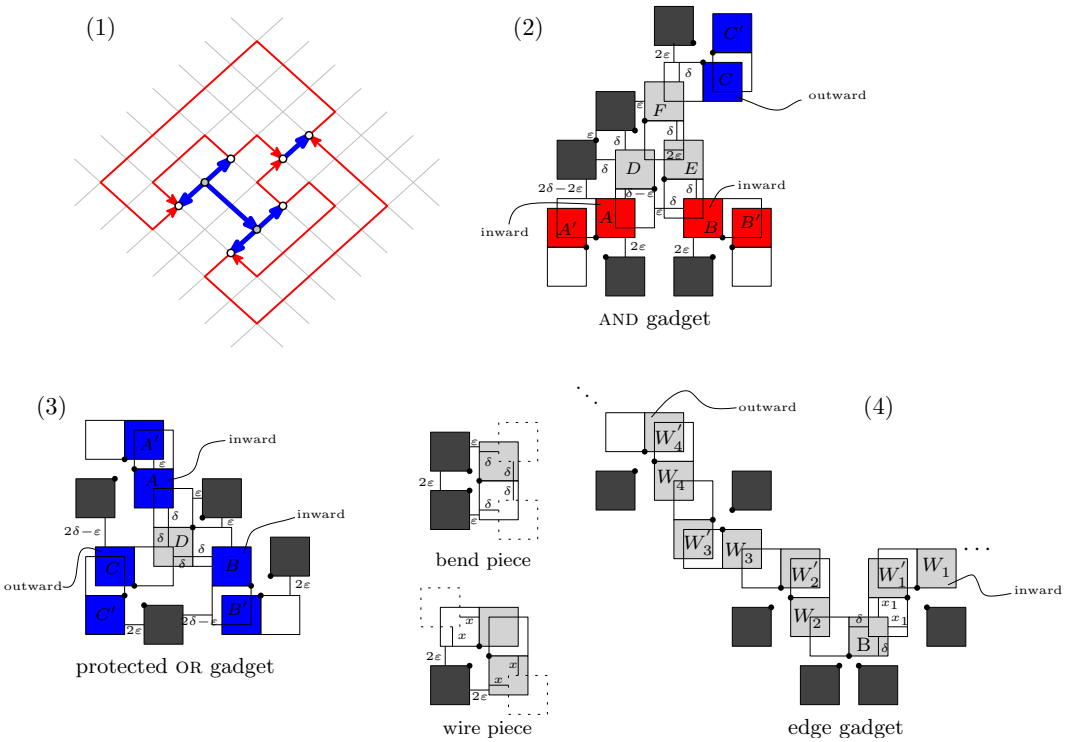


Figure 8 (1) shows a planar constraint graph for simulating *NCL* a dynamic point labeling. (2) depicts the gadget simulating an *NCL* AND vertex and (3) an *NCL* protected OR vertex. The edge gadget of (4) forms a polygonal chain of diagonal segments. It consists of bend and wire pieces connected at the variable dashed positions (depending on x whereas $x \in [\delta - \varepsilon, 1]$ (4P) or $x \in [\delta - \varepsilon, 1 - 3\varepsilon]$ (2S/4S)). It simulates an edge between different AND and protected OR gadgets. The entire construction works for unit-square labels of side length 1 up to $1 + \delta - \varepsilon$.

an AND resp. protected OR gadget simulates the same behaviour as their corresponding *NCL* vertex types (see Figure 8 (2) resp. (3)). An edge gadget (Figure 8 (4)) is used to connect AND and protected OR vertex gadgets over longer distances with each other so that a label moving out of this vertex gadgets lets another label moving inward "on the opposite side" and vice versa.

Dark gray labels are unmovable and called *blockers*. They restrict the placement/movement of the red and blue labels to two different orientations corresponding to the two different edge orientations. The light gray labels restrict the space usable for the red and blue labels to ensure the inflow constraint of the different vertex types. Label positions closest to the center of a vertex gadget are called *inward*, their counterparts are called *outward*. Thus edges directed *out of* a NCL vertex in G should be equivalent to labels placed inward and edges directed *into* a NCL vertex in G should be equivalent to labels placed outward. The AND and protected OR vertices in G have to be equivalent to their corresponding gadgets which represent a specific point configuration. That this is always the case we have to show that overlap-free gadgets always correspond to a legal constraint graph configuration and vice versa. In addition, movements between constraint graph configurations are legal iff their respective label movements can be done continuously whithout overlap. In short, we want to show that "connected" gadgets which represent an overlap-free labeling are equivalent to a legal constraint graph configuration. The following three Lemmata 11, 12 and 13 lead to Theorem 14 which exactly shows this equation. From now on we assume labels of size 1×1 unless specified differently.

▶ **Lemma 11.** *The* AND *gadget in Figure 8 satisfies the same constraints as NCL* AND *vertex with incident edges A, B and C, where A and B are the edges of weight 1. This holds for the 4P model when* $0 < \varepsilon \leq \delta < 1 - 3\varepsilon$*, and for the the 2S and 4S models when* $0 < \varepsilon \leq \delta < 1/3 - 4\varepsilon/3$*.*

Proof. We first show that in an overlap-free labeling which satisfies the inflow constraint of an *NCL* AND vertex either C must be placed outward or A as well as B must be placed outward. This is trivial to see in the 4P label model. In the 2S/4S label model we proceed as follows: Moving A and B down by 2ε allows moving D and E moving down by $\delta + \varepsilon$ and $\delta + 2\varepsilon$ respectively. Thus, F can be moved down by at least $2\delta + 2\varepsilon$ opening a gap of size $3\delta + 4\varepsilon$ above F which is smaller than $4(1/3 - 4\varepsilon/3) + 4\varepsilon = 1$. Consequently, C cannot fill this gap and moves inward. Larger label sizes than 1×1 would further narrow the gap. Contrary, moving D upward (downward) when A moves inward (outward) - the same works for B and C with E and F - shows that all legal game moves can be performed as label movements. ■

▶ **Lemma 12.** *The protected* OR *gadget in Figure 8 satisfies the same constraints as NCL protected* OR *vertex with incident edges A, B and C, where A and B form the protected pair. This holds for the 4P model when* $0 < \varepsilon \leq \delta < 1 - 3\varepsilon$*, and for the the 2S and 4S models when* $0 < \varepsilon \leq \delta < 1/3 - 4\varepsilon/3$*.*

Proof. Under the precondition that label A and B are never both placed inward, we have to show that it is impossible to place A, B and C inward at the same time. That this is not possible in the 4-position model is trivial. That this holds for the 2S and 4S model as well becomes clear when we create a maximum gap between A and C which is at most $2\delta + 3\varepsilon$ but at the same time smaller than $2(1/3 - 4\varepsilon/3) = 2/3 + \varepsilon/3 < 1$. The same argument applies to B and C. As a result, D does not fit in this gap and this holds for labels of increasing size as well because it will make the gaps narrower.

Placing A, B and C outward is possible in every situtation. Now moving A, B or C inward forces D to move out of the way which is a problem when C is already placed inward. That

would require A and B are placed outward before on of them moves inward. We assumed A and B to be the protected pair (with edges not both directed inward) and therefore not being placed outward. Thus this situation is impossible and this shows that all legal game moves can be performed as label movements. ■

▶ **Lemma 13.** *A label on one end of an edge gadget in Figure 8 may only move out of its vertex gadget if the label on the other end is placed into its vertex gadget. This holds for the 4P model when* $0 < \varepsilon \leq \delta < 1 - 3\varepsilon$*, and for the the 2S and 4S models when* $0 < \varepsilon \leq \delta < 1/3 - 4\varepsilon/3$.

Proof Idea. The label W_1 is placed inward relative to a vertex gadget which might be located next to it. It is to show that moving W_1 outward triggers a chain reaction which traverses the edge gadget like a signal. Thus the edge gadgets label on the other end must be placed inward. The other direction holds due to symmetrical reasons.

▶ **Theorem 14.** *Let G be a planar constraint graph with n vertices and let ε and δ be two real numbers with $0 < \varepsilon \leq \delta < 1 - 3\varepsilon$. One can then construct a point set $P = P(G, \delta, \varepsilon)$ that has the following properties for any $s \in [1, 1 + \delta - \varepsilon]$:*

- *(i) the size of P and the coordinates of the points in P are polynomially bounded in n.*
- *(ii) for any legal configuration of G there is an overlap-free static labeling of P with $s \times s$ in the 4P model and vice versa,*
- *(iii) there is a sequence of moves transforming one legal configuration of G into another iff there is a dynamic labeling transforming the corresponding static labelings of P into each other.*

When $\delta < 1/3 - 4\varepsilon/3$ the same results holds for the 2S and 4S models.

Proof. Given a constraint graph G we can achieve a planar grid graph embedding by the algorithm of [3] in $\mathcal{O}(n)$ time. After turning the grid by 45° relative to the coodinate axes we replace the vertices and edges of G by the appropriate gadgets of Lemma 11, Lemma 12 and Lemma 13. The points of the gadgets alltogether form the set P as wanted.

Claim (*i*) is true as the coordinates of all points in the gadgets are linear combinations of 1, δ and ε with integer coefficients. In addition, it remains to ensure that this property is still true when connecting gadgets with each other (by using edge gadgets). This holds if the number of wire pieces is polynomial and consequently none of the integer coefficients can be super-polynomial which is true since the algorithm of [3] places G on a $n \times n$ grid with not more than $2n + 2$ bend pieces.

Claim (*ii*) and (*iii*) follow from the equivalence of the gadgets defined in Lemma 11, Lemma 12 and Lemma 13 with their corresponding NCL vertex types. ■

3.2.1 Hardness of Dynamic Point Labeling

Now all basic tools are introduced and we are able to answer the question raised at the beginning of this section, namely why the decision problem of whether a solution of a dynamic point labeling exists is strongly PSPACE-complete. And therefore any optimizations concerning this problem are strongly PSPACE-hard.

▶ **Theorem 15.** *The following decision problem is strongly PSPACE-hard for the 4P, 2S and 4S model.*
Given: *A dynamic point set P, numbers a and b ($a < b$) and a static labeling $\mathcal{L}(a)$ for $P(a)$.* ***Decide:*** *Whether there exists a dynamic labeling $\mathcal{L}$ for P respecting $\mathcal{L}(a)$ that labels all points with non-overlapping 1×1 over the time interval $[a, b]$.*

The theorem above remains true even when all points are stationary and only *one* point is removed and only *one* point is added in $[a, b]$. This is shown by the following proof.

Proof. PSPACE-hardness is provable by reduction from configuration-to-edge NCL. First we construct a point set $P = P(G, \delta, \varepsilon)$ and a valid static labeling corresponding to the starting configuration $\mathcal{C}_A$ of the given constraint graph G. Secondly, we have a goal orientation for a single edge e_B. Now move one blocker $q \in P$ in this edge gadget of e_B towards the nearest non-blocked point $p \in P$ (shown as the red label in Figure 9 (1)) at constant speed $v = 2\varepsilon/(b-a)$. As a result, the edge gadget has become constrained to a single orientation around p as depicted in Figure 9. Figure 9 makes it clear that there must be a sequence of moves in $\mathcal{C}_A$ with the given direction of e_B iff there is a dynamic labeling for $\mathcal{L}$ starting with $\mathcal{L}_A$ from which we know that an algorithm solving this problem also solves this PSPACE-hard configuration-to-edge NCL problem. Furthermore, the decision problem is strongly PSPACE-hard since all coordinates are polynomial bounded in the size of G. ■

Note that due to symmetrical reasons, the hardness proof works analogously when $\mathcal{L}(b)$ instead of $\mathcal{L}(a)$ is given. If there is no static labeling given at all, the problem does not become easier.

▶ **Theorem 16.** *The following decision problem is strongly PSPACE-hard for the 4P, 2S and 4S model.*
Given: *A dynamic point set P and numbers a and b $(a < b)$.*
Decide: *Whether there exists a dynamic labeling $\mathcal{L}$ for P that labels all points with non-overlapping unit-square labels over the time interval $[a, b]$.*

The theorem above remains true even when all points are stationary and *two* points are removed and *two* points are added in $[a, b]$. This is shown by the following proof.

Proof. PSPACE-hardness is provable by reduction from edge-to-edge NCL. In contrast to the proceeding in the proof of Theorem 15 we now move two blockers in two different edge gadgets corresponding to the two edges e_A and e_B. Let the blocker q in e_B move into the edge gadget at constant speed $v = 2\varepsilon/(b-a)$ and let the blocker q' in e_A move outward the edge gadget at speed $v' = 3\varepsilon/(b-a)$. Vertex v' is chosen slightly faster than v what ensures that e_A is constrained only during $[a, a+\Delta]$ and e_B only during $(b-\Delta, b]$ with $\Delta = (b-a)/3$. The blockers always move in the same direction (indicated by the arrows in Figure 9) if the edge gadgets are laid out suitably on the grid. Further claims can be showed equal to those of Theorem 15. ■

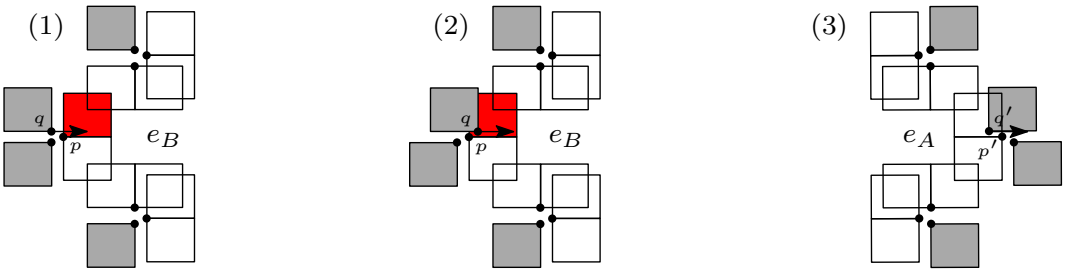


Figure 9 Reduction idea from configuration-to-edge NCL (1)/(2) and edge-to-edge NCL (3) to dynamic point labeling a set of moving points. (1) shows a part of the edge gadget of edge e_B at time a, (2) the modifed edge gadget at time b. (3) depicts the modified edge gadget for edge e_A at time a.

3.3 Predefinions for a Heuristic Algorithm for Dynamic Point Labeling

To get a dynamic point labeling $\mathcal{L}$ of a given dynamic point set P, we need to solve the dynamic point labeling problem (see definiton 6). $\mathcal{L}$ and the comprising static labelings $\mathcal{L}(t)$ should therefore comply with several quality criteria as described in the following list.

First, we require from the static labelings $\mathcal{L}(t)$:

1. The association between points and labels should be clear. *Solution*: Use a so called α-trailing model which is a subset of the 4S label model (see Figure 8 for details).
2. As many labels as possible should be as readable as possible. *Solution*: Labels should not overlap and be of significant size. Out of the optimization problems, we choose free-labeling maximization as the most promising approach for smooth dynamic labelings. (As already stated in the introductory section, size maximization however, would result in continously scaling labels which might be hard to read, number maximization makes labels appearing and disappearing whether now new points are added or existing points are removed).

In Addition, we require of the dynamic labeling $\mathcal{L}$:

3. The labels should not obscure the direction of movement of the points. *Solution*: Require that labels always "stay behind" their points as explained by Figure 10, the so called α-trailing model.
4. The labels should move slowly and continuously, where we measure the speed of a label relative to the point it labels. *Solution*: the question here is whether we should minimize the maximum label speed or the average label speed. It will become clear in Subsection 3.5.2 that we can fulfil both constraints at the same time (under some conditions).

This α-trailing model, a subset of the 4S model is now described in detail. Based on the point direction of movement (indicated by ray r_1) we draw a second (densely dashed) ray r_2 which goes from the center M of the label through the point. The requirement for the label position is that these two rays make an angle of at least $\alpha \in [0, \pi]$. Figure 10 for example shows an angle of $\alpha = \pi/2$ (indicated by the vertial densely dashed rays). Thus, the parameter α specifies how restrictive we are in terms of "behindness". The gray-shaded area in Figure 10 indicates the allowed label positions provide $\alpha = \pi/2$. Note, that due to the fact that this model is a subset of the 4S model, the point must stay on the border of the label. All possible point locations are indicated by the bold black subframe of the label.

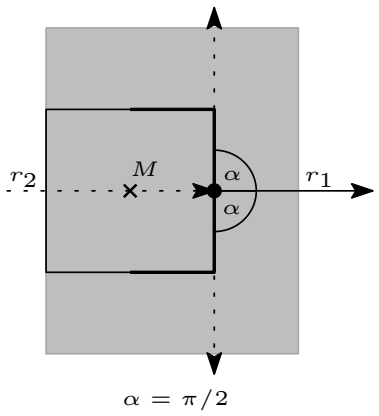


Figure 10 The α-trailing model which defines a variable "behindness" of labels and their corresponding points.

Imagine we would factor out the quality criteria (3) and (4) we require on the dynamic labeling $\mathcal{L}$. Consequently, we could run any fast static point labeling algorithm from scratch each and every time the display is refreshed to get $\mathcal{L}$. Unfortunately, confusing, potentially fast label movements which move contrary to their associated points would be the consequence.

3.4 Preliminary considerations on label speed, behindness, and freeness

In the last section we introduced (among other things) the behindness parameter α and required to minimize the maximum label speed and average label speed. Another possibility would be to limit the maximum label speed and minimizing α. On the one hand this would not prevent that the labeling algorithm always uses the maximum label speed. On the other hand we will see in the following theorem that the number of free labels can be even notably worse.

▶ **Theorem 17.** *Suppose we want to label a set of n moving points using a 1S or α-trailing model. If we then impose a maximum label speed, the best possible dynamic labeling can be substancially worse than if we did not limit label speed: the integral of free labels over time can be $\Theta(n)$ times worse, and the minimum number of free labels at any time can go from non-zero to zero.*

Proof. Figure 11 shows straightline trajectories of two $k = n/2$ (n is assumed to be odd) element sets $P = \{p_1, \ldots, p_k\}$ and $Q = \{q_1, \ldots, q_k\}$ of moving points. The points move horizontal with the same unit speed (i.e., $x(p_i(t)) = x(q_i(t))$ holds for all i and t) while trailing 1×1 labels behind them. The trajectory slopes in the time interval $[i-1, i]$ are chosen so that the lowest (highest) point is p_i (q_i) for all $i \in \{1, \ldots, k\}$. Overall timesteps $t \in [0, k]$ every static labeling of $(P \cup Q)(t)$ can have at least two free labels. This can be achieved by placing on label above the highest point, the other one below the lowest point. All other labels are between those two and overlap each other. Let $\mathcal{L}$ be a speed-restricted dynamic labeling of $P \cup Q$ with a maximum label speed of $1/(2k)$. If $\mathcal{L}$ assigns the point p_i a free label in the interval $[i-1, i]$ then a point p_j with $j \neq i$ will never have a free label. The reason is that at a speed of $1/(2k)$ p_i can move by $1/2$ at most in the interval $[0, k]$ which is not enough time to move it out of the way (assuming 1×1 labels). The same applies to the points of Q. Thus having a maximum label speed of $1/(2k)$ the integral of free labels over time cannot be larger than two in any dynamic point labeling $P \cup Q$.

Consider the maximum label speed is set to $1/\varepsilon$. As a result, a free label can be placed to p_i while $[i-1, i-\varepsilon]$ and to q_i while $[i-1+\varepsilon, i]$, there is everytime at least one free label and the integral of free labels over time is $2k(1-\varepsilon)$.

3.5 A Heuristic Algorithm for Dynamic Point Labeling

As we have seen in Section 3.2 each of the maximization problems of definition 2, definition 3 and definition 4 are PSPACE-hard for a dynamic point set. Thus, an approximation algorithm is desired in practise. Such an algorithm named InterpolativeLabeling (see algorithm 9) is presented in this section. To satisfy the four quality criteria mentioned in Section 3.3, we make use of the following heuristic approach. InterpolativeLabeling consists of two single algorithms which are executed successively at every timestep. Between two arbitrary timesteps t and $t + \Delta t$ InterpolativeLabeling first computes the two respective static labelings $\mathcal{L}(t)$ and $\mathcal{L}(t_{next})$ so that the number of free labels are maximized. Then the interpolation between this labelings attempts to minimize the movement speed of the labels. The input parameters of the algorithm are a dynamic points set P to be labeled in a given time range $t_{next} \in (0, t_{max}]$. $\mathcal{L}(t, t')$ denotes a part of the dynamic labeling $\mathcal{L}$ between two consecutive times t and t'.

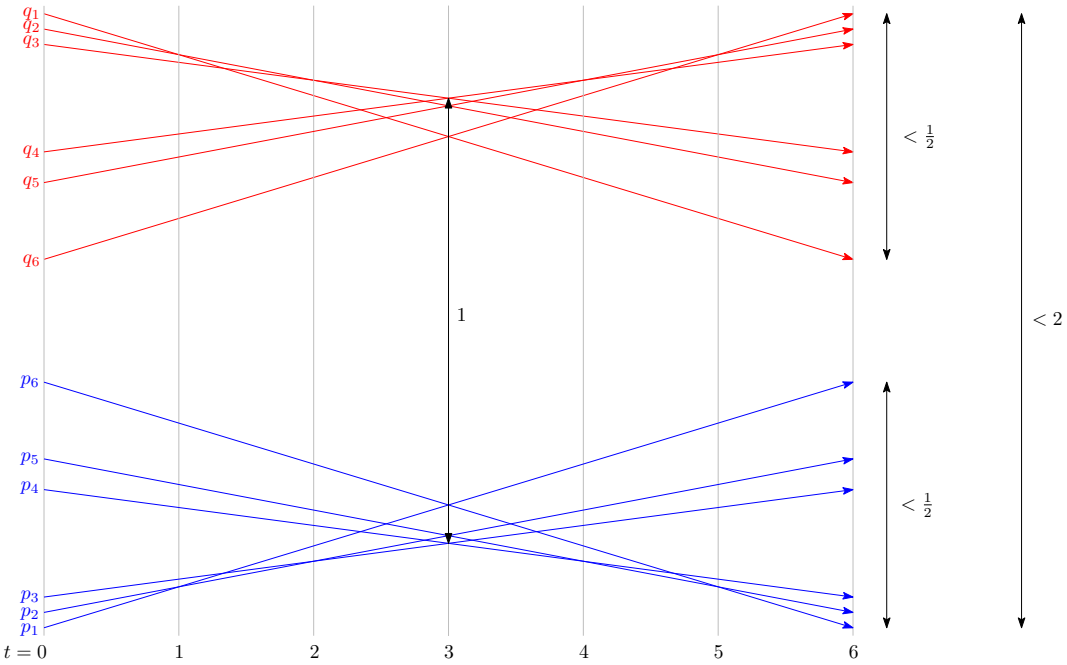


Figure 11 This example shows that the best dynamic labeling with α-trailing labels where the label speeds are restricted can be substantially worse compared to the best dynamic labeling having no speed restrictions. This becomes clear in the situation of points moving along the trajectories (straight lines) at unit speed so that they are always on a common vertical line.

```
1 InterpolativeLabeling(P, t_max) begin
2     L ← ∅, t ← 0
3     L(t) ← StaticLabeling(P, t)
4     while t < t_max do
5         t_next ← min(t + Δt, t_max)
6         L(t_next) ← StaticLabeling(P, t_next)
7         L(t, t_next) ← Interpolate(L(t), t, L(t_next), t_next)
8         t ← t_next
9     return L
```

Algorithm 4 : A heuristic algorithm for free-label maximization.

3.5.1 Static point labeling algorithm

The name of the algorithm we use for StaticLabeling in InterpolativeLabeling is FourGreedySweeps. FourGreedySweeps is an approximation algorithm for the free-label maximization problem with the following properties. It is an $\mathcal{O}(n \log n)$-time constant-factor approximation algorithm to free-label maximization of 1×1 labels, for all of the fixed-position and slider models. The approximation ratio is 6 for the 1S model, 7 for the 2-position model, 22 for the 4P and 2S models, and 32 for the 4S model. Note, however that we use the α-trailing model (a subset of the 4S model) so that the algorithm can be transferred (in contrast to the technique used to prove the aforementioned approximation ratio).

3.5.2 Interpolation algorithm

The INTERPOLATE algorithm comes with a linear time complexity as it basically computes shortest paths in rectilinear polygons using the so called funnel algorithm. Theorem 19 takes a precise look at the running time of INTERPOLATE. Hereafter is explained wherein, how and why using shortest paths.

Assuming the initial situation with given t_1 and t_2 as two moments in time and their corresponding static labelings $\mathcal{L}(t_1)$ and $\mathcal{L}(t_2)$ over all points p in the dynamic point set P. We now want a dynamic labeling $\mathcal{L}(t_1, t_2)$ which equals $\mathcal{L}(t_1)$ and $\mathcal{L}(t_2)$ respectively at times t_1 and t_2 for all moving points $p \in P$. Points $p \in P$ which exist, that is, $life(p) \cap [t_1, t_2] \neq \emptyset$ are of interest only. From now on we track such a point p at its lifetime $[t'_1, t'_2] = life(p) \cap [t_1, t_2]$ in the given interval of time. Figure 12 (1) shows a possible example of a moving point p and its trajectory (gray arrows). At the beginning t'_1 and in end t'_2 of the point trajectory we have fixed (i.e., static) label positions (gray striped rectangles). The half rectangle frames indicate valid label positions in the α-trailing model ($\alpha = \pi/2$ here, see Figure 10 for more details). A label's center has always to be on this frame. This ensures label centers are "behind" their points. Consequently, if we let p be always at the origin of the coordinate system then the frames circulate around them, depending on p's changes of direction.

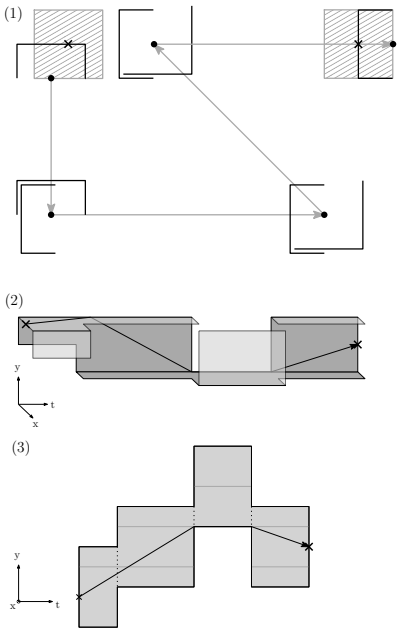


Figure 12 (1) shows a piecewise linear point trajectory (indicated by the gray lines with dots as vertices). The labels are placed at the beginning and end of the trajectory (gray hatched rectangles), crosses mark the label centers. (2) depicts the configuration space $\mathcal{C}$ of valid label positions regards to $\alpha = \pi/2$ and the label trajectory of (1). (3) shows the unfolded configuration space $\mathcal{C}$ of (2) as a rectilinear polygon R. The line depicts a shortest path from t'_1 to t'_2 through it.

Now consider the allowed positions of the labels to be segments cut out of a 3-dimensional tube as shown in Figure 12 (2) for the point trajectory in Figure 12 (1) with given $\alpha = \pi/2$. This is what we call a *configuration space* $\mathcal{C}$ for a point p over the interval $[t'_1, t'_2]$. All possible label trajectories for p are time-monotone paths through the configuration space $\mathcal{C}$. To find a preferred path, we can unfold $\mathcal{C}$ into a rectilinear polygone R as shown in Figure 12 (3). Why the preferred path is the shortest path here will be explained and proven later on. R consists of an union of $k - 1$ closed, axis-aligned rectangles if p consists of a trajectory of k vertices. The vertical intersection say at time t between two neighbored (indicated as dashed lines in Figure 12 (3)) is called a *portal* at t or $\Psi(t)$. Moreover, we include the vertical lines at t'_1 and t'_2 (which are label positions respecting $\mathcal{L}(t_1)$ and $\mathcal{L}(t_2)$) to the set of portals. Thus, a label trajectory for p corresponds to a time-monotone path through R from portal $\Psi(t'_1)$ to portal $\Psi(t'_2)$. The following Lemma 18 picks up the observation on the preferred path through R raised before.

▶ **Lemma 18.** *A shortest path from $\Psi(t'_1)$ to $\Psi(t'_2)$ through R obtains a label trajectory minimizing the average speed (i) as well as the maximum speed (ii) of p's label relative to p.*

Proof. (*i*) A time-monotone shortest path π can be splitted up in two components. $T(\pi)$

denotes the summed length covered in t-axis direction, $Y(\pi)$ the same but in y-axis direction. Every arbitrary path π' from $\Psi(t'_1)$ to $\Psi(t'_2)$ traverses the same distance on the t-axis, i.e., $T(\pi') = T(\pi) = t'_2 - t'_1$. A shortest path π has the property that it minimizes the distance traveled on the y-axis so that $Y(\pi) \leq T(\pi')$ holds for all paths π'. As a result, the average speed $Y(\pi)/T(\pi)$ of p's label is minimized by minimizing $Y(\pi)$.

(*ii*) Let ab be the link in the shortest path π that has maximum absolute slope overall neighbored portals in R. Figure 13 shows such a link ab with negative slope (the following arguments apply on positive slopes analogously). It should be noted that the steeper a link in R is, the faster the label moves. Consequently, it suffices to show that π minimizes the maximum speed $Y(ab)/T(ab)$ of p's label, that means there is no other path next to the shortest path which lowers this bound. The point t' (t'') on the t-axis is where the links starts in a (b). Imagine π makes a left turn at a as indicated by the dashed-dotted link between a and b. This cannot happen because it would increase the steepness of the preceding link. Thus π must do a right turn at a (or start at a). In case of a right turn at a, a has to be a bottom point or otherwise we can shorten π by lowering a to the bottom of the portal $\Psi(t')$. Since a lies above b, this makes ab the shortest path not only from a to b, but from $\Psi(t')$ to b as well. If π instead starts at a then $t'_1 = t'$ and the same condition holds. With same arguments, ab must be the shortest path from a to $\Psi(t'')$ due to symmetry reasons. As a consequence, ab is the shortest path from $\Psi(t')$ to $\Psi(t'')$. Thus any time-monotone path from $\Psi(t')$ to $\Psi(t'')$ must be at least as steep as ab somewhere between t' and t''. Hence the claim follows.

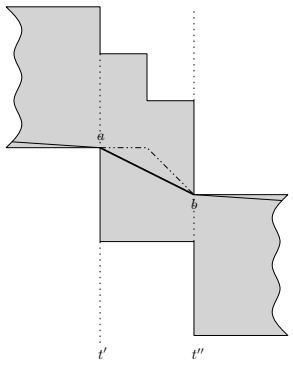


Figure 13 A path through R making a right turn not at the lowest of a portal (as indicated by dashed-dotted line) can be shortened. The shortest path π has right turns at the lowest point of portals only (and simmilary left turns at the highest point of portals). As a consequence, the steepest connection ab in the shortest path π must be the least steepest over all connections between two given portals at t' and t''.

▶ **Theorem 19.** *Suppose we are given a dynamic point set P with n points, along with static labelings $\mathcal{L}(t_1)$ and $\mathcal{L}(t_2)$ of it at times t_1 and t_2. For each p in P let k_p be the number of vertices in its polygonal trajectory between times t_1 and t_2. In $\mathcal{O}(\sum_{p\in P} k_p)$ time and $\mathcal{O}(\max\{k_p \mid p \in P\})$ space we can then compute a dynamic labeling $\mathcal{L}(t_1, t_2)$ respecting $\mathcal{L}(t_1)$ and $\mathcal{L}(t_2)$, that for each point minimizes both its average and its maximum label speed.*

Proof. The configuration space R is a simple, rectilinear polygon with $\mathcal{O}(k_p)$ vertices. We already proved by Lemma 18 that $\mathcal{L}(t_1, t_2)$ can be achieved by the shortest path for all points $p \in P$ which are alive (i.e., $\mathit{life}(p) \cap [t_1, t_2] = [t'_1, t'_2] \neq \emptyset$) through their configuration spaces R. We now pick one arbitrary point $p \in P$. In case of $\mathit{life}(p) \supseteq [t_1, t_2]$ we seek a shortest point-to-point path in R, in case of $\mathit{life}(p) \subset [t_1, t_2]$ a shortest edge-to-edge path in R or a shortest point-to-edge path in R otherwise. To compute a shortest path inside a rectilinear polygon use the algorithm presented by [12] combined with the results of [5]. This algorithm comes with a time complexity of $\mathcal{O}(n)$. ■

3.5.3 Hourglass Trimming

We know how to minimize both the average and the maximum label speed between two static labelings as described in Subsection 3.5.1. It is not excluded thereby that high label speeds

can still occur. This happens if poor static labeling choices are made. Consider a chosen label position which lies behind the point at time t and suddenly lies in front at time $t + \varepsilon$. This problem can be solved by narrowing the ranges of label positions the static labeling algorithm is allowed to use.

Thus we transfer the unfolded configuration space R into hourglasses H, exemplarily shown in Figure 14. $H(t', t'')$ emerges out of the shortest paths in R between the the vertical segments at t' and t'', more precisely $\pi(a, c)$ (red) and $\pi(b, d)$ (blue). Out of the shortest paths $\pi(\cdot)$ we call the upper one the hourglass's *upper chain* and the other its *lower chain*.

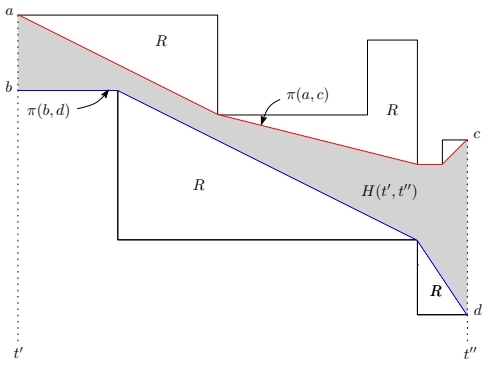


Figure 14 Hourglass construction by taking the interior between shortest paths connecting the two highest points (a and c) and the two lowest points (b and d) of two neighbored portals (t' and t'') in a configuration space.

On close inspection of hourclasses we can distinguish between two different kinds of them as depicted in Figure 15. $H(t^-, t)$ is called open since the upper and lower chain does not have a common intersection. In contrast to open hourglasses they can be also closed like $H(t, t^+)$ is. Closed hourglasses consists of two so called *funnels* connected by the common intersection named *string*.

Shortest paths between chosen points in time $t^- = \max(\mathit{birth}(p), t - \Delta t)$, $t \in \mathit{life}(p)$ and $t^+ = \min(t + \Delta t, \mathit{death}(p))$ always reside inside the corresponding hourglass. If $H(t^-, t)$ or $H(t, t^+)$ has steep edges as in Figure 15, fast label speeds may be the consequence. By trimming (i.e., narrowing the range of valid label positions) the hourglasses steep edges appropriately we can possibly reduce rapid label movements. Let v be the parameter that defines the desired label speed.

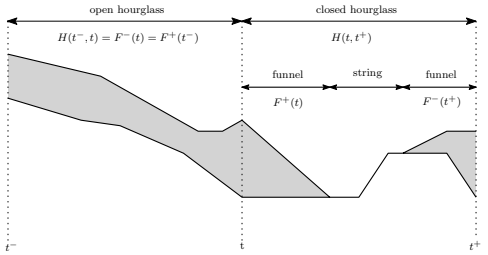


Figure 15 An example of the two types of hourglasses, the left one is open since the upper and lower chain does not intersect, the right one is closed and therefore consists of two funnels connected by a string.

The trimmed hourglass $H'(t^-, t)$ is created as follows: translate a line with slope $+v$ ($-v$) down (up) from positive (negative) infinity along the y-axis until it becomes tangent to one of the two chains defining $F^+(t^-)$ at t^-. Repeat the same procedure on the two chains defining F^{-t} at t. Each comprising two lines define a narrower interval on the vertical line at t^-, t and t^+ respectivly (indicated by curly braces in Figure 16 (1)).

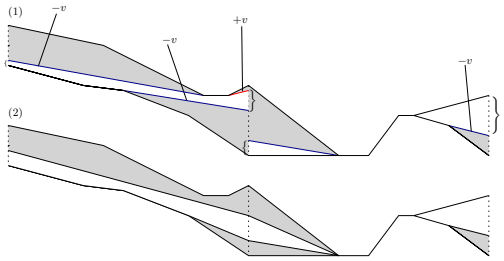


Figure 16 Hourglass trimming of the hourglasses of Figure 15. (1) shows the resulting hourglasses enclosed in curly parentheses but being disjoint. They are modified so that they connect each other as shown in (2).

As a result, the slowest label interpolation in the trimmed hourglass cannot be faster than v, except in two cases. In the first case $H'(\cdot)$ is closed and it's string has an edge steeper that v. Then the label speed cannot be reduced. In the second case $H'(\cdot)$ is open and further trimmable as a bi-tangent of its upper and lower chain is steeper than v. A further trimmed hourglass might be closed resulting in the situation of the first case. This is avoided by not trimming any further in such cases.

From example Figure 16 (1) we may infer that the intersections at t of the trimmed hourglasses $H'(t^-, t)$ and $H'(t, t^+)$ are disjoint. In other words, the curly parentheses does not overlap at a given point in time. If they overlap, we can simply take their intersection. Otherwise, we proceed as presented in Figure 16 (2) and take the interval in between those two. This can possibly lead to weaker trimmings again.

3.6 Experimental Results

The InterpolativeLabeling algorithm was implemented in C++[1] and tested on a desktop machine having an Intel Q6600 2.40GHz and 3GB of RAM. The computation of Interpolation consumes 0.4 resp. 2.2 ms between two 100 resp. 1,000 points per set. FourGreedySweeps needs 5 resp. 189 ms for 100 resp. 1,000 points. The complexity of the FourGreedySweeps implemention is $\mathcal{O}(n^2)$ it turns out that this is fast enough since labeling 1,000 points cannot practially displayed on most of the screens anyway making a $\mathcal{O}(n \log n)$ implementation unneccessary.

3.6.1 Experimental setup

To evaluate the quality of our heuristic algorithm, we chose the street network of the Dutch city of Eindhoven. Points moved on five polygonal paths through this network. The arrival times of the points on each route are produced by a Poisson process with parameter 5 s. The produced number of different random problem instances were 100, as label model we chose the α-trailing model with parameter $\alpha = \pi/2$. The trajectories of the points are chosen as polygonals because it is simpler (curved trajectories making the configuration space a splinegon and therefore requires a more sophisticated shortest path implementation but this should work as well). The time interval for the points to be labeled dynamically was set to $[t = 0, t = t_{max} = 60\,s]$. To determine the quality of the resulting dynamic labeling $\mathcal{L}$ continuously, the sample rate is set to 1/25.6 per second (as it is roughly the same framerate we can see in movies having 24 frames per second). Due to the fact that 1/24 (= 1/120) has no finite binary floating point representation we used 1/25.6 (= 5/128) instead.

The amount each label moved relative to its point and the number of free and non-free labels were recorded for every sample since its preceding sample. Furthermore, the area

[1] The implementation is available at `http://dirkgerrits.com/programming/flying-labels/`.

covered by all labels minus the area covered by more than one label (i.e., the overlapping parts) is recorded as well and called the *free label area*. This records were made for 30 different timesteps beginning with a smallest $\Delta t = 1/25.6\ s \approx 0$ up to the largest with $\Delta t = 61\ s$. The hypothesis to be experimentally tested was the following: small timesteps result in maximum label freeness without regard for label speed and large timesteps minimize the label speed without regard for label freeness. The label speed was set to $v = 35\ px/s$ resp. $v = 10\ px/s$ constantly without resp. with hourglass trimming.

3.6.2 Experimental Results

The experimental results are depicted in Figure 17 to ensure a good practical capability of InterpolativeLabeling. We observe a sharp decline in label speeds until around $\Delta t = 2\ s$ while the label freeness shows a moderate decrease. The label speed is close to zero at around $\Delta t = 5\ s$ and the label freeness decreases constantly slow for higher Δt. Consequently, a timestep Δt of around 2 seconds should generate most promising labelings. Hourglass trimming has proven to be the alternative to choosing a larger timestep as it has the same effect as increasing the timestep so that the label speed as well as the label freeness both decrease. In short, we obtain a trade-off between the number of free labels and the label speeds, whereas if we accept particulary faster label speeds we can achieve a large increase in the number of free labels.

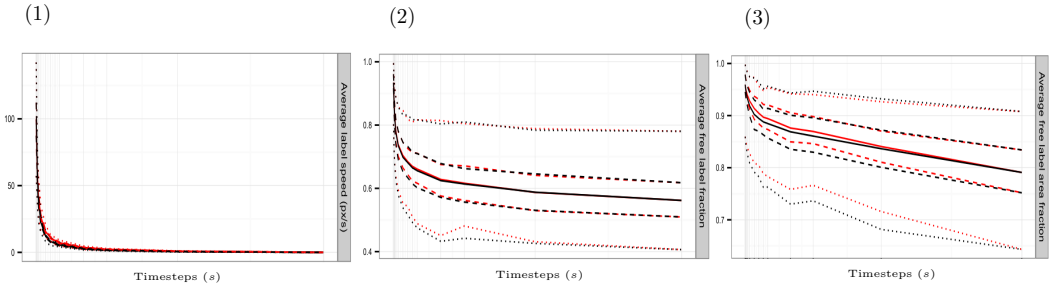


Figure 17 (1) - (3) plots how the label speed and label freeness is influenced by choosing different timesteps from Δt up to $t > 60\ s$. (1) Shows the label speed (in px/s) averaged over all moving points and over the whole interval of time $[0, t_{max}]$. (2) Depics the label speed for the faction of labels that are completely free. (3) displays (area covered by the union of the labels - area covered by more than one label) / (total area that would be spanned by the labels if they did not overlap). Note that (1) - (3) only show the minimum and the maximum (dotted lines), 25% and 75% quantiles (dashed lines) and the mean (solid line) over all 100 test instances. The black resp. red lines indicate deactivated resp. activated hourglass trimming.

3.7 Future Work

Unfortunately, algorithms for dynamic point labeling with proven approximation ratios are still unknown. In the author's opinion, the static labeling algorithm can be improved so that the high number of free labels does not decrease that fast when interpolating. Additional experiments such as how the quality of the labelings depend on the density of points in the set might be promising. Label freeness can be more restricted by calling a label free iff it remains so for i.e., more than one second. Due to the fact that the human eye is more focused on the center of view, one can penalize overlapping here more than at the periphery [14].

Last but not least, the linchpin of the algorithm and further improvements is that users evaluate the labeling as comfortable. Thus user studies are required.

References

1 Pankaj K. Agarwal, Marc J. van Kreveld, and Subhash Suri. Label placement by maximum independent set in rectangles. *Comput. Geom.*, 11(3-4):209–218, 1998.

2 Kenneth R. Allendoerfer, Joseph J. Galushka, and Richard H. Mogford. Display system replacement baseline reearch report. Technical Report DOT/FAA/CT-TN00/31, William J. Hughes Technical Center, Atlantic City International Airport (NJ), 2000.

3 Therese C. Biedl and Goos Kant. A better heuristic for orthogonal graph drawings. *Comput. Geom.*, 9(3):159–180, 1998.

4 Parinya Chalermsook and Julia Chuzhoy. Maximum independent set of rectangles. In Claire Mathieu, editor, *SODA*, pages 892–901. SIAM, 2009.

5 Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6:485–524, 1991.

6 C.Iturriaga. *Map Labeling Problems.* PhD thesis, University of Waterloo, Canada, 1999.

7 Michael Formann and Frank Wagner. A packing problem with applications to lettering of maps. In *Symposium on Computational Geometry*, pages 281–288, 1991.

8 Robert J. Fowler, Mike Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Inf. Process. Lett.*, 12(3):133–137, 1981.

9 Dirk H.P. Gerrits. *Pushing and Pulling - Computing push plans for disk-shaped robots, and dynamic labelings for moving points.* PhD thesis, Eindhoven University of Technology, aug 2013.

10 Robert A. Hearn and Erik D. Demaine. Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *CoRR*, cs.CC/0205005, 2002.

11 Hiroshi Imai and Takao Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *J. Algorithms*, 4(4):310–323, 1983.

12 Der-Tsai Lee and Franco P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14(3):393–410, 1984.

13 Joe Marks and Stuart Shieber. The computational complexity of cartographic label placement. Technical Report TR-05-91, Harvard University Center for Research in Computing Technology, Cambridge, Massachusetts, December 1991.

14 Stephen D. Peterson, Magnus Axholt, Matthew D. Cooper, and Stephen R. Ellis. Detection thresholds for label motion in visually cluttered displays. In Benjamin Lok, Gudrun Klinker, and Ryohei Nakatsu, editors, *VR*, pages 203–206. IEEE, 2010.

15 Marc J. van Kreveld, Tycho Strijk, and Alexander Wolff. Point labeling with sliding labels. *Comput. Geom.*, 13(1):21–47, 1999.

4 Flussbasiertes Zählen von Triangulierungen

Vitali Henne

Zusammenfassung

Diese Ausarbeitung beschäftigt sich mit dem von V. Alvarez und R. Seidel [5] vorgestelltem Algorithmus zur Zählung von Triangulierungen einer in allgemeiner Lage liegender Punktmenge. Der flussbasierte Algorithmus hat eine Laufzeit von $O(n^2 2^n)$ und einen Speicherverbrauch von $O(n2^n)$. Dadurch ist das hier beschriebene Verfahren das Erste, welches asymptotisch schneller ist als die zurzeit beste bekannte untere Schranke $\Omega(2.43^n)$ [2] für die Anzahl der Triangulierungen einer solchen Punktmenge.

4.1 Einleitung

Sei $S = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$ eine in allgemeiner Lage liegende Punktmenge im zweidimensionalem Raum. Allgemeine Lage bedeutet hierbei, dass keine drei Punkte kollinear sind und keine zwei Punkte die gleiche y-Koordinaten haben. Diese Eigenschaft erlaubt es Aussagen über S zu treffen, ohne die Menge einzuschränken. So kann man die Punkte in S anhand ihrer x-Koordinaten total ordnen, d.h. wir indizieren S so, dass p_1 bei uns den linkesten und p_n den rechtesten Punkt bezeichnet. Falls die Punktmenge S nicht in allgemeiner Lage vorliegt, so kann man durch infinitesimale Verschiebung der Punkte die geforderte Eigenschaft erreichen. Wir wollen die Anzahl aller möglichen planaren Graphen eines bestimmten Typs bestimmen, die als Knoten Punkte aus S haben.

Sei $PG(S)$ die Menge aller geradlinigen, kreuzungsfrei eingebetteten Graphen auf S. Graphen werden hierbei geometrisch eingebettet, d.h. die Position der Knoten ist fix. Die Menge $PG(S)$ kann man weiter unterteilen, indem man spezielle Familien von Graphen betrachtet:

- perfekte Matchings (1-reguläre Graphen)
- Spannzykel (2-reguläre Graphen mit einer Zusammenhangskomponente)
- Spannbäume (Graphen mit einer Zusammenhangskomponente und minimaler Anzahl Kanten)
- Triangulierungen (Graphen mit einer maximalen Anzahl Kanten)

Diese Liste ist keineswegs vollständig, reicht aber aus, um einen Einblick in die Art der betrachteten Graphfamilien zu bekommen.

Für alle oben genannten Klassen wurden untere und obere Schranken ihrer Kardinalität bewiesen. Diese sind von der Form: Für jede in allgemeiner Lage liegende Punktmenge S ist die Anzahl der Graphen in der Klasse $\Omega(c_1^n)$ und $O(c_2^n)$. Die beiden Konstanten c_1 und c_2 sind meistens sehr weit auseinander. Tabelle 1 zeigt die aktuell besten Schranken für die oben genannten Graphfamilien.

Uns interessiert speziell die Klasse der maximalen Graphen in $PG(S)$. Maximal bedeutet hierbei, dass man keine geradlinigen Kanten zu dem Graphen hinzufügen kann, ohne die Planarität zu verletzen. Diese maximale Anzahl von Kanten ist dabei für ein gegebenes S eindeutig, was folgendes Lemma zeigt.

▶ **Lemma 1.** *Sei $G = (V, E)$ ein geradlinig planar eingebetteter Graph. Sei h die Anzahl der Knoten auf der konvexen Hülle, $n = |V|$, $m = |E|$ und f die Anzahl der Facetten des Graphen. Dann hat G maximal $3n - 6 - (h - 3)$ Kanten und maximal $2n - h - 1$ Facetten.*

Klasse	untere Schranke	obere Schranke
$\lvert PG(S)\rvert$	$\Omega(41.18^n)$ [1]	$O(187.53^n)$ [15]
perfekte Matchings	$\Omega(3^n)$ [9]	$O(10.05^n)$ [18]
Spannzykel	$\Omega(4.64^n)$ [9]	$O(54.543^n)$ [17]
Spannbäume	$\Omega(12.52^n)$ [12]	$O(141.07^n)$ [11]
Triangulierungen	$\Omega(2.4317^n)$ [16]	$O(30^n)$ [14]

Tabelle 1 Untere und obere Schranken für verschiedene Klassen von Graphen in $PG(S)$

Beweis. Idee: Erweitere den Graphen $G = (V, E)$ zu einem planaren Graphen $G' = (V, E')$, bei dem alle Facetten Dreiecke sind. Dieser ist nicht notwendigerweise geradlinig planar.

Sei x die Anzahl der Kanten, die eingefügt werden müssen, damit alle inneren Facetten Dreiecke sind. Falls $h \geq 3$, so ist die äußere Facette ein h-Eck. Wir müssen $h-3$ Kanten einfügen, damit die äußere Facette ein Dreieck wird. G' hat folglich $h-3+x$ mehr Kanten als G. Für jede eingefügte Kante erhöht sich die Anzahl der Facetten um 1, G' hat also $h-3+x$ mehr Facetten als G.

Da G' ein planarer Graph ist, können wir die Euler-Charakteristik anwenden. Nach Konstruktion ist G' außerdem maximal planar. Für solche Graphen gilt $|E'| = 3|V| - 6$.

$$
\begin{aligned}
& n-(m+(h-3+x))+(f+(h-3+x)) = 2 \\
\Leftrightarrow\quad & n-(3n-6)+(f+(h-3+x)) = 2 \qquad \Leftrightarrow \quad -2n+3+f+h+x = 2 \\
\Leftrightarrow\quad & f = 2n-h-1-x
\end{aligned}
$$

G besitzt folglich $2n-h-1-x$ Facetten. Wenn man diese Anzahl in die Euler-Charakteristik von G einsetzt, erhält man:

$$
\begin{aligned}
& n-m+f = 2 \\
\Leftrightarrow\quad & n-m+2n-h-1-x = 2 \qquad \Leftrightarrow \quad 3n-m-h = 3+x \\
\Leftrightarrow\quad & m = 3n-h-3-x \qquad \Leftrightarrow \quad m = 3n-6-(h-3)-x
\end{aligned}
$$

Falls $h < 3$ ist, so folgt aus der geradlinigen Planarität von G, dass der Graph nur aus einem Knoten ($h = 1$) oder zwei Knoten und einer Kante ($h = 2$) bestehen kann. In beiden Fällen stimmt die Behauptung. ◀

Einen Graph $T = (S, E) \in PG(S)$ mit $|E| = 3n-6-(h-3)$ Kanten bezeichnen wir als eine *Triangulierung* T von S, da alle Facetten (bis auf die äußere) Dreiecke sind. Uns interessiert die Anzahl $tr(S)$ solcher Graphen in $PG(S)$.

4.1.1 Historisches / Entwicklung

Die Bestimmung von $tr(S)$ ist ein lange untersuchtes Problem, für welches es viele Lösungsvorschläge gibt. Die ersten Algorithmen [6, 7], welche $tr(S)$ bestimmt haben, basieren alle auf dem Ansatz, dass man die verschiedenen Triangulierungen möglichst effizient enumeriert. Der uns beste bekannte Algorithmus mit diesem Ansatz ist von Bespamyatnikh [7] und hat als Laufzeit $O(tr(S) \cdot \log(\log(|S|)))$.

Ray und Seidel [13] verwenden einen auf dynamischer Programmierung basierenden Ansatz mit Laufzeit $O(n^3 9^n)$, während Alvarez et al. [3] einen sweep-basierten Ansatz vorstellen. Dieser verwendet Triangulierungspfade und hat bisher die beste asymptotische Laufzeit mit $O(3.1414^n)$.

Alvarez et al. [4] approximieren $tr(S)$. Ihr Ansatz erreicht eine Approximationsgüte von $2^{O(n^{\frac{3}{4}}\sqrt{\log(n)})}$ mit einer Laufzeit von $2^{O(\sqrt{n}\log(n))}$. Ihr Algorithmus ist der Erste, welcher eine sub-exponentielle Approximationsgüte bei einer sub-exponentiellen Laufzeit erreicht.

Der hier vorgestellte Algorithmus hat eine Laufzeit von $O(n^2 2^n)$ und ist damit der Erste, der asymptotisch schneller ist, als die bis dahin beste bekannte untere Schranke $\Omega(2.4317^n)$ [16] für $tr(S)$.

4.2 Grundlagen

Dieser Abschnitt beschäftigt sich mit den notwendigen Definitionen und wichtigen Eigenschaften von Triangulierungen, die als Grundlage des Algorithmus dienen. Der Ansatz benutzt Kantenzüge in Triangulierungen, sogenannte monotone Ketten. Diese werden sweep-artig nach oben hin erweitert. Wir konstruieren einen azyklischen Graphen, in dem Knoten solchen Kantenzügen entsprechen. Pfade von einer ausgezeichneten Quelle zu einer ausgezeichneten Senke in diesem Graphen entsprechen Triangulierungen. Der eigentliche Algorithmus zählt nach Konstruktion des Graphen alle Pfade von der Quelle zur Senke und damit auch die Triangulierungen.

4.2.1 Monotone Ketten

Der verwendete Ansatz abstrahiert über die eigentlichen Triangulierungen von S. Dies wird mithilfe von monotonen Ketten erreicht, für die in Abschnitt 4.2.5 eine Korrespondenz zu Triangulierungen bewiesen wird.

▶ **Definition 2** (Monotone Kette für S)**.** Eine monotone Kette C für $S = \{p_1, \ldots, p_n\}$ ist ein Streckenzug der

- den linkesten Punkt p_1 mit dem rechtesten Punkt p_n verbindet,
- als Eckpunkte nur Knoten aus S enthält,
- jede vertikale Gerade maximal einmal schneidet.

Als alternative Definition kann man sich monotone Ketten als Teilmengen $S' \subset S$ vorstellen, wobei die Punkte die Ordnung und die Indizes aus S beibehalten. Dabei besteht zwischen je zwei aufeinanderfolgenden Punkten in S' eine Kante. Die oben genannten Bedingungen lauten umformuliert:

- $p_1 \in S'$ und $p_n \in S'$,
- $\forall p_i \in S' : p_i \in S$,
- für zwei aufeinanderfolgende Punkte p_i, p_j in S' gilt: $i < j$.

Monotone Ketten lassen sich für beliebige, in allgemeiner Lage liegende Punktmengen definieren. Da wir uns für Triangulierungen von Punktmengen interessieren, schränken wir die allgemeinen monotonen Ketten ein:

▶ **Definition 3** (Monotone Ketten für eine Triangulierung T von S)**.** Eine monotone Kette C für eine Triangulierung T von S ist eine monotone Kette, die nur Kanten der Triangulierung T als Streckensegmente beinhaltet.

4.2.2 Erweiterungen von monotonen Ketten

Monotone Ketten einer Triangulierung lassen sich zu anderen monotonen Ketten derselben Triangulierung erweitern. Diese Erweiterung definieren wir anhand von erweiternden Dreiecken:

▶ **Definition 4** (Erweiterndes Dreieck)**.** Sei C eine beliebige monotone Kette einer Triangulierung T von S und $\Delta_T(C)$ die Menge der Dreiecke von T, die unter C liegen. Ein erweiterndes Dreieck t in T für C ist ein Dreieck, welches

- über C liegt, also $t \notin \Delta_T(C)$
- $\Delta_T(C) \cup \{t\} = \Delta_T(C')$ für eine monotone Kette C' von T.

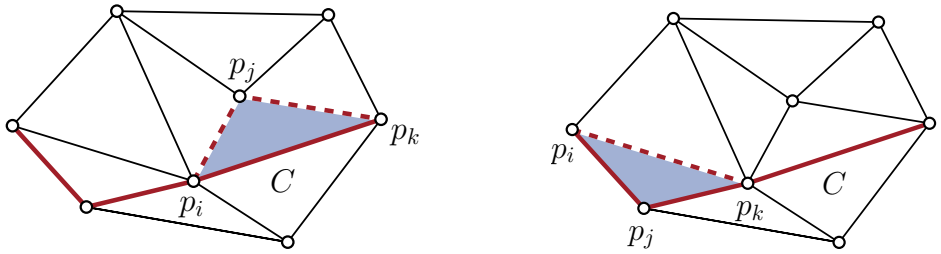
Sei t ein erweiterndes Dreieck mit den Eckpunkten p_i, p_j, p_k mit $i < j < k$. Es gibt zwei Klassen von erweiternden Dreiecken:

- p_j liegt über dem Segment $\overline{p_ip_k}$. In diesem Fall teilt sich das Dreieck eine Kante mit C (Abbildung 18a).
- p_j liegt unter dem Segment $\overline{p_ip_k}$. In diesem Fall teilt sich das Dreieck zwei Kanten mit C (Abbildung 18b).

Besitzt eine monotone Kette C ein erweiterndes Dreieck t, so lässt C sich zu einer monotonen Kette C' erweitern.

Falls t eine Kante mit C teilt (p_j liegt über dem Segment $\overline{p_ip_k}$), so erhält man C' indem man die Kante $\overline{p_ip_k}$ durch die zwei Kanten $\overline{p_ip_j}$ und $\overline{p_jp_k}$ ersetzt. Im anderen Fall teilt t zwei Kanten mit C (p_j liegt unter dem Segment $\overline{p_ip_k}$) und man erhält C' durch Ersetzung der zwei Kanten $\overline{p_ip_j}$ und $\overline{p_jp_k}$ mit der Kante $\overline{p_ip_k}$. Wir werden als Kurzschreibweise für solche Ersetzungen folgende Notation verwenden:

- $C' = C - \overline{p_ip_k} + \overline{p_ip_j} + \overline{p_jp_k}$ (im ersten Fall).
- $C' = C - \overline{p_ip_j} - \overline{p_jp_k} + \overline{p_ip_k}$ (im zweiten Fall).



(a) Eine geteilte Kante **(b)** Zwei geteilte Kanten

Abbildung 18 Die zwei Arten erweiternder Dreiecke

4.2.3 Kriterium für die Erweiterbarkeit von monotonen Ketten

Nach Definition 4 lassen sich monotone Ketten einer Triangulierung erweitern. Allerdings besitzen nicht alle monotonen Ketten eine Erweiterung. Das folgende Lemma liefert eine hinreichende Bedingung für die Erweiterbarkeit monotoner Ketten.

▶ **Lemma 5.** *Sei T eine Triangulierung von S. Für **jede** monotone Kette C in T existiert ein erweiterndes Dreieck, es sei denn C ist die obere konvexe Hülle.*

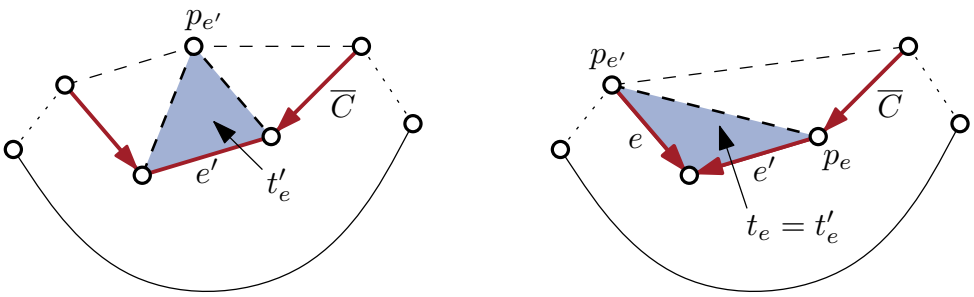
Beweis. Sei C eine beliebige monotone Kette in T und $\overline{C}$ eine maximale Teilkette von C, die keine Kanten der oberen konvexen Hülle enthält.

Für jede Kante $e = (p_i, p_j)$ von $\overline{C}$ gibt es ein Dreieck t_e, welches e enthält und über $\overline{C}$ liegt. Sei p_e der Knoten von t_e, der nicht inzident zu e ist. Wir orientieren alle Kanten e' von $\overline{C}$ bezüglich ihrer respektiven p'_e. Falls e' links von p'_e ist, so wird diese nach rechts orientiert. Analog wird e' nach links orientiert, falls p'_e rechts von e' ist.

Existiert eine Kante e' ohne Orientierung, so erfüllt das Dreieck $t_{e'}$ alle Anforderungen an ein erweiterndes Dreieck, welches eine Kante mit der monotonen Kette teilt (Abbildung 19a).

Nehmen wir also an, dass es keine Kante ohne Orientierung gibt. Wenn es zwei aufeinanderfolgende Kanten e, e' gibt, die zum selben Knoten hinzeigen, so sind die zwei Dreiecke t_e und $t_{e'}$ identisch. Das Dreieck $t_e = t_{e'}$ erfüllt alle Anforderungen an ein erweiterndes Dreieck, welches zwei Kanten mit der monotonen Kette teilt (Abbildung 19b).

Angenommen alle Kanten sind orientiert und solch eine Konfiguration tritt nicht auf. Da in diesem Fall die linkeste Kante nach rechts und die rechteste Kante nach links orientiert sein muss, können wir bei der linkesten Kante anfangen und der Orientierung folgen. Spätestens bei der rechtesten Kante von $\overline{C}$ finden wir eine Kante, die anders orientiert ist. Dies ist ein Widerspruch zu der Annahme, dass wir keine zwei aufeinanderfolgenden Kanten finden, die zum selben Knoten orientiert sind. ◀



(a) Eine Kante ohne Orientierung

(b) Alle Kanten sind orientiert

Abbildung 19 Die zwei möglichen Kategorien orientierter Kanten und die dadurch implizierten erweiternden Dreiecke.

4.2.4 Linkester erweiternder sweep

Lemma 5 besagt, dass jede monotone Kette C in T (außer der oberen konvexen Hülle) mindestens eine Erweiterung besitzt. Jede diese Erweiterungen ist über ein erweiterndes Dreieck definiert. Wir betrachten dasjenige Dreieck unter allen möglichen Erweiterungen, welches am weitesten links liegt. Dadurch erhalten wir für jede Triangulierung T von S eine eindeutige Folge von monotonen Ketten $C_0, \ldots, C_M$. Die Kette C_0 ist die untere konvexe Hülle, C_M die obere konvexe Hülle und man erhält C_{i+1} durch Erweiterung von C_i mit dem linkesten erweiternden Dreieck. Die Anzahl solcher monotonen Ketten $M + 1$ ist dabei für ein gegebenes S eindeutig, was folgende Überlegung zeigt: Wir betrachten die Anzahl der Dreiecke unterhalb der monotonen Kette. Die Kette C_0 ist die untere konvexe Hülle, hat also kein Dreieck unter sich. Bei jeder Erweiterung C_{i+1} einer beliebigen monotenen Kette C_i erhöht sich die Anzahl der Dreiecke unterhalb der monotonen Kette um genau eins. Die obere konvexe Hülle C_M besitzt keine Erweiterung und die Anzahl der Dreiecke unterhalb von C_M ist gerade die Anzahl der Facetten von $T - 1$. Lemma 1 besagt, dass jede Triangulierung von S genau $2|S| - h - 1$ Facetten besitzt, wobei h die Anzahl der Punkte auf der konvexen Hülle ist. Also ist M gerade $2|S| - h - 2$ und jede solche Folge besteht aus $2|S| - h - 1$ monotonen Ketten.

▶ **Definition 6** (Linkester erweiternder sweep)**.** Der **linkeste erweiternde sweep** für eine beliebige Triangulierung T von S ist die eindeutige Folge von linkesten erweiternden monotonen Ketten $C_0, \ldots, C_M$. Die Anzahl dieser monotonen Ketten, M, ist dabei die Anzahl der Facetten von $T - 1 = 2|S| - h - 2$ (h ist die Anzahl Punkte auf der konvexen Hülle von S).

Zu jeder Triangulierung T von S gibt es einen eindeutigen linkesten erweiternden sweep. Man kann also, statt die Anzahl verschiedener Triangulierungen von S zu zählen, die Anzahl verschiedener linkester erweiternder sweeps von S bestimmen.

Die Hauptidee des Algorithmus ist es, einen gerichteten azyklischen Graphen zu generieren, in dem source-sink Pfade linkesten erweiternden sweeps entsprechen. Die Anzahl solcher Pfade ist dann gerade $tr(S)$, also die Anzahl verschiedener Triangulierungen von S.

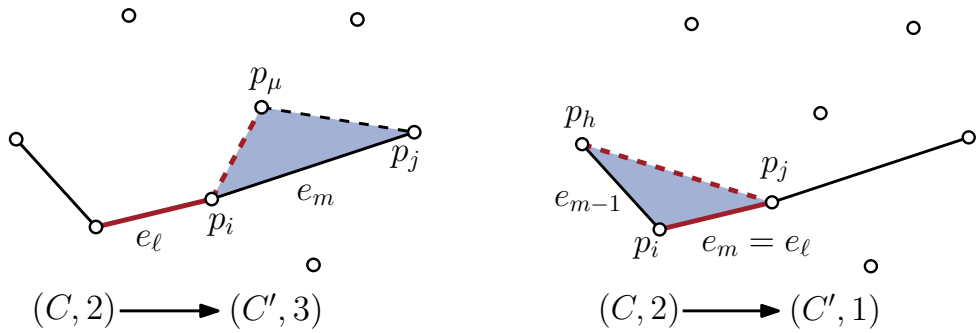
4.2.5 Graphenkonstruktion

Dieser Abschnitt beschäftigt sich mit der Konstruktion eines Graphen $\mathcal{G}_S = (V, E)$ für ein gegebenes S. Dieser Graph ist gerichtet, azyklisch und besitzt eine ausgezeichnete Quelle s und eine ausgezeichnete Senke t. Pfade von s zu t entsprechen 1-zu-1 linkesten erweiternden sweeps, welche nach Definition 6 eindeutig Triangulierungen von S entsprechen. Beachte: $\mathcal{G}_S$ kann weitere Senken und Quellen besitzen (im klassischen Sinne). Da allerdings kein Pfad von der ausgezeichneten Quelle s zu der ausgezeichneten Senke t über weitere Quellen und Senken führt, werden diese ignoriert.

▶ **Definition 7** (Markierte monotone Kette). Eine **markierte monotone Kette** für S ist ein Tupel (C, ℓ), wobei C eine monotone Kette und ℓ eine Ganzzahl ist, die besagt, dass die ℓ-te Kante von C markiert ist.

Wir definieren eine Nachfolgerrelation auf markierten monotonen Ketten: Sei (C, ℓ) eine markierte monotone Kette und $e_m = \overline{p_i p_j}$, $m \geq \ell$ eine Kante von C.

1. **Fall:** Es existiert ein p_μ zwischen p_i und p_j, welches über C liegt. Falls das Dreieck mit den Eckpunkten p_i, p_μ, p_j keinen Punkt aus S einschließt, betrachten wir die monotone Kette C', die man durch Erweiterung von C mit diesem Dreieck bekommt (Abbildung 20a). Genauer: $C' = C - \overline{p_i p_j} + \overline{p_i p_\mu} + \overline{p_\mu p_j}$. Dann ist (C', m) ein Nachfolger von (C, ℓ), wobei die Kante $\overline{p_i p_\mu}$ markiert wird.
2. **Fall:** Es existieren zwei Kanten $e_{m-1} = \overline{p_h p_i}$, $e_m = \overline{p_i p_j}$, sodass das Dreieck mit den Eckpunkten p_h, p_i, p_j über C liegt und keinen Punkt aus S einschließt. Betrachte die monotone Kette C', die man durch Erweiterung von C mit diesem Dreieck bekommt (Abbildung 20b). Genauer: $C' = C - e_{m-1} - e_m + \overline{p_h p_j}$. Dann ist $(C', m-1)$ ein Nachfolger von (C, ℓ), wobei die Kante $\overline{p_h p_j}$ markiert wird.



(a) Eine geteilte Kante **(b)** Zwei geteilte Kanten

Abbildung 20 Nachfolger einer markierten monotonen Kette. Die neuen Kanten des jeweiligen Nachfolgers sind gestrichelt dargestellt.

Wir konstruieren einen gerichteten, azyklischen Graphen $\mathcal{G}_S = (V, E)$ mit:

- $V = \{(C, \ell) \mid (C, \ell)$ ist eine markierte monotone Kette für $S\}$
- $E = \{((C, \ell), (C', m)) \mid (C', m)$ ist Nachfolger von $(C, \ell)\}$

Der Graph $\mathcal{G}_S$ ist azyklisch, da die Anzahl der Dreiecke unter der monotonen Kette C' von einem Nachfolger (C', m) eines Knotens (C, ℓ) immer genau um eins größer ist als die unter der Kette C.

In dieser Definition von $\mathcal{G}_S$ kann es mehrere Quellen und Senken geben. In Theorem 9 wird der Graph angepasst, damit er eine ausgezeichnete Quelle und Senke besitzt.

Das nächste Lemma stellt eine 1-zu-1 Korrespondenz zwischen Pfaden in $\mathcal{G}_S$ und Triangulierungen von Teilmengen von S her.

▶ **Lemma 8.** *Sei B die unterste monotone Kette (=untere konvexe Hülle), U die oberste monotone Kette (=obere konvexe Hülle) und C eine beliebige monotone Kette in S.*
Dann gilt:
Gerichtete Pfade von $(B, 1)$ zu (C, k) in $\mathcal{G}_S$ entsprechen 1-zu-1 Triangulierungen des Bereiches $\mathcal{A}$, der von $(B, 1)$ und (C, k) eingeschlossen wird. Die linkeste obere Kante des eindeutigen linkesten erweiternden sweeps einer solchen Triangulierung ist gerade die markierte (k-te) Kante von C.

Beweis. Wir führen eine Induktion über die Anzahl i der Dreiecke in $\mathcal{A}$.

$i = 1$: Ein Gebiet mit einem Dreieck besitzt nur eine mögliche Triangulierung. Sei (C, k) eine beliebige markierte monotone Kette, die zusammen mit $(B, 1)$ ein Dreieck einschließt. Eine Vorüberlegung zeigt, dass die Markierung an diesem eingeschlossenen Dreieck sein muss, da sonst dieser Knoten in $\mathcal{G}_S$ keine Vorgänger besitzt – es gibt also keinen Pfad von $(B, 1)$ zu dem Knoten.
Gemäß der Definition von erweiternden Dreiecken in Definition 4 gibt es zwei Arten von solchen Dreiecken: Dreiecke die sich eine oder zwei Kanten mit der monotonen Kette teilen.
Angenommen das erweiternde Dreieck teilt sich zwei Kanten mit B. Dann gibt es nur eine mögliche, gültige Markierung – die Seite des Dreiecks die nicht Teil von B ist. Der entsprechende Knoten in $\mathcal{G}_S$ hat als einzigen Vorgänger $(B, 1)$. Es gibt also in diesem Fall genau einen Pfad von $(B, 1)$ zu (C, k), der dem linkesten erweiternden sweep der Triangulierung entspricht, womit die Behauptung gezeigt wäre.
Falls sich das erweiternde Dreieck eine Kante mit der monotonen Kette teilt, so gibt es zwei mögliche Markierungen am Dreieck: links oben und rechts oben. Angenommen die Markierung wäre rechts oben. Der entsprechende Knoten in $\mathcal{G}_S$ hat keine Vorgänger, da die Nachfolgerrelation so definiert ist, dass sich die Markierung immer an der linkesten oberen Kante des erweiternden Dreieck befindet. Falls sich die Markierung links oben befindet, so hat der entsprechende Knoten in $\mathcal{G}_S$ nur den Vorgänger $(B, 1)$, woraus die Behauptung folgt.

$i \curvearrowright i + 1$: Sei (C, k) eine beliebige markierte monotone Kette, sodass $\mathcal{A}$ genau $i+1$ Dreiecke umfasst. Wir betrachten die Vorgänger (D_j, ℓ_j) des Knoten (C, k) in $\mathcal{G}_S$: Für jedes (D_j, ℓ_j) sei $\mathcal{A}_j$ der zwischen $(B, 1)$ und (D_j, ℓ_j) eingeschlossene Bereich. Dieser hat i Dreiecke – wir können also die Induktionsvoraussetzung anwenden. Anders gesagt: Jeder Pfad von $(B, 1)$ zu einem der (D_j, ℓ_j) entspricht einer Triangulierung des Bereiches $\mathcal{A}_j$, wobei die markierte Kante von (D_j, ℓ_j) gerade die linke obere Kante des letzten linkesten erweiternden Dreiecks ist.
Jede Triangulierung eines Bereichs $\mathcal{A}_j$ kann man nur auf eine eindeutige Art zu einer Triangulierung des Bereiches $\mathcal{A}$ erweitern, da nur ein weiteres Dreieck dazukommt. Anders gesagt gibt es nur einen Pfad von (D_j, ℓ_j) zu (C, k). Analog zu der Induktionsannahme

gibt es aufgrund der Definition der Nachfolgerrelation nur eine valide Markierung eines erweiternden Dreiecks. Daraus folgt, dass die Anzahl der Triangulierungen des Bereiches $\mathcal{A}$ gerade die Summe der Anzahl der Pfade von $(B, 1)$ zu jedem (D_j, ℓ_j) ist. ◀

Aus Lemma 8 folgt, dass je zwei unterschiedliche Pfade von $(B, 1)$ zu einem beliebigen Knoten (C, ℓ) die selbe Länge haben müssen, da diese Pfade verschiedenen Triangulierungen des selben Bereichs entsprechen. Da dieser Bereich eine eindeutige Anzahl von Dreiecken besitzt und diese Anzahl gleich der Länge des Pfades ist, müssen die beiden Pfade die selbe Länge haben.

Durch Hinzufügung eines Nachfolgerknotens $\top$ für alle markierten Ketten der oberen konvexen Hülle (U, k) in $\mathcal{G}_S$ erhalten wir das Hauptresultat, auf dem der Algorithmus aufbaut:

▶ **Theorem 9.** *Die Anzahl der Triangulierungen von S ist die Anzahl der Pfade in $\mathcal{G}_S$ von der Quelle $(B, 1)$ zur Senke $\top$.*

Beweis. Jede beliebige Triangulierung T von S hat einen eindeutigen linkesten erweiternden sweep. Die letzte Erweiterung in diesem sweep ist über ein erweiterndes Dreieck definiert. Dieses Dreieck hat Kanten, die Teil der oberen konvexen Hülle sind. Wir markieren die linke obere Kante dieses Dreiecks. Dadurch entspricht die letzte monotone Kette des linkesten erweiternden sweeps mit dieser Markierung einem Knoten (U, k) in $\mathcal{G}_S$ für ein k.

Lemma 8 besagt, dass jeder Pfad in $\mathcal{G}_S$ von $(B, 1)$ zu einem (C, k) einer Triangulierung des Bereiches $\mathcal{A}$ zwischen $(B, 1)$ und (C, k) entspricht. Da der zwischen $(B, 1)$ und (U, k) für ein beliebiges k eingeschlossene Bereich alle Punkte aus S umfasst, entspricht jeder Pfad von $(B, 1)$ zu (U, k) für ein beliebiges k einer Triangulierung von S. Da der Knoten $\top$ Nachfolger von allen (U, k) ist, folgt dass die Anzahl der Pfade von $(B, 1)$ zu $\top$ die Summe der Pfade von $(B, 1)$ zu allen (U, k) ist. Dieses ist nach Lemma 8 die Anzahl aller Triangulierungen von S. ◀

4.3 Der Algorithmus

Wir stellen einen Algorithmus vor, der schlimmstenfalls in Zeit $O(n^2 2^n)$ und Platzverbrauch $O(n2^n)$ die Anzahl der Triangulierungen von einer in allgemeiner Lage liegenden Punktmenge $S = \{p_1, \ldots, p_n\}$ berechnet. Aus Theorem 9 folgt, dass es ausreichend ist, die Anzahl Pfade von $(B, 1)$ zu $\top$ in $\mathcal{G}_S$ zu zählen, um das Problem zu lösen. Den Algorithmus kann man in zwei Teile spalten:

1. Konstruiere $\mathcal{G}_S$.
2. Zähle die Anzahl Pfade von $(B, 1)$ zu $\top$.

Wir werden für beide Teile Pseudocode angeben und anhand dessen, die oben behaupteten Schranken beweisen.

4.3.1 Konstruktion von $\mathcal{G}_S$

Sei $S = \{p_1, \ldots, p_n\}$ wie gehabt eine in allgemeiner Lage liegende Punktmenge. Die alternative Definition von monotonen Ketten (Definition 2) besagt, dass man jede monotone Kette als Teilmenge von S auffassen kann. Es gibt insgesamt 2^n mögliche Teilmengen von S. Aus der ersten Bedingung für monotone Ketten folgt, dass p_1 und p_n in der Teilmenge enthalten sein müssen. Es existieren also 2^{n-2} verschiedene monotone Ketten. S enthält n Elemente, also ist die Länge einer beliebigen monotonen Kette durch $n - 1$ nach oben beschränkt. Jede

monotone Kette besitzt folglich maximal $n-1$ verschiedene Markierungen. Die Knoten in $\mathcal{G}_S$ sind gerade markierte monotone Ketten, also folgt, dass die Anzahl dieser Knoten nach oben durch $O(n2^n)$ beschränkt ist.

Um die maximale Anzahl der Kanten abzuschätzen, definieren wir für eine beliebige markierte monotone Kette (C,k) zwei Mengen:

- $\zeta(C)$ ist die Menge der Punkte aus S, die auf (C,k) liegen
- $\zeta_\uparrow(C)$ ist die Menge der Punkte aus S, die oberhalb von (C,k) liegen.

Diese beiden Mengen sind disjunkt und haben insgesamt maximal n Elemente. Definition 4 besagt, dass es zwei Klassen von erweiternden Dreiecken gibt: Dreiecke, die sich eine, bzw. zwei Kanten mit der monotonen Kette teilen. Sei jetzt t ein beliebiges erweiterndes Dreieck für (C,k).

1. **Fall:** Das erweiternde Dreieck teilt sich eine Kante mit (C,k). In diesem Fall muss der dritte Punkt des Dreiecks oberhalb von (C,k) liegen, also in $\zeta_\uparrow(C)$ enthalten sein.
2. **Fall:** Das erweiternde Dreieck teilt sich zwei Kanten mit (C,k). In diesem Fall liegen alle drei Punkte des Dreiecks auf (C,k), sind also alle in $\zeta(C)$ enthalten.

Jeder Punkt p_i aus $\zeta_\uparrow(C)$ trägt maximal zu einem erweiternden Dreieck und damit einer Erweiterung bei: das potentielle erweiternde Dreieck ist dabei eindeutig durch p_i und die Kante in C bestimmt, die unter dem Punkt liegt. Dieses Dreieck muss nicht ein gültiges erweiterndes Dreieck sein, da es weitere Punkte aus S einschließen könnte und in dem Fall der zweiten Bedingung der Definition von erweiternden Dreiecken in Definition 4 widerspricht.

Analog kann man zeigen, dass jeder Punkt in $\zeta(C)$ zu maximal einer Erweiterung beiträgt: Sei p_i ein beliebiger Punkt aus $\zeta(C)$. Das potentielle erweiternde Dreieck ist durch p, den vorhergehenden und den nachfolgenden Punkt in C eindeutig bestimmt. Dieses Dreieck muss nicht gültig sein, da es analog wie im anderen Fall keine Punkte aus S einschließen darf und es außerdem oberhalb von C liegen muss.

Zusammengefasst können wir folgern, dass jeder Punkt aus $\zeta(C)$ und $\zeta_\uparrow(C)$ höchstens zu einer Erweiterung beiträgt. Da die Kardinalität von den beiden Mengen gemeinsam maximal n ist, folgt, dass jede markierte monotone Kette maximal n Nachfolger hat. Dadurch ist die Anzahl der Kanten in $\mathcal{G}_S$ nach oben durch $O(n^2 2^n)$ beschränkt.

Wir zeigen im Folgenden, dass die Kanten von $\mathcal{G}_S$ nicht explizit gespeichert werden müssen. Man kann alle Nachfolger einer markierten monotonen Kette in $O(n)$ on-the-fly berechnen, wenn man einen einmaligen Vorverarbeitungsschritt mit Laufzeit $O(n^4)$ durchführt.

Vorverarbeitung

Das Ergebnis des Vorverarbeitungsschrittes ist ein dreidimensionales Array valid(v,v',v''), welches zu drei gegebenen Punkten v,v',v'' aus S angibt, ob das Dreieck mit den Eckpunkten v,v',v'' keinen Punkt aus S einschließt.

Gegeben ein Dreieck mit den Eckpunkten v,v',v'' und einen Punkt p kann man wie folgt in $O(1)$ Zeit bestimmen, ob das Dreieck p einschließt: Wir orientieren die Kanten des Dreiecks im Uhrzeigersinn und schauen nach, ob p bezüglich jeder orientierten Kante auf derselben Seite ist. Falls dies der Fall ist, so ist p innerhalb des Dreiecks.

Der Vorverarbeitungsschritt hat eine Gesamtlaufzeit von $O(n^4)$, da insgesamt viermal (verschachtelt) über die Elemente in S iteriert wird. Der Test eines Punktes, ob dieser innerhalb eines Dreiecks liegt, kann in $O(1)$ Zeit überprüft werden.

Der nachfolgende Algorithmus liefert gegeben S, eines Knotens (C, ℓ) von $\mathcal{G}_S$ und des Arrays valid$(\cdot,\cdot,\cdot)$ aus dem Vorverarbeitungsschritt alle Nachfolger von (C, ℓ).

Input : $S = \{p_1, \ldots, p_m\}$, Knoten (C, ℓ), Array valid$(\cdot,\cdot,\cdot)$.
Output : Menge aller Nachfolger von (C, ℓ).

1 **foreach** $p_i \in S$ **do**
2 **if** p_i *ist über Segment* $e_j = \overline{p_x p_y} \bigwedge j \geq \ell \bigwedge$ *valid*(p_x, p_i, p_y) **then**
3 $(C - \overline{p_x p_y} + \overline{p_x p_i} + \overline{p_i p_y}, j)$ ist Nachfolger von (C, ℓ);
4 **foreach** $e_j = \overline{p_i p_{i+1}} \in (C, \ell)$ **do**
5 **if** p_i *ist unter Segment* $\overline{p_{i-1} p_{i+1}} \bigwedge j \geq \ell \bigwedge$ *valid*(p_{i-1}, p_i, p_{i+1}) **then**
6 $(C - \overline{p_{i-1} p_i} - \overline{p_i p_{i+1}} + \overline{p_{i-1} p_{i+1}}, j-1)$ ist Nachfolger von (C, ℓ);
7 **if** (C, ℓ) *ist obere konvexe Hülle* **then**
8 $\top$ ist Nachfolger von (C, ℓ);

Algorithmus 5 : Auffinden von Nachfolgern

Bei der Abschätzung für die maximale Anzahl Nachfolger einer markierten monotonen Kette (C, ℓ) wurden zwei Mengen definiert (vergleiche Abschnitt 4.3.1):

1. $\zeta_\uparrow(C)$, die Menge der Knoten aus S, die oberhalb von (C, ℓ) liegen
2. $\zeta(C)$, die Menge der Knoten aus S, die auf (C, ℓ) liegen.

Wir haben gezeigt, dass jedes Element aus einer dieser beiden Mengen maximal zu einem Nachfolger von (C, ℓ) beiträgt. Algorithmus 5 lässt sich in drei Teile gliedern, von denen jeder eine Laufzeit von $O(n)$ hat.

In der ersten Schleife werden die Elemente aus $\zeta_\uparrow(C)$ untersucht. Die erweiternden Dreiecke, die aus diesen Elementen resultieren teilen sich genau eine Kante mit (C, ℓ). Aus der Definition von Nachfolgern markierter monotonen Ketten folgt, dass die geteilte Kante e_j des erweiternden Dreiecks mit C nicht links von e_ℓ liegen darf. Des Weiteren darf das erweiternde Dreieck keine weiteren Punkte aus S einschließen. Diese zwei Bedingungen werden gerade in der If-Abfrage überprüft.

Würde man diese erste Schleife naiv implementieren, so hätte sie eine Laufzeit von $O(n^2)$, da man für jeden Punkt aus S die zugehörige Kante finden müsste, über welcher dieser Punkt liegt. Da S allerdings nach Voraussetzung in allgemeiner Lage vorliegt, können wir die Elemente aus S sortiert indizieren. Dadurch können wir bei dem linkesten Segment von (C, ℓ) anfangen und für jeden Punkt nur überprüfen, ob dieser oberhalb des aktuellen Segments liegt. Falls dies nicht der Fall ist, so ersetzen wir das aktuelle Segment durch das Nachfolgende in (C, ℓ). Dadurch, dass wir jedes Segment in (C, ℓ) maximal einmal anschauen und wir über alle Punkte in S iterieren, folgt, dass die Laufzeit für die erste Schleife gerade $O(n)$ ist.

Die zweite Schleife des Algorithmus 5 überprüft die Knoten aus der Menge $\zeta(C)$. Die erweiternden Dreiecke, die durch Betrachtung dieser Menge entstehen, teilen sich zwei Kanten mit (C, ℓ). Analog zu der ersten Schleife überprüft die If-Abfrage die notwendigen Bedingungen für einen Nachfolger der markierten monotonen Kette. Aus der alternativen Definition 2 für monotone Ketten folgt, dass die monotone Kette maximal $n - 1$ Kanten besitzen kann. In diesem Fall würde diese monotone Kette einen Streckenzug durch alle Punkte aus S darstellen. Da diese zweite Schleife über die Kanten der monotonen Kette iteriert und die If-Abfrage in konstanter Zeit durchgeführt werden kann, folgt, dass die Schleife eine Laufzeit von $O(n)$ hat.

Die ersten beiden Schleifen bestimmen, gemäß der definierten Nachfolgerrelation, alle Nachfolger für einen Knoten (C, ℓ). Da wir allerdings die Senke $\top$ künstlich hinzugefügt haben und definiert haben, dass alle (U, k) für ein beliebiges k diese als Nachfolger haben, müssen wir noch überprüfen, ob (C, ℓ) die obere konvexe Hülle ist. In diesem Fall ist ist $\top$ Nachfolger von (C, ℓ). Den Test auf Konvexität kann man durch Vergleich der Kanten mit der vorberechneten oberen konvexen Hülle erreichen. Diese Vorberechnung benötigt $O(n \log n)$ Zeit, wenn man den Graham Scan Algorithmus [10] verwendet. Da man diese Vorberechnung einmalig pro Punktmenge S machen muss und der Algorithmus exponentielle Laufzeit hat, verschwindet der Aufwand für die Berechnung der konvexen Hülle in der O-Notation.

Die Länge von (C, ℓ) ist durch $n-1$ nach oben beschränkt, also ist die Laufzeit der letzten Schleife ebenfalls $O(n)$. Die Gesamtlaufzeit von Algorithmus 5 ist $O(n)$, wenn man einen einmaligen Vorverarbeitungsschritt für die Bestimmung der konvexen Hülle und Erstellung des Arrays valid$(\cdot, \cdot, \cdot)$ mit Laufzeit $O(n^4)$ einfügt.

Da wir bei der Konstruktion von $\mathcal{G}_S$ nur die Knoten berechnen und speichern müssen, ist die Laufzeit des ersten Schritts $O(n2^n)$ und der Speicherverbrauch $O(n2^n)$. Durch die on-the-fly Berechnung von allen Nachfolgern für einen Knoten (C, ℓ) aus $\mathcal{G}_S$ in Zeit $O(n)$ (Algorithmus 5) haben wir im Vergleich zu der naiven Konstruktion des Graphen einen Faktor n im Speicherverbrauch gespart. Zusammengefasst gilt also folgendes Lemma:

▶ **Lemma 10.** *Die Konstruktion von $\mathcal{G}_S$ ist in Zeit $O(n2^n)$ und Speicherverbrauch $O(n2^n)$ möglich. Die Kanten werden dabei nicht explizit gespeichert. Alle Nachfolger eines Knotens können in Zeit $O(n)$ bestimmt werden, wenn man einen Vorverarbeitungsschritt mit Zeit $O(n^4)$ einfügt.*

4.3.2 Zählen der Anzahl Pfade von $(B, 1)$ zu $\top$

Bevor wir zum eigentlichen Algorithmus kommen, definieren wir den Begriff des topologischen Ranges eines Knotens in einem gerichtetem, azyklischen Graphen (DAG).

▶ **Definition 11** (Topologische Sortierung eines DAG)**.** Sei $G = (V, E)$ mit $V = \{p_1, \ldots, p_n\}$ ein gerichteter, azyklischer Graph. Eine topologische Sortierung von G ist eine bijektive Funktion $\Phi \colon V \to \{1, \ldots, n\}$ sodass gilt: $\forall (p_i, p_j) \in E \Rightarrow \Phi(p_i) < \Phi(p_j)$. In diesem Fall bezeichnen wir $\Phi(p_i)$ als den topologischen Rang von p_i bezüglich Φ.

Bildlich gesprochen weist man jedem Knoten einen eindeutigen Index zu, welcher niedriger sein muss, als die Indizes seiner Nachfolger. Die topologische Ordnung von G ist die durch den topologischen Rang induzierte Ordnung auf G.

Algorithmus 6 bestimmt in Zeit $O(|V| + |E|)$ eine topologische Sortierung eines gerichteten, azyklischen Graphen $G = (V, E)$. Da jeder Knoten und jede Kante maximal einmal angeschaut wird, folgt die Laufzeit von $O(|V| + |E|)$.

Algorithmus 7 beschreibt den zentralen Algorithmus dieser Ausarbeitung. Dieser bekommt als Eingabe den Graphen $\mathcal{G}_S$ und liefert die Anzahl Pfade von $(B, 1)$ nach $\top$ in $\mathcal{G}_S$.

Am Anfang wird pro Knoten in $\mathcal{G}_S$ ein Zähler mit 0 initialisiert, $(B, 1)$ bekommt als Startwert 1. Dieser Zähler symbolisiert gerade die Anzahl der Pfade von $(B, 1)$ zu dem jeweiligen Knoten. Wenn wir die Knoten in topologischer Ordnung besuchen, dann ist sichergestellt, dass wir einen Knoten erst dann behandeln, wenn wir alle anderen „Eingangspfade“ abgearbeitet haben, d.h. wir können folgende Schleifeninvariante formulieren:

Zu jedem Zeitpunkt verändert sich der Zähler bei den schon besuchten Knoten nicht mehr und dieser Zähler entspricht gerade der Anzahl Pfade von $(B, 1)$ zu dem jeweiligen Knoten.

Wir können eine Induktion über die Anzahl der schon besuchten Knoten führen, wobei wir uns hier aufgrund von Platzmangel nur auf den Induktionsschluss beschränken werden.

Input : $\mathcal{G}_S = (V, E), V = \{p_1, \ldots, p_n\}$ gerichteter, azyklischer Graph.
Output : $\Phi\colon V \to \{1, \ldots, n\}$, topologische Sortierung für G.

```
1  Q ← Menge aller Knoten mit Eingangsgrad 0;
2  i ← 1;
3  while Q ≠ ∅ do
4      Entferne einen Knoten p aus Q;
5      Φ(p) := i;
6      foreach q ∈ V mit (p, q) ∈ E do
7          E ← E ∖ {(p, q)};
8          if q hat Eingangsgrad 0 then
9              Q ← Q ∪ {q};
10     i ← i + 1;
```

Algorithmus 6 : Bestimmung einer topologischen Sortierung eines DAG

Betrachten wir den Zeitpunkt, wenn ein Knoten (C, ℓ) besucht wird. Für jeden Vorgänger von (C, ℓ) gilt die Induktionsannahme, d.h. der Zähler besagt gerade, wie viele Pfade von $(B, 1)$ zu dem jeweiligen Knoten führen. Da wir die Knoten in topologischer Ordnung durchlaufen, ist sichergestellt, dass es keinen Vorgängerknoten von (C, ℓ) gibt, der noch nicht besucht wurde.

Jeder Pfad von $(B, 1)$ zu einem Vorgänger von (C, ℓ) lässt sich zu einem Pfad zu (C, ℓ) erweitern. Da der Graph in topologischer Ordnung traversiert wird, folgt, dass alle Vorgänger schon abgearbeitet wurden. Also ist die Anzahl der Pfade zu (C, ℓ) gerade die Summe der Pfade seiner Vorgänger. Da für jeden Vorgänger der Zähler von (C, ℓ) um den Zähler des Vorgängers erhöht wurde, folgt die Behauptung und damit die Korrektheit von Algorithmus 7.

Pro Knoten wird ein Zähler gespeichert, es gibt $O(n2^n)$ Knoten, also ist ist der Gesamtplatzbedarf des Algorithmus 7 $O(n2^n)$.

Kommen wir nun zu der Laufzeitanalyse. Die Konstruktion von $\mathcal{G}_S$ braucht $O(n2^n)$ Zeit und einen Platzbedarf von $O(n2^n)$, wenn man auf die explizite Speicherung der Kanten verzichtet (Lemma 10). Das Initialisieren der Zähler in der ersten Schleife dauert, da wir $O(n2^n)$ Knoten haben, $O(n2^n)$ Zeit. Die topologische Sortierung des Graphen $\mathcal{G}_S$ ist in Zeit $O(|V| + |E|) = O(n2^n + n^2 2^n) = O(n^2 2^n)$ möglich (Algorithmus 6). Die zweite For-Schleife iteriert über jeden Knoten in $\mathcal{G}_S$, wobei das Auffinden der Nachfolger eines Knotens, nach

Input : $\mathcal{G}_S = (V, E)$.
Output : Anzahl Pfade von $(B, 1)$ zu $\top$ in $\mathcal{G}_S$.

```
1  foreach (C, i) ∈ V do
2      (C, i) ← 0;
3  (B, 1) ← 1;
4  foreach (C, i) ∈ V in topologischer Ordnung do
5      addiere den Zähler von (C, i) zu seinen Nachfolgern;
```

Algorithmus 7 : Zählen von Triangulierungen

Lemma 10, in Zeit $O(n)$ möglich ist (wenn man $O(n^4)$ Aufwand in Vorberechnungsschritte investiert). Die Gesamtlaufzeit der zweiten Schleife ist also $O(n^2 2^n)$, gegeben der Annahme, dass je zwei Zähler in konstanter Zeit aufaddiert werden können. Der Algorithmus hat exponentiellen Speicherverbrauch, es ist also sinnvoll anzunehmen, dass wir als Berechnungsmodell eine RAM mit linearer Wortlänge betrachten. Wir nehmen also an, dass ein Zähler in einem Wort gespeichert wird und dadurch je zwei Zähler in konstanter Zeit aufaddiert werden können.

Zusammengefasst gilt folgendes Theorem:

▶ **Theorem 12.** *Die Anzahl der Triangulierungen für eine gegebene, in allgemeiner Lage liegende Punktmenge kann nach Konstruktion von $\mathcal{G}_S$ in Zeit $O(n^2 2^n)$ mit einem Platzbedarf von $O(n2^n)$ bestimmmt werden (Algorithmus 7). Die Konstruktion von $\mathcal{G}_S$ ist in Zeit $O(n2^n)$ und Speicherverbrauch $O(n2^n)$ möglich (Lemma 10).*

Der Zähler jedes Knotens abstrahiert über alle Triangulierungen des Bereiches unterhalb von der jeweiligen monotonen Kette. Durch die Aggregation der Zähler im Laufe des Algorithmus betrachten wir immer größer werdende Mengen von Triangulierungen auf einmal und sind deswegen in der Lage die untere Schranke zu durchbrechen.

4.4 Implementierungsdetails und Ausblick

V. Alvarez und R. Seidel [5] haben den Algorithmus implementiert und anhand von zufällig gewählten Punkten evaluiert. Dieser Abschnitt befasst sich mit der Evaluierung und den Implementierungsdetails des Algorithmus.

4.4.1 Optimierungen

Ein Problem ergibt sich, da pro Knoten ein Zähler gespeichert wird, der exponentiell große Werte annehmen kann. Die zurzeit beste bekannte obere Schranke für die Anzahl der Triangulierungen ist $O(30^n)$ [14]. Wenn unsere Probleminstanz nur 30 Punkte groß ist, dann müssten wir schlimmstenfalls bereits

$$\underbrace{30 \times 2^{30}}_{\text{Anzahl Knoten}} \times \underbrace{\frac{\log_2 30^{30}}{8}}_{\text{Speicherbedarf pro Knoten in Byte}} \approx 552 \text{ Gigabytes} \tag{1}$$

Speicherplatz für Zähler aufwenden, was nicht praktikabel ist. Um dieses Problem zu umgehen, verwenden wir den chinesischen Restsatz (ohne Beweis).

▶ **Theorem 13** (Chinesischer Restsatz [8]). *Seien $m_1, \ldots m_n$ n paarweise teilerfremde Zahlen. Dann existiert für jedes Tupel ganzer Zahlen $(a_1, \ldots, a_n)$ eine Zahl x, welche die folgende simultane Kongruenz erfüllt:*

$$x \equiv a_1 \pmod{m_1} \quad \ldots \quad x \equiv a_n \pmod{m_n}$$

Das x ist dabei eindeutig (kongruent modulo $M := \prod_{i=1}^{n} m_i$) und kann durch den erweiterten euklidischen Algorithmus bestimmt werden.

Wir wählen uns m Primzahlen $p_1, \ldots, p_m$ mit $\prod_{i=1}^{m} p_i > 30^{|V|}$. Wir führen Algorithmus 7 m mal durch. Dabei speichern wir im Durchgang i die Zähler der einzelnen Knoten $\bmod p_i$ und merken uns am Ende jedes Durchgangs den Zählerwert von $\top$. Nach dem chinesischen

Restsatz können wir die ursprüngliche Lösung mit dem erweiterten euklidischen Algorithmus rekonstruieren.

Der Algorithmus iteriert bei der Summation der Zähler über die Knoten von $\mathcal{G}_S$ in topologischer Ordnung. In $\mathcal{G}_S$ haben je zwei Pfade von $(B, 1)$ zu einem beliebigen Zielknoten (C, ℓ) die gleiche Länge (vgl. Folgerungen aus Lemma 8). Es gibt also keine „Abkürzungen“ in $\mathcal{G}_S$ – falls ein Pfad von $(B, 1)$ zu einem Knoten existiert, so ist dessen Länge auch gleichzeitig minimal. Durch diese Eigenschaft können wir die Art der Iteration anpassen, ohne den Ausgang des Algorithmus zu verändern: Statt über die Knoten in topologischer Ordnung zu iterieren, reicht es aus eine Breitensuche von $(B, 1)$ zu starten, wobei alle Knoten in derselben Breitensuchschicht abgearbeitet werden müssen, bevor Knoten der nächsten besucht werden. Diese Art der Iteration induziert eine topologische Sortierung von $\mathcal{G}_S$: Alle Knoten in einer niedrigeren Breitensuchschicht haben einen kleineren Rang als die in der höheren. Die Ordnung der Knoten in derselben Breitensuchschicht kann willkürlich gewählt werden, ohne die Eigenschaften einer topologischen Sortierung zu verletzen. Dass solch eine topologische Ordnung für beliebige, azyklische, gerichtete Graphen nicht valide ist, zeigt Abbildung 21. Es ist hinreichend zu zeigen, dass falls alle Pfade von $(B, 1)$ zum selben Zielknoten eine

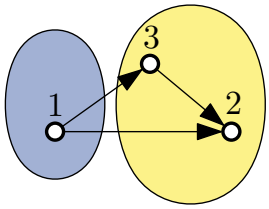


Abbildung 21 Ein gerichteter, azyklischer Graph, indem die von der Breitensuche induzierte topologische Sortierung falsch ist. Die einzelnen Breitensuchschichten sind farblich dargestellt. Die Zahlen über den Knoten entsprechen dem falschen topologischen Rang der Knoten.

eindeutige Länge haben, die durch Breitensuche induzierte topologische Sortierung von $\mathcal{G}_S$ korrekt ist. Angenommen es gibt einen Knoten (C, ℓ), dessen topologischer Rang invalide ist. Dies bedeutet, dass es einen Knoten (D, m) gibt, der einen höheren Rang als (C, ℓ) besitzt und von dem ein Pfad zu (C, ℓ) existiert. Betrachte den Pfad $\mathcal{P}_1$ von $(B, 1)$ zu (C, ℓ) und den Pfad $\mathcal{P}_2$ von $(B, 1)$ zu (C, ℓ) über (D, m). Diese Pfade können identische Teilpfade besitzen. Die Längen der Pfade $\mathcal{P}_1$ und $\mathcal{P}_2$ müssen nach Voraussetzung gleich sein. Dies bedeutet aber, dass (D, m) in einer niedrigeren Breitensuchschicht liegt als (C, ℓ). Damit hätte (C, ℓ) einen höheren topologischen Rang als (D, m), was ein Widerspruch zur Annahme ist.

Folglich gilt die Behauptung, dass die von $(B, 1)$ ausgehende Breitensuche eine valide topologische Sortierung von $\mathcal{G}_S$ induziert. Aus dieser validen topologischen Sortierung folgt wiederum die Korrektheit der angepassten Iteration.

Wenn wir $\mathcal{G}_S$ in dieser Art traversieren, hat dies den Vorteil, dass die von $(B, 1)$ unerreichbaren Knoten ignoriert werden. Außerdem sparen wir die explizite Berechnung der topologischen Sortierung von $\mathcal{G}_S$.

Da die Nachfolger der Knoten in $\mathcal{G}_S$ von einer Breitensuchschicht nur von den Knoten selbst abhängen, müssen wir $\mathcal{G}_S$ nie ganz im Speicher halten: Es reicht aus zwei Breitensuchschichten gleichzeitig zu speichern:

1. Die zurzeit bearbeitete Schicht (am Anfang ist es nur der Knoten $(B, 1)$)
2. Alle Nachfolgerknoten der aktuellen Schicht

Durch diese beiden Optimierungen, war die Implementierung von V. Alvarez und R. Seidel [5] in der Lage, die Anzahl der Triangulierungen für eine Punktmenge mit 50 Elementen

zu berechnen, was vorher nicht möglich gewesen ist.

4.4.2 Experimentelle Ergebnisse und Ausblick

Alle Experimente wurden auf einer Maschine mit 128 Gigabyte Hauptspeicher ausgeführt. Für die Evaluation wurde der Algorithmus mit zwei weiteren verglichen: dem auf dynamischer Programmierung basierenden Algorithmus von Ray und Seidel [13] und dem Algorithmus von Alvarez et al. [3]. Letzterer ist dabei auf Graphen spezialisiert, die sich in möglichst wenige innere, konvexe Polygone zerlegen lassen.

Die Experimente fanden auf drei verschiedenen Klassen von Punktmengen statt:

1. n uniform verteilte Punkte in einem Quadrat
2. n zufällige Punkte, die sich in $\lceil \frac{n}{3} \rceil$ konvexe Polygone zerlegen lassen
3. n zufällige Punkte auf drei kozentrischen Kreisen, mit $\approx \frac{n}{3}$ Punkten pro Kreis.

Der hier vorgestellte Algorithmus hat eine durchweg bessere, oder vergleichbare gemessene Laufzeit in den ersten beiden Klassen. Der dynamische Ansatz von Ray und Seidel [13] ist zwar bei kleineren Instanzgrößen besser, versagt aber bei größerem n. Der hier vorgestellte Ansatz ermöglicht es erstmals Instanzgrößen von 50 (erste Punktmengenklasse), bzw. 37 (zweite Punktmengenklasse) zu lösen. Der spezialisierte Algorithmus von Alvarez et al. [3] ist bei den ersten beiden Klassen durchgehend schlechter als die beiden anderen Algorithmen. Falls man jedoch die Anzahl der konvexen Polygone, in die der Graph zerlegt werden kann, auf drei reduziert (dritte Punktmengenklasse), erreicht dieser Ansatz die besten Laufzeiten, obwohl seine asymptotische Laufzeit mit $O(3.1414^n)$ schlechter ist, als von dem hier vorgestellten Algorithmus. Für die genauen Messergebnisse verweisen wir auf die Originalarbeit [5].

Alvarez und Seidel [5] zeigen, dass der Algorithmus durch leichte Anpassungen auf andere Probleme angewandt werden kann. Im Folgenden geben wir eine kurze Übersicht über diese verwandten Probleme.

- Der vorgestellte Ansatz lässt sich benutzen um gleichverteilt eine zufällige Triangulierung zu bestimmen. Dabei steigt der Platzverbrauch jedoch auf $O(n^2 2^n)$, da Kanten explizit gespeichert werden müssen.
- Man kann statt die Anzahl aller möglichen Triangulierungen zu zählen, nur Triangulierungen betrachten, die einen bestimmten Kantenzug besitzen. Die einzige Anpassung, die zur Lösung dieses Problems gemacht werden muss, ist die Modifikation der Nachfolgerrelation (Abschnitt 4.2.5). Die Kanten der erweiternden Dreiecke dürfen hierbei nicht den Kantenzug schneiden. Die Laufzeit und der Platzverbrauch des so angepassten Algorithmus ändern sich nicht.
- Das letzte verwandte Problem ist die Bestimmung einer *optimalen* Triangulierung. Optimal bedeutet hierbei, dass eine gegebene Funktion minimiert wird. Aufgrund des sweep-artigen Vorgehens des Algorithmus, werden an die Funktion gewisse Anforderungen gestellt. Wir gehen nicht weiter ins Detail, verweisen allerdings den interessierten Leser auf die Originalarbeit [5].

4.5 Fazit

In dieser Ausarbeitung stellten wir den Algorithmus von Alvaraz und Seidel [5] zur Berechnung der Anzahl der Triangulierungen für eine gegebene, in allgemeiner Lage vorliegenden Punktmenge vor. Der Algorithmus basiert auf markierten monotonen Ketten. Wir zeigten, dass jede Triangulierung einer eindeutigen Folge monotoner Ketten entspricht. Außerdem

konstruierten wir einen gerichteten, azyklischen Graphen, in dem Pfade von der Quelle zur Senke 1-zu-1 solchen Folgen und damit Triangulierungen, entsprechen. Das Zählen aller Pfade in diesem Graphen ist in Zeit $O(n^2 2^n)$ und Speicherverbrauch von $O(n2^n)$ möglich. Der Algorithmus erreicht als Erster eine asymptotisch schnellere Laufzeit, als die zurzeit beste bekannte untere Schranke $\Omega(2.4317^n)$ für die Anzahl Triangulierungen einer n-elementigen Punktmenge [2]. Durch leichte Abwandlungen ist der Algorithmus in der Lage verwandte Probleme, wie das gleichverteilte Bestimmen einer zufälligen Triangulierung, zu lösen.

Referenzen

1 Oswin Aichholzer, Thomas Hackl, Clemens Huemer, Ferran Hurtado, Hannes Krasser, and Birgit Vogtenhuber. On the number of plane geometric graphs. *Graphs and Combinatorics*, 23(1):67–84, 2007.

2 Oswin Aichholzer, Ferran Hurtado, and Marc Noy. A lower bound on the number of triangulations of planar point sets. *Computational Geometry*, 29(2):135–145, 2004.

3 Victor Alvarez, Karl Bringmann, Radu Curticapean, and Saurabh Ray. Counting crossing-free structures. In *Proceedings of the 2012 Symposium on Computational Geometry*, pages 61–68. ACM, 2012.

4 Victor Alvarez, Karl Bringmann, Saurabh Ray, and Raimund Seidel. Counting triangulations and other crossing-free structures approximately. *arXiv preprint arXiv:1404.0261*, 2014.

5 Victor Alvarez and Raimund Seidel. A simple aggregative algorithm for counting triangulations of planar point sets and related problems. In *Proceedings of the 29th annual Symposium on computational geometry*, pages 1–8. ACM, 2013.

6 David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1):21–46, 1996.

7 Sergei Bespamyatnikh. An efficient algorithm for enumeration of triangulations. *Computational Geometry*, 23(3):271–279, 2002.

8 Cunsheng Ding. *Chinese remainder theorem*. World Scientific, 1996.

9 Alfredo Garcıa, Marc Noy, and Javier Tejel. Lower bounds on the number of crossing-free subgraphs of $k \subset n$. *Computational Geometry*, 16(4):211–221, 2000.

10 Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information processing letters*, 1(4):132–133, 1972.

11 Michael Hoffmann, André Schulz, Micha Sharir, Adam Sheffer, Csaba D Tóth, and Emo Welzl. Counting plane graphs: Flippability and its applications. In *Thirty Essays on Geometric Graph Theory*, pages 303–325. Springer, 2013.

12 Clemens Huemer and Anna de Mier. Lower bounds on the maximum number of non-crossing acyclic graphs. *arXiv preprint arXiv:1310.5882*, 2013.

13 Saurabh Ray and Raimund Seidel. A simple and less slow method for counting triangulations and for related problems. In *Proceedings of the 20th European Workshop on Computational Geometry, Seville*, pages 77–80, 2004.

14 Micha Sharir and Adam Sheffer. Counting triangulations of planar point sets. *Electr. J. Comb*, 18(1):P70, 2011.

15 Micha Sharir and Adam Sheffer. Counting plane graphs: Cross-graph charging schemes. In *Graph Drawing*, volume 7704 of *Lecture Notes in Computer Science*, pages 19–30. Springer Berlin Heidelberg, 2013.

16 Micha Sharir, Adam Sheffer, and Emo Welzl. On degrees in random triangulations of point sets. In *Proceedings of the 2010 annual symposium on Computational geometry*, pages 297–306. ACM, 2010.

17 Micha Sharir, Adam Sheffer, and Emo Welzl. Counting plane graphs: Perfect matchings, spanning cycles, and kasteleyn's technique. *Journal of Combinatorial Theory, Series A*, 120(4):777–794, 2013.

18 Micha Sharir and Emo Welzl. On the number of crossing-free matchings, cycles, and partitions. *SIAM Journal on Computing*, 36(3):695–720, 2006.

5 Superpatterns and Universal Point Sets for Graphs with Bounded Pathwidth

Fabian Klute

Abstract

This paper presents a detailed excerpt of the paper [1] and [2] from Bannister et al. which was published in 2013. The original work presents a connection between the universal point set problem and superpatterns of permuations. From these techniques several new and better bounds for the universal point set problem emerged. Most notably perhaps the improved general bound of $\frac{n^2}{4} - \Theta(n)$ and the better bound for graphs with bounded pathwidth of $O(n \log^{O(1)} n)$. The later one will be the focus of this paper, since a complete coverage is not possible in the course of a seminar. Still the general techniques are very interesting and for the most part applicable to general planar graphs.

5.1 Introduction

This paper is the final result of the algorithmics seminar at the Insitute of Theoretical Informatics at the KIT. The underlying work is a paper from Bannister et al.: Superpatterns and universal point sets [1] and [2]. In the original paper a new approach to the classical graph problem of finding universal point sets for planar graphs is presented. The authors found a connection between superpatterns of permutations and universal point sets. The universal point set problem asks for every integer $n > 0$ for a set of points in $\mathbb{R}^2$ which allows to draw every n-vertex planar graph into it using the points as vertices and only straight lines for edges, which are not allowed to cross. In the original paper better bounds for the general case as well as for several special classes of graphs are presented. Prior to this the best known upper bound for arbitrary planar graphs was a subset of a $\frac{4}{3}n \times \frac{2}{3}n$ grid of size $\frac{4}{9}n^2 + O(n)$ [3], this bound was improved to $\frac{n^2}{4} - \Theta(n)$. For simply nested graphs the bound got improved from $O(\log n / \log \log n^2)$ [4] to $O(n \log n)$. But the result we will focus on in this paper is for graphs with bounded pathwidth. Here Bannister et al presented a bound of $O(n \log^{O(1)} n)$.

As said this write up only focuses on graphs with bounded pathwidth, since it is not possible to cover all results, especially the one for arbitrary graphs, in a seminar paper. In Section 5.2 we introduce several definitions concerning permutations, their superpatterns and a few graph related concepts. The next part, Section 5.3, focuses on the general way of deriving a universal point set from a planar graph. This part is still valid for every planar graph, but does not provide any proofs for size bounds. After we introduced the general concepts we turn to chessboard representations, Section 5.4, and a very handy integer sequence (Section 5.5). Just afterwards, in Section 5.6, we give the first size bounds for superpatterns of permutations with fixed Strahler number, again connecting graph related concepts with permutations. Finally the last Section 5.7 introduces graphs with bounded pathwidth and provides a proof for the above stated upper bound.

5.2 Definitions

First we need to define a few general concepts. We start with the universal point set problem itself and continue with a few graph-related terms, permutations and superpatterns of permutations. As usual $G = (V, E)$ refers to a graph with vertices V and edges E. A planar graph is a graph, which can be drawn in the plane without any crossings. Although the final result in this work is only for graphs with bounded pathwidth, the techniques discussed in Section 5.3 can be used for any general planar Graph. Because of this we delay the definition of bounded pathwidth and other concepts until later.

▶ **Definition 1** (Universal point set problem)**.** A universal point set U_n is a set of points in $\mathbb{R}^2$, such that any planar graph with n vertices can be drawn without crossings using only points from U_n as vertices and straight lines as edges. The universal point set problem asks for such a set U_n with as few points as possible.

5.2.1 Permutations

Now that we have defined the general problem we turn away from graphs and towards permutations. For us S_n will be the set of all permutation of the numbers 1 to n. We will write a permutation as a sequence of numbers. For example there are six permutations with length three:

$$S_3 = \{123, 132, 213, 231, 312, 321\}.$$

Let π be a permutation in S_n, then π_i denotes the value at position $i (i \leq n)$ and $|\pi| = n$ the length of $|\pi|$. Next we need some further terminology for permutations. The first one is that of a subpattern. As a general example we will use the permutation $\pi = 3124$.

▶ **Definition 2** (Subpattern)**.** Given a permutation π we define a subpattern σ of π as another permutation, such that we can find a sequence of integers $1 \leq l_1 < l_2 < ... < l_{|\sigma|} \leq n$ with $\pi_{l_i} < \pi_{l_j}$ if and only if $\sigma_i < \sigma_j$. Additional we say that a permutation π avoids another permutation ϕ if ϕ is not found as a subpattern in π.

An example for a subpattern of our example permutation 3124 would be the length three permutation 213 which we find at the red marked positions 3124. A pattern our example avoids would be 231. We can organize permutations in so called *permutation classes*. A class consists of a set of permutations and its subpatterns or alternatively we state the forbidden permutations of the class. The result would be the set of all length $1, ..., n$ permutations avoiding the forbidden patterns. A permutation class, where the patterns ϕ_1 to ϕ_k are avoided is written, as $S(\phi_1, \phi_2, ..., \phi_k)$. For example the class $S_3(231, 321)$ would look like this:

$$S_3(231, 321) = \{123, 132, 213, 312\}.$$

The last definition for permutations is a superpattern:

▶ **Definition 3** (Superpatterns)**.** Let $P \subseteq S_n$ be a permutation class. A P-superpattern is a permutation σ, such that every $\pi \in P$ is a subpattern of σ.

For example 25314 would be a superpattern for all length three permutations. We will connect the universal point set problem with permutations, superpatterns and their size. Something we will need is a transformation from a permutation to coordinates. The easiest way would be to simply say for every element in a permutation σ we take its position as x- and its value as y-coordinate, but for our purposes the following definition is better:

▶ **Definition 4** (Stretch). Let σ be a permutation and $q = |\sigma|$ its length, then

$stretch(\sigma) = \{(i, q^{\sigma_i}) | 1 \leq i \leq q\}$.

Something else needed in a later section is the augmentation of a permutation:

▶ **Definition 5** (Augment). Let σ be a permutation. Then $augment(\sigma)$ is another permutation with length $|\sigma| + 3$ where the first entry of $augment(\sigma)$ is a 1 followed by $|\sigma| + 3$, then the permutation σ itself and at the end a 2. Since 1 and 2 are the new smallest elements σ has to be modified, here we simply add a 2 to every element.

An example would be the augmentation of $\sigma = 2143$ to $augment(\sigma) = 1743652$.

5.2.2 Canonical representation

A concept from graph theory we use a lot, is the canonical representation of planar graphs. Given a maximal planar graph G with n vertices, we say that G is canonically represented if:

- The three vertices $v_1v_2v_n$ are on the outer face of G, which is a triangle and they are ordered in clockwise order v_1, v_n, v_2.
- For each vertex $v_i, i \geq 3$ we find two or more neighbours v_j with $j < i$. We call those neighbours *earlier neighbours* of v_i.
- The earlier neighbours of a vertex v_i form a contiguous subset in the cyclic order of edges around v_i.

Let G_k be the induced subgraph of G with the vertices $\{v_1, v_2, ..., v_k\}, k \leq n$. Since G is in canonical order the outer face of G_k consists at least of the vertices v_1, v_2, v_k, their clockwise order around the outer face is v_1, v_k, v_2. Note that every maximal planar graph has a canonical representation.

For a vertex v_i with $i > 2$ the parent of v_i is the most clockwise smaller numbered neighbour of v_i and we will denote it with $parent(v_i)$. The tree $ctree(G)$ of a graph G is the tree we get by following the parents from every vertex until we reach v_1. Another way of obtaining $ctree(G)$ is to give each edge a direction from smaller to bigger indices (see Figure 22) and perform a depth-first traversal from v_1, visiting the vertices in clockwise order. Performing a pre- and reverse postorder traversal on this tree gives us the two values $pre(v_i)$ and $post(v_i)$ for every vertex in G. The value $pre(v_i)$ is the position of v_i in a preorder and $post(v_i)$ the position in a reverse postorder traversal (see Figure 22). A preorder traversal visits the parent first and then the children. A reverse postorder traversal is obtained by traversing the tree in postorder, meaning children first and last the parent, and reversing the resulting sequence.

▶ **Lemma 6.** *Let G be a graph and $ctree(G)$ is defined as above then a vertex v_i is an ancestor of vertex v_j in the $ctree(G)$ if and only if $pre(v_i) < pre(v_j)$ and $post(v_i) < post(v_j)$.*

Proof. Let v_i and v_j be two vertices with $pre(v_i) < pre(v_j)$, $post(v_i) < post(v_j)$. Then the vertex v_i has to be the ancestor of v_j since if this was not the case v_j would have been visited before v_i in the depth first search, but that would result in $pre(v_i) > pre(v_j)$ and $post(v_i) > post(v_j)$. Now if v_i is the ancestor of v_j in a $ctree(G)$ v_i had to be visited before v_j resulting in *pre*- and *post*-values as above. ◀

5.3 From Superpatterns to universal point sets

In this section we focus on connecting 213-avoiding superpatterns and universal point sets. Throughout the section it is assumed that the graph G with n vertices is maximal planar,

meaning no edge can be added without loosing planarity. If we want to embed another graph with n vertices which is not maximal it can be done using the same set of points since we only lose edges, but not get new ones. Since the graph is maximal planar we can get an embedding just by choosing the outer face. Additionally keep in mind that all the faces in a maximal planar graph are triangles. Before the central theorem of this section can be proven, we need several lemmas. Lemmas 7 and 10 will provide a connection between graphs and permutations, while Lemmas 11 to 13 show how coordinates can be derived from a permutation and why the resulting drawing has no crossings.

▶ **Lemma 7.** *If we renumber the vertices of a graph G by their number $post(v_i)$, we again have a canonical representation.*

Proof. First we will show, that *post* gives us a topological ordering of the graph G: If this was not the case we would find an edge (u, v) in $ctree(G)$ such that $post(u) > post(v)$. Since *post* is a reversed postorder this would result in $post'(u) < post'(v)$, if $post'$ is the underlying non-reversed postorder. This can't exist, since in a postorder every parent comes after its children.
Let v be a vertex in G and $u_1, ..., u_k$ its earlier neighbours. Renumber all vertices by their number in *post*. We find by definition the edge (u_i, v) in the tree $ctree(G)$ and since *post* is a topological ordering $post(u_i) < post(v)$. Now if there was any other edge (u_j, v) not in the tree $ctree(G)$ with $post(u_j) > post(v)$ the vertex u_j would have been a child of v in the tree $ctree(G)$, but then u_j could not have been an earlier neighbour of v. ◀

The next lemma is needed in a later proof:

▶ **Lemma 8.** *The vertices on the cycle C_k around the outer face of the subgraph G_k of G are ordered in clockwise order by their values in pre.*

Proof. An induction over k gives the result. The first interesting case is $k = 3$, which is simply a triangle. Obviously the three vertices are ordered by their values in a preoder traversal as above. Adding another vertex to the outer face such that it preserves the canonical ordering increases k by one. Now that new vertex has a parent with smaller *pre*-value and the next vertex around the outer face has to have bigger *pre*-value, since it will be discovered from our new vertex. ◀

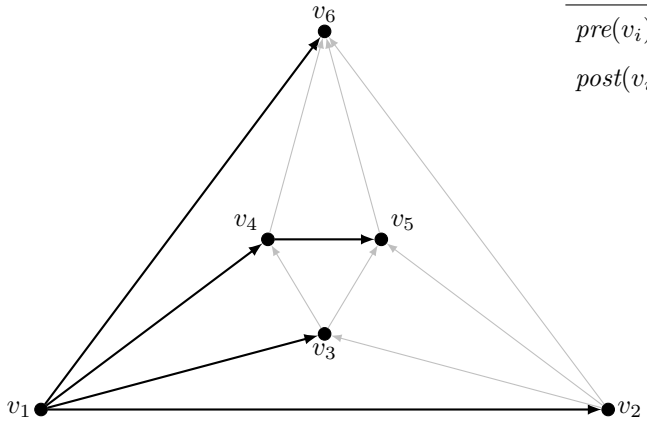
Now we can define the special permuation to build a connection between permutations and universal point sets.

▶ **Definition 9** (cperm)**.** Let G be a maximal planar graph and v_I a vertex in G, then we define $cperm(G)$ as the permutation which has at position $pre(v_i)$ the value $post(v_i)$.

Another way to derive $cperm(G)$ is to say that we traverse $ctree(G)$ in preorder and list the values of *post* for every vertex (See Figure 22 for an example).

▶ **Lemma 10.** *If G is a canonically-represented maximal planar graph, the permutation $\pi = cperm(G)$ is 213-avoiding.*

Proof. Take three arbitrary indices i, j, k with $1 \leq i < j < k \leq |\pi|$ and let π_i, π_j, π_k be their corresponding elements in π. Now the only interesting case is, when π_j is the smallest of those three values. We need to show that in this case $\pi_i > \pi_k$. Let's put together any property already known. Since $i < j < k$ the preorder value of v_i, v_j, v_k are in order $pre(v_i) < pre(v_j) < pre(v_k)$, if this was not the case the order of the indices had to be different. Additionally π_j is smaller than π_i and π_k, meaning that $post(v_j)$ has to be smaller than



	v_1	v_2	v_3	v_4	v_5	v_6
$pre(v_i)$	1	6	5	3	4	2
$post(v_i)$	1	2	3	4	5	6

$cperm(G) = 164532$

Figure 22 An example graph, its DFS tree, *pre* and *post* traversal as well as the permutation *cperm*.

$post(v_i)$ and $post(v_k)$. Using Lemma 6 we see that v_j is an ancestor of v_k, but no ancestor or descendant of v_i and with that v_i can not be an ancestor or descendant of v_k. But the only constellation making this possible is $post(v_i) > post(v_k)$, which leads in return to the permutation π having not a 213, but a 312 pattern. ◀

Note that $cperm(G)$ always has as its first element a 1, followed by the number of vertices of G and as a last element a 2. Additionally we can say that if σ is an $S_{n-3}(213)$ superpattern, we find $cperm(G)$ as pattern in $augment(\sigma)$.

5.3.1 Deriving coordinates from a permutation

After connecting permutations and graphs it is necessary to talk about how coordinates can be derived from a permutation. At a first glance it seems obvious to choose (i, σ_i) as coordinates, where i is an index in the permutation σ and σ_i the element at this position, but for us it will be better to use a stretched set of coordinates, defined in Definition 4. During the next three lemmas σ will be a permutation, q the length of σ, i, j, k, h four indices with $1 \leq i, j, k, h \leq q$ and p_i, p_j, p_k, p_h the corresponding points in $stretch(\sigma)$.

▶ **Lemma 11.** *Let $\sigma_i < \sigma_j$ and m the absolute slope of the line segment $p_i p_j$. Then the following equation holds: $q^{\sigma_j - 1} \leq m < q^{\sigma_j}$.*

Proof. Consider the minimum value of m. It is obtained, when $|j - i| = q - 1$ and $\sigma_i = \sigma_j - 1$. Putting this into the formula for m we get

$$m = \frac{q^{\sigma_j} - q^{\sigma_i}}{|j - i|} = \frac{q^{\sigma_j} - q^{\sigma_j - 1}}{q - 1} = q^{\sigma_j - 1}.$$

Next the maximum of m is reached if $|j - i| = 1$ and $\sigma_i = 1$. Evaluating m at these values gives us:

$$m = \frac{q^{\sigma_j} - q^{\sigma_i}}{|j - i|} = \frac{q^{\sigma_j} - q^1}{1} = q^{\sigma_j} - q < q^{\sigma_j}.$$

◀

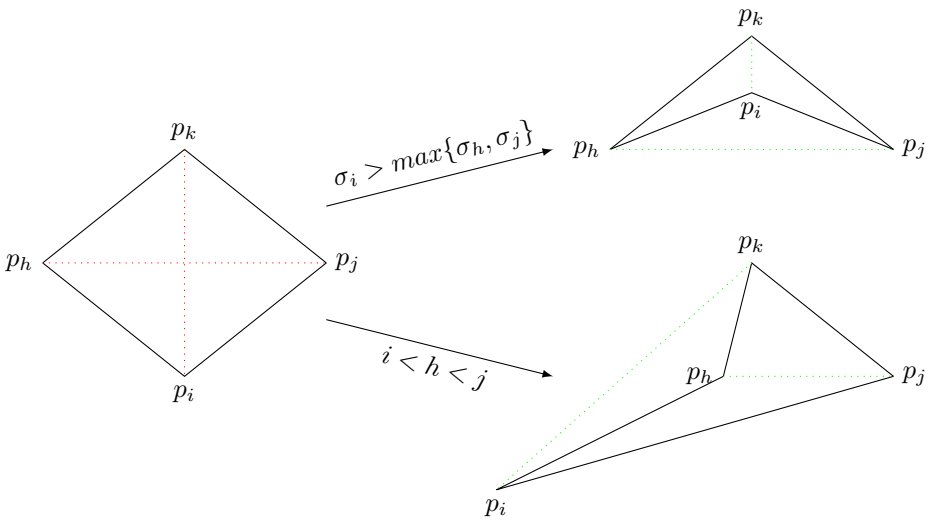


Figure 23 A rough sketch of the situation in Lemma 13. The points have no exact coordinates.

▶ **Lemma 12.** *Let $max\{\sigma_i, \sigma_j\} < \sigma_k$ and $i < j$. Then the points p_i, p_j and p_k are in the clockwise order p_i, p_k, p_j.*

Proof. This can be proven by comparing the slopes of p_ip_j and p_ip_k. Since p_i is always left of p_j and p_k is always above p_i and p_j, we only need to show that $m_{p_ip_k} > m_{p_ip_j}$. Consider the minimum value of $m_{p_ip_k}$ and the maximum value of $m_{p_ip_j}$:

$$min\ m_{p_ip_k} \geq q^{\sigma_k - 1} \geq q^{\sigma_j} > max\ m_{p_ip_j}$$

◀

▶ **Lemma 13.** *Let $max\{\sigma_h, \sigma_i, \sigma_j\} < \sigma_k$ and $h < j$. Then p_hp_j and p_ip_k cross if and only if $h < i < j$ and $max\{\sigma_h, \sigma_j\} > \sigma_i$.*

Proof. See Figure 23 as a sketch of the above situation. Keep in mind, the coordinates are stretched and that the stretching is an essential part in the proof, however since the stretch leads to exponential drawing space it is impossible to give an exact example. The proof can be split up into two parts. The first uses the condition, that σ_i has not the second-largest value, the second that $h < i < j$.
For the first case consider the triangles $p_hp_ip_j$ and $p_hp_kp_j$. If the lines p_ip_k and p_hp_j cross those triangles have to have opposite orientations, which can only be the case if the clockwise ordering of these triangles is p_h, p_j, p_i and p_h, p_k, p_j. Again meaning, by Lemma 12, that $\sigma_i < max\ \{\sigma_h, \sigma_j\}$.
The second case uses the triangles $p_ip_hp_k$ and $p_ip_jp_k$. Those triangles have opposite directions if and only their orientation is p_h, p_j, p_i and p_h, p_k, p_j, again, by Lemma 12, this can only be the case if $h < i < j$. ◀

5.3.2 Putting together a universal point set

Finally we are able to put a universal point set together for an arbitrary maximal planar graph. Note that at this point nothing is known about the size of such an universal point set.

▶ **Theorem 14.** *Let σ be a $S_{n-3}(213)$-avoiding superpattern and let $U_n = stretch(augment(\sigma))$. Then U_n is a universal point set for planar graphs with n vertices.*

Proof. The proof has two parts. In the first we state how to go from a graph through permutations to coordinates and in the second we show why those points give a drawing without crossings.
Let G be a maximal planar graph with canonical representation $v_1, v_2, ..., v_n$. Then we have a permutation $cperm(G)$ as above. Since σ is a superpattern we can find $cperm(G)$ as subpattern in $augment(\sigma)$. Let x_i denote positions in $augment(\sigma)$ such that the values at the x_i's form a pattern of type $cperm(G)$ and choose $y_i = |\, augment(\sigma)|^j = q^j$ with $j = augment(\sigma)_j$. The point set we get is $UG_n = \{(x_i, y_i) | 1 \leq x_i \leq q \text{ and } y_i = q^j\}$.
The remaining work is to show that there can not be a crossing with these points for any arbitrary maximal planar graph G. Consider four vertices $v_i v_j v_h v_k$ with edges $v_h v_j$ and $v_i v_k$ existing in G. Let's choose w.l.o.g. $post(v_k)$ as the largest $post$ value among those vertices and $pre(v_h) < pre(v_j)$. By lemma 13 a crossing occurs if and only if $pre(v_h) < pre(v_i) < pre(v_j)$ and $post(v_i) < max\{post(v_h), post(v_j)\}$. Assume this was the case, since the graph is canonically ordered and $v_i v_k$ is an edge in G, v_i has to be on the outer face of the graph G_l induced by $l = max\{post(v_h), post(v_j)\}$.
Now we will show that this can not be the case and that v_i has to lie in the interior of G_l. Assume $post(v_h) > post(v_j)$ and consider the graph G_{l-1} with v_i on its outer face. If we add v_h to G_{l-1} we get that v_h, since $pre(v_h) < pre(v_i)$, has an edge to a vertex before v_i. Additionally we get a new edge between v_h and v_j. Since G_l is in canonical order this edge has to be on the outer face and with $pre(v_i) < pre(v_j)$ we find v_i to not longer lie on the outer face of G_l and therefore the edge $v_i v_k$ can not exist. The case $j > h$ can be proven symmetrically. ◀

5.4 Chessboard representation

After the general approach it is time to turn towards graphs with bounded pathwidth, but before the final results can be proven we need some more definitions and a slightly different way to think about permutations.

▶ **Definition 15** (Chessboard representation)**.** Let π be a permutation, then define a column as a maximal ascending run in π and a row as a maximal ascending run in π^{-1}. The chessboard representation of the permutation is a $r \times c$ matrix M where r is the number of rows and c the number of columns. Every cell M_{ij} in the matrix has as value the number of elements in the intersection of the i-th row and j-th column.

An example for a chessboard representation can be seen in Figure 24. The represented permutation is $\pi = 165342$ and its inverse permutation is $\pi^{-1} = 164532$ can be derived as (i, π_i). This representation is now used to get the chessboard graph of π. This is simply the graph we get by connecting all non-empty neighbours in a row or column of M with an edge. To make this graph directed we let all edges point up- respectively rightwards.
The following lemma provides a characterization of 213-avoiding permutations and its chessboard representations. This result can later be used to prove size bounds for superpatterns of $\{213, 3412\}$-avoiding permutations, see Lemma 21. But first another definition:

▶ **Definition 16** (Directed plane forest)**.** Let G be a graph. We call G a directed plane forest, if G is planar, directed and every connected component of G is tree.

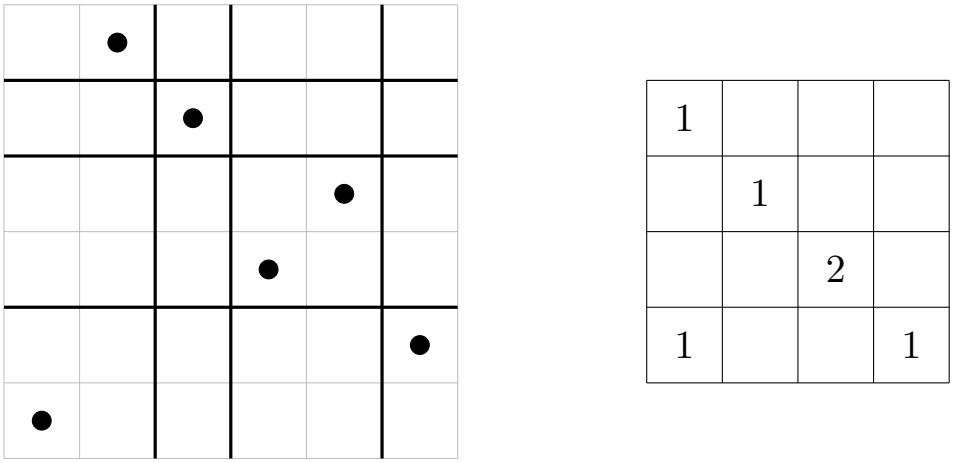


Figure 24 Scatterplot (left) and chessboard representation (right) of permutation 165342

▶ **Lemma 17.** *Given a permutation π, then it is* 213*-avoiding if and only if its chessboard representation and graph fulfil these three requirements:*

1. *equal number of rows and columns*
2. *the main diagonal (top left to bottom right) has only nonzero squares*
3. *the chessboard graph is a directed plane forest*

Proof. As a first case we will show that every permutation, whose chessboard representation fulfils these conditions is 213-avoiding. Condition three means above the diagonal there can only be squares with zeros, else we would not have a forest. Since condition two gives us a nonzero diagonal, every nonzero square below the diagonal has two outgoing edges, one upwards and one rightwards. Now if there was a 213 pattern the cells corresponding to the 2 and 1 would have to be below the diagonal, since else the cell corresponding to the 3 would be above the diagonal, but there all cells are 0. So we can assume the cell with the 3 to be below or on the diagonal. By definition there is an edge going to the right from the cell with the 2 and an edge going upward from the cell with the 1. So we can follow an upward path from the 1 until we hit a cell on the diagonal and a rightward path from the 2. But since the 2 is necessarily above and to the left of the 1 the paths meet at a nonzero cell or cross. Both cases violate the third condition.

The other direction is a bit more difficult. Look at the maximal decreasing subsequence L in a 213-avoiding permutation π. Maximal means there is no index i in L with the property that there exist another index i' not in L, such that $L \setminus \{i\} \cup \{i'\}$ and $i' > i$ is an equally long decreasing subsequence. What follows is, that there can not be any $i' > i$ with $\pi_{i'} > \pi_i$, since such an i' would either violate the maximality of L or form a 213 pattern with two elements in L.

We can see that for every column the maximal element has to be in L since if it was not, it would form a 213 pattern together with the next element in the permutation and the next element in L. Symmetrically one can argue that for every row its maximal element is in L. Now both the maximal elements of all columns and all rows are in L, which in return means that every square on the diagonal has an element in L, because above the diagonal no nonzero square can be found since such a square directly violates the assumption that π is 213 avoiding. With the same argument no two edges can cross below the diagonal. ◀

5.5 Subsequence majorization

The next part introduces a very useful sequence, which we use later to majorize other finite integer sequences. Let ξ be the sequence, where $\xi_i = i \oplus (i-1)$ and $\oplus$ is the bitwise xor operation. Applying this rule leads to the sequence:

$$1, 3, 1, 7, 1, 3, 1, 15, 1, 3, 1, 7, 1, 3, 1, 31, ...$$

Apparently this sequence has at every position i which is a power of two the value $\xi_i = 2 * i - 1$ and after such a value we have a copy of the $i-1$ values from before the i-th position. Now we can show that for the sum $\zeta_n = \sum_{i=1}^{n} \xi_n$ the following holds:

▶ **Lemma 18.** *$\zeta_n \leq n \log_2 n + n$.*

Proof. Let $n = 2^x$ for any $x \in \mathbb{N}$. Then $\zeta_n = \sum_{i=1}^{2^x} \xi_i = 2^x(x+1)$. An induction gives the result. Test it explicitly for $x = 0$:

$$\zeta_1 = \sum_{i=1}^{1} \xi_i = 1 = 2^0(0+1).$$

Execute the induction step $x \to x+1$:

$$\begin{aligned}
\sum_{i=1}^{2^{x+1}} \xi_i &= \sum_{i=1}^{2^x} \xi_i + \sum_{i=2^x+1}^{2^{x+1}} \xi_i \\
&= 2^x(x+1) + 2^x(x+1) + (2^{x+2} - 1) - (2^{x+1} - 1) \\
&= 2^x(x+1) + 2^x(x+1) + 2^{x+1} \\
&= 2^{x+1}((x+1)+1) \\
&= n \log_2 n + n
\end{aligned}$$

If n is not a power of two we will use the binary representation of a number n. Let $k = \lfloor \log_2 n \rfloor$ then $n = \sum_{i=1}^{k} 2^i b_i$ with $b_i \in \{0, 1\}$. We can express ζ_n in terms of its binary representation as:

$$\zeta_n = \sum_{i=0}^{k} i b_i 2^i + b_i 2^i$$

and finally get the result:

$$\zeta_n \leq \sum_{i=0}^{k} \log_2 n b_i 2^i + b_i 2^i = n \log_2 n + n.$$

◀

The other property of ξ we are interested in, is that any finite sequence can be majorized by a subsequence of ξ.

▶ **Lemma 19** (Majorization). *If $\alpha_1, ..., \alpha_k$ is a finite sequence with sum n, then we find a subsequence $\beta_1, \beta_2, ..., \beta_k$ in the first n terms of ξ such that $\alpha_i \leq \beta_i$ for every $i \leq k$.*

Proof. Let $q = 2^x$ be the largest power of two which is still smaller or equal than n. The value of ξ_q is $2 * 2^x - 1$ which is already bigger or equal than n and with that bigger or equal than the maximum element α_i of the sequence. Next, search the smallest i, such that

$\sum_{j=1}^{i} \alpha_j \geq 2^x$, choose $\beta_i = \xi_q$. Continue recursively on both sides of ξ_q and both sides of α_i. This works since:

$$\sum_{j=1}^{i-1} \alpha_j < q$$

$$\sum_{j=i+1}^{k} \alpha_j < q$$

else the chosen i would not have been minimal or we chose a q not big enough. Choose the next β's as above. ◀

5.6 Strahler Number and Applications

All the above lemmas and theorems never showed any size bound for superpatterns, this will change now as we are going to prove that any 213-avoiding permutation π with Strahler number at most s has superpatterns in $O(n \log^{s-1} n)$.

▶ **Definition 20** (Strahler number)**.** Let T be a tree, then the Strahler number of a vertex in T is either the maximum of the Strahler number s of its children or if there are multiple children with equal Strahler number it is $s+1$. If the vertex itself is a leaf its Strahler number is 1.

Following this definition the Strahler number can easily be computed by a bottom-up traversal of T. As we have seen above, every 213-avoiding permutation has a directed plane forest as its chessboard graph. What we want to do now is define Strahler numbers for permutations, but to reach that goal an augmentation for those forests is needed. Lets look at the graph obtained from the chessboard representation of a 213-avoiding permutation π. We can connect two trees simply by adding an element to the representation. If a square in the chessboard graph contains more than one element we give the corresponding vertex a weight equal to the number of elements. The resulting new permutation π' has length at most $2n-1$, since the worst case would be that all vertices are on the main diagonal and we need to add $n-1$ new ones to build a tree. Additionally it is still a 213-avoiding permutation, which we can check by using the three requirements from Lemma 17. First we never add a new row or column, secondly since we add neither a row nor a column and we started with a 213-avoiding permutation, the diagonal still has to be filled with nonzero squares and the third statement simply follows from the fact, that a tree is a forest with only one element (See Figure 25 for an example). Finally we will call such an augmentation the *tree augmentation* of π and obviously every superpattern for a tree augmentation is also a superpattern for π itself.
With the graph augmented to a tree, we can define the Strahler number of an arbitrary permutation π as the minimum Strahler number over all tree augmented chessboard graphs of π.

▶ **Lemma 21.** *If π is a permutation and has Strahler number s, the chessboard graph has a pattern which is a complete binary tree of height $\Omega(s)$.*

Proof. Augment the permutation π in the way shown above. The resulting permutation π' has the same Strahler number as π. A binary tree with height in $\Omega(s)$ can be found as a subset of vertices in π' and all these vertices belong to π itself. As a start vertex we choose a

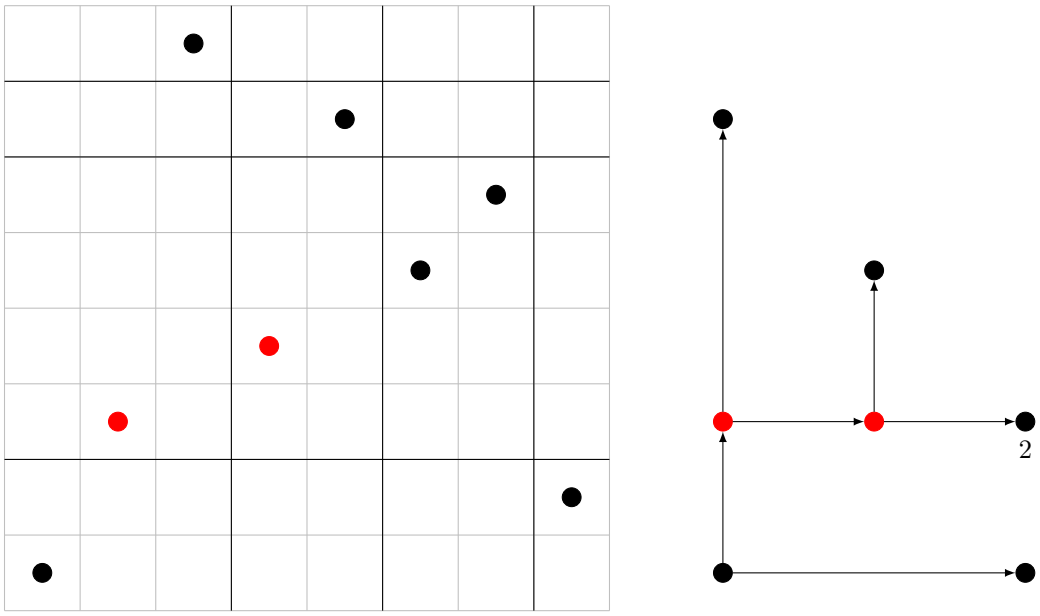


Figure 25 Tree augmentation and the resulting chessboard graph of the same permutation as in Figure 24. The two red vertices were added to augment the graph to a binary tree.

vertex with Strahler number s. If it belongs to π itself the two children have Strahler number $s-1$, if this was not the case a smaller tree augmentation could have been chosen. If such a vertex doesn't exist all vertices with Strahler number s belong to π'. When we find a subtree of such a vertex whose root has Strahler number $s-1$ this vertex has to belong to π with the same argument as above. Since we are in a binary tree there have to be two children with stahler number $s-1$. Choosing one of those subtrees and recurring on it gives a binary tree with at least $s/2$ vertices on its shortest root-to-leaf path. ◀

▶ **Theorem 22.** *The* 213*-avoiding permutations with Strahler number* s *have superpatterns of length* $O(n \log^{s-1} n)$.

Proof. The result can be shown by two inductions over s. The first one enables us to find a superpattern for a general 213-avoiding permutation, the second one gives the size bound.
Let $s = 1$, the only tree with this Strahler number consists of a directed path from root to leaf. The single permutation with such an augmented chessboard graph is the identity permutation. Following our definition we only have one vertex with weight equal to the length of the permutation as the chessboard graph, which can be replaced by a path from root to leaf of length equal to the size of the permutation. With this the permutations with Strahler number $s = 1$ have superpatterns of length n.
The induction step needs some preparation. Let $S(x, y)$ be a superpattern of an augmented permutation of length x and Strahler number y. We know such a superpattern exists for permutations with Strahler number $y < s$.

Let S_{n+1} be the empty permutation and $a_1, a_2, ..., a_k$ a sequence of increasing integers. Inductively define S_i for $i = n, n-1, ..., 1$ as:

$$S_i = a_1, a_2, ..., a_{\xi_i}\ S(\xi_i, s-1)\ S_{i+1}\ S(\xi_i, s-1).$$

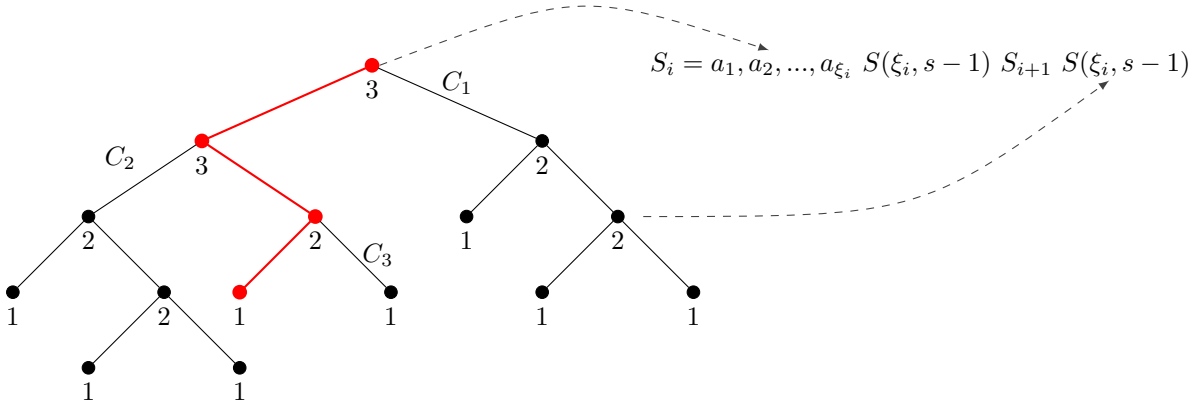


Figure 26 An example tree with the C_i's as in Lemma 21. The root of C_1 is mapped to the initial sequence, while the subtree is mapped onto one of the $S(\xi_i, s-1)$ copies.

Set $S(n,s) = S_1$. Let π' be an augmented permutation of length n' with Strahler number smaller or equal s. We can find π' as a pattern in $S(n',s)$ like this: Take a root-to-leaf path in the chessboard graph of π'. We can choose this path in such a way, that all vertices with Strahler number s are on it (See Figure 26). Let v be the i-th vertex on this path and C_i the subgraph of the chessboard graph which consists of v, the child not on the root-to-leaf path and the remaining tree below this child. Now if c_i denotes the size of such a subgraph the sum over all the c_i's is the size of the graph n'. Applying Lemma 19 we get a sequence t_i as a subsequence of ξ such that $c_i \leq t_i$ for all i. Now map the elements in the root vertex of any C_i to the initial increasing sequence of the S_i, which corresponds to t_i. Map the remaining part of C_i to one of the two $S(t_i, s-1)$, whichever is on the correct side of S_{i+1}. The mapping of the subtree is correct by induction hypothesis and the elements in the root vertex can be mapped to the initial sequence, since $c_i \leq t_i$ and the elements in the root vertex can only be an ascending sequence of elements in the permutation.
Finally the size bound for those superpatterns follows from another induction. As a start consider $s = 1$, as we saw above we can choose superpatterns of length n and this obviously holds with our considered bound $n \log^{1-1} n = n$. For the induction step we have to analyse S_1:

$$S_1 = \underbrace{a_1 a_2 ... a_k}_{\leq n \log n + n} \underbrace{S(\xi_1, s-1)S(\xi_2, s-1)...S(\xi_n, s-1)}_{=\sum_{i=1}^{n} \xi_i \log^{s-2} \xi_i} \underbrace{S(\xi_n, s-1)...S(\xi_2, s-1)S(\xi_1, s-1)}_{=\sum_{i=1}^{n} \xi_i \log^{s-2} \xi_i}.$$

The first a_i elements are simply all elements from all S_i in the $S(n,s)$ and with lemma 18 we know that there can only be $O(n \log n)$ of them. All the recursively defined $S(\xi_i, s-1)$ have size $O(\xi_i \log^{s-2} \xi)$ by induction. The only thing left is to show that:

$$\sum_{i=1}^{n} \xi_i \log^{s-2} \xi_i = O(n \log^{s-1} n).$$

Let ξ_m be the maximum ξ_i, then we can do the following estimation

$$\begin{aligned}\sum_{i=1}^{n} \xi_i \log^{s-2} \xi_i &\leq \sum_{i=1}^{n} \xi_i \log^{s-2} \xi_m \\ &\leq \log^{s-2} \xi_m \sum_{i=1}^{n} \xi_i \\ &\leq \log^{s-2} \xi_m (n \log n) \\ &= O(n \log^{s-1} n).\end{aligned}$$

◀

With Theorem 22 and Lemma 21 we get the following corollary and with it a size bound for superpatterns depending on the Strahler number of the permutation:

▶ **Corollary 23.** *Let π be a 213-avoiding permutation, π' a tree augmentation and h the number of vertices on the longest root-to-leaf path in π'. If we choose the augmentation such that h is minimized the $\{213, \pi\}$-avoiding permutations have superpatterns of length $O(n \log^{2h-1} n)$.*

Proof. To use Theorem 22 here we first need to show, that the Strahler number of a $\{213, \pi\}$-avoiding permutation σ with the given conditions is at most $2h - 1$. This follows directly from Lemma 21 since, if σ's tree augmented chessboard graph σ' had Strahler number bigger than $2h - 1$ the chessboard graph of σ' already would have a complete binary tree of height h and π would be a subpattern of σ. Now simply use the result from Theorem 22. ◀

5.7 Graphs with bounded Pathwidth

The last section of the paper will introduce graphs with bounded pathwidth and the resulting superpatterns. To get the *pathwidth* of a graph G we look at an interval supergraph of G. If we choose such a supergraph to minimize the size of its maximum clique, the pathwidth of G is the size of the maximum clique minus one.

5.7.1 Two Lemmas for graphs with bounded pathwidth

Before the final result, two preparing lemmas are needed. The first gives a construction to augment any planar graph to a maximal planar one, the second one is needed to apply the methods from Section 5.3.

▶ **Lemma 24.** *Given a non-maximal planar graph G, there exists a maximal planar supergraph G' on the same set of vertices and a canonical labeling of G'. Let T be the depth-first search tree of G', then all edges in T which do not have v_1 as an endpoint belong to G.*

Proof. In the proof we assume that G has a planar embedding, thus we can talk about left and right. Figure 27 displays the general idea. We start by choosing arbitrary edge v_1v_2 as a base. At all times we maintain a subgraph S of G with all already added vertices and edges and a cycle C_k representing the outer face of S after adding k vertices, ensuring v_1v_2 to be on C_k. After the base edge was chosen S consists of the two vertices v_1, v_2 and the edge connecting them. Further let $right_k(v)$ be the rightmost neighbour of a vertex v on C_k and $left_k(v)$ its leftmost.

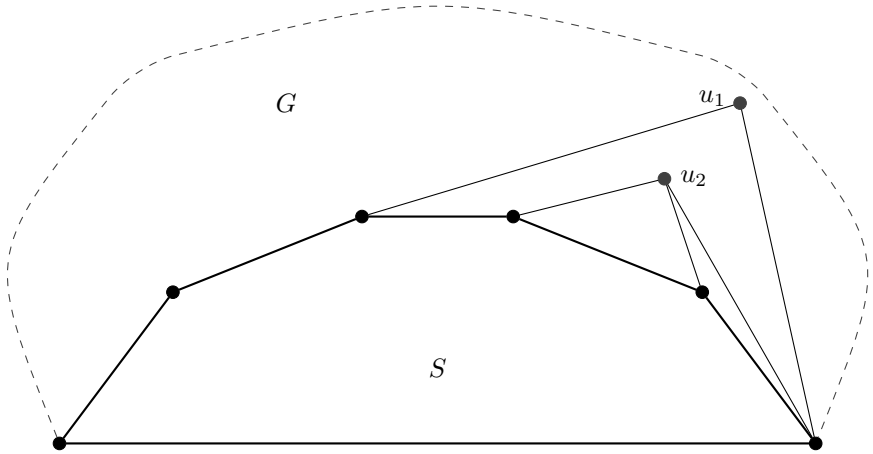


Figure 27 Example of a graph being augmented to a maximal planar graph with a canonical embedding. In this case the next vertex to add would be u_2 by case one.

We need to choose vertices u and add them to S until no vertex is left in $G \setminus S$. Note that a vertex, which is adjacent to S, has to be connected to vertices from C_k and shares no edges with vertices in S. Distinguish two cases:

- If vertices u_i exist with an edge $(u_i, v_j) \in E$ and $v_j \neq v_2$ we choose among them the vertex which has the rightmost $\mathit{left}_k(u_i)$. Ties can be broken up by choosing the u_i most to the right in the embedding. Again distinguish two cases:
  - u_i has two or more neighbours in S: add u_i with edges to all vertices between $\mathit{right}_k(v)$ and $\mathit{left}_k(v)$.
  - u_i has only one neighbour ($\mathit{left}_k(v) = \mathit{right}_k(v)$): add u_i together with edges to $\mathit{left}_k(v)$ and the right neighbour of $\mathit{left}_k(v)$ on C_k.

  Both cases can not make S nonplanar since two vertices u_i and u_j can not have crossing intervals of neighbours on C_k. Else G could not have been planar from the start or we would have chosen the vertex with the nested interval of neighbours before the surrounding one.
- If all candidate vertices u_i have v_2 as their only neighbour simply choose the leftmost neighbour of v_2, which is not in S. Now add the chosen vertex u and edges to all vertices in C_k.

The last vertex added to S will be connected to all vertices in C_{n-1} and the resulting graph $S = G'$ is a supergraph of G with the desired properties. We still have to show that if we do a depth-first search as in section 5.3 we only traverse edges not in the original G which have v_1 as their starting point. This is rather easy: We start from v_1, and if we traverse the graph we only visit the leftmost edges of any vertex which is either connected to v_1 by the second case or is an original edge, since we only added edges on the right in case one. ◀

▶ **Lemma 25.** *Derive a permutation π from a planar graph G like we did in Section 5.3 and let σ be a pattern in π whose chessboard graph is a tree T' itself. Then we can already get a tree isomorphic to T' by contracting edges in a DFS tree T of G.*

Proof. Once more we will use Lemma 6. Let π be a permutation derived from a maximal planar graph G and T its DFS tree as in section 5.3. Additionally let σ be a subpattern of π and T' a tree and chessboard graph of σ. Take two vertices u and v from T'. Following the

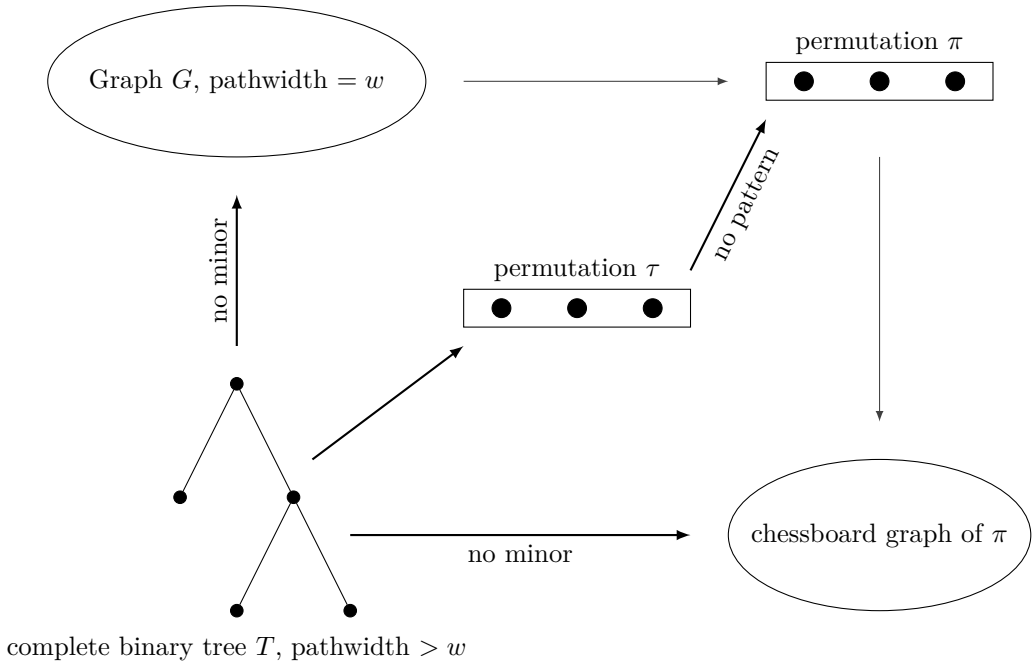


Figure 28 A rough diagramm displaying proof of Theorem 26

definition of a chessboard graph v has to be above and to the right of u to be a descendant. For their position in σ this means, that u has to be after v and that its value has to be bigger. For the corresponding vertices in T it means, that v has to be after u in *pre* and *post*. We see that the ancestor-descendant relationship of two vertices in T and T' is always the same. To get T' from T only contract every vertex in T which is not in T'. ◀

5.7.2 Size bound for graphs with bounded pathwidth

Now to our final theorem and the desired size bound for universal point sets of graphs with bounded pathwidth.

▶ **Theorem 26.** *Graphs with pathwidth w have universal point sets of size $O(n \log^{O(1)} n)$ (You can find a rough sketch of what we do in this proof in Figure 28).*

Proof. Let G be a maximal planar graph with pathwidth w (if it is not maximal planar use Lemma 24 to augment it). Now use Lemma 25. Derive a permutation π from G as in Section 5.3. One property of pathwidth, which we will use, is that it is closed under minors. This means a graph with pathwidth w can not have a subgraph with bigger pathwidth. Let T be a complete binary tree with pathwidth at least $w + 1$. T can not be a minor of G since G's pathwidth is smaller than T's. Additionally the chessboard graph of π can not have a pattern isomorphic to T. But that in return means, that the pattern τ derived from T can not be present as a pattern in π. Since if it was we could choose the corresponding vertices in the chessboard graph and use them to find T as a minor of G.
Finally we use corollary 23 and the method as in Section 5.3 with our permutation avoiding 213 and τ, leaving us with a universal point set of size $O(n \log^{O(1)} n)$. ◀

5.8 Summary

Lets recollect what we saw throughout the write up. First we talked about a connection between superpatterns and universal point sets. This connection was built through reverse postorder and preorder traversal, which we used to derive a permutation for a planar graph. Next we proved that this permutation can be used to create a universal point set if it is 213-avoiding. After we created this connection we turned to several smaller things like subsequence majorization, chessboard representation and Strahler number. With them in our box of tools we were able to build a superpattern closely related to the Strahler number of a permutation. Finally we presented the promised bound of $O(n \log^{O(1)} n)$ for graphs with bounded pathwidth in the last section of the write up.

References

1 Michael J. Bannister, Zhanpeng Cheng, William E. Devanny, and David Eppstein. Superpatterns and universal point sets. *CoRR*, abs/1308.0403, 2013.

2 Michael J. Bannister, Zhanpeng Cheng, William E. Devanny, and David Eppstein. Superpatterns and universal point sets. In Stephen Wismath and Alexander Wolff, editors, *Graph Drawing*, volume 8242 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 2013.

3 Franz-Josef Brandenburg. Drawing planar graphs on (8/9)*n^2 area. *Electronic Notes in Discrete Mathematics*, 31:37–40, 2008.

4 P. Gritzmann, B. Mohar, J. Pach, and R. Pollack. E3341. *The American Mathematical Monthly*, 98(2):pp. 165–166, 1991.

6 Flips in Triangulations

Lea Köckert

In dieser Arbeit wird untersucht, ob zwei unterschiedliche, kombinatorische Triangulierungen durch Flippen der Kanten ineinander überführt werden können und wie viele Flips eine solche Umwandlung maximal und mindestens benötigt. Hierbei ist es bereits ausreichend, wenn die durch Flippen veränderte Triangulierungen isomorph zu der anderen ist.

Eine *Triangulierung* ist eine mit Kanten verbundene Knotenmenge, so dass jede Facette aus drei Knoten besteht und somit ein Dreieck bildet. Eine Triangulierung ist planar, d.h. dass keine der Kanten sich kreuzt. Außerdem sind die im Folgendem betrachteten Triangulierungen maximal planar, das bedeutet, dass jede Kante, die ohne eine andere zu kreuzen, in der Triangulierung vorhanden sein kann, auch vorhanden ist.

Im kombinatorischen Bereich sind die Knoten nicht an feste Punkte in einer Ebene gebunden, sondern können verschoben werden bzw. es sind gebogene Kanten erlaubt. Jeder Knoten kennt demzufolge nur die Reihenfolge seiner Nachbarknoten, aber nicht deren feste Position. Beim Flippen einer Kante wird das Viereck betrachtet, dass zwei Dreiecke der Triangulierung mit einer geteilten Kante bilden. Nun wird in diesem Viereck bestehend aus den Punkten a, b, c und d, die geteilte Kante e, die zugleich einer Diagonalen entspricht, *geflippt*, sodass sie nicht mehr zu a und c inzident ist, sondern zu b und d (Abbildung 29).

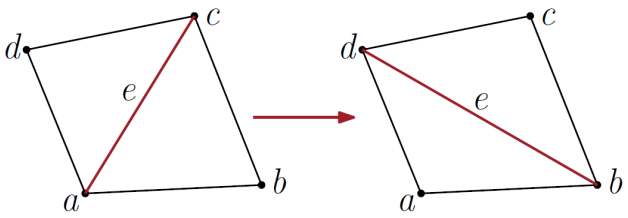


Abbildung 29 Flip der Kante e

6.1 Einleitung

Zunächst hat Wagner [11] im Zusammenhang mit dem Vierfarbenproblem bewiesen, dass zwei Triangulierungen durch das Flippen der Kanten ineinander transformiert werden können und sich dann mit der Frage beschäftigt, wie viele Flips dazu maximal benötigt werden. Bei dem Vierfarbenproblem wird im Allgemeinen eine Landkarte betrachtet, die mit vier Farben so eingefärbt werden soll, dass keine zwei benachbarten Länder dieselbe Farbe haben.

Um zu beweisen, dass zwei Triangulierungen durch eine Sequenz von Flips ineinander überführbar sind, wandelt Wagner diese zunächst in eine allgemeine Darstellung um, indem er zwei Knoten dominant macht. Das heißt, dass diese zwei Knoten zu allen anderen benachbart sind. Er zeigt, dass alle Triangulierungen in diese Darstellung, die *kanonische Triangulierung*, umgeformt werden können. Abbildung 30 veranschaulicht diesen Vorgang. Demnach können die Triangulierungen auch ineinander transformiert werden, da eine dieser Sequenzen dazu einfach rückwärts betrachtet werden kann.

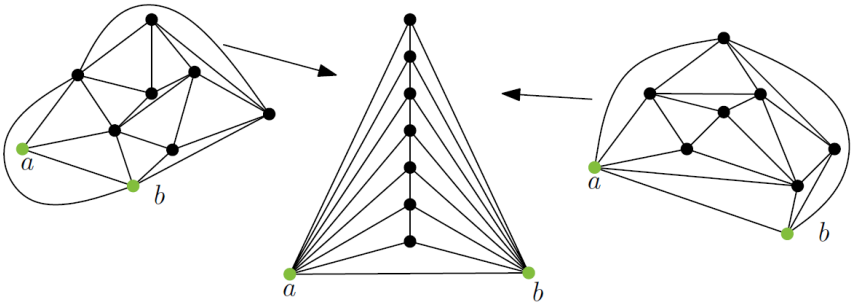


Abbildung 30 Transformation zweier Triangulierungen über die kanonische Triangulierung

6.2 Ähnliche Themen

Das dieser Ausarbeitung zu Grunde liegende Paper von Bose et al [5] führt sämtliche Forschungsergebnisse zu dem Thema Flipdistanz zweier Triangulierungen im kombinatorischen Bereich zusammen, wobei Distanz die minimale Anzahl benötigter Flips für eine Transformation bezeichnet.

In diesem Abschnitt möchten wir uns der Frage widmen, was für Ergebnisse es in den Bereichen der Geometrie und der Kombinatorik im Bezug auf die Bestimmung der benötigten Flips, um zwei verschiedene Triangulierungen ineinander zu überführen, gibt.

Im geometrischen Bereich gilt zu beachten, dass die Knoten an festen Orten im Raum sind und nur geradlinige Kanten erlaubt sind.

Es kann allgemein noch zwischen der Distanz von isomorphen Triangulierungen und sogenannten *edge-labelled* Triangulierungen unterschieden werden. Der erste Fall wird in dieser Arbeit im Bereich Kombinatorik näher untersucht und im zweiten Fall handelt es sich um den Abstand von Triangulierungen mit fest bezeichneten Kanten, die es gilt an die richtige Stelle zu „Flippen“. Es ist offensichtlich, dass dieses Problem potentiell mehr Flips benötigt.

Bereits 1992 haben Sleator et al [10] für den kombinatorischen Fall von „edge-labelled“ Triangulierungen gezeigt, dass die Anzahl benötigter Flips in $\Theta(n \log n)$ liegt. Wobei sie ebenfalls die Vorarbeit von Wagner benutzen und zunächst die Triangulierung in die kanonische Triangulierung überführen. Diese Ergebnisse wurden 2013 von Bose et al. [4] bestätigt. Bose et al. haben ebenfalls für den geometrischen Fall gezeigt, dass diese Schranke auch für „edge-labelled“ Triangulierungen gilt, die wie konvexe Polygone aufgebaut sind.

Außerdem wurde in demselben Paper die Anwendung von simultanen Flips untersucht. Hierbei handelt es sich um die Frage, wie viele Flips gleichzeitig, also *simultan*, ausgeführt werden können, wobei pro Facette maximal eine Kanten auf einmal geflippt werden darf und in wiefern sich das auf die Gesamtanzahl von simultanen Flips auswirkt. Hierfür wurde die *simultane Flipdistanz* zweier geometrischer Triangulierungen für „unlabelled“ und für „edge-labelled“ Dreiecksnetze bestimmt. Im „unlabelled“ Fall beträgt die Schranke $O(\log n)$. Allerdings wurde ebenfalls gezeigt, dass es auch Fälle gibt in denen $\log n$ simultane Flips notwendig sind. Für die „edge-labelled“ Triangulierungen wurde eine maximale Anzahl von $\log^2 n$ benötigten simultanen Flips bewiesen. Für nähere Informationen zum Bereich der simultanen Flips siehe [4].

Über zwei Triangulierungen von konvexen Polygonen konnte bis jetzt keine Aussage getroffen werden, ob die Berechnung des minimalen Abstands in P liegt oder NP-vollständig ist. 2012 wurde von Lubiw et al. [7] gezeigt, dass zwei allgemeinere Probleme als jenes NP-vollständig sind. Bei diesen Verallgemeinerungen handelt es sich um die Berechnung der

minimalen Anzahl an benötigter Flips im Falle eines Polygons mit Löchern und im Falle einer Menge an Punkten in einer Ebene. Im darauffolgendem Jahr erschien eine Publikation von Aichholzer et al [2] in der bewiesen wurde, dass die Berechnung für ein einfaches Polygon ebenfalls NP-vollständig ist.

In der Geometrie sind die Kantenflips vor allem wegen der Delaunay Triangulierung so populär. Hierbei handelt es sich um eine spezielle Triangulierung von Punkten mit der der Graph in Zellen aufgeteilt werden kann, für die jeweils gilt, dass der Knoten im Mittelpunkt dieser Zelle zu jedem anderen Punkt außerhalb seiner Zelle einen größeren Abstand hat, als der Mindestabstand von diesem Punkt zu einem anderen Knoten. Der Vorteil von solchen Triangulierungen ist, dass sie eher gleichseitige Dreiecke beinhaltet als lange dünne Dreiecke.

6.3 Maximale Anzahl benötigter Flips für eine Transformation

Aus Wagners Paper [11], veröffentlicht 1936, geht für die Transformation zweier Triangulierungen eine quadratische Anzahl benötigter Flips hervor. In diesem Abschnitt wird diese obere Schranke durch den Beweis von Negami und Nakomoto [9] gezeigt. Diese formulieren dazu folgendes Lemma 1:

▶ **Lemma 1** (Negami and Nakamoto [9], Theorem 1). *Jede Triangulierung auf n Knoten kann durch eine Sequenz von $O(n^2)$ Flips in die kanonische Triangulierung Δ_n transformiert werden*

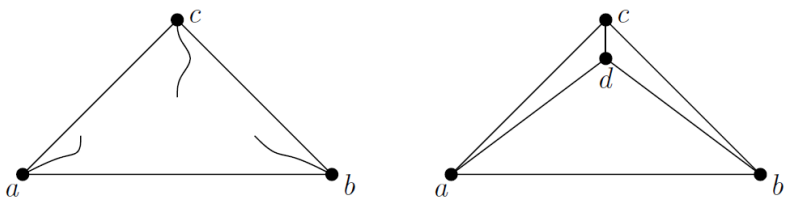


Abbildung 31 Die äußere Facette abc einer Triangulierung, sobald c angepasst wurde, kann der Subgraph G/c betrachtet werden

Beweis. Dieser Beweis folgt durch eine vollständige Induktion.

Zunächst wird gezeigt, dass es in dem äußersten Dreieck der Triangulierung immer einen Flip gibt, der den Grad des Knotens c verringert der nicht zu den zu *dominierenden* Knoten a und b gehört. Dieser Schritt wird solange durchgeführt, bis der Grad von c auf drei reduziert wurde. Vergleiche hierzu Abbildung 31.

Im nächsten Schritt wird der Graph ohne den Knoten c, der zuvor betrachtet wurde, untersucht. Wir sehen uns wieder die äußere Facette an, diese wird nun durch abd gebildet. Dies ist analog zu der vorherigen Situation. Wieder wird der Grad immer weiter verringert, in diesem Fall allerdings nicht bis drei sondern vier. Es existiert eine weitere Kante, die den Knoten d mit der eigentlich äußeren Facetten abc verbindet. Dieser Vorgang wird solange wiederholt, bis unsere aktuell betrachtete Facette keinen weiteren Knoten enthält.

Im Folgenden wird die Behauptung bewiesen, dass es immer einen legalen Flip gibt, der es ermöglicht den Grad des Knotens c zu verringern.

Dazu wird betrachtet wie die Kante ausgewählt wird, die geflippt werden muss, um den Grad von c zu verringern.

Angenommen c hat die Knoten $a, u, v, w, \ldots, b$ entgegen des Uhrzeigersinns als Nachbarn. Aufgrund der Tatsache, dass es sich um eine Triangulierung handelt, sind a und u bereits verbunden (siehe hierzu Abbildung 32). Falls v noch nicht mit a verbunden ist, Abbildung 32 oben links, kann die Kante uc zu av geflippt werden, wobei der Grad von c verringert wird, wie in Abbildung 32 oben rechts zu sehen. Wenn diese Kante schon existiert, bilden $cuvw$ ein Viereck mit der Kante cv als Diagonale, dies ist in Abbildung 32 unten links veranschaulicht. Nun kann der Grad von c reduziert werden, indem diese Kante cv des Vierecks geflippt wird (Abbildung 32 unten rechts).

Dieser Vorgang wird für jeden Nachbarn von c solange durchgeführt bis der Grad von c drei ist. Danach wird der noch übriggebliebene Nachbar von c, der nicht a oder b ist, auf dieselbe Weise behandelt. Dieser ist zusätzlich noch zu c verbunden, weswegen der Grad dieses Knoten nur bis vier reduziert. Dies wiederum wird solange durchgeführt bis alle Knoten der Triangulierung bis auf a und b einen Grad von drei bzw. vier haben.

In diesem Fall befindet sich der Graph in der kanonischen Triangulierung mit den dominanten Knoten a und b. Hierbei kann die Anzahl benötigter Flips mit $O(n^2)$ abgeschätzt werden, da wir den Grad jedes Knotens außer für a und b, also $(n-2)$-Mal, im schlimmsten Fall $(k_i - 4)$-Mal reduzieren müssen, wobei $k_i = (n-i)$ die Anzahl Knoten im jeweiligen Subgraphen ist.

◀

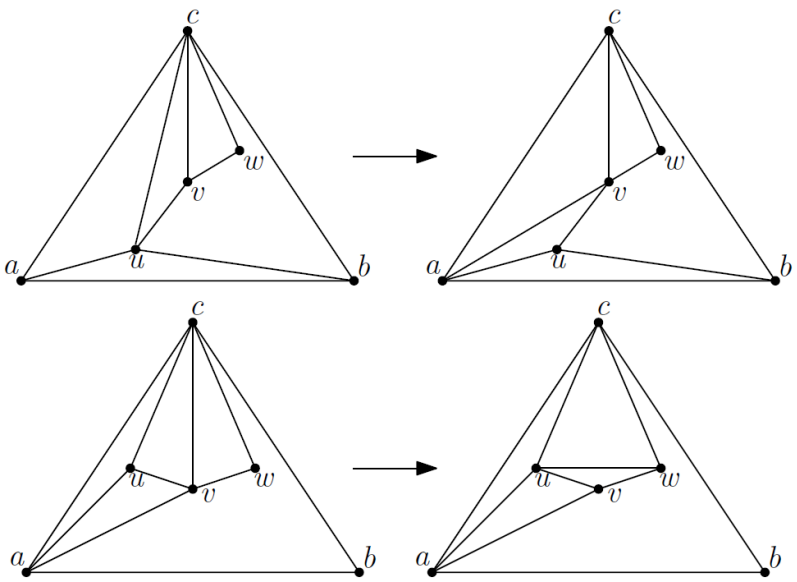


Abbildung 32 Entweder ist a mit dem Nachbar v von c verbunden oder nicht. Davon hängt ab, welche der Kanten geflippt wird.

6.3.1 Verbesserung der oberen Schranke

Komuro [6] zeigt, dass eine lineare Anzahl an Flips ausreicht um eine Triangulierung in die kanonische Repräsentation zu bringen (Ohne Beweis).

▶ **Theorem 2** (Komuro [6], Theorem 1). *Zwei Triangulierungen mit n Knoten können bei $n \geq 13$ durch höchstens $8n-54$ Kantenflips und bei $n \geq 7$ durch höchstens $8n-48$ Kantenflips in einander umgeformt werden.*

Im Folgenden Abschnitt wird zunächst der Ansatz von Mori et al. [8] vorgestellt, wie eine hamiltonische Triangulierung in eine kanonische Darstellung überführt werden kann. Im darauffolgendem Abschnitt wird der darauf aufbauende Ansatz von Bose et al. [3] betrachtet, bei dem zuerst eine Triangulierung in eine hamiltonische umgewandelt wird und danach mit Mori et al.s Ergebnissen in die kanonische überführt wird.

6.3.2 Spezialfall: Hamiltonische Triangulierung

Die obere Schranke von Komuro (Theorem 2) konnte durch Mori et al. [8] noch weiter verbessert werden. Dabei nutzen diese die Eigenschaften von hamiltonischen Triangulierungen. *Hamiltonische Triangulierungen* haben immer einen hamiltonischen Kreis. Dieser Kreis enthält jeden Knoten der Triangulierung genau einmal. Ein Beispiel eines solchen Kreises ist in Abbildung 33 veranschaulicht.

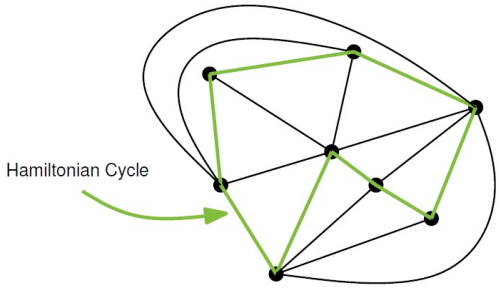


Abbildung 33 Eine hamiltonische Triangulierung mit hamiltonischen Kreis

Ein hamiltonischer Kreis kann dazu benutzt werden die Triangulierung in zwei außerplanare Subgraphen zu zerlegen.

▶ **Definition 3** (Außerplanare Graphen). *Außerplanare Graphen* sind Graphen, deren Knoten sich alle auf der konvexen Hülle befinden.

Dabei besteht der eine Subgraph aus dem Kreis und allen innenliegenden Kanten (Abbildung 34 links) und der andere ebenfalls aus diesem Kreis und allen außenliegenden Kanten (Abbildung 34 rechts). Somit sind bei beiden Subgraphen alle Knoten auf dem Kreis.

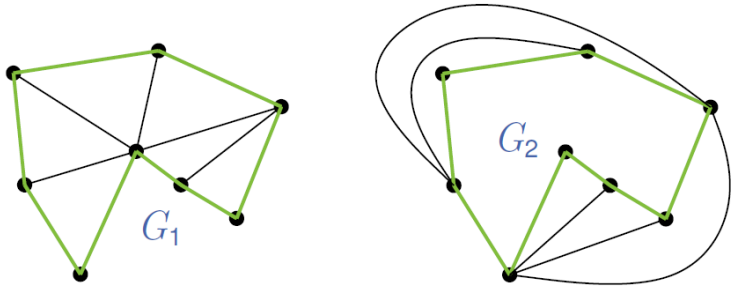


Abbildung 34 Durch die Zerlegung entstehen zwei außerplanare Subgraphen

Der Vorteil von außerplanaren Graphen ist, dass sie leicht in die kanonische Triangulierung überführt werden können. Mori et al. zeigten dazu:

▶ **Lemma 4** (Mori et al. [8], Lemma 8 and Proposition 9). *Jeder Knoten v in einem maximalen außerplanaren Graphen auf n Knoten kann durch $n - 1 - deg(v)$ Flips dominant gemacht werden.*

Beweis. Angenommen a soll dominant gemacht werden. Wenn a zu allen anderen Knoten benachbart ist, dann ist a bereits dominant. Wenn nicht dann gibt es eine Kante xy, die a von einem Knoten z abschneidet. Wie in Abbildung 35 auf der linken Seite zu sehen ist. Um den Grad von a um eins zu erhöhen flippen wir die Kante xy, so dass nun a und z miteinander verbunden sind. Diese beiden Knoten können aufgrund der Außerplanarität nicht bereits verbunden gewesen sein, sonst würden sich diese Kanten schneiden. Somit kann dieser Flip durchgeführt werden um den Grad von a um eins erhöhen. Der Grad von a kann pro Flip um maximal eins erhöht werden. Demnach sind $n - 1 - deg(a)$ Flips sowohl notwendig als auch hinreichend um a dominant zu machen. Bei diesem Vorgehen werden immer nur die Kanten innerhalb des hamiltonischen Kreises geflippt und niemals die Kanten des Kreises. Ein Flip ist immer nur dann erlaubt, wenn er nicht zu Doppelkanten führt. ◀

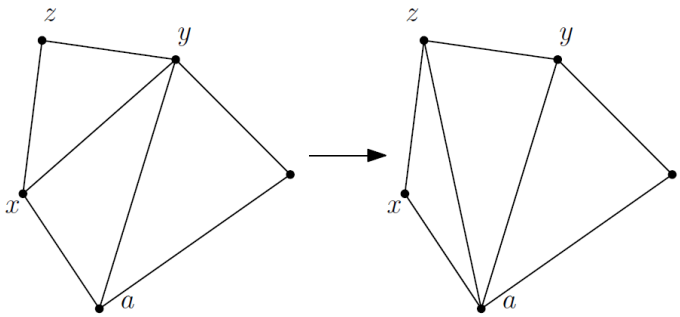


Abbildung 35 Knoten a wird dominant gemacht

Für beide außerplanare Graphen wird je einer der Knoten dominant gemacht. Daraus folgt:

▶ **Theorem 5** (Mori et al. [8], Proposition 9)**.** *Jede hamiltonische Triangulierung auf n Knoten kann durch höchstens* $2n - 10$ *Flips in eine kanonische transformiert werden, wobei der hamiltonische Kreis erhalten bleibt.*

Beweis. Wie bereits erwähnt gehen Mori et al [8] so vor, dass sie die hamiltonische Triangulierung in zwei außerplanare Graphen G_1 und G_2 zerlegen (Abbildung 34), die beide den hamiltonischen Kreis enthalten. Wir untersuchen für diesen Beweis einen Knoten a, der in G_2 einen Grad von zwei hat und somit nur zu Knoten benachbart ist, die sich auf dem hamiltonischen Kreis direkt neben diesem befinden. Ein solcher Knoten ist in einem außerplanaren Graphen immer vorhanden. In unserem Fall besteht jede Facette aus drei Knoten, damit die Knoten der Dreiecke die zwei Kanten auf der konvexen Hülle haben, alle einen Grad von drei haben, müsste der Knoten der zu den beiden Kanten des Dreiecks inzident ist, die auf der konvexen Hülle liegen, entweder eine andere Kante des gesamten Graphens schneiden oder eine Kante außerhalb des Graphens bilden, allerdings so dass keine Doppelkante entsteht. Dies würde entweder der Planarität widersprechen oder der Tatsache, dass sich danach nicht mehr alle Knoten auf der konvexen Hülle befänden. Mit dieser Voraussetzung stellen wir sicher, dass sämtliche Flips in G_1 keine bereits existierenden Kanten erzeugen können. Dadurch dass der Graph eine kombinatorische Triangulierung ist und demnach jeder Knoten einen Mindestgrad von drei hat, können wir folgern, dass der Grad von a in G_1 mindestens drei sein muss. Nach Lemma 4 werden also $n - 1 - 3$ Flips benötigt.

Knoten die in einem außerplanaren Graphen einen Grad von zwei haben, können entfernt werden, ohne dass der Restgraph seine Außerplanarität verliert. Dadurch können wir Knoten a aus G_2 entfernen und reduzieren die Anzahl Knoten in G_2 somit um eins. Des Weiteren gilt für außerplanare Graphen mit einer Knotenzahl von $n \geq 6$, dass sie immer einen Knoten mit einem Grad von mindestens vier haben. Mit diesen Ergebnissen kommen wir auf eine Maximalanzahl benötigter Flips für G_2 von $n - 2 - 4$.

Daraus können wir folgern, dass insgesamt maximal $(n - 4) + (n - 6) = 2n - 10$ Flips benötigt werden.

◀

6.3.3 Umwandlung in eine hamiltonische Triangulierung

Über Triangulierungen, die keine separierenden Dreiecke enthalten, ist bekannt, dass sie hamiltonisch und vierfach zusammenhängend sind.

▶ **Definition 6** (Separierende Dreiecke). Separierende Dreiecke sind Dreiecke, die die Triangulierung in zwei Subgraphen teilt, wenn die zugehörigen Kanten und Knoten entfernt werden würden. Diese sind in Abbildung 36 hellblau hervorgehoben.

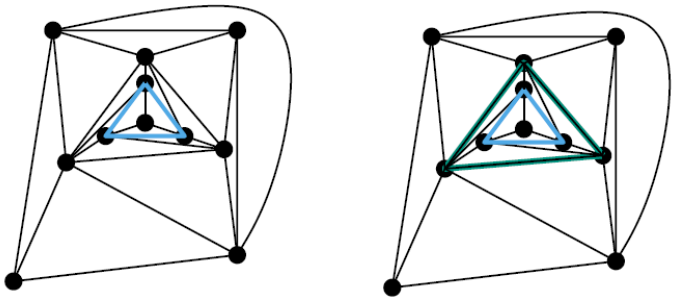


Abbildung 36 Ein separierendes Dreieck und ein enthaltendes Dreieck

Diese Eigenschaft nutzen Bose et al. in ihrem Paper [3] um eine Triangulierung hamiltonisch zu machen. Die Idee ist in einer vorhandenen Triangulierung, die separierenden Dreiecke durch Kantenflippen zu entfernen. Den Beweis dass durch das Flippen einer Kante eines separierenden Dreiecks kein neues entsteht, liefern Mori et al. (ohne Beweis)

▶ **Lemma 7** (Mori et al. [8], Lemma 11). *In einer Triangulierung mit $n \geq 6$ Knoten, wird durch das Flippen jeder Kante eines separierenden Dreiecks $D = abc$ dieses separierende Dreieck entfernen. Dadurch entsteht niemals ein neues separierendes Dreieck, vorausgesetzt dass die ausgewählte Kante zu mehreren separierenden Dreiecken gehört.*

Um die Ergebnisse aus dem Paper von Bose et al. [3] zu verstehen, müssen zuvor einige Begriffe definiert werden:

▶ **Definition 8** (Enthaltende Dreiecke). Enthaltende Dreiecke sind separierende Dreiecke, die selbst separierende Dreiecke enthalten. Diese sind in Abbildung 36 blaugrün hervorgehoben.

▶ **Definition 9** (Tiefstes Dreieck). Ein tiefstes Dreieck ist ein separierendes Dreieck, das von den meisten separierenden Dreiecken enthalten ist.

▶ **Definition 10** (Freie Kanten). Freie Kanten sind Kanten, die nicht zu einem separierendem Dreieck gehören. In Abbildung 37 sind diese hellgrün gekennzeichnet.

▶ **Lemma 11** (Bose et al. [3], Lemma 2)**.** *In einer Triangulierung T ist jeder Knoten v eines separierenden Dreiecks D zu mindestens einer freien Kante innerhalb von D inzident.*

Beweis. Angenommen es gibt eine Kante von D die inzident zu einem Knoten v ist. Über D wissen wir, dass es mindestens einen Knoten enthalten muss, da es separierend ist. Außerdem gibt es eine dreieckige Facette innerhalb von D, die diese Kante enthält. Im Folgenden betrachten wird die Kante e dieser Facette, welche ebenfalls zu v inzident ist.

Der Rest erfolgt durch vollständige Induktion über die Anzahl separierender Dreiecke in D.

Induktionsanfang: Angenommen D enthält keine weiteren separierenden Dreiecke, dann ist e eine freie Kante und wir sind für diesen Fall fertig.

Induktionsschritt: Wir betrachten zwei weitere Fälle.

1. Wenn e nicht zu einem separierenden Dreieck gehört, ist e eine freie Kante und wir sind fertig.
2. e ist eine Kante eines separierenden Dreiecks D'. D' selbst ist in D enthalten. Diese *Enthalten-Relation* ist transitiv, d.h. separierende Dreiecke die in D' enthalten sind, müssen auch in D enthalten sein. Demnach ist die Anzahl an separierenden Dreiecken in D' kleiner als die von D. Da v ebenfalls ein Knoten von D' ist, können wir mit Hilfe der Induktionsvoraussetzung folgern, dass es eine freie Kante innerhalb von D' geben muss, die inzident zu v ist. Diese Kante ist ebenfalls in D, da D D' enthält.

◀

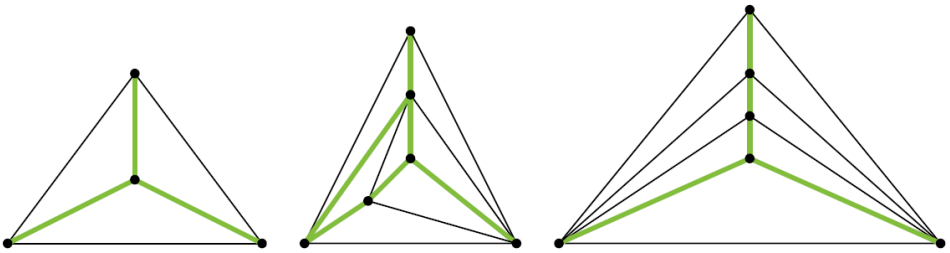


Abbildung 37 Jeder Knoten eines separierenden Dreiecks hat eine zu sich inzidente freie Kante innerhalb dieses Dreiecks

Mit Hilfe eines Kreditschemas wird im Folgendem bewiesen, dass zur Umwandlung einer Triangulierung in eine vierfach zusammenhängende maximal $\lfloor(3n-6)/5\rfloor$ Flips benötigt werden.

Dazu wird zunächst jede Kante mit einer Münze versehen.

Im Folgendem werden zwei Invarianten betrachtet, die zu Beginn und nach jedem Entfernen eines separierendem Dreiecks eingehalten werden müssen:

1. Jede Kante eines separierenden Dreiecks hat eine Münze.
2. Jeder Knoten eines separierenden Dreiecks hat eine inzidente frei Kante, die sich innerhalb des Dreiecks befindet und ebenfalls eine Münze hat.

Nach dem Satz von Euler hat eine planare Triangulierung mit n Knoten maximal $3n-6$ Kanten. Zu zeigen ist also, dass jeder Flip mit fünf Kanten und somit fünf Münzen bezahlt wird. Dabei ist wichtig, dass die zweite Invariante nach jedem Flip weiterhin gülitg ist.

Es wird immer das tiefste, separierende Dreieck betrachtet und die Fälle untersucht in denen Kanten mit separierenden Dreiecken oder Knoten von enthaltenden Dreiecken geteilt werden, die das aktuell betrachtete Dreieck enthalten. Da immer nur mit inneren Kanten bzw. mit Kanten des separierenden Dreiecks bezahlt werden, werden für Dreiecke, für die diese Voraussetzung nicht gilt, die benötigten Kante zum Bezahlen für die Flips unberührt bleiben. Daraus kann gefolgert werden, dass diese Fälle irrelevant sind, da sie die Invarianten nicht verletzen können.

Im Folgenden gilt für die Abbildungen 38, 39, 40, 41 und 42, dass die zu flippende Kante durch eine gestrichelte Linie dargestellt ist. Allgemein haben die Kanten mit denen für den Flip gezahlt wird die Farben rot, gelb oder blau. Die grauen Linien stehen für mögliche enthaltenden Dreiecke, für die bewiesen werden muss, dass die Invarianten nach dem Flip immer noch eingehaltet werden. Dass diese Kanten nicht zum Bezahlen des Flips benutzt werden, wird durch gelbe Kreise auf den noch benötigten Kanten gekennzeichnet.

Fall 1: Das separierende Dreieck teilt keine Kante mit einem anderem separierendem Dreieck

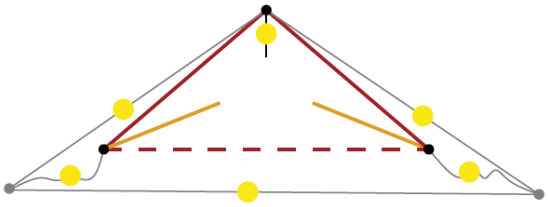


Abbildung 38 Fall 1: Das separierende Dreieck teilt keine Kante mit einem anderem separierendem Dreieck

Beweis. Drei Münzen können direkt über die Kanten des Dreiecks D bezahlt werden, da im betrachtetem Fall (Abbildung 38) nach Voraussetzung keine mit einem anderem separierendem Dreieck geteilt wird. Diese sind in Abbildung 38 rot gekennzeichnet. Nach Lemma 7 von Mori et al. kann kein neues separierendes Dreieck durch das Flippen einer Kante eines separierenden Dreiecks entstehen, demnach kann eine Münze immer mit der zu flippenden Kante bezahlt werden. Die restlichen zwei Münzen können mit zwei der drei inneren, freien Kanten bezahlt werden. Diese sind in Abbildung 38 gelb hervorgehoben. Da im schlimmsten Fall maximal ein Knoten des Dreiecks mit einem enthaltendem Dreieck geteilt werden kann, reicht es aus eine unbelastete, freie Kante beizubehalten, welche im Falle eines existierenden, enthaltenden Dreiecks der Kante entspräche, die inzident zu dem geteilten Knoten wäre. Eine solche hätte nach diesem Schritt noch eine Münze wie in Abbildung 38 zu sehen ist. Würde das Dreieck zwei Knoten mit zwei unterschiedlichen enthaltenden Dreiecken teilen, würden sich die Kanten dieser kreuzen, was der Planarität widerspräche. ◀

Fall 2: Das separierende Dreieck teilt keine Kante mit einem enthaltenden Dreieck, aber mindestens eine Kante mit einem separierenden Dreieck

Beweis. In diesem Fall (Abbildung 39) wird die Kante geflippt, die von beiden Dreiecken geteilt wird. Dadurch werden beide separierenden Dreiecke entfernt.

Wieder kann die Münze der geflippten Kante zum Bezahlen genommen werden. Des Weiteren werden die inneren, freien Kanten des betrachteten Dreiecks D genommen, welche

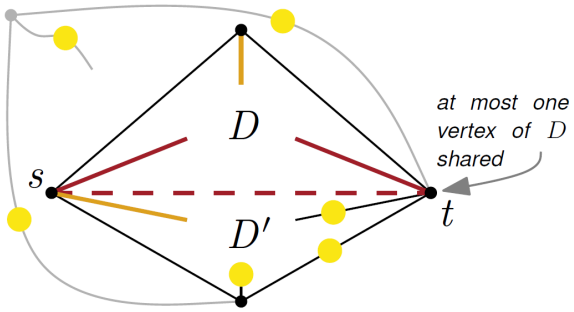


Abbildung 39 Fall 2: Das separierende Dreieck teilt keine Kante mit einem enthaltenden Dreieck, aber mindestens eine Kante mit einem separierenden Dreieck

inzident zu den Knoten s, t sind (diese entsprechen den roten Kanten in Abbildung 39). Das anliegende, separierende Dreieck D' enthält für diese beiden Knoten noch freie, unbenutzte Kanten, die einem enthaltendem Dreieck nach diesem Schritt noch zur Verfügung ständen. Folgende Aussage kann unter den Voraussetzungen getroffen werden, entweder wird kein Knoten mit einem enthaltenden Dreieck geteilt oder genau einer. Von ein und demselben enthaltendem Dreieck können keine zwei Knoten geteilt werden, da dieses sonst eine Kante mit D teilen würde, das widerspräche den Voraussetzungen für diesen Fall. Bei zwei verschiedenen enthaltenden Dreiecken würde dies bedeuten, dass sich die Kanten der beiden Dreiecke kreuzen würde. Dies wiederum würde der Planariät widersprechen. Demnach reicht es aus, wenn für einen der beiden Knoten der geflippten Kante eine unbenutzte, freie Kante übrig bleibt. Mit der anderen inneren Kante kann für den Flip gezahlt werden (in Abbildung 39 die gelbe Kante in D')

Nichtsdestotrotz kann maximal eine Kante des anliegenden, separierenden Dreiecks mit einem enthaltendem Dreieck geteilt werden, dieser Fall ist in Abbildung 39 gräulich dargestellt. Wenn zwei Kanten geteilt werden würden, würde dies dazu führen, dass das anliegende Dreieck dem enthaltendem entspräche und die geflippte Kante somit zu einem enthaltendem Dreieck gehören müsste, das widerspräche der Voraussetzung. Die inneren Kanten die inzident zu einem Knoten sind, der mit einem enthaltendem Dreieck geteilt wird, dürfen nicht genommen werden, weil diese noch für das enthaltende Dreieck benötigt werden.

Allerdings befindet sich innerhalb von D noch eine freie, innere Kante, die wir benutzen können, da der dazu inzidente Knoten nicht ebenfalls von einem enthaltendem Dreieck geteilt werden kann, die obere gelbe Kante in Abbildung 39.

◀

Fall 3: Das separierende Dreieck D teilt eine Kante mit einem enthaltendem Dreieck A, aber keine mit einem weiterem separierendem Dreieck.

Beweis. Hierbei wird ebenfalls die geteilte Kante geflippt und dadurch beide separierenden Dreiecke entfernt. Da nach Voraussetzung keine weitere Kante von einem separierendem Dreieck geteilt wird, können die ersten drei Münzen mit den Kanten des betrachtetem Dreiecks D bezahlt werden, in Abbildung 40 rot hervorgehoben. Im nächsten Schritt müssen etwaige enthaltende Dreiecke beachtet werden, die Knoten mit D teilen, die nicht durch den Flip entfernt werden.

Wieder kann durch Widerspruch gezeigt werden, dass maximal einer der Knoten s, t von einem weiteren enthaltendem Dreieck geteilt werden kann, welches ebenfalls alle enthaltenden

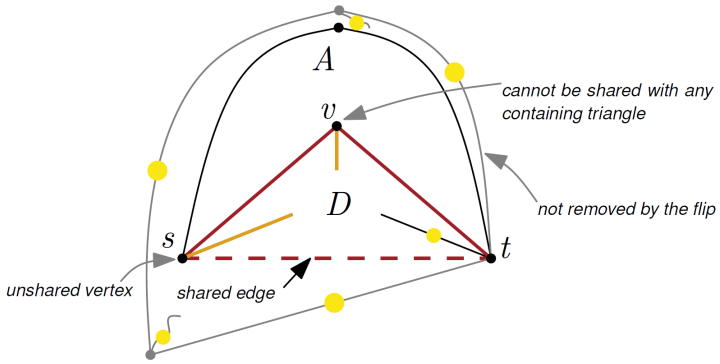


Abbildung 40 Fall 3: Das separierende Dreieck D teilt eine Kante mit einem enthaltenden Dreieck A, aber keine mit einem weiterem separierenden Dreieck.

Dreiecke beinhaltet, die eine Kante mit Dreieck D teilen, dieser Fall ist in Abbildung 40 gräulich dargestellt. Würden die beiden Knoten inzident zu der zu flippenden Kante von zwei verschiedenen enthaltenden Dreiecken geteilt werden, die A inklusive D enthalten, müssten sich deren Kanten kreuzen, was in einer planaren Einbettung nicht der Fall sein kann.

Der Knoten v von D, der nicht von dem enthaltenden Dreieck geteilt wird, kann nicht von einem weiteren enthaltenden Dreieck geteilt werden, da dieses wiederum D enthalten müsste. Wäre das nicht der Fall, würde es ein Dreieck geben, das tiefer als D wäre, womit eine der Voraussetzungen verletzt wäre. Außerdem müsste dieses Dreieck ebenfalls die geflippte Kante enthalten, womit es identisch zum betrachtetem Dreieck wäre und dies ebenfalls bedeuten würde, dass D nicht das tiefste Dreieck sein kann.

Demzufolge gibt es zwei innere, freie Kanten von D, die nicht von einem enthaltendem Dreieck geteilt werden können und mit diesen können die noch fehlenden zwei Münzen bezahlt werden, diese sind Abbildung 40 gelb hervorgehoben.

◀

Fall 4: Das separierende Dreieck teilt eine Kante mit einem enthaltenden Dreieck und eine weitere Kante mit einem nicht-enthaltenden, separierenden Dreieck.

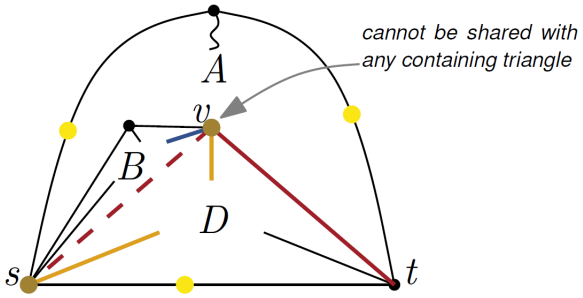


Abbildung 41 Fall 4: Das separierende Dreieck teilt eine Kante mit einem enthaltenden Dreieck und eine weitere Kante mit einem nicht-enthaltenden, separierenden Dreieck.

Beweis. Dieser Fall (Abbildung 41) ist ähnlich zu Fall 3, abgesehen davon, dass diesmal noch eine Kante mit einem separierenden Dreieck B geteilt wird, welches selbst kein enthaltendes sein kann, da dies dazu führen würde, dass es ein Dreieck gäbe, dass tiefer wäre als D.

Diesmal wird die Kante zwischen B und D geflippt. Das bedeutet wir können die Kante, die von D und A, dem enthaltenden Dreieck, geteilt wird, nicht zum Bezahlen verwenden. Die noch übrige Kante von D ist nach Voraussetzung nach dem Flip nicht mehr zu einem separierendem Dreieck zugehörig. Demnach können die ersten zwei Münzen von dieser und der geflippte Kante genommen werden, diese sind in Abbildung 41 rot hervorgehoben. Für die inneren, freien Kante inzident zu s und v gilt das Gleiche wie in Fall 3, womit zwei weitere Münzen bezahlt wären, in Abbildung 41 sind diese gelb dargestellt. Die letzte Münze wird mit der freien, noch nicht benutzten inneren Kante aus B, welche inzident zu Knoten v ist, dieser gehört nicht zu der Kante die von D und A geteilt wird (die blaue Kante in Abbildung 41). Diese innere Kante kann nach dem Flip keine innere Kante zu einem separierendem Dreieck sein. Ansonsten würde dies bedeuten, dass B von einem weiteren Dreieck enthalten wird, das wiederum hieße, dass B und nicht D das tiefste Dreieck wäre. ◀

Fall 5: Das separierende Dreieck teilt eine Kante mit einem enthaltenden Dreieck und zwei Kanten mit nicht-enthaltenden separierenden Dreiecken.

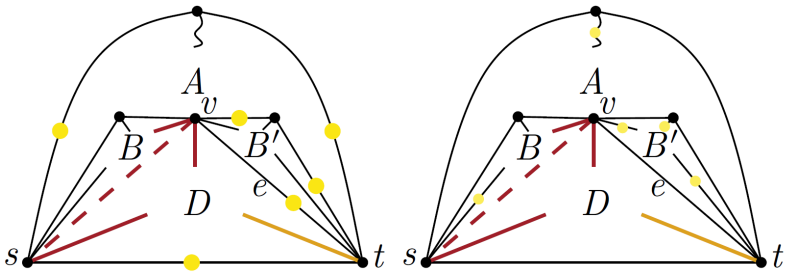


Abbildung 42 Fall 5: Das separierende Dreieck teilt eine Kante mit einem enthaltenden Dreieck und zwei Kanten mit nicht-enthaltenden separierenden Dreiecken.

Beweis. Im letzten Fall (Abbildung 42) wird zusätzlich zum Fall 4, die noch „freie“ Kante e von D ebenfalls mit einem separierenden Dreieck geteilt. Demnach kann mit dieser nicht für diesen Flip gezahlt werden, die anderen vier Münzen sind wie in Fall 4 und in Abbildung 42 rot hervorgehoben. Allerdings befindet sich innerhalb von D noch eine freie, unbenutzte, innere Kante, die diesmal zum Bezahlen herangezogen wird. Dieser Schritt kann vollzogen, da durch B' für den entsprechenden Knoten noch eine freie, unbenutzte, innere Kante vorhanden ist. ◀

Somit ist gezeigt, dass für jeden Fall die Invarianten erfüllt sind und fünf Kanten verbraucht werden, aber zugleich keine Kanten zum Bezahlen verwendet werden, die für andere Flips benötigt werden. Damit kommen wir auf maximal $\lfloor (3n-6)/5 \rfloor$ Flips zum Entfernen aller separierenden Dreiecke und dementsprechend auf einen vierfach zusammenhängenden Graphen. Auf diesen kann Lemma 5 von Mori et al. angewendet werden, welches besagt, dass dreifach zusammenhängende Graphen maximal $2n-10$ Flips benötigen, um in die kanonische Triangulierung überführt zu werden. In unserem Fall haben wir um eine hamiltonische Triangulierung zu erreichen, alle separierende Dreiecke entfernt, solche Graphen sind nach Definition vierfach zusammenhängend. Somit kann der Minimalgrad des Startknoten von drei auf vier erhöht werden, womit wir bei einer maximalen Flipanzahl von $2n-11$ wären.

Somit muss für beide Triangulierungen die ineinander transformiert werden sollen, der Graph zunächst in einen vierfach zusammenhängenden umgewandelt werden und danach

mit Hilfe von Mori et al.'s [8] Ansatz in die kanonische Triangulierung konvertiert werden. Daraus kann folgende maximale Flipanzahl bestimmt werden:

$$2 \cdot (2n - 11 + \frac{3n-6}{5}) = 4n - 22 + \frac{6n-12}{5} = \frac{26n-122}{5} = 5.2n - 24.4$$

Demnach können zwei Triangulierungen über die kanonische Triangulierung ineinander überführt werden und dafür werden höchstens $5.2n - 24.4$ Flips benötigt.

6.4 Untere Schranken

In den vorherigen Abschnitten haben wir die obere Schranke für die Anzahl an Flips mit dem Ansatz betrachtet, die Triangulierungen zunächst in die kanonische Repräsentation zu überführen. Im Folgenden möchten wir untersuchen, wie viele Flips mindestens für diesen Ansatz benötigt werden.

Hierfür wurde die beste, untere Schranke bereits in einer Veröffentlichung aus dem Jahr 1997 von Komuro [6] bewiesen. Dazu hat er den Ansatz verfolgt den jeweiligen maximalen Grad einer Triangulierung zu betrachten und mit dessen Hilfe die Anzahl der Flips abzuschätzen.

Dafür hat Komuro folgendes Theorem aufgestellt und in seinem Paper [6] bewiesen.

▶ **Theorem 12** (Komuro [6], Theorem 5). *Sei G eine Triangulierung auf n Knoten. Dann werden mindestens $2n - 2\Delta(G) - 3$ Flips gebraucht um G in die kanonische Triangulierung zu transformieren, wobei $\Delta(G)$ den maximalen Grad von G bezeichnet.*

Da es Triangulierungen mit einem Maximalgrad von sechs gibt, kommt man auf folgende untere Schranke für die Anzahl Flips: $2n - 2 \cdot 6 - 3 = 2n - 15$.

Des Weiteren hat Komuro ein allgemeineres Theorem aufgestellt, wobei er untersucht hat, wie viele Flips mindestens benötigt werden um zwei Triangulierungen ineinander zu überführen ohne dabei über die kanonische Darstellung zu gehen (ohne Beweis, siehe hierzu [6]).

▶ **Theorem 13** (Komuro [6], Theorem 4). *Seien G und G' Triangulierungen auf n Knoten. Seien $v_1, \ldots, v_n$ und $v'_1, \ldots, v'_n$ die Knoten von G und G', jeweils, sortiert nach aufsteigendem Grad. Dann werden mindestens $\frac{1}{4}D(G, G')$ Flips benötigt um G in G' zu transformieren, wobei $D(G, G') = \sum_{i=1}^{n} |deg(v_i) - deg(v'_i)|$.*

Im Folgenden wird die untere Schranke des Ansatz von Mori et al., welcher in Abschnitt 6.3.2 beschrieben wurde, im Zusammenhang mit Bose et al.s Vorgehen, für welchen in Abschnitt 6.3.3 die obere Schranke bereits betrachtet wurde, kurz näher beleuchtet. Für die obere Schranke wurden die Vorteile von hamiltonischen Triangulierungen angewendet und Bose et al. [3] untersuchten den Ansatz eine Triangulierung hamiltonisch zu machen, indem alle separierenden Dreiecke durch Flippen entfernt werden.

Um nun die untere Schranke zu erhalten, gehen Bose et al. so vor, dass sie eine Triangulierung erstellen, die so viele separierende Dreiecke wie möglich enthält, die keine Kanten miteinander teilen. Sind die separierenden Dreiecke disjunkt können wir pro Flip maximal eines davon entfernen. Somit beschreiben wir den Worst Case und können untersuchen wie viele Flips mindestens benötigt werden.

Die Erstellung einer solchen Triangulierung wird nachfolgend beschrieben und ist außerdem in Abbildung 43 veranschaulicht.

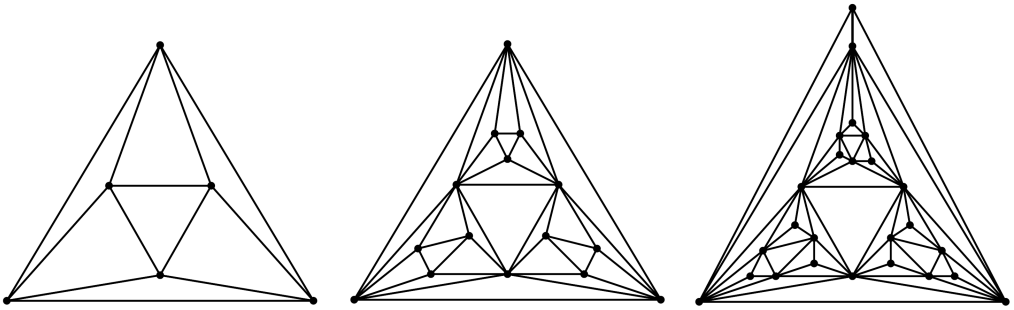


Abbildung 43 Erstellung einer Triangulierung mit einer maximalen Anzahl an kantendisjunkten, separierenden Dreiecken.

▶ **Theorem 14** (Bose et al. [3], Theorem 5). *Es gibt Triangulierungen die mindestens* $\lceil(3n-10)/5\rceil$ *Flips benötigen um vierfach zusammenhängend gemacht zu werden.*

Beweis. Bei diesem Vorgehen handelt es sich um einen rekursiven Ansatz, bei dem nach und nach Knoten hinzugefügt und die entstehenden Facetten trianguliert werden. Der Anfang besteht aus einem einfachen Dreieck, in dieses werden drei Knoten so platziert, dass sie ein Dreieck bilden, welches invers zu dem Äußeren ist. Dieses Dreieck wird so trianguliert, dass jeder dieser Knoten inzident zu den Knoten des äußeren Dreiecks ist, die inzident zu der Kante sind, welche dem Knoten gegenüber liegt. Siehe hierzu Abbildung 43 ganz links. Im darauffolgenden Schritt wird das Gleiche für die neuentstandenen Dreiecke durchgeführt, die inzident zu den Knoten des äußeren Dreiecks sind. Dies kann in Abbildung 43 in der Mitte betrachtet werden. Dies wird bis zum vorletzten Durchgang genauso weitergeführt. Im letzten Schritt wird in jedes neuerstellte Dreieck ein Knoten platziert und trianguliert (Abbildung 43 ganz rechts). Zum Schluss wird statt Dreiecke in die bisherige Triangulierung einzufügen, ein Knoten oberhalb des bisherigen Ergebnis gesetzt, welcher durch Kanten mit den Knoten des äußersten Dreiecks verbunden wird (ebenfalls Abbildung 43 ganz rechts). Durch dieses Vorgehen wird das Startdreieck zu einem separierenden Dreieck.

Dadurch dass in jedem Schritt Dreiecke in ein Dreieck bzw. im letzten ein einzelner Knoten gesetzt werden, werden die äußeren Dreiecke automatisch zu separierenden Dreiecken.

Insgesamt werden hierdurch $(3n-10)/5$ disjunkte, separierende Dreiecke erzeugt. Da pro Flip maximal ein separierendes Dreieck entfernt werden kann, führt dies dazu, dass mindestens $\lceil(3n-10)/5\rceil$ Flips dazu benötigt werden. ◀

Damit wäre gezeigt, dass diese Schranke nach oben und nach unten scharf ist. Die obere und die untere Schranke unterscheiden sich nicht einmal um einen Flip, die obere Schranke ist bei $\lfloor(3n-6)/5\rfloor$.

Allerdings kann eine Triangulierung bereits hamiltonisch sein, auch wenn sie noch separierende Dreiecke enthält. Dazu haben Aichholzer et al. [1] einen weiteren Ideenansatz verfolgt um eine Triangulierung hamiltonisch zu machen. Dabei sind sie zu folgendem Ergebnis gekommen (ohne Beweis, siehe hierzu [1]):

▶ **Theorem 15** (Aichholzer et al. [1], Corollary 5). *Es gibt Triangulierungen auf n Knoten die mindestens* $(n-8)/3$ *Flips benötigen um hamiltonisch gemacht zu werden.*

Wenn man diese Ergebnisse mit der unteren Schranke von Komuro aus Theorem 12 vergleicht, fällt auf, dass selbst durch diese Verbesserungen, diese nicht an dessen untere Schranke von $2n-15$ Flips herankommen. Das liegt vor allem daran, dass wir nach diesen

Schritten von Aichholzer [1] erst bei einer hamiltonischen Triangulierung sind, die zusätzlich noch in die kanonische überführt werden muss. Für hamiltonische Triangulierungen gilt ebenfalls, dass es welche gibt, die einen Maximalgrad von sechs haben. Mit Hilfe von Mori et al.s Lemma 4 kann die Mindestanzahl an benötigten Flips somit durch $(n-1-6)+(n-1-6-1) = 2n-15$ abgeschätzt werden. Vergleiche hierzu den zugehörigen Beweis 6.3.2, in diesem Falle wird nicht der Mindestgrad sondern der Maximalgrad eingesetzt. Mit diesem Ergebnis kommen wir insgesamt auf eine Mindestanzahl von $(n-8)/3+2n-15$ Flips für die Konvertierung in die kanonische Darstellung. Daraus kann gefolgert werden, dass dieser Ansatz nicht so weit verbessert werden kann, dass er ähnlich der unteren Schranke von Komuro ist.

6.5 Zusammenfassung

In dieser Arbeit wurde gezeigt, dass es möglich ist im kombinatorischen Bereich zwei Triangulierungen mit jeweils n Knoten durch Flips ineinander zu überführen. Außerdem wurde bewiesen, dass die Anzahl der benötigten Flips linear ist. Allen hier vorgestellten Beweisen liegt die Grundidee voraus, dass jede der beiden Triangulierungen zunächst in die kanonische Triangulierung konvertiert wird und um eine Sequenz von Flips zu erhalten, die Konvertierungsschritte eines der Dreiecksnetze invertiert wird.

Das Problem hierbei besteht offensichtlich darin, dass durch die Konvertierung möglicherweise deutlich mehr Flips durchgeführt werden müssen, als die beiden Triangulierungen tatsächlich voneinander entfernt sind. Um die Schranken weiter zu verbessern ist es notwendig einen neuen Ansatz zu finden, durch den direkt die Sequenz von benötigten Flips bestimmt werden kann. Leider ist bis jetzt noch kein solcher Ansatz bekannt.

Abstract

Diese Arbeit handelt von der grundsätzlichen Frage, ob zwei unterschiedliche Triangulierungen mit jeweils n Knoten in einer kombinatorischen Einbettung durch eine endliche Folge des Flippens von Kanten ineinander überführt werden können und wie viele Flips für die Umwandlung benötigt werden. Es konnte gezeigt werden, dass hierfür eine lineare Anzahl von Flips ausreichend ist. Die niedrigste obere Schranke liegt bei $5.2n-24.4$ und die beste untere Schranke bei $2n-15$ benötigter Flips.

Referenzen

1 Oswin Aichholzer, Clemens Huemer, and Hannes Krasser. Triangulations without pointed spanning trees. *Computational Geometry*, 40(1):79 – 83, 2008.

2 Oswin Aichholzer, Wolfgang Mulzer, and Alexander Pilz. Flip distance between triangulations of a simple polygon is np-complete. *CoRR*, abs/1209.0579, 2012.

3 Prosenjit Bose, Dana Jansens, André van Renssen, Maria Saumell, and Sander Verdonschot. Making triangulations 4-connected using flips. *CoRR*, abs/1110.6473, 2011.

4 Prosenjit Bose, Anna Lubiw, Vinayak Pathak, and Sander Verdonschot. Flipping edge-labelled triangulations. *CoRR*, abs/1310.1166, 2013.

5 Prosenjit Bose and Sander Verdonschot. A history of flips in combinatorial triangulations. In Alberto Márquez, Pedro Ramos, and Jorge Urrutia, editors, *Computational Geometry*, volume 7579 of *Lecture Notes in Computer Science*, pages 29–44. Springer Berlin Heidelberg, 2012.

6 H Komuro. The diagonal flips of triangulations on the sphere. *Yokohama Math. J*, (44), 1997.

7 Anna Lubiw and Vinayak Pathak. Flip distance between two triangulations of a point-set is np-complete. *CoRR*, abs/1205.2425, 2012.

8 Ryuichi Mori, Atsuhiro Nakamoto, and Katsuhiro Ota. Diagonal flips in hamiltonian triangulations on the sphere. *Graphs and Combinatorics*, 19(3):413–418, 2003.

9 Negami and Nakamoto. Diagonal transformations of graphs on closed surfaces. *Sci. Rep. Yokohama Nat. Univ. Sect. I Math. Phys. Chem.*, pages 71–97, 1993.

10 D. Sleator, R. Trajan, and W. Thurston. Short encodings of evolving structures. *SIAM Journal on Discrete Mathematics*, 5(3):428–450, 1992.

11 K. Wagner. Bemerkungen zum vierfarbenproblem. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 46:26–32, 1936.

7 Schnittapproximation durch Graphdekomposition in j-Bäume

Tobias Maier

Basierend auf: „Fast Approximation Algorithms for Cut-based Problems in Undirected Graphs“ von A. Madry

Zusammenfassung

In dieser Ausarbeitung werden wir ein Ergebnis von A. Madry [3] vorstellen, das sich mit schnittbasierten Problemen beschäftigt. Der Ansatz unseres Verfahrens besteht darin, den Graph G in strukturell einfachere Graphen zu zerlegen. Danach genügt es, die optimalen Schnitte der vereinfachten Graphen zu berechnen, um eine $\widetilde{O}(\log n)$ Approximation des optimalen Schnitts von G zu erhalten.

Im Gegensatz zu anderen Verfahren zerlegen wir nicht fix in Bäume, stattdessen definieren wir die Graphfamilie der j-Bäume, diese haben eine variable Komplexität. Dadurch erhalten wir eine Flexibilität zwischen der Laufzeit und der Komplexität der Graphen, in unserer Zerlegung.

Außerdem stellen wir eine Variante des Verfahrens vor, die durch Rekursion und zufällige Abtastung in annähernd linearer Laufzeit eine polylogarithmische Approximationsgüte $\widetilde{O}(\log^k n)$ erreicht.

7.1 Einleitung

Ein Schnitt eines ungerichteten Graphen mit Kantengewichten $G = (V, E, u)$ kann als eine Knotenmenge C definiert werden. Das Gewicht $u(C)$ des Schnittes ist die Summe der Kantenkapazitäten aller Kanten, die von einem Knoten aus C zu einem Knoten aus $\overline{C} = V \backslash C$ verlaufen ($u(C) = \sum_{e=(v,w)\in E \mid v\in C \ w\in\overline{C}} u(e)$).

Wir nennen ein Problem genau dann schnittbasiertes (Minimierungs)-Problem, wenn jede Instanz I des Problems darauf zurückgeführt werden kann, in einem Graphen G_I den Schnitt C^* zu finden, der das Produkt $f_I(C)u(C)$ minimiert. Dabei sei $f_I : 2^V \to \mathbb{R}$ eine beliebige Funktion, die nicht von der Struktur des Graphen G_I, sondern nur von der Instanz I abhängt.

Ein sehr bekanntes schnittbasiertes Problem ist z.B. das *balanced separator* Problem. Dabei ist ein Schnitt C gesucht, so dass $\frac{u(C)}{\min\{C,\overline{C}\}}$ minimal ist und außerdem $\min\{C, \overline{C}\} \geq c|V|$. Dieses Problem kann man als schnittbasiertes Problem formulieren, mit $f(C) = \frac{1}{\min\{C,\overline{C}\}}$ (falls: $\min\{C, \overline{C}\} \geq c|V|$) und $f(C) \approx +\infty$ (sonst).

Schnittbasierte Probleme gehören zu den bekanntesten Optimierungsproblemen. Dementsprechend wurde viel Forschung auf diesem Bereich betrieben. Die meisten Ergebnisse sind jedoch auf die effiziente Lösung eines einzelnen Problems beschränkt.

Die Hauptinspiration von unserem Ergebnis ist die schnittbasierte Graphdekomposition von Räcke [4]. Diese wurde ursprünglich in Zusammenhang mit *oblivious routing schemes* entwickelt.

In [4] liefert Räcke eine Möglichkeit Graphen in Polynomialzeit in sogenannte Dekompositionsbäume zu zerlegen. Hierzu wird eine Konvexkombination $\sum_i \lambda_i T_i$ von Dekompositionsbäumen T_i gefunden, so dass G einbettbar ist in jeden Dekompositionsbaum T_i und die Konvexkombination $O(\log n)$-einbettbar ist in G (siehe Definition 3).

Angelehnt an dieses Ergebnis, bestimmen wir eine ähnliche Konvexkombination. Unser Ziel ist jedoch das Verfahren so zu beschleunigen, dass die Laufzeit konkurrenzfähig

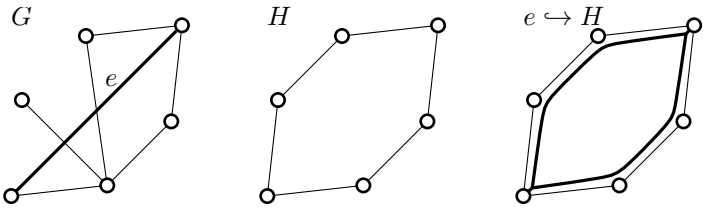


Abbildung 44 zeigt, wie die Kante e in Graph H eingebettet werden kann.

mit spezialisierten Schnittalgorithmen ist. Hierzu sind wir auch bereit, Abstriche bei der Approximationsgüte zu machen.

7.2 Theoretische Grundlagen des Verfahrens

Wir betrachten in dieser Ausarbeitung Schnitte von ungerichteten Graphen mit Kantengewichten $G = (V, E, u)$. Dabei sei V die Knotenmenge mit $|V| = n$, E sei die Kantenmenge mit $|E| = m$ und $u : E \to \mathbb{Z}^+$ sei die Kapazitätsfunktion. Wir sagen U ist die maximale Kantenkapazität innerhalb des Graphen ($U = \max_e u(e)$).

Eine wichtige Grundlage der Betrachtung von Schnitten sind Flüsse. Zum Beispiel lässt sich das sehr bekannte Problem des minimalen st-Schnitts mit Hilfe eines einfachen st-Flusses lösen.

Ein Fluss f in $G = (V, E, u)$ transportiert eine Menge von $|f|$ Einheiten eines Gutes von einer Quelle s zu einer Senke t. Dabei darf an keiner Kante $e \in E$ die Menge $|f(e)|$ des an der Kante geleiteten Gutes die Kapazität $u(e)$ überschreiten ($|f(e)| \leq u(e)$). Außerdem muss der Fluss an Knoten erhalten werden, d.h. die Menge des eingehenden Flusses an einem Knoten muss gleich der Menge des ausgehenden Flusses an diesem Knoten sein (ausgenommen Quelle und Senke).

▶ **Definition 1** (Mehrgüterfluss (multicommodity flow)). Ein Mehrgüterfluss $f = (f^1, ..., f^k)$ auf G ist eine Menge von k einzelnen Flüssen. Ein solcher Mehrgüterfluss heißt *realisierbar*, wenn an jeder Kante $e \in E$ die aufsummierte Menge der einzelnen Flüsse kleiner ist als die Kapazität der Kante ($\Sigma_{i=1}^{k} |f^i(e)| = |f(e)| \leq u(e)$).

Eine weitere wichtige Grundlage, die wir zur Vereinfachung von Graphen benötigen, sind Einbettungen. Sie sind eine Möglichkeit die Schnitte zweier gegebener Graphen G und H in Zusammenhang zueinander zu bringen. Anschaulich stellt eine Einbettung dar, wie ein Graph G in einem Graphen H enthalten ist.

▶ **Definition 2** (Einbettungen (embedding)). Eine Einbettung von einem Graphen $G = (V, E_G, u_G)$ in einen Graphen $H = (V, E_H, u_H)$ ist ein realisierbarer Mehrgüterfluss f in H. Dabei existiert für jede Kante $e = (v, w) \in E_G$ ein Fluss f^e, der in H $|f^e| = u(e_i)$ Einheiten eines Guts von Knoten v zu Knoten w transportiert.

Ein Graph G heißt *einbettbar* in einen Graphen H genau dann, wenn eine solche Einbettung existiert. Beachte, beide Graphen besitzen dieselbe Knotenmenge. Ein Schnitt auf dem Graphen H induziert deshalb auch immer einen Schnitt auf G und umgekehrt. Außerdem lässt sich leicht einsehen, dass das Gewicht eines Schnittes C in H immer mindestens so groß ist wie das Gewicht des selben Schnittes in G ($u_H(C) \geq u_G(C)$).

Ein Graph G heißt *t-einbettbar* in einen Graphen H genau dann, wenn G einbettbar in den Graphen $H_t = (V, E_H, u_{H_t})$ ist (wobei: $u_{H_t}(e) = t \cdot u_H(e)$). Analog zu oben gilt

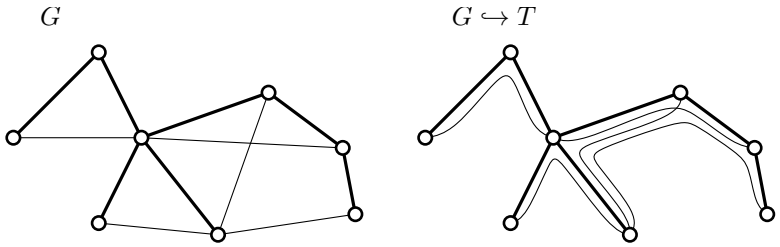


Abbildung 45 zeigt, die eindeutige Einbettung des Graphen G in den Spannbaum T (gegeben durch die dicken Kanten)

hier $t \cdot u_H(C) = u_{H_t}(C) \geq u_G(C)$, d.h. die Schnitte im Graphen G sind maximal um einen Faktor t größer, als die Schnitte in H.

Falls G einbettbar in H ist und umgekehrt H t-einbettbar in G ist, dann folgt daraus $u_G(C) \leq u_H(C) \leq t \cdot u_G(C)$. Somit sind in diesem Fall die Schnitte von G und H gleich bis auf einen Faktor von t.

Im Laufe des Verfahrens benötigen wir die Einbettung von einem Graphen $G = (V, E_G, u_G)$ in einen Spannbaum $T = (V, E_T, u_T)$ von G. Für jede Kante $e = (v, w) \in E_G$ gibt es einen eindeutigen Weg $path_T(e)$ in T von v nach w. Daraus folgt, dass der Mehrgüterfluss einer Einbettung von G in T eindeutig ist. Wie in Abbildung 45 zu sehen, muss jede Kante entlang dieses Weges $path_T(e)$ in T eingebettet werden. Soll also G in T einbettbar sein, so müssen die Kapazitäten u_T entsprechend gewählt werden, damit dieser eindeutige Mehrgüterfluss realisierbar ist.

$$u_T(e) = \sum_{\forall e' : e \in path_T(e')} u_G(e')$$

Die Kantenkapazitäten u_T können in $\widetilde{O}(m)$ Zeit berechnet werden. Hierzu lässt sich ein Verfahren von Spielman und Teng [5] anpassen, das ursprünglich dazu gedacht ist die Streckung eines Spannbaums zu berechnen. Es handelt sich dabei um ein *Divide and Conquer* Verfahren, dass den Baum in balancierte Teile spaltet.

7.2.1 Graphdekomposition

Graphdekompositionen sind das zentrale Konzept unseres Verfahrens. Eine $(\alpha, \mathcal{G})$-Dekomposition von G ist eine Zerlegung von G in Graphen der Familie $\mathcal{G}$, so dass die Schnittstruktur von G in der Dekomposition erhalten bleibt.

▶ **Definition 3** ($(\alpha, \mathcal{G})$-Dekomposition)**.** Sei $\alpha \geq 1$ und $\mathcal{G}$ eine Graphfamilie. Wir nennen eine Menge $\{(\lambda_i, G_i)\}_i$ genau dann $(\alpha, \mathcal{G})$-Dekomposition von einem Graphen G, wenn alle $\lambda_i > 0$, alle $G_i \in \mathcal{G}$ und:

- $\sum_i \lambda_i = 1$
- G ist einbettbar in jeden G_i
- Für jeden Graphen G_i existiert eine Einbettung f_i in G, so dass $\sum_i \lambda_i \cdot |f_i(e)| \leq \alpha u(e)$.

Wir nennen eine $(\alpha, \mathcal{G})$-Dekomposition *k-dünn* wenn sie maximal k unterschiedliche Graphen G_i enthält.

In einer $(\alpha, \mathcal{G})$-Dekomposition bleiben die Schnitte von G bis auf einen Faktor α erhalten. Dies lässt sich anschaulich erkennen, indem man die $(\alpha, \mathcal{G})$-Dekomposition als konvexe Linearkombination der Graphen G_i betrachtet. Der kombinierte Graph $\sum_i \lambda_i G_i$ ist dabei

α-einbettbar in G (Auch klar: G ist einbettbar in die Linearkombination). Das folgende Lemma präzisiert diese Aussage, außerdem lässt sich mit seiner Hilfe ein Verfahren generieren, um den optimalen Schnitt in G zu approximieren.

▶ **Lemma 4** (Schnittapproximation durch $(\alpha, \mathcal{G})$-Dekomposition). *Sei $\{(\lambda_i, G_i)\}_i$ eine $(\alpha, \mathcal{G})$-Dekomposition von G. Dann gilt für jeden Schnitt C in G:*

- **untere Schranke***: $u_i(C) \geq u(C)$ in jedem Graphen G_i*
- **obere Schranke***: es gibt einen Graphen G_j mit $u_j(C) \leq \alpha u(C)$*

Wir sagen auch, die Dekomposition α-erhält die Schnitte von G.

Beweis. Die **untere Schranke** folgt direkt aus der Einbettbarkeit von G in jeden G_i.

Um die **obere Schranke** zu beweisen, betrachten wir die dritte Anforderung an eine $(\alpha, \mathcal{G})$-Dekomposition. Diese lautet, für alle Graphen G_i existiert eine Einbettung f_i in G, so dass $\sum_i \lambda_i f_i(e) \leq \alpha u(e)$. Daraus folgt insbesondere, dass $\sum_i \lambda_i u_i(C) \leq \alpha u(C)$. Da sich die Linearfaktoren λ_i zu eins aufaddieren, muss zumindest für einen der Graphen G_i gelten $u_i(C) \leq \alpha u(C)$. ◀

Im Folgenden wollen wir dieses Lemma verwenden, um optimale Schnitte in G zu approximieren. Gegeben eine Dekomposition des Graphen G, in eine strukturell einfachere Graphfamilie $\mathcal{G}$, berechnen wir in jedem Graphen G_i den optimalen Schnitt C_i. Aus diesen Schnitten wählen wir den minimalen Schnitt C_j aus. Der Schnitt C_j ist eine α-Approximation des optimalen Schnittes C^* in G.

$$u(C^*) \leq_1 u(C_j) \leq_2 u_j(C_j) \leq_3 u_k(C_k) \leq_4 u_k(C^*) \leq_5 \alpha u(C^*)$$

1: Optimalität von C^* 2: Untere Schranke 3: gilt für alle k da C_j optimal ist unter allen Graphen G_i
4: C_k ist optimal in G_k 5: gilt zumindest für einen Graph G_k

Analog erreicht man eine $\alpha\beta$-Approximation, indem man in den Graphen G_i nicht die optimalen Schnitte berechnet, sondern stattdessen in jedem Graphen eine β-Approximation des optimalen Schnitts bestimmt.

7.2.2 j-Bäume

Wie bereits beschrieben, wollen wir den Ausgangsgraphen in Graphen einer strukturell einfacheren Graphfamilie zerlegen. Von Räcke [4] gibt es bereits ein Ergebnis über eine Zerlegung in Bäume. Der dort vorgestellte Algorithmus berechnet jedoch $\widetilde{O}(m)$ mal das *all-pair shortest path* Problem und hat damit eine Laufzeit von ca. $\widetilde{O}(m \min\{mn, n^{2,376}\})$. Um eine bessere Laufzeit zu ermöglichen, gestatten wir in der Zerlegung komplexere Graphen als Bäume, die sogenannten j-Bäume.

▶ **Definition 5** (j-Bäume). Ein Graph G heißt j-Baum, wenn er aufgeteilt werden kann in einen Subgraphen K der Größe j ($|V_K| \leq j$) und einen Wald L (vergleiche Abbildung 46). Wobei jeder Baum des Waldes mit einem Knoten aus K verbunden ist. Wir nennen K den Kern und L die Hülle von G. Außerdem bezeichnen wir die Menge aller j-Bäume mit $\mathcal{J}[j]$.

Warum wählen wir gerade diese Graphfamilie? Die Familie der j-Bäume ist eine Verallgemeinerung von Bäumen, die Menge aller Bäume entspricht genau der Menge aller 1-Bäume. Auf Bäumen sind schnittbasierte Probleme oft sehr leicht lösbar. Dies wollen wir ausnutzen. Wenn man ein schnittbasiertes Problem auf einem j-Baum lösen will, so besteht das Hauptproblem darin, gute Schnitte innerhalb des Kerns effizient zu berechnen. Der Kern ist nach einer geeigneten Zerlegung kleiner als der ursprüngliche Graph. Dadurch ist das

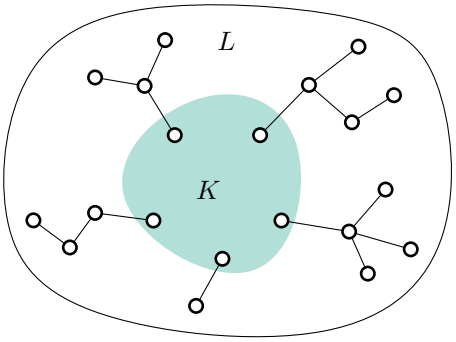


Abbildung 46 zeigt die schematische Darstellung eines j-Baums.

schnittbasierte Problem auf den j-Bäumen der Zerlegung einfacher als auf dem ursprünglichen Graphen.

Im Folgenden seien in jedem j-Baum der Kern und die Hülle bekannt. Dies ist keine Einschränkung, da wir einerseits bei der Konstruktion eines j-Baums die entsprechenden Teilgraphen markieren können und andererseits die Einteilung in linearer Zeit bestimmt werden kann ($L = \{v \mid \textit{nicht Teil eines Zykels}\}$ berechenbar mit einer Tiefensuche).

7.3 Existenz einer $(\alpha, \mathcal{J})$-Dekomposition

Das nun folgende Theorem stellt das zentrale Ergebnis dieser Ausarbeitung dar. Es liefert eine Dekomposition des Ausgangsgraphen G in j-Bäume. Die Komplexität der j-Bäume kann durch den Parameter t geregelt werden. Der Parameter t hat nicht nur Einfluss auf die Komplexität, auch die Laufzeit und die Menge der Graphen in der Dekomposition sind von t abhängig.

▶ **Theorem 6** (Existenz einer $(\alpha, \mathcal{J})$-Dekomposition). *Für einen Graphen $G = (V, E, u)$ und ein $t \geq 1$, können wir in $\widetilde{O}(tm)$ Zeit eine t-dünne $(\widetilde{O}(\log n), \mathcal{J}[\widetilde{O}(\frac{m \log U}{t})])$-Dekomposition $\{(\lambda_i, G_i)\}_i$ von G berechnen. Die maximale Kantenkapazität jedes G_i liegt dabei in $O(mU)$.*

Das Theorem liefert eine $(\widetilde{O}(\log n), \mathcal{J}[\widetilde{O}(\frac{m \log U}{t}])$-Dekomposition. Wir erhalten also eine $\alpha = \widetilde{O}(\log n)$ Approximation der Schnitte von G. In der Dekomposition ist der Graph G in t verschiedene $\widetilde{O}(\frac{m \log U}{t})$-Bäume zerlegt. Dabei werden die Kantenkapazitäten nur beschränkt größer. Dies ist vor allem interessant, wenn die Kantenkapazität in die Laufzeit des Algorithmus einfließt, der auf den vereinfachten Graphen der Dekomposition verwendet wird, um dort optimale Schnitte zu berechnen.

Dieses Theorem ist besonders interessant, da der Parameter t frei gewählt werden kann. Eine Veränderung von t hat immer positive und negative Folgen. Sowohl die Laufzeit, als auch die Menge und Komplexität der Graphen G_i hängt von t ab. Vergrößert man t, so wird der Kern und damit die Komplexität der j-Bäume kleiner. Dafür wird der Algorithmus proportional langsamer, und es werden mehr Graphen G_i erzeugt. Verringert man t, so geschieht das Gegenteil, der Algorithmus wird schneller und es werden weniger Graphen für die Zerlegung benötigt, aber die einzelnen Graphen werden komplexer.

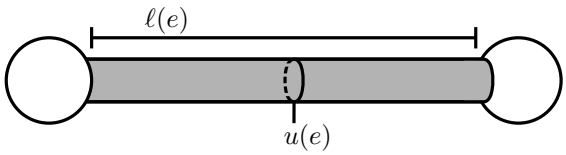


Abbildung 47 Anschaulich lässt sich das Volumen eines Graphen mit Rohren visualisierten

7.3.1 Beweisüberblick

Der Beweis von Theorem 6 besteht aus mehreren Abschnitten, die in zwei große Schritte eingeteilt werden können.

Im ersten Schritt berechnen wir eine Dekomposition von G in eine andere Graphfamilie, die Graphfamilie $\mathcal{H}[j]$. Auch hier beschreibt j ungefähr, wie sehr sich die Graphen von Bäumen unterscheiden. Der große Unterschied der beiden Graphfamilien ist, dass Graphen der Familie $\mathcal{H}$ immer Teilgraphen von G sind. Mit Hilfe von Theorem 9 werden wir zeigen, dass wir eine $(\widetilde{O}(\log n), \mathcal{H}[\widetilde{O}(\frac{m \log U}{t}])$-Dekomposition in $\widetilde{O}(tm)$ Zeit finden können.

Im zweiten Schritt werden wir die Dekomposition $\{(\lambda_i, H_i)\}_i$, die wir im ersten Schritt bestimmt haben, umwandeln in eine $(\widetilde{O}(\log n), \mathcal{J}[\widetilde{O}(\frac{m \log U}{t}])$-Dekomposition $\{(\lambda_i, G_i)\}_i$. Hierzu wird jeder Graph $H_i \in \mathcal{H}[j]$ der ursprünglichen Dekomposition in einen $\widetilde{O}(j)$-Baum G_i umgewandelt. Dabei muss jeder G_i die Schnitte des zugehörigen H_i approximieren, dies stellen wir sicher, indem jeder H_i einbettbar ist in G_i und jeder G_i c-einbettbar ist in H_i.

Der Einfachheit halber halten wir in diesem Kapitel sowohl den Ausgangsgraphen $G = (V, E, u)$ als auch die Zahl t fest ($n = |V|$, $m = |E|$, $U = \max_e u(e)$).

7.3.2 Erster Schritt

Der erste Schritt des Beweises lässt sich weiter unterteilen. Zuerst wollen wir Theorem 9 vorstellen, mit dessen Hilfe wir zunächst eine Dekomposition von G in Teilgraphen erhalten. Danach wollen wir die Graphfamilie $\mathcal{H}[j]$ vorstellen, die aus Teilgraphen von G besteht.

Bei der Anwendung von Theorem 9 müssen wir Teilgraphen von G konstruieren, die bestimmte Anforderungen erfüllen. Anschließend müssen wir zeigen, dass man entsprechende Graphen in der Graphfamilie $\mathcal{H}[j]$ finden kann.

Theorem zur Dekomposition in Teilgraphen Im Folgenden werden wir die Zerlegung durch ein Theorem konstruieren, dass ähnlich schon von Räcke in [4] verwendet wurde. Das Theorem liefert eine Zerlegung von G unter der Annahme, dass wir schnell Teilgraphen mit bestimmten Eigenschaften finden können. Um diese Eigenschaften zu formulieren, benötigen wir jedoch einige Definitionen.

Wir verwenden eine Längenfunktion $\ell : E \to \mathbb{R}^+$ auf den Kanten des Graphen, mit deren Hilfe wir das Volumen eines Teilgraphen definieren. Anschaulich lässt sich das Volumen eines Graphen visualisieren, indem man alle Kanten als Zylinder/Rohre betrachtet. Dabei entspricht die Grundfläche der Kantenkapazität und die Länge des Zylinders entspricht der Länge $\ell(e)$ der Kante (siehe Abbildung 47). Das Volumen des Graphen ist das aufsummierte Volumen aller Kanten/Zylinder.

▶ **Definition 7** (Volumen eines Graphen $\ell(H)$)**.** Wir definieren das Volumen eines Graphen $H = (V, E_H, u_H)$ bezüglich der Längenfunktion $\ell : E_H \to \mathbb{R}$.

$$\ell(H) = \sum_{e \in E_H} \ell(e) u_H(e)$$

Wie schon mehrfach beschrieben, ist nicht nur die Einbettbarkeit von G in die Graphen der Zerlegung wichtig. Auch anteilige Einbettungen von jedem Graphen der Zerlegung in G sind notwendig. Im ersten Schritt wollen wir eine Zerlegung in Teilgraphen H von G berechnen. Teilgraphen lassen sich sehr einfach in G einbetten, indem man jede Kante entlang der entsprechenden Kante in G einbettet. Wir nennen dies auch die Identitätseinbettung.

▶ **Definition 8** (inverse Auslastung γ (inverse congestion))**.** Für einen Graphen G und einen Teilgraphen H definieren wir die inverse Auslastung $\gamma_H(e)$ der Kante e (bei Verwendung der Identitätseinbettung). Das Minimum aller inversen Kantenauslastungen nennen wir auch die Auslastung des Teilgraphen H. Außerdem definieren wir die Menge $\kappa(H)$ als die Menge aller Kanten von H, die ähnlich stark ausgelastet sind wie die am stärksten ausgelastete Kante.

$$\begin{aligned}\gamma_H(e) &= \frac{u(e)}{u_H(e)}\\ \gamma(H) &= \min_{e\in E_H} \gamma_H(e)\\ \kappa(H) &= \{e \in E_H |\ \gamma_H(e) \leq 2\gamma(H)\}\end{aligned}$$

▶ **Theorem 9** (Konstruktion einer Dekomposition in Teilgraphen)**.** *Sei $\alpha \geq \ln m$ und $\mathcal{G}$ eine Graphfamilie. Wenn wir für jede Längenfunktion ℓ auf G in $\widetilde{O}(m)$ Zeit einen Teilgraphen $G_\ell \in \mathcal{G}$ von G finden, so dass:*

- *$\ell(G_\ell) \leq \alpha\ell(G)$*
- *G ist einbettbar in G_ℓ*
- *$|\kappa(G_\ell)| \geq \frac{4\alpha m}{t}$*

dann kann eine t-dünne $(2\alpha, \mathcal{G})$-Dekomposition von G in $\widetilde{O}(tm)$ Zeit berechnet werden.

Bei der Anwendung dieses Verfahrens ist zu beachten, dass G_ℓ ein Teilgraph von G ist. Da außerdem G einbettbar in G_ℓ ist muss $\gamma(G_\ell) \leq 1$ gelten. Wir werden hier das Theorem nicht vollständig beweisen, stattdessen wollen wir kurz eine Intuition geben, wie das Verfahren abläuft. Für einen vollständigen Beweis siehe Räcke [4].

Die $(2\alpha, \mathcal{G})$-Dekomposition von G wird iterativ aufgebaut, indem man einen Teilgraphen G_ℓ mit dem Faktor $\gamma(G_\ell)$ zur Dekomposition hinzufügt. Durch den Faktor $\gamma(G_\ell)$ wird sichergestellt, dass alleine durch G_ℓ keine Kante von G überlastet wird. Nach jeder Iteration wird die Längenfunktion aktualisiert, dabei verlängern wir stark ausgelastete Kanten. Diese können dementsprechend in folgenden Iterationen nicht so stark ausgelastet werden, da ansonsten die Volumeneinschränkung $\ell(G_\ell) \leq \alpha\ell(G)$ nicht erfüllt wird. Auf diese Weise stellen wir sicher, dass über alle t Iterationen keine Kante durchschnittlich mehr als α-ausgelastet wird.

Durch die Anforderung $|\kappa(G_\ell)| \geq \frac{4\alpha m}{t}$ erzwingen wir, dass jeder Iterationsschritt Fortschritt erzeugt. Denn die Auslastung von $|\kappa(G_\ell)|$ Kanten steigt um mindestens $\frac{1}{2}$ an. Da außerdem keine Kante eine Auslastung größer 2α erhält, lässt sich die Anzahl der benötigten Iterationsschritte und damit die Anzahl der Graphen der Zerlegung beschränken.

Nun werden wir zunächst die Graphfamilie $\mathcal{H}[j]$ vorstellen. Im Folgenden wählen wir für die Graphfamilie $\mathcal{G} = \mathcal{H}[\widetilde{O}(\frac{m \log U}{t})]$ und dementsprechend $G_\ell = H_\ell$. Dann werden wir zeigen, dass wir für ein $\alpha \in \widetilde{O}(\ln m)$ Teilgraphen $H_\ell \in \mathcal{H}[\widetilde{O}(\frac{m \log U}{t})]$ mit den geforderten Eigenschaften erzeugen können. Somit erhalten wir mit Hilfe von Theorem 9 die gewünschte Dekomposition.

Die Graphfamilie $\mathcal{H}[j]$ Um einen Graphen $H \in \mathcal{H}[j]$ zu konstruieren betrachten wir zunächst einen Spannbaum $T = (V, E_T)$ von G (dargestellt durch die dicken Kanten in Abbildung 48 links – unabhängig von der Farbe). Wir wählen bis zu j Kanten aus dem

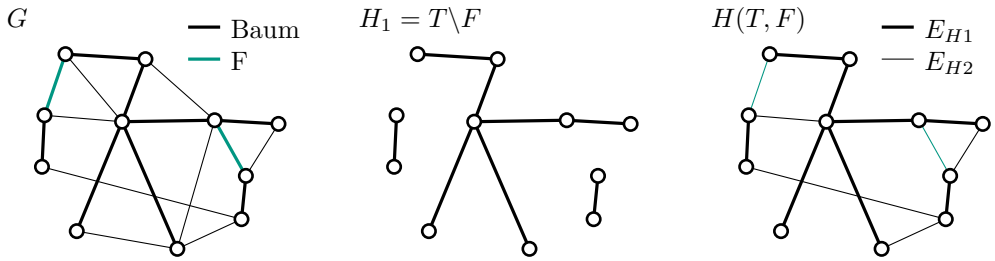


Abbildung 48 stellt die Konstruktion des Graphen $H(T, F)$ dar. Links: der Graph G mit dem Baum T dick hervorgehoben und den Kanten F farblich markiert; Mitte: der Baum T ohne die Kanten F; Rechts: der fertige Graph $H(T, F)$, die Kanten E_{H1} sind dick hervorgehoben, die Kanten E_{H2} sind schmal ($F \subseteq E_{H2}$ sind farblich markiert)

Baum T, diese bilden die Menge F (dargestellt durch die farbig markierten Kanten in Abbildung 48 links).

Betrachten wir zunächst $E_{H1} = E_T \backslash F$, damit wäre $H_1 = (V, E_{H1})$ ein Wald (siehe Abbildung 48 mitte). Um H_1 wieder zusammenhängend zu machen, fügen wir außerdem alle Kanten aus E hinzu, die zwischen unterschiedlichen Bäumen verlaufen. Diese bilden die Menge $E_{H2} = \{e \in E \mid \exists f \in F : f \in path_T(e)\}$, also alle Kanten, deren eindeutiger Pfad in T über mindestens eine Kante von F verläuft (siehe Abbildung 48 rechts dünn). Beachte, dass alle Kanten von F in E_{H2} liegen, also $F \subseteq E_{H2}$. Abhängig von T und F definieren wir den Graphen $H(T, F) = (V, E_H, u_H)$ mit $E_H = E_{H1} \cup E_{H2}$ (siehe Abbildung 48 rechts).

Zur Erfüllung des Theorems, müssen die Kantenkapazitäten so gewählt werden, dass G in $H(T, F)$ einbettbar ist. Wie in Abschnitt 7.2 beschrieben, lassen sich für den Baum T die Kantenkapazität u_T so wählen, dass G darin einbettbar ist. Wir definieren die Kapazitäten von H durch:

$$u_H(e) = \begin{cases} u_T(e) & \textit{falls } e \in E_{H1} \\ u(e) & \textit{falls } e \in E_{H2} \end{cases}$$

Die Menge aller Graphen $H(T, F)$, die auf diese Weise aufgebaut sind bilden die Graphfamilie $\mathcal{H}[|F|]$.

G ist einbettbar in $H(T, F)$. Für jede Kante $e = \{v, w\} \in E$ liegen entweder v und w innerhalb der selben Zusammenhangskomponente von H_1 oder e liegt per Definition in der Menge E_{H2}. Im ersten Fall liegt auch der eindeutige Pfad $path_T(e)$ komplett in dieser Zusammenhangskomponente. Alle Kanten e' auf diesem Pfad gehören zu E_{H1}, deshalb gilt $u(e') = u_T(e')$. Die Kante e lässt sich dementsprechend in $H(T, F)$ genauso einbetten wie in T. Im zweiten Fall gilt $e \in E_{H2}$ und damit ist die Kante auch im Graphen $H(T, F)$ enthalten und besitzt dort dieselbe Kapazität wie im Graphen G. Da keiner der sonstigen Flüsse diese Kante verwendet, lässt sich e in die entsprechende Kante von $H(T, F)$ einbetten.

Kanten aus F werden entlastet. In H_ℓ darf keine Kante deutlich stärker ausgelastet sein als andere Kanten, dass folgt aus $|\kappa(H_\ell)| \geq \frac{4\alpha m}{t}$. Dies werden wir erreichen, indem wir stark ausgelastete Kanten für F wählen, um diese zu entlasten.

Wir müssen also zeigen, dass alle Kanten aus F entlastet werden und keine stark ausgelastete Kante in $H(T, F)$ eingefügt wird. Alle Kanten, die wir betrachten müssen, liegen in der Menge E_{H2}. Jede Kante der Menge E_{H2} besitzt eine inverse Auslastung von eins, da $\gamma_H(e) = \frac{u(e)}{u_H(e)} = \frac{u(e)}{u(e)} = 1$.

Weiterhin gibt es in $H(T, F)$ keine Kante mit inverser Auslastung größer eins, da $u_H(e) \geq u(e)$. Dementsprechend ist klar, dass die Kanten von E_{H2} nicht zu stark ausgelastet sind. Außerdem müssen wir betrachten, wie lange die Konstruktion eines solchen Graphen benötigt.

▶ **Lemma 10** (Konstruktion von $H(T,F)$)**.** *Gegeben dem Ursprungsgraph G, einem Spannbaum $T = (V, E_T)$ und einer Kantenmenge $F \in E_T$ lässt sich der Graph $H(T,F) = (V, E_H, u_H)$ in $\widetilde{O}(m)$ Zeit berechnen.*

Beweis. Man kann in Linearzeit alle Zusammenhangskomponenten des Graphen H_1 markieren ($O(n)$). Danach kann man für jede Kante überprüfen, ob sie zu der Menge E_{H2} gehört ($O(m)$). Die Kantenkapazitäten u_T lassen sich, wie in Abschnitt 7.2 beschrieben, in $\widetilde{O}(m)$ berechnen. ◀

Um einen Graphen H_ℓ zu finden, der den Anforderungen von Theorem 9 mit $\mathcal{G} = \mathcal{H}[\widetilde{O}(\frac{m \log U}{t})]$ genügt, müssen wir nun in $\widetilde{O}(m)$ Zeit T_ℓ und F_ℓ so finden, dass $|F_\ell| \in \widetilde{O}(\frac{m \log U}{t})$, $\ell(H(T_\ell, F_\ell)) \leq \alpha\ell(G)$ und $|\kappa(H(T_\ell, F_\ell))| \geq \frac{4\alpha m}{t}$. Wir werden in zwei Schritten vorgehen. Zuerst wählen wir einen speziellen Spannbaum T_ℓ. Danach werden wir zeigen, dass wir F_ℓ so finden, dass $H(T_\ell, F_\ell)$ allen Anforderungen genügt.

Wähle einen guten Spannbaum für T_ℓ Als erstes werden wir einen Spannbaum T_ℓ bestimmen, der die Anforderung $\ell(T_\ell) \leq \alpha\ell(G)$ erfüllt. Danach werden wir zeigen, dass $\ell(H(T_\ell, F)) \leq \alpha\ell(G)$ unabhängig von F erhalten bleibt.

Um einen solchen Spannbaum zu finden verwenden wir einen Algorithmus für Spannbäume mit geringem durchschnittlichen Streckungsfaktor (low-average-stretch spanning trees). Wir wollen kurz definieren, was wir unter dem Streckungsfaktor verstehen, und wie man Bäume mit geringem durchschnittlichem Streckungsfaktor erhält. Sei $d^{\ell'}_{G'}(v,w)$ die kürzeste Wege Metrik in G' bezüglich der Längenfunktion ℓ', $d^{\ell'}_{G'}(v,w)$ ist also die Länge des kürzesten Weges von v nach w in G'. Der Streckungsfaktor einer Kante e bezüglich dem Spannbaum T' von G' ist dementsprechend

$$stretch^{\ell'}_{T'}(e) = \frac{d^{\ell'}_{T'}(u,v)}{d^{\ell'}_{G'}(u,v)}.$$

Der durchschnittliche Streckungsfaktor eines Spannbaums ist $\frac{1}{|E'|}\sum_e stretch^{\ell'}_{T'}(e)$. Das folgende Theorem 11 stammt von Abraham, Bartal und Neiman [1].

▶ **Theorem 11** (Spannbäume mit geringem durchschnittlichen Streckungsfaktor)**.** *Sei ℓ' eine Längenfunktion auf dem Graph $G' = (V', E')$. Dann lässt sich in $\widetilde{O}(|E'|)$ Zeit ein Spannbaum T' konstruieren, für den gilt*

$$\frac{1}{|E'|}\sum_{e\in E} stretch^{\ell'}_{T'}(e) \in \widetilde{O}(\log |V|).$$

Mit Hilfe dieses Theorems wollen wir nun das folgende Lemma Beweisen.

▶ **Lemma 12** (Finden eines geeigneten Spannbaums)**.** *In $\widetilde{O}(m)$ Zeit lässt sich ein Spannbaum $T_\ell = (V, E_{T_\ell}, u_{T_\ell})$ von G finden, so dass $\ell(T_\ell) \leq 2\overline{\alpha}\ell(G)$ für ein $\overline{\alpha} \in \widetilde{O}(\log n)$*

Beweis. Betrachten wir einen beliebigen Spannbaum $T = (V, E_T)$ von G, mit Kantenkapazitäten u_T.

$$\ell(T) = \sum_{f\in E_T} \ell(f)u_T(f) = \sum_{f\in E_T} (\ell(f) \sum_{e\in E | f\in path_T(e)} u(e)) = \sum_{e=(v,w)\in E} d^\ell_T(v,w)u(e)$$

Für jede Kante e gilt $d^\ell_G(e) \leq \ell(e)$ und deshalb $1 \leq \frac{\ell(e)}{d^\ell_G(e)}$

$$\ell(T) = \sum_{e=(v,w)\in E} d^\ell_T(v,w)u(e) \;\leq\; \sum_{e\in E} stretch^\ell_T(e)\ell(e)u(e)$$

Um das Lemma zu beweisen, müssen wir also zeigen, dass wir in $\widetilde{O}(m)$ Zeit einen Spannbaum finden, für den $\sum_{e\in E} stretch_T^{\ell}(e)\ell(e)u(e) \leq 2\overline{\alpha}\ell(G)$.

Zu diesem Zweck modifizieren wir den Graphen G, indem wir bereits vorhandene Kanten duplizieren. Dadurch erhalten wir den Multigraphen $\overline{G}$, in diesem kommt jede Kante $e \in E$ $r(e)$ mal vor.

$$r(e) \;=\; 1 + \lfloor \frac{\ell(e)u(e)|E|}{\ell(G)} \rfloor$$

Es ist klar, dass ein Spannbaum von $\overline{G}$ auch immer ein Spannbaum von G ist. Die zusätzlichen Kanten verändern jedoch den durchschnittlichen Streckungsfaktor $\frac{1}{|\overline{E}|}\sum_{e\in\overline{E}} stretch_T^{\ell}$, da die Streckungsfaktoren von duplizierten Kanten stärker gewichtet werden. Wir werden zeigen, dass wenn wir für T_ℓ einen Spannbaum mit geringem durchschnittlichen Streckungsfaktor von $\overline{G}$ wählen die Anforderungen des Lemmas erfüllt werden.

Zuerst wollen wir jedoch zeigen, dass wir mit Hilfe von Theorem 11 in $\widetilde{O}(m)$ Zeit diesen Spannbaum finden. Zunächst ist dazu wichtig, dass sich Theorem 11 auch auf Multigraphen übertragen lässt es hat jedoch auf $\overline{G}$ dementsprechend die Laufzeit $\widetilde{O}(|\overline{E}|)$. Wir müssen also zeigen, dass $\widetilde{O}(|\overline{E}|) = \widetilde{O}(m)$:

$$|\overline{E}| \;=\; \sum_e r(e) \;\leq\; |E| + \sum_e \frac{\ell(e)u(e)|E|}{\ell(G)} \;=\; |E| + |E|\frac{\sum_e \ell(e)u(e)}{\ell(G)} \;=\; 2|E|$$

d.h. wir haben die Kantenanzahl von G zu $\overline{G}$ maximal verdoppelt.

Nun betrachten wir einen Spannbaum T_ℓ von $\overline{G}$ mit geringem durchschnittlichem Streckungsfaktor. Nach Theorem 11 ist der durchschnittliche Streckungsfaktor kleiner als ein $\overline{\alpha} \in \overline{O}(\log n)$.

$$\frac{1}{|\overline{E}|}\sum_{e\in E_{\overline{G}}} stretch_{T_\ell}^{\ell}(e) \;=\; \frac{1}{|\overline{E}|}\sum_{e\in E} stretch_{T_\ell}^{\ell}(e)r(e) \;\leq\; \overline{\alpha}$$

Für jede Kante e gilt

$$r(e) \;=\; 1 + \lfloor \frac{\ell(e)u(e)|E|}{\ell(G)} \rfloor \;\geq\; \frac{\ell(e)u(e)|E|}{\ell(G)} \;\geq\; \frac{\ell(e)u(e)|\overline{E}|}{2\ell(G)}$$

und damit

$$\frac{\sum_e stretch_{T_\ell}^{\ell}(e)\ell(e)u(e)}{2\ell(G)} \;=\; \sum_e stretch_{T_\ell}^{\ell}(e)\frac{\ell(e)u(e)}{2\ell(G)} \;\leq\; \frac{1}{|\overline{E}|}\sum_e stretch_{T_\ell}^{\ell}(e)r(e) \;\leq\; \overline{\alpha}.$$

Es gilt für ein $\overline{\alpha} \in \widetilde{O}(\log n)$

$$\ell(T_\ell) \;\leq\; \sum_e stretch_{T_\ell}^{\ell}(e)\ell(e)u(e) \;\leq\; 2\overline{\alpha}\ell(G).$$

Der Baum T_ℓ erfüllt alle Anforderungen von Lemma 12. Er kann in $\widetilde{O}(m)$ Zeit erzeugt werden, indem man den Graphen $\overline{G}$ erzeugt und auf ihn Theorem 11 anwendet. ◀

Betrachten wir den Graphen $T_\ell = (V, E_{T_\ell}, u_{T_\ell})$, so erkennen wir, dass er zwei der drei Anforderungen von Theorem 9 erfüllt. Sein Volumen ist beschränkt durch $\alpha\ell(G)$ für ein $\alpha \in \widetilde{O}(m)$ und G ist einbettbar in T_ℓ.

Wir werden zeigen, dass unabhängig von F für das Volumen gilt: $\ell(H(T_\ell, F)) \leq \alpha\ell(G)$. Betrachte die zwei Teilgraphen $H_1 = (V, E_{H1}, u_{T_\ell})$ und $H_2 = (V, E_{H2}, u)$. Wobei E_{H1}

und E_{H2} definiert sind wie in Abschnitt 7.3.2. Es ist klar, dass $\ell(H(T,F)) = \ell(H_1) + \ell(H_2)$. Weiterhin gilt $E_{H1} \subseteq T_\ell$ und $E_{H2} \subseteq E$.

$$\ell(H(T,F)) \;=\; \ell(H_1) + \ell(H_2) \;\leq\; \ell(T_\ell) + \ell(G) \;\leq\; 2\overline{\alpha}\ell(G) + \ell(G) \;=\; (2\overline{\alpha}+1)\ell(G)$$

Für $\alpha \;=\; 2\overline{\alpha}+1 \;\in \widetilde{O}(\ln m)$ ist egal welche Kanten in der Menge F liegen, das Volumen des Graphen $H(T,F)$ genügt den Anforderungen von Theorem 9 – unabhängig von F.

Reduzieren der Auslastung Bisher haben wir mit $H(T_\ell, F)$ unabhängig von F einen Graphen, der bereits zwei Anforderungen von Theorem 9 erfüllt. Erstens hat er ein beschränktes Volumen und zweitens lässt sich G in ihn einbetten. Wenn wir also folgendes Lemma beweisen, können wir Theorem 9 anwenden und erhalten dadurch die gewünschte $(2\alpha, \mathcal{H}[\widetilde{O}(\frac{m \log U}{t})])$-Dekomposition.

▶ **Lemma 13** (Finden von F_ℓ)**.** *In $\widetilde{O}(m)$ Zeit lässt sich eine Menge $F_\ell \subseteq E_{T_\ell}$ finden, so dass der Graph $H(T_\ell, F_\ell)$ alle Anforderungen von Theorem 9 für $\alpha = 2\overline{\alpha}+1$ und $\mathcal{G} = \mathcal{H}[\widetilde{O}(\frac{m \log U}{t})]$erfüllt. Dabei ist $|F_\ell| \in \widetilde{O}(\frac{m \log U}{t})$.*

Beweis. Wir zeigen mit Hilfe des Schubfachprinzips, dass in den $\widetilde{O}(\frac{m \log U}{t})$ am stärksten ausgelasteten Kanten von T_ℓ, zumindest eine Menge K von mehr als $\frac{4\alpha m}{t}$ Kanten existiert, deren inverse Auslastung $\gamma_{T_\ell}(e)$ sich maximal um einen Faktor von zwei unterscheidet.

Wir wählen F_ℓ als die Menge aller Kanten, die stärker ausgelastet sind als die am stärksten ausgelastete Kante von K. Wie bereits in Abschnitt 7.3.2 beschrieben, sind in dem Graphen $H(T_\ell, F_\ell)$ die Kanten F_ℓ entlastet. Deshalb liegt die am stärksten ausgelastete Kante von $H(T_\ell, F_\ell)$ in K. Da sich außerdem die Auslastung der Kanten aus K nur um einen Faktor von 2 unterscheidet, gilt $K \subseteq \kappa(H(T_\ell, F_\ell))$ und damit $|\kappa(H(T_\ell, F_\ell))| \geq \frac{4\alpha m}{t}$.

Die inverse Auslastung der Kanten in T_ℓ lässt sich eingrenzen durch $\frac{1}{mU} \leq \gamma_{T_\ell}(e) = \frac{u(e)}{u_{T_\ell}(e)} \leq 1$. Die Kapazität $u_{T_\ell}(e)$ ist kleiner als mU, da maximal m Kanten im Baum T_ℓ über e geleitet werden und jede einzelne Kante hat eine Kapazität kleiner U. Außerdem gilt $u_{T_\ell}(e) \geq u(e)$.

Wir partitionieren die Kanten des Baumes T_ℓ abhängig von ihrer inversen Auslastung $\gamma_{T_\ell}(e)$ in die Mengen $F_0, ..., F_{\lfloor \log(mU) \rfloor}$.

$$F_j \;=\; \{e \in E_{T_\ell} \mid \frac{2^j}{mU} \leq \gamma_{T_\ell} < \frac{2^{j+1}}{mU}\}$$

Jede Kante aus E_{T_ℓ} liegt in genau einer Menge F_j, dementsprechend gilt $\dot{\bigcup}_j F_j = E_{T_\ell}$. Wir wählen j^* maximal, so dass immernoch gilt:

$$\sum_{j=0}^{j^*} |F_j| \;\leq\; \frac{4(2\overline{\alpha}+1)m(\lfloor \log(mU) \rfloor + 1)}{t} \;\in\; \widetilde{O}(\frac{m \log U}{t}).$$

Wenn wir $F_\ell \subseteq \dot{\bigcup}_{j=0}^{j^*} F_j$ wählen, dann ist klar $|F_\ell| \in \widetilde{O}(\frac{m \log U}{t})$ und damit $H(T_\ell, F_\ell) \in \mathcal{H}[\widetilde{O}(\frac{m \log U}{t})]$.

Falls $j^* = \lfloor \log(mU) \rfloor$, wählen wir $F_\ell = \dot{\bigcup}_{j=0}^{j^*} F_j = E_{T_\ell}$. Somit ist $H(T_\ell, F_\ell) = G$ und erfüllt alle Bedingungen ($\gamma_G(e) = 1$ für alle e und damit $|\kappa(G)| = m$).

Wenn $j^* < \lfloor \log(mU) \rfloor$, so folgt daraus,

$$\sum_{j=0}^{j^*+1} |F_j| \;>\; \frac{4(2\overline{\alpha}+1)m(\lfloor \log(mU) \rfloor + 1)}{t}.$$

Mit Hilfe des Schubfachprinzips lässt sich nun einfach zeigen, dass eine der Partitionen $F_0, ..., F_{j^*+1}$ mindestens $\frac{4(2\overline{\alpha}+1)m}{t}$ Kanten enthält ($j^* + 2 < \lfloor \log(mU) \rfloor + 1$). Wir nennen diese Partition $\overline{j}$. Wir definieren $F_\ell = \dot{\bigcup}_{j=0}^{\overline{j}-1} F_j$

In $H(T_\ell, F_\ell)$ gilt $\gamma(H(T_\ell, F_\ell)) \geq \frac{2^{\overline{j}}}{mU}$, da jede Kante e mit $\gamma_{T_\ell}(e) < \frac{2^{\overline{j}}}{mU}$ in F_ℓ liegt. Die Kanten von F_ℓ sind in $H(T_\ell, F_\ell)$ entlastet (siehe Abschnitt 7.3.2). Daraus folgt, dass $F_{\overline{j}} \subseteq \kappa(H(T_\ell, F_\ell))$ und deshalb $|\kappa(H(T_\ell, F_\ell))| \geq |F_{\overline{j}}| \geq \frac{4(2\overline{\alpha}+1)m}{t}$ wie gesucht.

Um F_ℓ zu konstruieren, verteilen wir die Kanten entsprechend ihrer inversen Auslastung $\gamma_{T_\ell}(e)$ auf die Partitionen $F_0, ..., F_{\lfloor \log(mU) \rfloor}$. Danach müssen wir nur die passende Partition $F_{\overline{j}}$ finden. Beides funktioniert in Zeit $\widetilde{O}(m)$. ◀

Zusammenfassung erster Schritt Nach der Konstruktion, von $H(T, F)$ gilt unabhängig von dem Spannbaum T und der Menge F, dass G einbettbar ist in $H(T, F)$. Mit Lemma 12 haben wir gezeigt, dass wir in $\widetilde{O}(m)$ Zeit einen Spannbaum T_ℓ finden, so dass $\ell(T_\ell) \leq \alpha\ell(G)$. Wie in Abschnitt 7.3.2 gezeigt, folgt daraus, dass $\ell(H(T_\ell, F)) \leq \alpha\ell(G)$, unabhängig von F. Desweiteren haben wir in Lemma 13 gezeigt, dass wir ebenfalls in $\widetilde{O}(m)$ Zeit eine Kantenmenge $F_\ell \subseteq E_{T_\ell}$ mit $|F_\ell| \in \widetilde{O}(\frac{m \log U}{t})$ finden, so dass $|\kappa(H(T_\ell, F_\ell)| \geq \frac{4\alpha m}{t}$.

Wie wir in Lemma 10 gezeigt haben, lässt sich der Graph $H(T_\ell, F_\ell)$ gegeben T_ℓ und F_ℓ in Zeit $\widetilde{O}(m)$ konstruieren. Wenn man all diese Ergebnisse kombiniert, lässt sich mit Hilfe von Theorem 9 die gewünschte $(\widetilde{O}(\log n), \mathcal{H}[\widetilde{O}(\frac{m \log U}{t})])$-Dekomposition konstruieren.

7.3.3 Zweiter Schritt

Der zweite Schritt unseres Beweises dient dazu, einen beliebigen Graphen $H_i \in \mathcal{H}[j]$ in einen Graphen $G_i \in \mathcal{J}[O(j)]$ umzuwandeln. Wobei wir sicherstellen wollen, dass Schnitte von G_i die Schnitte in H_i bis auf einen konstanten Faktor c approximieren. Dies stellen wir sicher, indem wir G_i so konstruieren, dass H_i einbettbar ist in G_i und G_i c-einbettbar ist in H_i.

Wir gehen erneut in zwei Schritten vor. Zuerst transformieren wir den Graphen H_i in einen fast j-Baum, anschaulich reduzieren wir dabei die Menge der Knoten von denen Nicht-Baumkanten ausgehen. Danach müssen wir sicherstellen, dass Nicht-Baumkanten nur innerhalb des Kerns vorkommen.

Transformation in einen fast j-Baum Unter einem *fast-j-Baum* verstehen wir einen Baum mit zusätzlichen Kanten zwischen maximal j unterschiedlichen Knoten.

In $H(T, F)$ kann von jedem Knoten eine Nicht-Baumkante ausgehen (also eine Kante, die nicht in dem Baum T liegt). Bei der Konstruktion des fast-j-Baumes müssen wir diese Kanten auf $O(j)$ Knoten zusammenfassen. Unser Ansatz wird sein, die Kanten an den maximal $2j$ Knoten zu konzentrieren, die in H an Kanten aus F liegen.

▶ **Lemma 14.** *Für einen gegebenen Spannbaum T von G und die Kantenmenge $F \in E_T$ lässt sich in $\widetilde{O}(m)$ Zeit ein fast-$2|F|$-Baum $G(T, F)$ konstruieren, so dass $H(T, F)$ einbettbar ist in $G(T, F)$ und $G(T, F)$ 3-einbettbar ist in $H(T, F)$. Dabei wird die maximale Kantenkapazität höchstens verdoppelt.*

Beweis. Seien E_{H1} und E_{H2} wie in Abschnitt 7.3.2 definiert, also $E_{H1} = E_T \backslash F$ und $E_{H2} = E_{H(T,F)} \backslash E_{H1}$. Wir markieren die Knotenmenge $J = \{v \in V \mid \exists e = (v, w) \in F\}$ aller Knoten, die inzident zu einer Kante in F liegen (siehe Abbildung 49 links). Nun betrachten wir zu jeder Kante $e = (v, w) \in E_{H2}$ den eindeutigen Weg $path_T(e)$, der in T von v nach w führt (siehe Abbildung 49 mitte). Sei $v_1(e)$ jeweils der erste markierte Knoten auf diesem Weg und $v_2(e)$ der letzte markierte Knoten auf dem Weg. Beachte, für jede

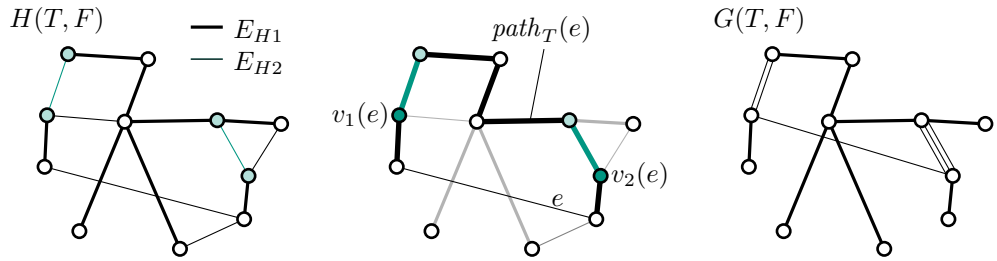


Abbildung 49 zeigt die Transformation von $H(T,F)$ zu $G(T,F)$. Links: der Graph $H(T,F)$ vgl. Abbildung 48 mit farblich markierten Knoten; Mitte: zu einer Kante e wird der zugehörige Weg $path_T(e)$ durch den Baum T betrachtet (dick), auf diesem liegen die zwei Knoten $v_1(e)$ und $v_2(e)$; Rechts: der fertige fast-2$|F|$-Baum $G(T,F)$, Mehrfachkanten deuten an wie viele Kanten $e \in E_{H2}$ zu dieser Kante gebündelt werden;

Kante in E_{H2} existieren mindestens zwei markierte Knoten. Als Kantenmenge wählen wir $E_{G(T,F)} = E_{H1} \cup \{(v_1(e), v_2(e)) \mid e \in E_{H2}\}$ (siehe Abbildung 49).

$H(T,F) \hookrightarrow G(T,F)$: Wenn wir $H(T,F)$ in $G(T,F)$ einbetten, kann jede Kante $e \in E_{H1}$ in der entsprechenden Kante von $G(T,F)$ eingebettet werden. Jede Kante $e \in E_{H2}$ leiten wir entlang der neuen Kante $(v_1(e), v_2(e))$. Sei $e = (v,w) \in E_{H2}$ wir betten e zunächst von v zu $v_1(e)$ entlang $path_T(e)$ ein, dann über die Kante $(v_1(e), v_2(e))$ zu $v_2(e)$ und von dort zu w erneut entlang $path_T(e)$. Um die Flüsse dieser Einbettung realisierbar zu machen wählen wir die Kantengewichte $u_{G(T,F)}$ wie folgt:

$$u_{G(T,F)}(e) = \begin{cases} 2u_T(e) & \text{falls } e \in E_{H1} \\ \sum_{e' \mid (v_1(e'),v_2(e'))=e} u(e') & \text{sonst} \end{cases}$$

Mit diesen Kantengewichten ist $H(T,F)$ einbettbar in $G(T,F)$. Wir müssen noch zeigen, dass $G(T,F)$ 3-einbettbar ist in $H(T,F)$.

$G(T,F) \hookrightarrow H(T,F)$: Wir können erneut jede Kante $e \in E_{H1} \subseteq E_{G(T,F)}$ entlang ihrer Entsprechung in $H(T,F)$ einbetten, dies lastet jede Kante doppelt aus, da $u_{G(T,F)}(e) = 2u_{H(T,F)}(e)$. Jede andere Kante e ist $(v_1(e'), v_2(e'))$ von zumindest einer Kante $e' \in E_{H2}$. Für jedes solche e' betten wir einen Anteil $u(e')$ von e entlang von e' ein. Sei $e' = (v,w)$ wir betten den Anteil von e zunächst von $v_1(e)$ zu v ein (über $path_T(e')$), von dort über e' zu w und von dort zu w (über $path_T(e)$). Das Gewicht dieser Flüsse (zusammen) ist an jeder Baumkante e_T kleiner $u_T(e_T)$ (beachte Definition von u_T in Abschnitt 7.2). Insgesamt wird bei dieser Einbettung keine Kante mehr als dreifach ausgelastet.

Wir müssen nur noch zeigen, dass wir den Graphen $G(T,F)$ in $\widetilde{O}(m)$ Zeit berechnen können. Die einzige Schwierigkeit besteht darin, die Knoten $v_1(e)$ und $v_2(e)$ zu bestimmen. Zu diesem Zweck passen wir erneut der Algorithmus von Spielman und Teng [5] an (siehe Abschnitt 7.2 zur Berechnung von u_T). ◀

Von einem fast-j-Baum zu einem j-Baum Der Unterschied zwischen einem fast-j-Baum und einem j-Baum besteht darin, dass Nicht-Baumkanten in einem fast-j-Baum lange Zykel bilden können. Ein großer Zykel ist ein fast-2-Baum, da man den Zykel als lange Kette mit einer Nicht-Baumkante betrachten kann. Es ist jedoch klar, dass ein Zykel in einem j-Baum vollständig im Kern liegt und deshalb maximal j Knoten enthalten kann.

▶ **Lemma 15.** *Für einen fast-j-Baum $G' = (V, E', u')$ lässt sich in $\widetilde{O}(m)$ Zeit ein $O(j)$-Baum $\overline{G} = (V, \overline{E}, \overline{u})$ konstruieren, so dass G' einbettbar ist in $\overline{G}$ und $\overline{G}$ ist 3-einbettbar in G'. Bei der Konstruktion wird die maximale Kantenkapazität höchstens verdoppelt.*

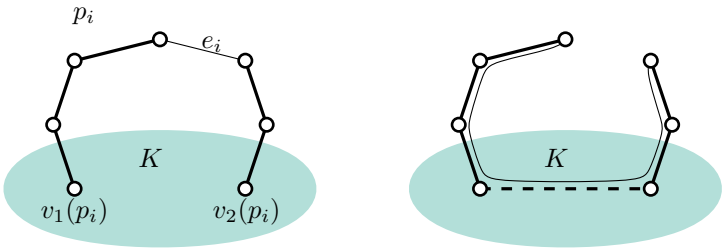


Abbildung 50 zeigt das Verfahren, mit dem die Pfade p_i getrennt werden.

Beweis. Wir werden den j-Baum $\overline{G}$ im Verlauf dieses Beweises erzeugen, indem wir Schritt für Schritt Kanten in $\overline{E}$ einfügen und die entsprechenden Kapazitäten $\overline{u}$ setzen. Gleichzeitig konstruieren wir den Kern K und die Hülle L des j-Baums.

Sei L_1 die Menge aller Knoten, die nicht Teil eines Zyklus in G' sind. Diese lassen sich in Zeit $O(m)$ finden. Der von L_1 induzierte Teilgraph ist ein Wald und soll Teil der Hülle von $\overline{G}$ werden. Deshalb fügen wir L_1 zu L hinzu. Außerdem fügen wir alle Kanten e die zu L_1 inzident sind zu $\overline{E}$ hinzu und setzen ihre Kapazität auf $u'(e)$.

Sei $G'_C = (V_C, E_C, u_C)$ der von $V \backslash L_1$ induzierte Graph. Dieser Graph besteht aus verbundenen Zyklen. Jeder Knoten in G'_C hat einen Grad von mindestens zwei. Wir betrachten die Menge aller Knoten mit einem Grad echt größer zwei in G'_C.

Aus diesen Knoten werden wir den Kern K von $\overline{G}$ aufbauen. Wir werden zeigen, dass $|K| \leq 3j - 2$. Der Graph G'_C ist immernoch ein fast-j-Baum. Es gibt also minimal $|V_C| - j$ Knoten in G'_C die nur inzident zu ursprünglichen Baumkanten sind. Die Gradsumme dieser Knoten beträgt maximal $2(V_C - 1)$ (Anzahl der Baumkanten mal zwei). Da in G'_C jeder Knoten einen Grad größer eins besitzt, kann es nur maximal $2(j - 1)$ Knoten mit einem Grad größer als zwei geben, die nur zu Baumkanten inzident sind. Gemeinsam mit den j Knoten, die zu Nicht-Baumkanten inzident sind gibt es also maximal $3j - 2$ Knoten in G'_C die einen Grad größer zwei besitzen. Es gilt also wie gefordert $|K| \leq 3j - 2 \in O(j)$. Wir fügen alle Kanten e zwischen zwei Knoten aus K zu $\overline{E}$ hinzu und initialisieren ihre Kapazitäten auf $u'(e)$.

Der von $F_2 = V_C \backslash K$ induzierte Graph besteht aus Pfaden p_i. Die Knotenmenge F_2 soll zur Hülle von $\overline{G}$ gehören. Ohne Anpassung der inzidenten Kanten ist dies jedoch nicht möglich, da jeder Pfad mit zwei Knoten aus der Menge K verbunden ist. Deshalb wiederholen wir für jeden Pfad p_i die folgenden Schritte (siehe Abbildung 50):

1. Wir finden die leichteste Kante e_i des Pfades.
2. Jede andere Kante e des Pfades fügen wir in $\overline{E}$ ein und setzen ihre Kapazität auf $u'(e) + u'(e_i)$, dabei wird die Kapazität jeder Kante maximal verdoppelt.
3. Seien $v_1(p_i)$ und $v_2(p_i)$ die Knoten in K mit denen p_i verbunden ist. Wir fügen die Kante $(v_1(p_i), v_2(p_i))$ in $\overline{E}$ ein und setzen ihre Kapazität auf $u'(e_i)$. Beziehungsweise wir erhöhen die Kapazität, falls die Kante schon existiert hat.

$G' \hookrightarrow \overline{G}$: Bei der Einbettung von G' in $\overline{G}$ kann jede Kante $e \in E'$, die auch in $\overline{E}$ liegt in sich selbst eingebettet werden. Jede andere Kante $e \in E'$ die nicht in $\overline{E}$ liegt muss die leichteste Kante eines Pfades p_i gewesen sein, e kann entlang dem Pfad p_i zusammen mit Kante $(v_1(p_i), v_2(p_i))$ eingebettet werden (siehe Abbildung 50).

$\overline{G} \hookrightarrow G'$: Bei der Einbettung von $\overline{G}$ in G' werden wir erneut jede Kante $e \in \overline{E}$, die auch in E' liegt in sich selbst einbetten. Kanten, die in Schritt drei der obigen Konstruktion erzeugt wurden, können wir entlang dem jeweiligen Pfad p_i einbetten. Bei dieser Konstruktion

wird keine Kante e mehr als dreifach ausgelastet, da $\overline{u}(e) \leq 2u'(e)$ und $u(e_i) \leq u'(e)$ (falls $e \in p_i$). ◀

Zusammenfassung zweiter Schritt Aus dem ersten Schritt erhalten wir eine t-dünne $(\alpha, \mathcal{H}[\widetilde{O}(\frac{m \log U}{t})])$-Dekomposition $\{(\lambda_i, H_i)\}_i$ in Zeit $\widetilde{O}(tm)$. Durch Lemma 14 können wir zu jedem Graphen H_i einen fast-j-Baum G_i' konstruieren. Aus diesem fast-j-Baum gewinnen wir mit Hilfe von Lemma 15 einen j-Baum G_i.

Dann ist $\{(\lambda_i, G_i)\}_i$ eine $(9\alpha, \mathcal{J}[\widetilde{O}(\frac{m \log U}{t})])$-Dekomposition. Denn G ist einbettbar in G_i durch die Verknüpfung der Einbettungen $G \hookrightarrow H_i \hookrightarrow G_i' \hookrightarrow G_i$. Analog ist G_i 9-einbettbar in H_i. Zusammen mit den Einbettungen f_i von H_i in G ergeben sich Einbettungen f_i' von G_i in G so dass $\sum_i \lambda_i |f_i'(e)| \leq \sum_i \lambda_i 9 |f_i(e)| \leq 9\alpha u(e)$ für jede Kante e.

7.4 Weiterentwicklungen des Ergebnisses

Durch Theorem 6 erhalten wir eine t-dünne $(\widetilde{O}(\log n), \mathcal{J}[\widetilde{O}(\frac{m \log U}{t})])$-Dekomposition des Graphen G. Mit dieser Dekomposition können wir minimale Schnitte von G approximieren, indem wir für jeden Graph G_i der Dekomposition den minimalen Schnitt berechnen/approximieren. Wir möchten kurz zwei Möglichkeiten vorstellen, mit denen sich das Verfahren optimieren lässt.

Rekursion auf dem Kern Der Kern eines j-Baums ist ein Graph mit j Knoten ohne strukturelle Besonderheit. Wir können demnach mit Theorem 6 auch eine Dekompositon des Kerns eines j-Baums bestimmen. Anschaulich zerlegen wir den Graphen rekursiv, dabei werden die Kerne der j-Bäume in jedem rekursiven Schritt kleiner.

Dies lässt sich verwenden um Dekompositionen zu verfeinern. Sei $\{(\lambda_i, G_i)\}_i$ eine $(\alpha, \mathcal{J}[j])$-Dekomposition des Graphen G. Wenn wir für jeden Kern K_i eines Graphen G_i der Dekomposition eine $(\beta, \mathcal{J}[k])$-Dekomposition $\{(\lambda_j^i, K_j^i)\}_j^i$ rekursiv berechnen, dann ist die Menge $\{(\lambda_i \lambda_j^i, H_i \cup K_j^i)\}_{i,j}$ eine $(\alpha\beta, \mathcal{J}[k])$-Dekomposition des ursprünglichen Graphen G.

Abtasten einer Dekomposition Bisher müssen wir um den minimalen Schnitt des Ausgangsgraphen zu approximieren, die minimalen Schnitte aller Graphen der Dekomposition bestimmen. Einerseits sind diese Graphen strukturell einfach, andererseits kann es sehr viele Graphen in der Dekomposition geben.

Unser Ziel ist es, nicht für jeden Graph in der Dekomposition den minimalen Schnitt zu berechen. Um dies zu erreichen werden wir nur die minimalen Schnitte einer Stichprobe aller Graphen berechnen. Es ist klar, dass die Qualität unserer Approximation darunter leidet. Es lässt sich jedoch über Markovungleichungen folgendes Lemma beweisen.

▶ **Lemma 16** (Abtasten einer Dekomposition). *Sei $\{(\lambda_i, G_i)\}_i$ eine $(\alpha, \mathcal{G})$-Dekomposition von G. Wenn wir mit der Wahrscheinlichkeitsverteilung gegeben durch λ_i zufällig einen Graphen G' aus der Menge $\{G_i\}_i$ auswählen ($P[G' = G_i] = \lambda_i$), dann gilt für jeden Schnitt C von G mit einer Wahrscheinlichkeit von $\frac{1}{2}$, dass $u_i(C) \leq 2\alpha u(C)$. Wir sagen auch, G' 2α-erhält die Schnitte mit einer Wahrscheinlichkeit von $\frac{1}{2}$.*

Aus Lemma 16 folgt, dass wenn wir eine Stichprobe von $O(\ln n)$ G_is nehmen, die nach der obigen Wahrscheinlichkeitsverteilung ausgewählt wurden, und auf diesen die minimalen Schnitte β-approximieren, dann erreichen wir mit hoher Wahrscheinlichkeit eine $2\alpha\beta$-Approximation des minimalen Schnitts von G.

Evolution des Ergebnisses Wenn wir diese beiden Ansätze kombinieren und in Theorem 6 integrieren, so erhalten wir das folgende Verfahren.

Kern des Verfahrens ist die Funktion *CutPreservingTree*(G,t,j) die für einen Graphen G genau einen j-Baum berechnet, der mit einer Wahrscheinlichkeit von p die Schnitte von G α-erhält. Dabei hängen α und p von den Parametern t und j ab. Die Funktion *CutPreserving-Tree*(G,t,j) hat einen einfachen Ablauf. Zuerst wird mit Hilfe von Theorem 6 eine t'-dünne Dekomposition bestimmt (t' abhängig von t). Diese wird wie in Lemma 16 beschrieben abgetastet, dass liefert den Graph G'. Falls G' bereits ein j-Baum ist wird er ausgegeben, ist dies nicht der Fall extrahieren wir den Kern $\overline{G}$ von G'. Auf diesem Kern rufen wir rekursiv *CutPreservingTree*($\overline{G}$,t^2,j) auf und fügen das Ergebnis wieder in die Hülle von G' ein. Der Parameter t wird quadriert, da $\overline{G}$ um einen Faktor t kleiner ist, als G.

Ziel des Verfahrens ist es, eine Menge $\{G_i\}_i$ zu erzeugen, so dass diese die Schnitte von G gut approximiert. Zunächst wird G ausgedünnt hierzu verwenden wir ein Verfahren von Benczúr und Karger [2], dieses verringert die Anzahl der Kanten ohne die Schnitte von G signifikant zu verändern. Danach rufen wir $(2^{k+1} \ln n)$ mal die Funktion *CutPreservingTree*(G,t,j) mit einem geschickt gewählten Parameter t auf. Dadurch erhalten wir die Menge $\{G_i\}_i$. Dieses Vorgehen liefert uns das folgende Theorem.

▶ **Theorem 17** (Evolution des Hauptergebnisses). *Für jedes $1 \geq l \geq 0$, jede ganze Zahl $k \geq 1$ und jeden Graph G, können wir in $\widetilde{O}(m + 2^k n^{(1+\frac{1-l}{2^k-1})} \log U)$ Zeit eine Menge von $(2^{k+1} \ln n)$ n^l-Bäumen $\{G_i\}_i$ finden, so dass:*

- **untere Schranke:** *für jeden Schnitt C und jeden Graphen G_i gilt: $u_i(C) \geq u(C)$*
- **obere Schranke:** *für jeden Schnitt C gibt es mit hoher Wahrscheinlichkeit ein i mit: $u_i(C) \leq (\log^{(1+o(1))k} n) u(C)$*

Außerdem liegt die maximale Kantenkapazität U_i jedes Graphen G_i in $O((\log^{(1+o(1))k} n)U)$.

Die Stärke dieses Theorems erkennt man zum Beispiel durch Einsetzen von $l = 0$ und $k = 3$. Dann folgt aus Theorem 17 in $\widetilde{O}(m + n^{\frac{8}{7}} \log U)$ Zeit eine Menge von $(16 \ln n)$ (1-)Bäumen. Die Laufzeit verbessert sich bei höherem k und wird quasi linear.

Wir werden Theorem 17 nicht ausführlich beweisen (für einen vollständigen Beweis siehe Madry [3]).

7.5 Zusammenfassung

Wir haben in dieser Ausarbeitung gezeigt, wie man für einen Graph G in $\widetilde{O}(tm)$ Zeit eine Dekomposition in t unterschiedliche j-Bäume bestimmt. Mit Hilfe dieser Dekomposition lassen sich optimale Schnitte des Ausgangsgraphen G effizient $\widetilde{O}(\log n)$ approximieren.

Außerdem haben wir motiviert, dass es sinnvoll ist, nicht nur Dekompositionen in Bäume zu betrachten. Durch die Einführung von j-Bäumen erreichen wir eine Flexibilität zwischen der Laufzeit und der Komplexität der Graphen in der Zerlegung.

Zuletzt haben wir angedeutet, wie man dieses Ergebnis anpassen kann, um eine annähernd lineare Laufzeit zu erreichen. Um dies zu erreichen, bauen wir die Dekomposition schrittweise durch Rekursion auf und verwenden zufälliges Abtasten um weniger Graphen der Komposition zu betrachten.

Referenzen

1 Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. *CoRR*, abs/0808.2017, 2008.

2 András A Benczúr and David R Karger. Approximating st minimum cuts in õ (n 2) time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 47–55. ACM, 1996.

3 A. Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 245–254, Oct 2010.

4 Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 255–264, New York, NY, USA, 2008. ACM.

5 Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *CoRR*, abs/cs/0607105, 2006.

8 Approximation Schemes for Maximum Weight Independent Set of Rectangles

Yassine Marrakchi

Abstract

Given a set of weighted rectangles, which subset of non-overlapping rectangles has the biggest total weight? This problem is a special case of the classical independent set problem. The proposed algorithm, originally presented by Adamaseck and Wiese [15] takes benefit from the geometric properties of the problem to define an equivalent problem with some more specification. They used a dynamic approach consisting on testing all possibilities to split recursively the original in smaller subproblems. Thanks to the suggested data structure, the algorithm is to the best of our knowledge the first $(1+\epsilon)$-approximation algorithm solving the considered problem in quasi-polynomial time.

8.1 Introduction

In graph theory, an independent set is a set of vertices in a graph such that any two vertices are not connected with an edge [16]. The INDEPENDENT SET problem is an ubiquitous optimization problem which has been extensively studied in many special settings as well as in the most general formulation. While the problem is originally intractable, better results are available in the geometric setting. Despite the restriction on a 2D plane, the problem remains NP-complete [7][10]. The improvement is reached in terms of approximation factors which depend heavily on the considered shapes. While for arbitrary shapes, the approximation factor reaches n^{ϵ} [8], a $(1+\epsilon)$-approximation is available for squares [6].

The special case assuming that shapes are rectangles is an interesting case for its practical use in data mining [9] [11] [12], channel admission control [13] and map labeling [1] [5]. A rudimentary requirement in the context of map labeling is that labels do not overlap. For reasons of genericity, we assume that each label is contained in a rectangle having a given position which means that some rectangles have to be omitted from the original set to insure disjunction. As labels can have different importance, we add a weight to each rectangle. The wanted subset of labels must have maximum total weight additionally to disjunction. The special case using rectangles has also been studied in previous works. $\mathcal{O}(\log n)$-approximation algorithms were proposed [11]. Chalermsook and Chuzhoy [3] consider the cardinality case and present a $\mathcal{O}(\log\log n)$-approximation algorithm. For the general weighted case, Chan and Har-Peled [4] give a $\mathcal{O}(\log n/\log\log n)$-approximation algorithm.

The best known approximation algorithm, which will be considered in this paper, is proposed by Adamaseck and Wiese [15] which is a $(1+\epsilon)$-approximation algorithm running in quasi-polynomial time: $\mathcal{O}(2^{poly(\log n/\epsilon)})$. The proposed algorithm uses a dynamic approach by breaking down the original problem into problems of smaller size recursively until all subproblems are trivial. Solutions of those subproblems contributes to build a solution for the original problem having the claimed approximation factor.

In the next section, we will present a formal definition of the considered problem. In section 3, we give a detailed of the proposed algorithm. The main lemma leading to prove the claimed approximation ratio is presented in section 4 as well as a proof that the required conditions are always held. The last section is devoted to recall the main results.

8.2 Problem definition

Let $\mathcal{R} = \{R_1, \ldots, R_n\}$be a set of n weighted parallel-axis rectangles in the plane. Each rectangle is defined as

$$R_i = \{(x, y) \in \mathbb{R} \times \mathbb{R} | \, x_i^1 < x < x_i^2, \, y_i^1 < y < y_i^2\}$$

where (x_i^1, y_i^1) and (x_i^2, y_i^2) are the coordinates of the left most bottom most and right most top most corner respectively (see Figure 1). That means that rectangles are open sets and boundaries are not included in the rectangles. Moreover each rectangle is assigned a weight $w(R_i) \in \mathbb{R}_+$. For the given set we want to find a disjoint subset $\mathcal{R}' \subseteq \mathcal{R}$, i.e,

$$R_i \in \mathcal{R}', \, R_j \in \mathcal{R}' \Rightarrow R_i \cap R_j = \emptyset$$

such that the total weight $w(\mathcal{R}') = \sum_{R \in \mathcal{R}'} w(R)$ is maximized.

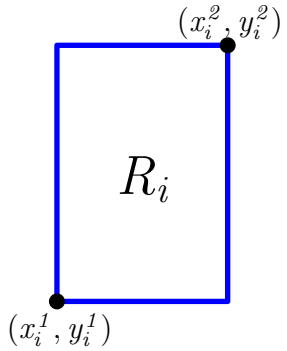


Figure 51 Rectangle specification

8.3 Algorithm

In this section, we will introduce the geometric dynamic algorithm proposed by Adamaszek and Wiese which is called GEO-DP. We start first by rescaling the weights of all rectangles. Then we redistribute all rectangles such that every rectangle has integer coordinates and start testing recursively possibilities to subdivide the current polygon. Each possible subdivision is evaluated and the subdivision having the best objective value is stored in the proposed data structure.

8.3.1 Rescaling

We start by rescaling the weights of rectangles such that $max_{R \in \mathcal{R}} w(R) = n/\epsilon$ for some $\epsilon > 0$. We have then for every instance I: $OPT(I) \geq n/\epsilon$ because we can select at least the heaviest rectangle. Then we remove $\mathcal{D} = \{R \in \mathcal{R} | \, w(R) < 1\}$ from the set of the given rectangles. Removing this subset of rectangles reduces the weight of the optimal solution by at most $\epsilon \cdot OPT(I)$ because

$$w(\mathcal{D}) = \sum_{R \in \mathcal{D}} w(R) \leq 1 \cdot n = \epsilon \cdot n/\epsilon \leq \epsilon \cdot OPT(I)$$

8.3.2 Redistribution

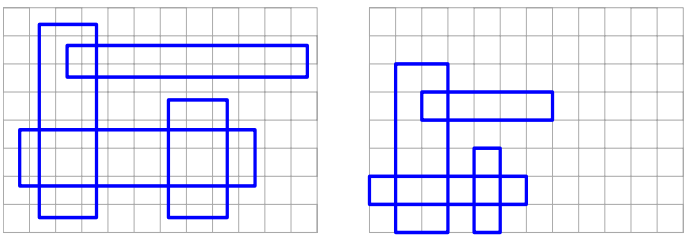


Figure 52 Example of combinatorially equivalent sets

The next step consists on finding a combinatorially equivalent set of rectangles to the given set such as the new rectangles have integer coordinates. Two sets of rectangles $\mathcal{R} = \{R_1, \ldots, R_n\}$ and $\mathcal{R}' = \{R'_1, \ldots, R'_n\}$ are called *combinatorially equivalent* when:

- $\forall\, i \in \{1, \ldots, n\} : w(R_i) = w(R'_i)$
- The order of x-values remains the same
- The order of y-values remains the same

An example of two combinatorially equivalent sets of rectangles is given in Figure 2. The targeted combinatorially equivalent set of rectangles has to satisfy the condition that all rectangle coordinates have to be integer ranging from 0 to $2n-1$. Such a set can be efficiently found: As we are considering at most n rectangles, abscissas of specification corners can take at most $2n$ different values (from 0 to $2n-1$ for example). We sort the abscissas and assign the smallest possible new abscissa to each corner such that all equivalences above hold. In this case, we need at most $2n$ different integers which occurs if and only if we are working with a set of n rectangles having pairwise different corner abscissas. The same strategy has to be applied for ordinates as well. The polygon $P_0 = [0,\, 2n-1] \times [0,\, 2n-1]$ is the input square.

8.3.3 Polygon subdivision

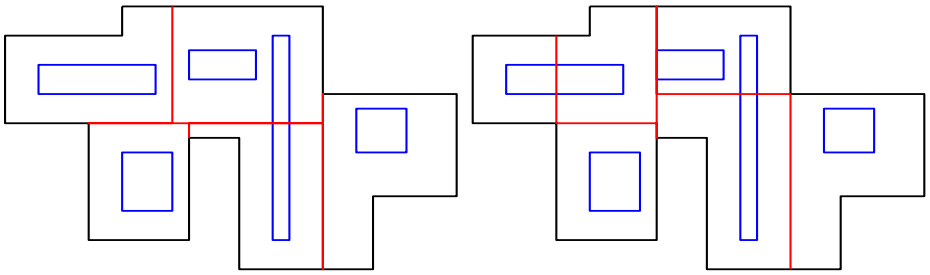


Figure 53 Two possible partitions of a polygon

In this step, we need a simple data structure called DP-table which will be introduced at first before discussing the algorithmic details.

The proposed algorithm needs a fixed global parameter k. Let $\mathcal{P}$ be the set of all parallel-axis polygons having integer coordinates in $\{0, \ldots, 2n-1\}$ and at most k edges. Notice that polygons may contain holes. In this case, the number of edges is defined as the sum of outer edges of the considered polygon and the number of edges bounding holes. The DP-table contains a cell for each polygon of $\mathcal{P}$. In each cell, we store a near optimal solution $sol(P)$ for the sub problem defined with the corresponding polygon P and the subset of rectangles $\mathcal{R}_P \subseteq \mathcal{R}$ such that $\forall\, R \in \mathcal{R}_P,\, R \subseteq P$.

As we are assuming that a polygon might have at most k edges, then every polygon has at most k corners. Since all corners have integer coordinates, the possibilities to choose a corner are bounded by $(2n)^2$ which means that there are at most $((2n)^2)^k \in n^{\mathcal{O}(k)}$ possible polygons, i.e, the DP-cell has at most $n^{\mathcal{O}(k)}$ cells. The computation starts from the cell defined by the input square P_0.

Computing the value of each DP-cell uses the following strategy: Let P and $\mathcal{R}_P$ be the polygon and the set of rectangles defining the problem for an arbitrary DP-cell. The trivial case occurs when $\mathcal{R}_P$ is an empty set. We set the DP-cell to $\emptyset$ and the weight of $sol(P)$ to 0. Otherwise, we compute all possibilities to partition P into at most k polygons having

at most k edges each, i.e, we consider every subset $\{P_1, \ldots, P_{k'}\}$ such that $k' \leq k$ and $\forall\, i \in \{1, \ldots, k'\} : P_i \in \mathcal{P}$ and check if the following conditions are held:

- Any two rectangles are disjoint: $\forall\, i, j \in \{1, \ldots, k'\} : i \neq j \Rightarrow R_i \cap R_j = \emptyset$
- Each polygon is contained in P: $\forall\, i \in \{1, \ldots, k'\} : P_i \subseteq P$
- P is totally covered: $\bigcup_{i=1}^{i=k'} P_i = P$

In Figure 3, we present two feasible partitions of a polygon. Since we pick up at most k polygons from $\mathcal{P}$, there is at most $n^{\mathcal{O}(k^2)}$ possible subsets. (Recall that $|\mathcal{P}|$ is bounded by $n^{\mathcal{O}(k)}$). To find out which subset has the best objective value, we need to calculate it for each.

8.3.4 Choosing the best partition

We initialize each DP-cell defined by a polygon P by setting $w(sol(P)) = max_{R \in \mathcal{R}_P} w(R)$ which means that we consider the trivial partition using only one polygon and the rectangle having the biggest weight is the only rectangle that would be added to searched subset $\mathcal{R}'$. Then for each feasible partition $\mathcal{P}_P = \{Par_1, \ldots, Par_{k'}\}$ of a polygon P, we calculate $w(\mathcal{P}_P) = \sum_{i=1}^{i=k'} w(Par_i)$ by looking for $w(Par_k)$ in the DP-cell defined by the polygon Par_k. If all subproblems defined by the polygons in $\mathcal{P}_P$ are already calculated, then we set the considered DP-cell if and only if $w(\mathcal{P}_p)$ is bigger than the current stored value. Otherwise, we start a new subproblem defined with the polygon Par_k which is not calculated up to now. The computation continues recursively. For checking if the subproblem defined with a polygon Par_k is already solved, we can use a table of boolean flags which has the same size as the DP-table. A cell is set when the corresponding problem is solved. When all possible partitions of P are entirely tested, the subproblem defined by P is solved and the corresponding boolean flag is set. At the end, the algorithm returns the value stored in the DP-cell corresponding to the input square P_0.

As the rescaling and the redistribution of rectangles occur in polynomial time, testing the feasibility of each partition is efficient because of the geometric properties of the problem and calculating $sol(P)$ occurs only once for each polygon P, the running time is upper bounded by $n^{\mathcal{O}(k^2)}$.

8.4 Bounding the approximation ratio

In this section, we want to find a specification of the global parameter k to get the wanted approximation ratio. Actually, we do not need to consider the original set of rectangles any more, but only the optimal subset of rectangles $\mathcal{R}^*$ because the algorithm tests all feasible partitions without taking care about the position of rectangles from $\mathcal{R}$. Obviously, the algorithm may find a subset $\mathcal{R}'$ containing rectangles from $\mathcal{R}^*$ and rectangles from $\mathcal{R}/\mathcal{R}^*$. In this section, we will show that rectangles from $\mathcal{R}^* \cap \mathcal{R}'$ are enough to reach an "acceptable" approximation ratio. We define first a special cut having some properties called balanced cheap cut: That is a polygon with bounded number of edges, which intersects rectangles of marginal weight and separates rectangles into two parts of similar size or there exists a rectangle R such that $w(R) \geq w(\mathcal{R}^*)/3$.. Then we show constructively that any set of feasible rectangles has a balanced cheap cut. From now on, we will work only on rectangles from $\mathcal{R}^*$.

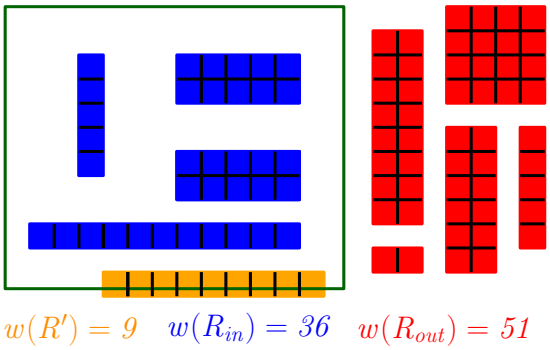


Figure 54 Example of balanced 0.1-cheap 4-cut

8.4.1 Balanced cheap cut

▶ **Definition 1.** Let $\ell \in \mathbb{N}$ and $\alpha \in \mathbb{R}$ with $0 < \alpha < 1$. Let $\mathcal{R}$ be a set of pairwise non-overlapping rectangles. A polygon P with axis-parallel edges is a *balanced* $\alpha - cheap\ \ell - cut$ if:

- P has at most ℓ edges
- $w(R') \leq \alpha \cdot w(R)$
- $w(R_{in}) \leq 2/3 \cdot w(R)$
- $w(R_{out}) \leq 2/3 \cdot w(R)$

where R', R_{in} and R_{out} are respectively subsets of $\mathcal{R}$ intersecting boundaries of P, contained in P and not intersecting or contained in P. Figure 4 presents an example of a balanced cheap cut.

The bound for approximation ratio is given in the following lemma.

▶ **Lemma 2.** *Let $\epsilon > 0$ and α and ℓ be values such that for any set of non-overlapping rectangles $\mathcal{R}^*$ there exists:*

- *a balanced α-cheap ℓ-cut, or*
- *$\exists R \in \mathcal{R}^*$ such that $w(R) \geq \cdot w(\mathcal{R})/3$*

Then the approximation ratio is $(1+\alpha)^{O(log(n/\epsilon))}$ for $k = \ell^2 \cdot O(log^2(n/\epsilon))$.

To make sure that the algorithm reaches the given bound for every initial set of rectangles, we need to prove that for any set of non overlapping rectangles, one of the two properties is held. In the reminder of this paper, we will focus on this point. The proof of the approximation ratio boundary is proposed and presented in details by Adamaszek and Wiese [15]. The remaining challenge is the proof that for any set of non-overlapping rectangles $\mathcal{R}^*$ exists:

- a balanced α-cheap ℓ-cut, or
- $\exists R \in \mathcal{R}^*$ such that $w(R) \geq \cdot w(\mathcal{R})/3$

We will show constructively that one of both conditions is held beginning with stretching the rectangles and partitioning the plane. The next step is to transform the partition $\mathcal{L}$ into a graph $G(\mathcal{L})$ and finally we apply the weighted variety of planar separator theorem proposed by Arora et al. [2] on $G(\mathcal{L})$. In this context, algorithmic aspects will be emphasized, while mathematical poofs will be omitted as they are available in the mentioned references respectively.

8.4.2 Stretching the plane

In this step, we aim to redistribute the rectangles such that weights are "smoothly" distributed. We formalize this aspect in the next definition:

▶ **Definition 3.** Let $\mathcal{R}$ be a given set of rectangles with integer coordinates in $\{0, \ldots, N\}$. $\mathcal{R}$ is $well - distributed$ if and only if $\forall \gamma > 0; \forall t \in \{0, \ldots, N\}$: The weight of rectangles contained in $[0, N] \times [t, t + \gamma \cdot N]$ is smaller than $2\gamma w(\mathcal{R})$. The same is required in the area $[t, t + \gamma \cdot N] \times [0, N]$.

Obviously, rectangles can be non-uniformly squeezed or stretched until the new set of rectangles is well distributed as shown in Figure 5. Note that the new set of rectangles is combinatorially equivalent to the original set. It can even be shown that such well-distributed combinatorially equivalent sets of rectangles use only coordinates in $\{0, \ldots, 4 \cdot |\mathcal{R}|\}$. A proof of this assertion is proposed in the original paper [15].

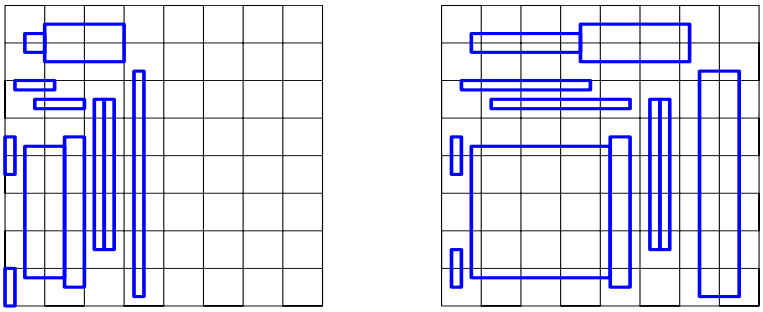


Figure 55 Example of stretching

Another important observation is that a stretched instance has a balanced α-cheap ℓ-cut if and only if the original instance has such a cut. This equivalence is evident because the polygon defining the cut does not need to have corners with integer coordinates. From now on, we can assume that the considered set of rectangle $\mathcal{R}$ has to be a well-distributed set of pairwise non-overlapping rectangles using integer coordinates ranging from 0 to N. Let $\delta > 0$ be a given real number such that $\frac{1}{\delta} \in \mathbb{N}$. As the choice of N is not upper bounded, assume also that $\delta^2 \cdot N$ is an integer.

8.4.3 Partitioning the plane

The purpose of this step is to build the set of given rectangles to a set of a straight axis-parallel lines $\mathcal{L}$. We consider the input square $[0, N] \times [0, N]$ and subdivide it into identical square cells of size $\delta^2 \cdot N \times \delta^2 \cdot N$. Then there are overall $\frac{1}{\delta^2} \times \frac{1}{\delta^2}$ $grid\,cells$. All corners of these cells have integer coordinates as $\delta^2 \cdot N$ is an integer. Lines defining the boundaries of grid cells are called $grid\,lines$. $R\,crosses\,Q$ a grid cell if and only if it intersects two opposite edges of the cell. In the next part of this paper we will need to distinguish between rectangles according to some properties. A graphical representation of each category is given in Figure 6. For this purpose, let $h(R)$ the height of R and $g(R)$ its width.

▶ **Definition 4.** If $h(R) > \delta^2 \cdot N$ or $g(R) > \delta^2 \cdot N$ then R is called a *large rectangle*. Otherwise, the rectangle is called a *small rectangle*.

Obviously, a large rectangle intersects at least 2 grid cells. Small rectangles intersects at most 4 grid cells and do not cross any grid cell.

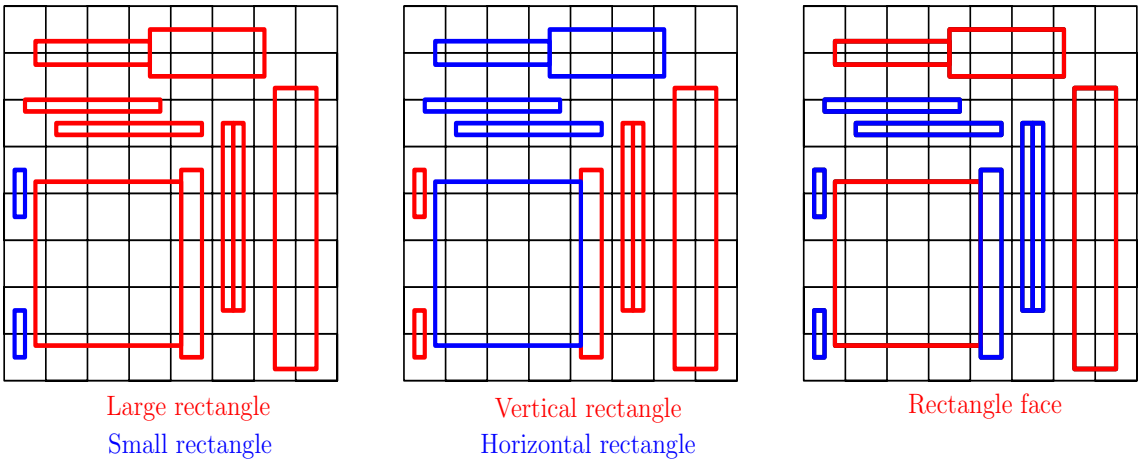


Figure 56 Categories of rectangles

▶ **Definition 5.** If $h(R) > g(R)$ the rectangle is called *vertical rectangle*. Otherwise the rectangle is called a *horizontal rectangle*.

Then, squares are horizontal rectangles. A grid cell can not be crossed at the same time by horizontal and vertical rectangles as we assume that rectangles are not allowed to be overlapping.

▶ **Definition 6.** R is called a *rectangle face* if and only if R is a large vertical rectangle crossed by a vertical grid line or R is a large horizontal rectangle crossed by a horizontal grid line.

There are $(\frac{1}{\delta})^2$ vertical lines that can cross rectangle faces. Each line crosses less than $(\frac{1}{\delta})^2$ rectangles. The same bounds are also valid for horizontal lines. Therefore, the number of rectangle faces is bounded by $2 \times (\frac{1}{\delta})^4$.

To construct $\mathcal{L}$, we start first by considering the rectangle faces. We add all edges of rectangle faces to $\mathcal{L}$. (step 1 in Figure 7) We consider then grid cells separately. Let $\mathcal{R}_\mathcal{Q}$ be the set of rectangles crossing a cell $\mathcal{Q}$ which are not rectangle faces. Obviously, $\mathcal{R}_\mathcal{Q}$ contains only large rectangles which means $\mathcal{R}_\mathcal{Q}$ contains only vertical or horizontal rectangles, otherwise, rectangles from $\mathcal{R}_\mathcal{Q}$ would overlap. According to the cardinality of $\mathcal{R}_\mathcal{Q}$ and the direction of rectangles in $\mathcal{R}_\mathcal{Q}$, we distinguish between three categories of cells:

- $\mathcal{R}_\mathcal{Q} = \emptyset$: all boundaries of $\mathcal{Q}$ are added to $\mathcal{L}$. Consider that a cell can be intersected by a rectangle face. In this case, we add exactly those portions from the boundaries that are not contained in a rectangle face. (step 2 in Figure 7)

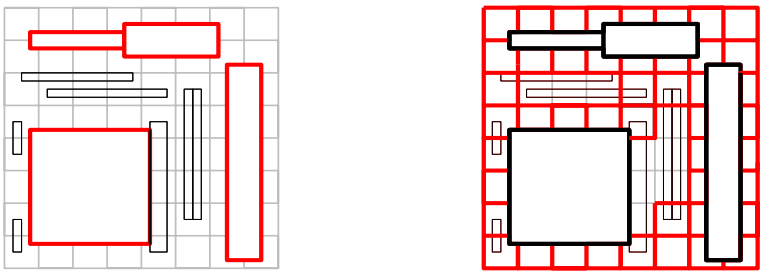


Figure 57 Construction of $\mathcal{L}$: step 1 and 2

- $\mathcal{R}_\mathcal{Q} \neq \emptyset$ and the rectangles are vertical: let L_r be the right most edge of all rectangles of $\mathcal{R}_\mathcal{Q} \cap \mathcal{Q}$ and L_ℓ be the left most edge of all rectangle of $\mathcal{R}_\mathcal{Q} \cap \mathcal{Q}$. We add L_r, L_ℓ as well as the rest of boundaries of $\mathcal{Q}$ which are not lying between L_r and L_ℓ or contained in a rectangle face. An example of construction of lines from $\mathcal{L}$ is given in Figure 8.(see also step 1 in Figure 9 for the allying of this step for a whole set of grid cells.)

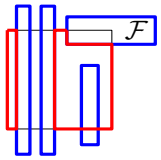


Figure 58 Adding edges when $\mathcal{R}_\mathcal{Q}$ contains large vertical rectangles

- $\mathcal{R}_\mathcal{Q} \neq \emptyset$ and the rectangles are horizontal: this case is analog to the last one. The only difference is that we consider the top-most and the bottom-most edges of rectangles from $\mathcal{R}_\mathcal{Q}$ instead of the right-most and left-most edges. (step 2 in Figure 9)

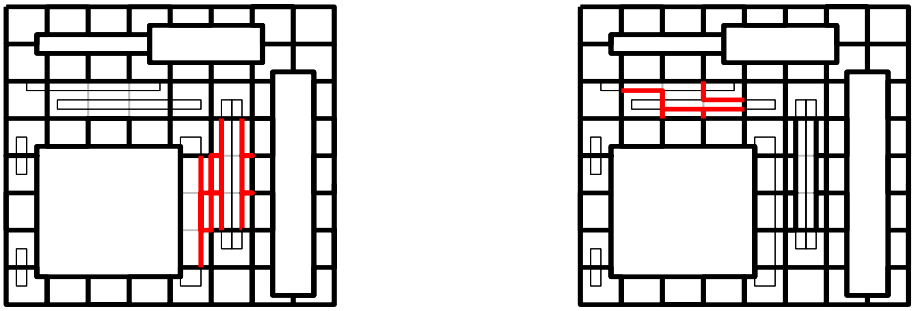


Figure 59 Construction of $\mathcal{L}$: step 3 and 4

The set $\mathcal{L}$ of lines is a set of pairwise properly non-intersecting lines. That means that the intersection of two lines contains at most one point, which is a common end point.

8.4.4 Construction of $G(\mathcal{L})$ based on $\mathcal{L}$

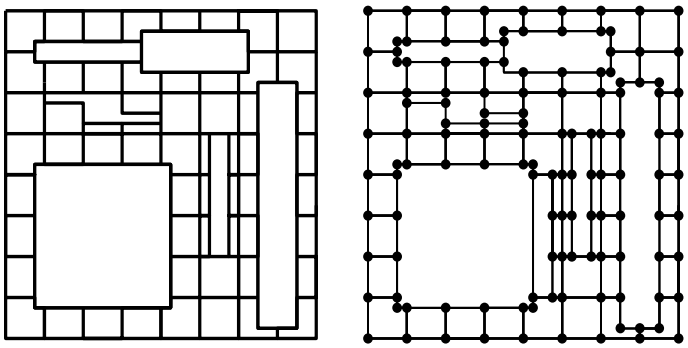


Figure 60 Construction of $G(\mathcal{L})$from $\mathcal{L}$

From the set of lines $\mathcal{L}$, we define the graph $G(\mathcal{L}) = (V, E)$ with respect to the following two rules:

- For any point $p \in [0, N] \times [0, N]$: p is an endpoint of one line $L \in \mathcal{L} \Rightarrow p$ becomes a vertex v_p of $G(\mathcal{L})$
- For any pair $v_p, v_q \in V : \exists L \in \mathcal{L}$ such that p and q are directly linked end points of $L \Rightarrow \{v_p, v_q\} \in E$

Graphically, we can start from a representation of $\mathcal{L}$ and assign a vertex at each corner as shown in Figure 10. The resulting representation contains all vertices as well as all edges which are induced by the lines of $\mathcal{L}$.

In the next lemmata, we present some properties of $G(\mathcal{L})$

▶ **Lemma 7.** *The graph $G(\mathcal{L})$ is planar and has $O(1/\delta^4)$ vertices and $O(1/\delta^4)$ edges.*

Beweis. The planarity of $G(\mathcal{L})$ follows since we start from a 2D embedding of L which is free from intersections as noticed above. We can get a vertex in $G(\mathcal{L})$ from a corner of a rectangle face of an intersection point of a rectangle and a boundary of a cell. Each rectangle face has 4 corners (dark red nodes in Figure 11) and we have at most $2.(\frac{1}{\delta})^4$ rectangle faces which gives at most $8 \cdot (1/\delta)^4$ from the first type. From the second type, we can get at most 12 vertices per cell (red vertices in Figure 11) and we have $(1/\delta)^4$ cells. Overall $|V| \leq 20 \times (1/\delta)^4$. As we allow only axis-parallel lines in L, each vertex has at most degree 4 which gives $|E| \leq 80 \cdot (1/\delta)^4$. ◀

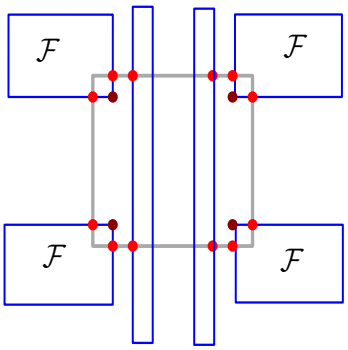


Figure 61 Maximum number of vertices per cell

▶ **Lemma 8.** *Each rectangle from $\mathcal{R}$ can be intersected by at most 4 edges of $G(\mathcal{L})$*

Beweis. We distinguish between 3 types of rectangles:

- R is a small rectangle $\Rightarrow R$ is intersected by at most 4 edges
- R is a large rectangle contained in one row or column $\Rightarrow R$ is intersected by at most 2 edges
- R is a large rectangle which is not contained in one row or column $\Rightarrow R$ is contained in a rectangle face $\Rightarrow$ R is not intersected by any edge

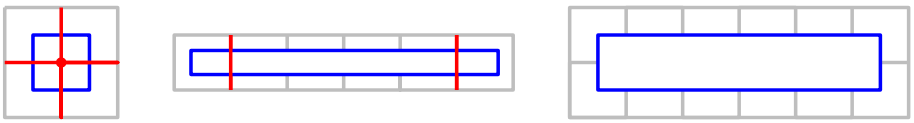


Figure 62 Maximal number of crossing edges per rectangle

◀

▶ **Lemma 9.** *The boundary of a face $\mathcal{F}$ intersects rectangles from $\mathcal{R}$ of total weight at most $8\delta^2 w(\mathcal{R})$.*

Beweis. The construction of $\mathcal{L}$ ensures the following properties:

- If x is a corner of a grid cell that does not belong to any line $L \in \mathcal{L}$ then $\exists R \in \mathcal{R}$: R is a rectangle face *and* $p \in R$.
- Let e be an edge of a grid cell $\mathcal{Q}$, $x \in e$ such that x is not a corner of $\mathcal{Q}$. If x does not belong to any rectangle face and $x \notin \mathcal{L}$ then $\mathcal{Q}$ is crossed with a large rectangle. Note also that if e is a horizontal edge then large rectangles crossing $\mathcal{Q}$ are vertical and if e is a vertical edge then large rectangles crossing $\mathcal{Q}$ are horizontal.

We distinguish between three types of faces in $\mathcal{L}$:

- $\mathcal{F}$ is a rectangle face then the upper bound is proven in Definition 6.
- Let $\mathcal{F}$ be a face from L contained in one grid cell $\mathcal{Q}$ (see Figure 13). Rectangles intersecting $\mathcal{F}$ are either small or large. Large rectangles are contained in two stripes of width $\delta^2 N$ and 4 other grid cells. Under the assumption that rectangles are well-distributed, the weight of rectangles intersecting $\mathcal{F}$ is less than $8\delta^2 w(\mathcal{R})$.

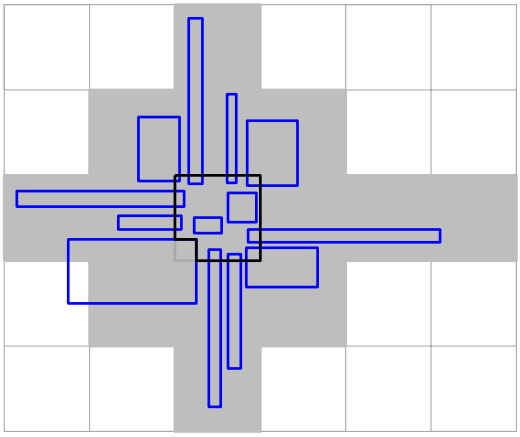


Figure 63 $\mathcal{F}$ contained in one grid cell

- $\mathcal{F}$ is not a rectangle face and is not contained in one grid cell (see Figure 14). In this case, $\mathcal{F}$ must be contained either in one vertical column or in one horizontal column. Otherwise, we get from the above indicated properties that a cell contains horizontal and vertical large rectangles at the same time which is forbidden. Assume without loss of generality that the considered face is contained in one vertical stripe. The construction of $\mathcal{L}$ ensures that rectangles intersecting $\mathcal{F}$ are also contained in the same vertical stripe as $\mathcal{F}$ which means that the total weight of rectangles intersecting $\mathcal{F}$ is upper bound with $2\delta^2 w(\mathcal{R})$.

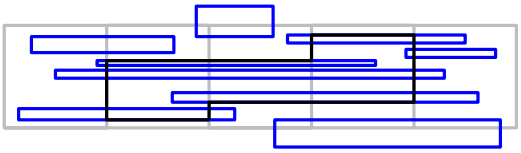


Figure 64 $\mathcal{F}$ contained in more than one grid cell and is not a rectangle face

◀

The graph $G(\mathcal{L})$ gets weights on its vertices, edges as well as faces. Vertices get all the default value 0. The cost of each edge is defined as the total weight of rectangles intersecting its corresponding line in $\mathcal{L}$. Each Face $\mathcal{F}$ is weighted with the total weight of rectangles that are included and a fraction m from the weight of each rectangle intersecting at least one other face. An example of assignment of costs and weights is given in Figure 15. Let R be a rectangle intersecting ℓ faces such that $R \cap \mathcal{F} \neq \emptyset$. In this case, $m = w(R)/\ell$.

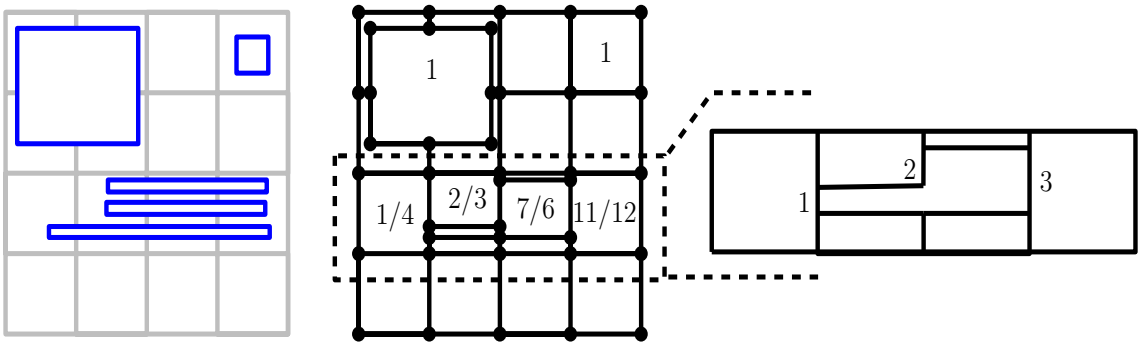


Figure 65 Assigning weights and costs in $G(\mathcal{L})$

▶ **Lemma 10.** *The total cost of edges in $G(\mathcal{L})$ is at most $4w(\mathcal{R})$. The weight of each non-rectangle face is at most $8\delta^2$. The total weight of the faces equals $w(\mathcal{R})$.*

Beweis. We prove those upper bounds separately

- Each rectangle is intersected by at most 4 Edges (Lemma 8)⇒ Total cost of edges is at most $4w(\mathcal{R})$
- Each face is intersected by rectangles of total weight at most $8\delta^2 w(\mathcal{R})$ (Definition 6) ⇒ Cost of each face is at most $8\delta^2 w(\mathcal{R})$
- Each rectangle R intersecting m faces contributes $w(R)/m$ to the weight of each of the m faces ⇒ Total cost of faces is $w(\mathcal{R})$

◀

8.4.5 Finding the cut in $G(\mathcal{L})$

In this subsection, we will give a constructive approach to find the wanted balanced cheap cut under the assumption that there is no rectangle $R \in \mathcal{R}$ with $w(R) > w(\mathcal{R})/3$. For this purpose we need to recall a definition and a previous result.

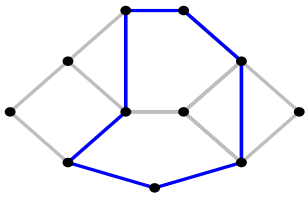


Figure 66 Example of V-cycle

▶ **Definition 11.** Let G be an embedded planar graph. We denote V-cycle each curve in G which might :

- go along edges of G

- cross faces of G

Edges crossing faces are called *face edges*. (see Figure 16)

▶ **Theorem 12.** *Let G be a planar embedded graph such that edges have costs and vertices and faces have weights. We call M the total cost of the graph and W its total weight. For any parameter k, we can find in polynomial time a separating V-cycle C such as:*

- *The interior and exterior of C have a weight of at most $2W/3$ each*
- *C uses at most k face edges*
- *C uses ordinary edges of total cost $O(M/k)$*

This theorem means that for any graph, there exists a low-cost cycle splitting the considered graph into to disconnected sets of similar weights. Those properties are analog to the wanted balanced cut. Therefore, to find the targeted balanced cut, we first apply the planar separator theorem to the graph $G(\mathcal{L})$ and set $k = 1/\delta$. The resulting $V - cycle$ C may contain face edges. The next step is to change each face edge $v_{\mathcal{F}}$ by edges from the boundary of the crossed face $\mathcal{F}$ such that the new set of edges forms a cycle $C/\{v_{\mathcal{F}}\}$. Under the assumption that for all rectangles: $w(R) \leq w(\mathcal{R})/3$ and that $\delta < 1/5$, lemma 10 insure that each face has a weight smaller than $8\delta^2 w(\mathcal{R}) < w(\mathcal{R})/3$. Changing face edges can satisfy that each component contains rectangles of total weight smaller than $2 \cdot w(\mathcal{R})/3$. Let C' denote such a cut. We can start from an arbitrary assignment of crossed faces to cut sides. If one side has a total weight bigger than $2 \cdot w(\mathcal{R})/3$, then start moving crossed faces to the other side by changing the edges from the boundary of the considered face by its remaining edges. At a given step k, we start from $\mathcal{R}_1^{k-1}$ and $\mathcal{R}_2^{k-1}$ such that

$$w(\mathcal{R}_1^{k-1}) \geq 2.w(\mathcal{R})/3$$

and reach the new partition $\mathcal{R}_1^k$ and$\mathcal{R}_2^k$ such that

$$w(\mathcal{R}_1^k) < 2.w(\mathcal{R})/3$$

by moving a face $\mathcal{F}$. Clearly,

$$w(\mathcal{R}_2^{k-1}) \leq w(\mathcal{R})/3 \, and \, w(\mathcal{R}_2^k) = w(\mathcal{R}_2^{k-1}) + w(\mathcal{F}).$$

As $w(\mathcal{F}) \leq w(\mathcal{R})/3$, we get the inequality $w(\mathcal{R}_2^k) \leq 2 \cdot w(\mathcal{R})/3$. With this construction, we are able to prove the next lemma:

▶ **Lemma 13.** *Assume that $0 < \delta < 1/5$ and $\forall R \in \mathcal{R}$, $w(R) < \sum_{R \in \mathcal{R}} w(R)/3$, there exists a balanced $O(\delta)$-cheap $(1/\delta)^4$-cut*

Beweis. We know that C' uses $\mathcal{O}((1/\delta)^4)$ edges because $|E| \in \mathcal{O}((1/\delta)^4)$ (lemma 7). We know also that the construction of C ensures that the total cost of ordinary edges is smaller than $\mathcal{O}(M/k)$ which is equal to $\mathcal{O}(\delta w(\mathcal{R}))$. We know that C crosses at most $k = 1/\delta$ faces and each faces crosses rectangles of total weight at most $\mathcal{O}(\delta^2 w(\mathcal{R}))$. That means that C' crosses overall rectangles of total weight in $\mathcal{O}(\delta w(\mathcal{R}))$. The construction of C' ensures that $w(C'_{in}) \leq 2.w(\mathcal{R})/3$ and $w(C'_{out}) \leq 2.w(\mathcal{R})/3$. ◀

The combination of lemma 1 and lemma 13 ensures that GEO-DP is a QPTAS when parametrized with $\delta = \Theta(\epsilon/(\log(n/\epsilon)))$ and $k = (\log n/\epsilon)^{\mathcal{O}(1)}$. We get then the following theorem

▶ **Theorem 14.** *The algorithm GEO-DP parametrized with $k = (\log n/\epsilon)^{\mathcal{O}(1)}$ yields a quasi-polynomial time approximation scheme for the maximum weight independent set of rectangles problem.*

8.5 Conclusion

In this paper, we presented the first $(1+\epsilon)$-algorithm the maximum weighted independent set of rectangles running in quasi-polynomial time as well as an overview on the proof of the approximation factor. The dynamic approach let us test each combination only once. The global parameter let us consider only a restricted number of possibilities and avoid the exponential running time.

References

1 P.K.Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Computational Geometry*, 209:218. 1998.

2 S. Arora, M. Grigni, D. Karger, P. Klein, and A. Woloszyn. A polynomial-time approximation scheme for weighted planar graph tsp. *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, 33:41. SIAM, 1998.

3 P. Chalermsook and J. Chuzhoy. Maximum independent set of rectangles. *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, 892:901. SIAM, 2009.

4 T. M. Chan and S. Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Proceedings of the 25th annual symposium on Computational geometry*, 333:340. SCG, 2009.

5 J. S. Doerschler and H. Freeman. A rule-based system for dense-map name placement. *Communications of the ACM*, 68:79, 1992.

6 T. Erlebach, K. Jansen, and E. Seidel. Polynomial-time approximation schemes for geometric graphs. *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, 671:679. SIAM, 2001.

7 R. J. Fowler, M. S. Paterson, and S. L. Tanimoto.. Optimal packing and covering in the plane are np-complete. *Information processing letters*, 133:137, 1981.

8 J. Fox and J. Pach. Computing the independence number of intersection graphs. *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, 1161:1165. SIAM, 2011.

9 T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Data mining with optimized two-dimensional association rules. *ACM Transactions on Database Systems (TODS)*, 179:213, 2001.

10 H. Imai and T. Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *Journal of algorithms*, 310:323, 1983.

11 S. Khanna, S. Muthukrishnan, and M. Paterson. On approximating rectangle tiling and packing. *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, 384:393. SIAM, 1998.

12 B. Lent, A. Swami, and J. Widom. Clustering association rules. *13th International Conference on Data Engineering Proceedings*, 220:231. IEEE, 1997.

13 L. Lewin-Eytan, J. Naor, and A. Orda. Routing and admission control in networks with advance reservations. *Approximation Algorithms for Combinatorial Optimization*, 215:228, 2002.

14 D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 103:128, 2007.

15 Anna Adamaszek and Andreas Wiese. Approximation Schemes for Maximum Weight Independent Set of Rectangles. *FOCS*, 400:409, 2013.

16 Korshunov, A.D. Coefficient of Internal Stability. *Kibernetika*, 17:28. 1974.

9 Segmentation of Trajectories on Non-Monotone Criteria

Lothar Fabian Weichert

Abstract

Trajectory Segmentation is the process of identifying intervals of a trajectory that are similar according to a certain criterion. The optimization problem at hand is to achieve this segmentation with as few intervals as possible. For monotone criteria the problem can be solved in $O(n \log n)$ time. The general problem, allowing non-monotone criteria, is *NP-hard*, as this paper shows. But under specific conditions the segmentation on these non-monotone criteria can still be solved in polynomial time. The here presented outlier-tolerant and standard deviation criteria allow segmentation in $O\left(n^2 \log n + kn^2\right)$ and $O\left(kn^2\right)$ time, respectively. To get these results, the segmentation problem is transformed into an equivalent geometric problem in the parameter space.

9.1 Introduction

Due to the large collections of trajectory data, collected by researchers in various fields, the automated segmentation of trajectories is a common problem. Unfortunately, the tractable monotone segmentation often yields inferior results. It strictly requires that any criterion, used to determine similarity within a segment, holds for the complete segment. Considering real-world applications and their noisy data, combined with the unavoidable measurement errors, good results can only be received by criteria that can handle noise and outliers. Such criteria are not monotone anymore and thus non-monotone segmentation is required. We show that non-monotone segmentation is in fact NP-hard. However, the geometric approach of the *start-stop diagram* makes the problem tractable under some conditions – without excluding real-world applications. We will present the conditions, together with the approaches to solve them. Followed by proofs for the run-times in these cases.

All results presented in this article (unless stated otherwise) are based on the work of Aronov et al. [1] on non-monotone trajectory segmentation.

9.2 Trajectory Segmentation

A *trajectory* is the path of an object through space, as a function of time. Examples could be the movement of birds or airplanes. Those trajectories are continuous functions. In general, they are collected with some kind of tracking method (e.g., GPS sensors). These tracking methods produce time-stamped, discrete points in space. Consequently, one usually considers the linear interpolation between those points as an approximation for the underlying continuous function. The measurements of the position are usually accompanied by other information, the *attributes*. These are either derived from the time and space data or taken by dedicated sensors. Possible attributes might be the speed, heading or altitude of the object and are described by an *attribute function*. This function is likely to be piecewise linear for the linear interpolation of the trajectory, but not necessarily, e.g., if the acceleration of the object is the attribute. The times of measurement define *breakpoints* of the trajectory and any associated attribute function. Any trajectory T can be defined as a function over an interval $I = [0, 1]$ to $\mathbb{R}^d$.

For the segmentation the notion of similarity is crucial. It is defined by a *criterion* which declares a possible segment *valid* or *invalid*, if the segment at hand is considered similar or not similar, respectively. The decision is based on the attribute function. The goal of a *trajectory segmentation* then is to split the considered trajectory (or more likely its linear interpolation) into as few pieces as possible, while guaranteeing that each segments describes a similar part of the trajectory. For example the criterion for segmenting a bird trajectory might enforce that the minimal and maximal speed of the bird within each segment only differs by 5mph. Such segments can then be utilized to further analyze the considered trajectory. Speed or altitude of a bird for example might reveal if it is hunting, sleeping or on a long distance flight during this part of the trajectory. Formally, a segment $T[a, b]$ is the trajectory function restricted to the subinterval $[a, b] \subseteq I$, and a criterion C is a function $C : I \times I \to \{\text{TRUE}, \text{FALSE}\}$, where $C(a, b) = \text{TRUE}$ if and only if the segment is valid.

The segmentation can be *discrete* or *continuous*. Discrete segmentation only allows to split the trajectories at breakpoints, while continuous allows segments to begin and end in-between points of measurement.

We define a segmentation as *minimal* if it splits up the complete trajectory into as few segments as possible, while still complying with the criterion. The *segmentation problem* is then to compute a minimal segmentation.

9.2.1 Non-Monotone Segmentation

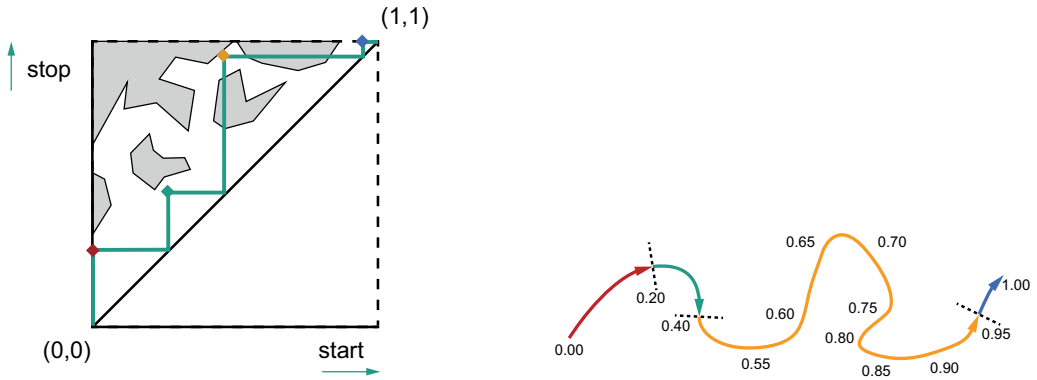
A criterion is monotone, if the sub-segments of any valid segment are valid. A simple example for monotone segmentation would be a criterion, that only allows the speed of a tracked animal to change by a maximal threshold h within each segment. This criterion is easily identified as monotone, as any speed-changes within the sub-segments of a (valid) segment will be smaller or equal to the segment's changes.

This paper is focused on non-monotone criteria. Staying in the example above, the animal's overall movement speed might be required to stay within threshold h, but insignificant, temporary spikes are ignored. E.g., quick stops for reorientation while it is running, in the case of a feline. That this criterion is not monotone, is due to the sub-segments, that mainly cover said temporary changes. These sub-segments alone do not comply if the outbalancing remainder of the segment is missing.

The monotone segmentation problem is solvable in $O(n \log n)$ time [2]. It is easier to solve then the NP-hard problems proposed here (Section 9.5).

9.3 The Start-Stop Diagram

It is possible to represent the full segmentation problem in a $[0, 1] \times [0, 1]$ diagram. The trajectory T is defined on the Interval $I = [0, 1]$. The x-coordinate of a point in the diagram then represents the beginning of a segment and its y-coordinate the end of said segment. In this way every possible segment $T(a, b)$ is represented as a point (a, b) in the diagram – more precisely in its top left half, since $a < b$. To account for the fact, that only some of those possible segments are valid, the concept of free and forbidden space is used: Defined by the criterion $C(a, b)$ the *free space* consists of all valid segments ($C(a, b) = \text{TRUE}$), while the rest are invalid, *forbidden* segments ($C(a, b) = \text{FALSE}$). An example can be found in Figure 67a. This figure already contains a *staircase*, which represents a valid segmentation of the trajectory (Figure 67b). The segmentation consists of four segments, represented by the four convex vertices of the staircase. Since each segmentation splits up the complete



(a) A start-stop diagram with a possible valid segmentation (green staircase).

(b) The trajectory segmented according to the segmentation on the left side.

Figure 67 A start-stop diagram and the represented trajectory.

trajectory, each segment begins exactly at the end of its predecessor, hence the staircase structure. The lines between the vertices are only for visualization – the concave vertices have to lie on the diagonal. The points on the diagonal represent segments that begin and end at the same point. A valid segmentation can be distinguished from an invalid segmentation by checking if all convex vertices lie in the free space (the visualizing lines are allowed to cross the forbidden space).

Thanks to the construction above, a minimal, valid staircase (as few steps as possible) represents a minimal segmentation and thereby solves the segmentation problem.

Using this geometrical representation of the segmentation problem, the search for the minimal segmentation can now be divided into two sub-problems. First, the construction of the start-stop diagram, including its forbidden and free spaces and second, the computation of the minimal staircase through the diagram.

In the following we give a brief presentation of discrete segmentation (Section 9.4) and then the NP-hardness-proof of the abstract segmentation problem (Section 9.5). The approach for finding the segmentation in the start-stop-diagram is covered in Section 9.6. In Section 9.7 the construction of the diagram follows.

9.4 Discrete Segmentation

The discrete case only allows to split the trajectory at specific breakpoints. As it is shown below, the discrete segmentation problem is comparably easy to solve using the start-stop diagram, thanks to the specific structure of the forbidden space. Depending on the application, this restriction however might yield considerable worse segmentations, as discussed in Section 9.4.2.

9.4.1 Solving the Discrete Segmentation

▶ **Theorem 1.** *Given an $n \times n$ start-stop matrix, one can check if a valid discrete segmentation exists, and if so compute a minimal one, in $O\left(n^2\right)$ time and space.*

If the segments can only begin and end at the n breakpoints of the trajectory, the free space is a set of discrete points. The start-stop diagram can thus be translated into a binary

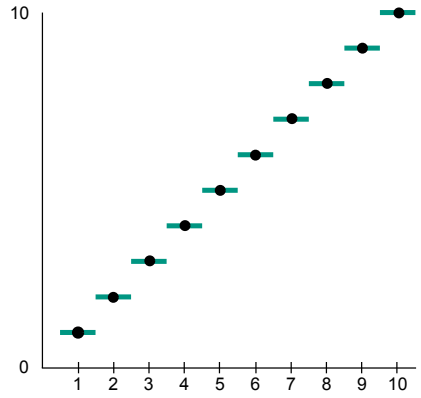


Figure 68 The two artificial attribute functions $g(i)$ (black dots) and $f(x)$ (green lines).

$n \times n$ *start-stop-matrix*. The criterion C is now calculated for each potential pair of break points and a valid segment is represented as a 1 in the matrix. If the indices are chosen accordingly ($(i, j) = (0, 0)$ in the bottom left corner), all potential segments are represented by the top-left half of the matrix.

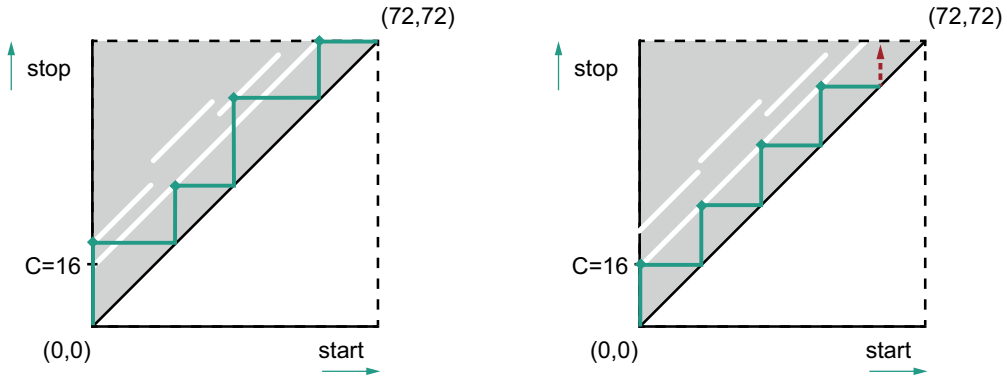
The constructed binary matrix can be interpreted as the adjacency matrix of a directed, unweighted, acyclic graph. Then the shortest path from node 1 to node n is in fact the equivalent of a staircase through the start-stop diagram and solves the discrete segmentation problem. The shortest path can be found using breadth-first-search in $O\left(n^2\right)$ time, leading to the theorem stated above.

9.4.2 Discrete vs. Continuous Segmentation

Considering the comparably easy solution for the discrete segmentation problem, the question arises, why a discrete approximation is not used to solve the continuous case. While this arguably might be a reasonable approach for some applications, e.g., where the time spans between measurements are relatively short, the following will show, that there exists cases where it is not.

In favor of simplicity, we will define the two artificial attribute functions $g(i)$ and $f(x)$. Discrete function $g(i)$ is defined as $g(i) = i$ for all $i \in \{1, \ldots, n\}$ and the very similar continuous function as $f(x) = \left\lfloor x + \frac{1}{2} \right\rfloor$ for $x \in \left[\frac{1}{2}, n + \frac{1}{2}\right]$. Therefore $f(x)$ is a step function, where each step encloses the respective discrete value of $g(i)$ ($\frac{1}{2}$ to the left and $\frac{1}{2}$ to the right). Suppose we segment these attribute functions in regard to standard deviation and a threshold of 0.4999, therefore the standard deviation over all values within each segment has to be below 0.4999. Since any pair of consecutive indices i already has a standard deviation of 0.5, no discrete segment may include more than one i. The continuous segmentation on the other hand will include the complete first step of $f(x)$ and nearly all of the second step in the first segment. The second segment will include the small remainder of the second step, all of the third and most of the fourth – just a small portion less than before. In consequence the very similar functions have been split up into nearly twice as much segments in the discrete case, compared to the continuous case.

A more practical argumentation is based on the underlying application. We will take the GPS-tracked bird as an example again. The bird's heading shall be tracked every five minutes. However the bird will change its heading anytime between those measurements, oblivious to the five minutes intervals. Continuous segmentation tries to model this fact, even



(a) A valid segmentation can be found, therefore $\langle\{3,7,5\},8\rangle \in$ SUBSET-SUM.

(b) No valid segmentation can be found, therefore $\langle\{6,7,5\},8\rangle \notin$ SUBSET-SUM.

Figure 69 Illustration of the reduction. The examples model the instances $\langle\{3,7,5\},8\rangle$ and $\langle\{6,7,5\},8\rangle$ of SUBSET-SUM.

when working with a linear interpolation of the actual trajectory. Discrete segmentation on the other hand enforces a non-existing periodicity on the segmentation.

9.5 Complexity of Trajectory Segmentation

In the last section we argued for the higher quality of continuous segmentation using non-monotone criteria. However, this quality in segmentation comes with a price in run-time of the algorithms. In this section we prove that this general form of segmentation, which allows monotone as well as non-monotone criteria, called the *abstract segmentation problem* is NP-hard.

The proof is a reduction from SUBSET-SUM, showing that even testing for the existence of a valid segmentation is NP-hard.

SUBSET-SUM is defined as follows: Given a collection of numbers $S = \{a_0, \ldots, a_{n-1}\}$ and a target number B, is there a subset $\{y_0, \ldots, y_k\} \subseteq S$, such that $\sum y_i = B$? SUBSET-SUM is NP-complete [3].

SUBSET-SUM will be represented in the start-stop diagram, by defining the free space in a way, that the minimal staircase will decide, which summands a_i will be included in the subset. It follows the reduction:

As visible in Figure 69a, there are several short line-segments and one long baseline-segment below them. These segments are placed so that each step ending in the baseline will cause a constant shift up the diagonal. This shift can be increased by a_i if the step lies in the respective short line-segment above the baseline. Choosing the constant shift carefully, one assures that each line-segment s_i can only host one step. We now define formally $A = a_0 + a_1 + \cdots + a_{n-1}$ and $C = A + 1$. The diagram is set up with side lengths of $((n+1)\,C + B, (n+1)\,C + B)$. The free space consists of $n+1$ line segments s_i. Each of them, except for s_n, represents one a_i. If a convex vertex of the staircase lies in the respective s_i, a_i is included in the subset. If the vertex lies on the parallel *base-line* s_n, it is not included. This line segment s_n shall have its endpoints at $(0, C)$ and $(nC + B, (n+1)\,C + B)$. The line segments s_i $(i \in \{0, n-1\})$ have their endpoints at $(iC, (i+1)\,C + a_i)$ and $(iC + A, (i+1)\,C + a_i + A)$. Using summand A one assures that each line segment is long enough to contain the convex

vertex, independently of the summands included or not included before, while the C enforces that each segment will include none or exactly one vertex. This construction assures that the $(n+1)$th vertex of a valid subset will lay exactly in $(nC+B, (n+1)\,C+B)$, so the a_i shifts from before add up exactly to B. Only in this case the staircase can be completed in $((n+1)\,C+B, (n+1)\,C+B)$.

Consequently if and only if we have a valid subset, the staircase can be valid. Hence, we presented a clearly polynomial reduction from SUBSET-SUM and proved the following theorem.

▶ **Theorem 2.** *The abstract segmentation problem is NP-hard.*

9.6 Compute the Minimal Staircase

This section will present the second part of the geometrical approach to solve the segmentation problem, the computation of a minimal staircase through the start-stop diagram. For this purpose we will present the general ReachableSpace algorithm (Algorithm 8) and then discuss two conditions for the forbidden space, that make the NP-hard problem tractable, as proved in the last two subsections.

9.6.1 The Reachable Space Algorithm

The algorithm ReachableSpace generates a set of *reachable spaces* R_i. These are defined as the set of intervals on the diagonal, that can be reached from the origin $(0,0)$ of the start-stop diagram using at most i steps of a valid staircase. This algorithm consequently considers intervals of possible start- and end-points for segments instead of concrete points (and so will the following sections). After ReachableSpace calculated the minimal k, where R_k contains the end $(1,1)$ of the start-stop diagram, the actual minimal staircase can easily be found by going backwards through the R_i starting with R_k and $(1,1)$. In the set R_i one identifies a point, that can also be reached from R_{i-1}, this point now defines the searched segment s_i of the solution. This iteration is repeated until $(0,0)$ is reached.

The algorithm presented here will always find a valid staircase through the diagram, if one exists. However, it might have exponential run-time, considering the results from above (Section 9.5).

The ReachableSpace algorithm calculates the R_0 to R_k iteratively. It uses the intervals of R_i as input for the computation of R_{i+1}. We differentiate between incoming and outgoing intervals. *Incoming intervals* are those that define the possible start-points of a segment, while *outgoing intervals* are the projection on the diagonal (to the right) of the possible end-points of such a segment. The *vertical slab* induced by an interval X is the intersection of all vertical lines through the diagram and this interval, graphically it's the column above the interval. The *horizontal slab* is defined analogously for horizontal lines. The formal, but high level description of ReachableSpace is given in Algorithm 8.

To compute R_{i+1} ReachableSpace iterates over all incoming intervals of R_i. These represent all points on the diagonal, that can be reached with an i step-staircase. For each incoming interval it checks, how far a new segment, starting in this interval, could reach on the trajectory, while still complying with the criterion (staying in the free space). These define the outgoing intervals. For one incoming interval, there can be several outgoing intervals. The union over all outgoing intervals, R_{i+1}, then defines all the points on the diagonal, that can be reached with an $i+1$ step-staircase. As one can see in Algorithm 8, each R_i contains

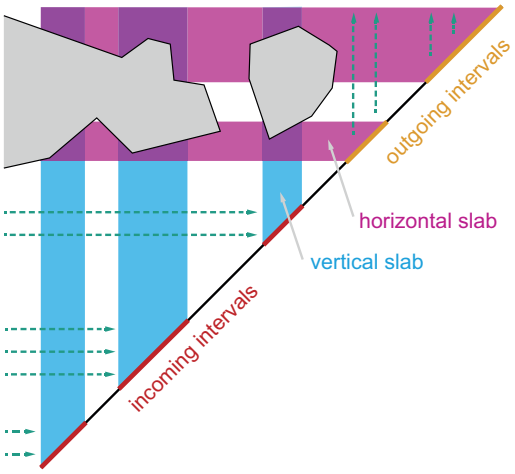


Figure 70 Detail of a start stop diagram with incoming and outgoing intervals and their respective slabs.

Input : A start-stop diagram S.
Output : The sets of reachable space $R_0, \ldots, R_k$, where R_k is the first R_i with an interval including $(1, 1)$

$i \leftarrow 0$; $R_0 \leftarrow \{(0, 0)\}$;
while $(1, 1) \notin R_i$ **do**
  foreach *interval* X *in* R_i **do**
    Consider the intersection of the free space in S with the vertical slab induced by X and project it horizontally back to the diagonal;
  The union of R_i and these projections, over all X, forms R_{i+1};
  $i \leftarrow i + 1$;
return $R_0, \ldots, R_i$

Algorithm 8 : ReachableSpace (S)

all R_j where $j \leq i$. This is sensible considering the definition of R_i (at most i steps) and that there might exist staircases of different forms, that all reach the same point on the diagonal. Possibly with additional steps, making them non-minimal.

The potentially exponential run-time of the algorithm originates here: Each incoming interval has to be checked and might generate several outgoing intervals. These have to be checked in successive iterations, again generating several new intervals each. Hence if each interval generates generates two disjunct outgoing interval, the number of intervals is doubles with every step of the iteration leading to the exponential run-time.

The following sections will discuss cases where the number of generated outgoing intervals is limited and by that the run-time is polynomial.

9.6.2 The Grid Decomposition and Special Cells

To define the conditions where the segmentation problem is tractable, we decompose the start-stop diagram into a $n \times n$-grid (Abbildung 71). The n breakpoints of the trajectory serve as natural cuts for the decomposition. However, any decomposition of the diagram into n^2 or less cells (using fewer breakpoints) works. It is important to note that this grid and its

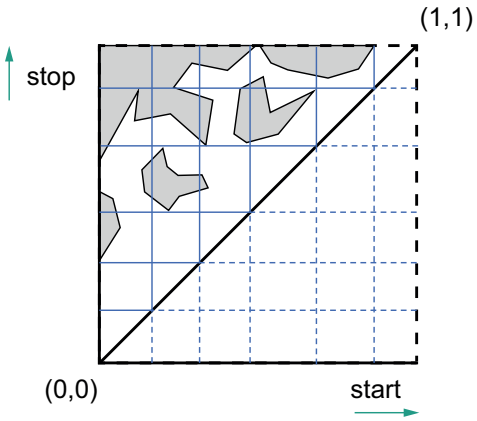
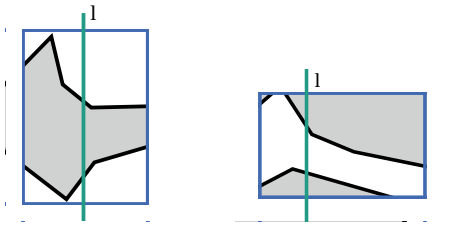


Figure 71 A start-stop diagram decomposed according to the breakpoints.



(a) A verticonvex cell with vertical line l marked.

(b) A tunnel cell with vertical line l marked.

Figure 72 The two special cells required to guarantee tractability of the segmentation.

breakpoints does not in any way affect the actual segmentation. The decomposition is only required to classify the segmentation problems.

Some of the cells produced by the grid can be categorized as verticonvex cells and tunnel cells. In a *verticonvex cell* (Figure 72a) any vertical line l through the cell will intersect the forbidden space at most once. In a *tunnel cell* (Figure 72b) such a vertical line l will cross the forbidden space at most twice.

In the following section we show that if the whole diagram consists of verticonvex cells, the minimal segmentation can be found in $O\left(kn^2\right)$ time, where k is the number of segments (steps of the staircase).

If the diagram contains tunnel cells, the problem becomes harder to solve. As long as only one tunnel cell exists in every row of the grid, a segmentation can be found in $O\left(k^2n^2\right)$ time. Aronov et al. ([1]) prove that one tunnel cell per row is the maximum this approach can handle efficiently. As soon as a second tunnel cell exists, the reachable space R_i might consist of $\Omega\left(2^i\right)$ intervals, which would cause exponential run-time. However, the outlier-tolerant and standard deviation criteria introduced in Section 9.7 show, that these conditions do apply to real world applications (both criteria in fact produce verticonvex cells).

9.6.3 Only Verticonvex Cells

In the following, we assume that each cell in our start-stop diagram is verticonvex. Using this assumption, we prove that the algorithm ReachableSpace produces only a linear number of outgoing intervals in each iteration. This limits the number of intervals that have to be

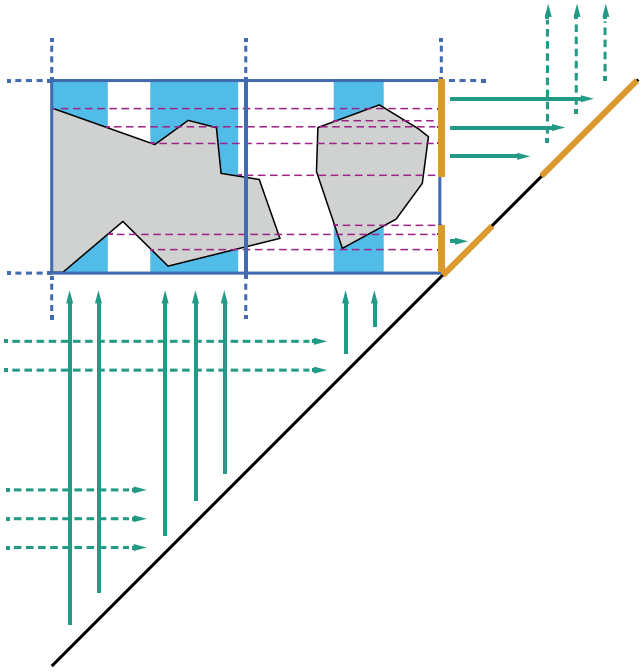


Figure 73 A detail of the start-stop diagram with two verticonvex cells. The incoming intervals and the outgoing ones generated by them are marked.

checked and by that the runtime of the algorithm is limited by $O\left(kn^2\right)$.

Before we give a more specific version of the ReachableSpace algorithm that will also serve as the proof, the following observation should be noted. Any vertical line through a verticonvex cell will intersect the free space in this cell at most twice. This generates two (possibly empty) intervals, one connected to the top of the cell, the other one connected to the bottom of the cell. This stays true for the horizontal projection of these intervals onto the diagonal. Further, the union with intervals from additional vertical lines might increase the size of the intervals, but never introduce a third disjunct interval. The observation is visualized in Abbildung 73. It can be explained with the shape of the forbidden space: Being verticonvex it can intersect any vertical line exactly once, leaving intervals above and below the intersection. Since this holds for any vertical line, it is not possible to produce a third interval. Consequently any row in the start-stop diagram will produce at most two outgoing intervals and thereby limit the number of intervals in R_i by $O\left(n\right)$.

We can now prove, that the minimal segmentation can be found in $O\left(kn^2\right)$ time. From the observation above follows, that a given R_i is a set of $O\left(n\right)$ intervals. R_{i+1} is calculated on this basis in a row by row manner. Starting with the lowest row in the start-stop diagram grid, the cells are swept from left to right, iterating over each incoming interval. We assume that the forbidden space has constant complexity in every cell, accordingly the size of the two outgoing intervals produced by an incoming interval can be computed in constant time. Using the observation from above, one knows, that the outgoing intervals can be fully described by their highest, respective lowest, point on the diagonal (they will always begin at the top, respective bottom of the cell). When the next interval is checked, its outgoing intervals are again computed in constant time. If the union with the intervals before is larger, the two bounds will be updated. The observation grants, that every cell will have at most two incoming intervals (following from the two outgoing ones in the respective row below, that is

projected upwards from the diagonal). Therefore the number of intervals checked in each row is limited by $O(n)$. We repeat the procedure for each of the n rows, hence one R_{i+1} is calculated in $O(n^2)$ time, which has to be repeated k-times until R_k includes $(1,1)$. We obtain the following theorem.

▶ **Theorem 3.** *Given an $n \times n$ start-stop diagram, in which the forbidden space in each cell is verticonvex and has constant complexity, a minimal segmentation can be computed in $O(kn^2)$ time, where k is the number of segments of the minimal segmentation.*

As noted above, the actual minimal staircase and thereby segmentation has to be extracted from the set of intervals R_k.

9.6.4 One Tunnel Cell per Row

In this section one tunnel cell shall be allowed in each row of the start-stop diagram grid. This overturns the fundamental observation from the proof above, as a single interval might now produce several disjoint outgoing intervals. How these additional intervals are generated will be shown first, followed by a proof, that the total number of intervals will still remain polynomial.

Challenges Caused by Tunnel Cells

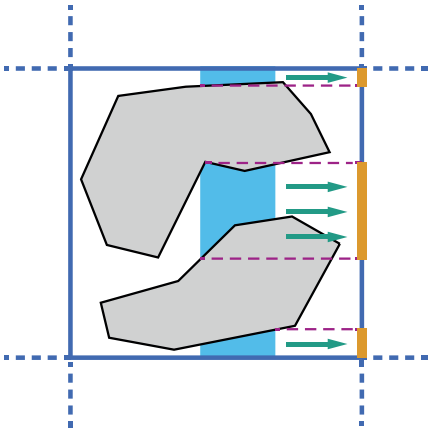
A tunnel cell is defined by allowing a vertical line l to have two intersections with the forbidden space. Thus, three intersections with the free space are allowed. If this vertical line l is the element of an incoming interval, the horizontal projection of these three intersections consequently yields three disjoint outgoing intervals. Two of them still connected to top and bottom of the cell, but the third one free-floating in between them (Figure 74a). With *free-floating* we therefore denote an interval that is neither connected to the bottom- nor the top-interval of the cell. Consequently another incoming interval in the same cell (all other cells in the row are verticonvex) might produce a fourth interval and so forth. While it will again only increase the size of the intervals at the bottom and top, this second incoming interval might generate an additional free-floating interval disjoint from all of the existing intervals (Figure 74b). Note that one tunnel cell can generate four outgoing intervals from two incoming ones – the number of outgoing intervals generated by a usual verticonvex cell. A special case arises, as soon as the free space in the tunnel cell is disconnected, as illustrated in Figure 74c. These cells do comply with the definition of a tunnel cell, since any vertical line l only intersects their forbidden space twice – which is true as long as the disjoint spaces are not located above each other. Assuming incoming intervals of appropriate size, these cells can even produce several free-floating outgoing intervals from just one incoming interval.

Since we allowed one tunnel cell per row the described effects might multiply through the diagram. The following section discusses why the number of interval still remains polynomial.

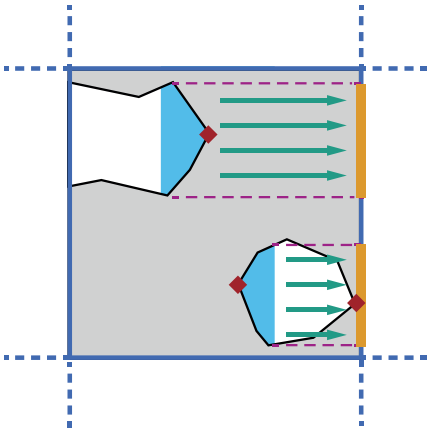
The Number of Additional Intervals is Bounded

Although the effects might multiply, the number of additional intervals produced by the tunnel cells is linear with the number of segments in the minimal segmentation k. The number of intervals in R_k then lies in $O(kn)$ and the run-time of Algorithm 8 is polynomial. To prove this, we will discuss how to handle each of the three cases presented above.

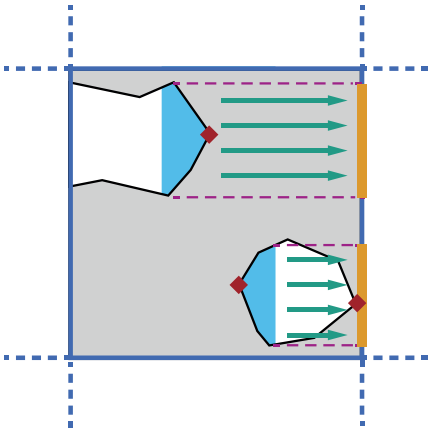
As in the proof above (Section 9.6.3) the forbidden space $F \subseteq C$ in each cell has constant complexity. This can be formally expressed as follows: The forbidden space in each cell is described by a constant number of vertices. These vertices are connected by x-monotone



(a) A third free-floating outgoing interval is induced.



(b) Additional incoming intervals may generate additional *free-floating* outgoing intervals.



(c) If the free space is disconnected, one incoming interval might even generate several outgoing intervals.

Figure 74 The three special cases observed, if one tunnel cell per row is allowed.

polynomial curve segments of bounded degree (monotone to the horizontal axis of the diagram).

We will now prove a set of Lemmas. These lead to the result, that the run-time of the ReachableSpace algorithm allowing one tunnel cell per row lies in $O\left(k^2n^2\right)$.

We first exclude the third case, that is the case where one interval reaches across several disjoint free spaces (it is discussed separately below). Formally, it is assumed that, if one draws a horizontal line through any vertex of F, it will not intersect the free-floating outgoing interval. For these cases we will show, that any incoming interval will produce at most three outgoing intervals.

We prove by contradiction: Assume that the incoming interval X produces two disjoint, free-floating outgoing intervals I and J (without loss of generality I shall lie above J). These intervals are the result of the horizontal projection of the intersection of X's vertical slab with the free space $C\backslash F$. Since the intervals I and J are disjoint, these intersections have to be disjoint. We excluded the special case, where the vertical slab includes a vertex of F.

Consequently each intersection's lower and upper bounds will reach across the complete width of the incoming interval (the same holds for the considered free space being a single curve-segment). The vertical slab therefore consists of at least five layers:

1. The forbidden space below the lower bound of the intersection producing I
2. the intersection itself
3. the forbidden space above its upper bound, which is again bounded by the lower bound of the intersection producing J
4. this second intersection
5. the forbidden space above its upper bound

Since J and I are free-floating, the lower bound of the intersection for I or the upper bound of the intersection for J cannot be the cell boundaries. Otherwise they would describe the top or bottom interval. The five layers above however include three forbidden spaces, accordingly a vertical line l would have three intersections with the forbidden space and the considered cell cannot be a tunnel cell. We just proved the following lemma:

▶ **Lemma 4.** *In a tunnel cell C, with forbidden space $F \subseteq C$, an incoming interval X can produce at most one outgoing interval I, such that (i) I is neither incident to the top nor the bottom of the row (free-floating); and (ii) I does not intersect a horizontal line through a vertex of F.*

The next step is to handle the special case excluded above. We will prove, that if vertices are allowed in the vertical slabs, the number of outgoing intervals out of the considered tunnel cell is limited by $u \leq c + m + 2$, where c is the number of vertices in the considered forbidden space and m the number of incoming intervals.

The proof uses the constant complexity of the forbidden space: We draw horizontal lines through each vertex of F, each of these lines can only intersect one disjoint outgoing interval (otherwise it would not be disjoint). Consequently we can use the constant number of vertices c to account for the additional outgoing intervals. All other outgoing intervals but the top and bottom interval do not intersect a vertex of F and Lemma 4 guarantees, that they will at most produce one additional free-floating interval each. As soon as one accounts for the top and bottom interval, the following lemma is proven.

▶ **Lemma 5.** *In a tunnel cell C, with forbidden space $F \subseteq C$, m is the number of incoming intervals and u the size of the union of all outgoing intervals produced by them. Then $u \leq c + m + 2$, where c is the number of vertices of F.*

Lemma 5 shows that the number of outgoing intervals is limited by two constant summands (c and 2) and the incoming intervals m. It remains to prove, that the summand m stays tractable over the run-time of the ReachableSpace algorithm.

We prove using a recurrence: In most cases m will be two, since the verticonvex cells can only produce two outgoing intervals, which are reflected into the now considered cell as incoming intervals (compare Section 9.6.3). The exception are the incoming intervals produced by tunnel cells, these their-self however are limited by the inequality above. This leads to the recurrence $u_k \leq c + u_{k-1} + 2$, where k is the number of steps in the staircase, that produces the outgoing intervals considered here. Hence each staircase-step might increases the number of incoming intervals. But since $u_1 \leq 3 + c$ (the first sub-segment always begins at 0, so one incoming interval only), this increasing summand is constant and $u_k \in O(k)$. Recurrence factor u_k describes the outgoing intervals per row, the worst case evaluation time for n rows is consequently in $O(kn)$, which results in the following lemma for the reachable space R_k.

▶ **Lemma 6.** *In an $n \times n$ start-stop diagram in which (i) the forbidden space in each cell has constant complexity, (ii) each row contains at most one tunnel cell, and (iii) the remaining cells are verticonvex, R_k consists of at most $O(kn)$ intervals.*

The ReachableSpace algorithm initially presented (Section 9.6.1) – according to this lemma – has to scan $O(k)$ incoming intervals per iteration, which adds a linear factor k into its runtime compared to the diagram with verticonvex cells only. We conclude:

▶ **Theorem 7.** *Given an $n \times n$ start-stop diagram, in which (i) the forbidden space in each cell has constant complexity, (ii) each row contains at most one tunnel cell, and (iii) the remaining cells are all verticonvex, a minimal segmentation can be computed in $O\left(k^2n^2\right)$ time, where k is the number of segments of the minimal segmentation.*

9.7 Computing the Start-Stop Diagram

When solving the segmentation problem, the first sub-problem to solve is the computation of the start-stop diagram. Particularly, the definition of the free and forbidden space in the diagram. As the forbidden space is based on the criterion, the computation of the diagram is specific for each criterion. This can make the segmentation easier or harder to solve. In this section two specific example criteria and the generation of their diagrams will be presented briefly.

Both criteria, outlier-tolerant and standard deviation criterion, produce verticonvex start-stop diagrams. Any algorithm for traversing the diagram is independent from the specific criterion, as long as the structure of the diagram complies with the discussed conditions. Therefore the algorithm for verticonvex diagrams from above can be used to solve the segmentation, yielding a total runtime of $O\left(n^2 \log n + kn^2\right)$ and $O\left(kn^2\right)$ for the calculation of the minimal segmentation.

9.7.1 Outlier-Tolerant Criterion

The outlier-tolerant criterion allows that some part of the considered segment does not comply with the criterion, to accommodate the eponymous outliers. Specifically, the criterion requires the minimum and maximum value within each segment $[a, b]$ to differ by at most h for ρ of the values, where $0 \leq \rho \leq 1$. The fraction $1 - \rho$ of values may have any value (the outliers). Obviously this criterion is non-monotone, considering the $1 - \rho$ fraction of the segment.

Aronov give an explicit definition for the forbidden space in each cell for given values of a,b,h, and ρ. Each cell can be computed in $O(\log n)$ time, or the whole $n \times n$ diagram in $O\left(n^2 \log n\right)$ time. This leads to the following theorem for solving the segmentation problem.

▶ **Theorem 8.** *A piecewise-constant attribute function f with n breakpoints, a threshold value $h > 0$, and a ratio $\rho \in [0, 1]$ shall be given. A minimal segmentation can be computed for the condition, that on a fraction of a least ρ of a segment, the difference between the maximum and minimum function value is at most h. This can be achieved in $O\left(n^2 \log n + kn^2\right)$ time, where k is the size of a minimal segmentation.*

9.7.2 Standard Deviation Criterion

The standard deviation criterion, as the name implies, is based on the standard deviation of the attribute function values. It requires the standard deviation within the considered

segment of the piecewise-constant attribute function not to exceed a given threshold. Again an obviously non-monotone criterion, as sub-segments with high variation have to be outbalanced by those with low variation to comply.

For this criterion the forbidden space is limited by piecewise cubic curves in respect to the individual cell, which can be computed in $O(1)$ time. Thus the complete start-stop diagram can be build in $O(n^2)$ time. Combined with the knowledge on finding segmentations in verticonvex diagrams (Section 9.6.3) follows Theorem 9.

▶ **Theorem 9.** *Given a piecewise-constant function f with n breakpoints and a threshold value $h > 0$, we can compute a minimal segmentation for the criterion, that the standard deviation of a segment is at most h, in $O(kn^2)$ time, where k is the size of a minimal segmentation.*

9.8 Conclusion

The presented approach has some limitations. Primarily in regard to the computation of the minimal staircase: As soon as the start-stop diagram contains more than one tunnel cell per row, or even more complex cell structures, the ReachableSpace algorithm might have exponential runtime. The two given criteria have a pure verticonvex structure for piecewise-constant attribute functions. To be piecewise-constant is a reasonable assumption for many interpolated attributes, e.g., speed or height. However, as soon as the attribute function is piecewise-linear, the attribute functions easily produce start-stop diagrams with several tunnel cells per row (at least for the here presented non-monotone criteria) leading to potentially exponential run-times.

As has been proven by Aronov et al. [1], the approach works right below tight limits. Consequently, it is expected that any problem not covered by the stated conditions will call for a completely new approach.

Notwithstanding the stated limitations, solving the segmentation by this geometric approach has not yet been exhaustively researched: Other non-monotone criteria might be of interest (Aronov et al. are looking for suggestions) and each of them will require specific procedures to efficiently set-up the start-stop diagram. Further, it might be valuable to investigate alternative procedures to split up the attribute function and the diagram, as none of the proofs required them to be split at the points of measurement. This would potentially allow a faster segmentation (e.g., by reducing the number of cells). Finally, the segmentation problem is not limited to trajectories, but is generally applicable to any piecewise-constant function and thereby might be of interest for other areas requiring automated analysis.

Concluding, this paper presented an approach to trajectory segmentation on non-monotone criteria, based on a geometric representation, that allowed to elegantly prove the NP-hardness of the abstract segmentation problem. In spite of the complexity of the problem, the authors were able to define procedures to efficiently find segmentations in polynomial time, under reasonable conditions. While tight, these conditions cover real-world applications and hence the approach is of theoretical as well as practical interest.

References

1 Boris Aronov, Anne Driemel, Marc van Kreveld, Maarten Löffler, and Frank Staals. Segmentation of trajectories for non-monotone criteria. In *SODA*, pages 1897–1911, 2012.

2 Maike Buchin, Anne Driemel, Marc van Kreveld, and Vera Sacristan. Segmenting trajectories: A framework and algorithms using spatiotemporal criteria. *Journal of Spatial Information Science*, 0(3):33–63, 2011.

3 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, January 1972.