Technische Universität Graz
Institut für Softwaretechnologie
Bachelorstudium Informatik
LV-Nummer: 716.404

# Counting Convex 5-Holes

Bachelorarbeit Informatik
von

**Manfred Scheucher**

Matrikelnummer: 0930956
Studienkennzahl: 033 521

Betreuer: Assoc.Prof. Dipl.-Ing. Dr.techn. Oswin Aichholzer
Dipl.-Ing. Dr.techn. Thomas Hackl
Institut für Softwaretechnologie
Technische Universität Graz

August 2013

# Inhaltsverzeichnis

# 1 Übersicht

In meiner Bachelorarbeit wird das Zählen leerer konvexer 5-Ecke ("convex 5-holes", "c5h") in Punktmengen in der Ebene behandelt, mit der Beschränkung auf Punkte in allgemeiner Lage. Für beliebige Punktmengen mit $n$ Punkten ist die Mindestanzahl $h_5(n)$ an leeren konvexen 5-Ecken gesucht. Durch diese Mindestanzahl ist zugleich auch eine scharfe untere Schranke an leeren konvexen 5-Ecken für $n$ Punkte gegeben.

Zu Beginn meiner Arbeit waren die exakten Werte $h_5(n)$ bis $n = 12$ Punkte bereits bekannt [13, 12, 11, 9]. Mittels meiner Implementation, der Erweiterung vom bestehenden Tool `Merged Cape` [16, 15], konnten die exakten Werte $h_5(13) = 3$, $h_5(14) = 6$ und $h_5(15) = 9$ bestimmt werden.

Zudem konnte für $n = 16$ gezeigt werden, dass es immer $\geq 10$ convex 5-holes geben muss. Da einige $n = 16$ Punktmengen gefunden werden konnten, die 11 convex 5-holes aufweisen, ist die Frage nach dem exakten Wert von $h_5(16)$ zwar noch offen, jedoch konnte dieser auf 2 mögliche Werte eingeschränkt werden: $h_5(16) \in \{10, 11\}$.

Ebenfalls konnten die bisherigen oberen Schranken für $h_5(n)$ für $n > 16$ verbessert werden und besonders gute Verbesserungen ($\geq 10\%$) waren für $n > 20$ möglich. Da die aktuell besten oberen Schranken für $h_5(n)$ für $n > 30$ vermutlich noch weit vom tatsächlichen Wert von $h_5(n)$ entfernt sind, wurde die Suche nur bis $n = 36$ Punkte durchgeführt.

In den folgenden Kapiteln werden die Tools, mit denen ich die Ergebnisse erzielen und auswerten konnte, und deren Funktionalität beschrieben. Des weiteren werden auch einige Punkte aufgelistet, wie man diese Tools noch verbessern und erweitern könnte.

# 2 Einleitung

## 2.1 Order Types

Da es unendlich viele Möglichkeiten gibt, $n$ Punkte in der Ebene zu platzieren, werden *Order Types* [13, 8] als Repräsentanten von Punktmengen verwendet. Order Types ermöglichen es, nur mit der Orientierung, der Lage von je 3 Punkten zueinander, zu arbeiten. Die genaue Positionierung der Punkte ist nicht relevant. Da nur Punktmengen in allgemeiner Lage betrachtet werden, kann der dritte Punkt nur links oder rechts von der gerichteten Geraden durch den ersten und den zweiten Punkt liegen, nicht aber auf der Geraden selbst. Entscheidend ist hierbei die Reihenfolge der Punkte, da sich beim Vertauschen der Ordnung zweier Punkte die Orientierung umkehrt. Die Datenstruktur, welche alle Orientierungs-Tripel zu je 3 Punkten beinhaltet, wird als *Groß-Lambda Matrix* $\Lambda$ bezeichnet.

Eine weitere Datenstruktur ist die *Klein-Lambda Matrix* $\lambda$. Ihre Einträge $(\lambda_{ij})$ geben zu je 2 Punkten $p_i$ und $p_j$ der Punktmenge an, wieviele Punkte links von der gerichteten Geraden durch $p_i$ und $p_j$ liegen. Auch hier ist die Reihenfolge der Punkte entscheidend. Da keine 3 Punkte auf einer Geraden liegen dürfen und jeder der $n-2$ anderen Punkte entweder links oder rechts von der gerichteten Geraden liegt, muss $\lambda_{ij} + \lambda_{ji} = n - 2$ gelten.

Eine Klein-Lambda Matrix und die dazugehörige Groß-Lambda Matrix beinhalten dieselben Informationen und können in $\Theta(n^3)$ Zeit ineinander umgerechnet werden. Ein Order Type kann somit auch durch eine Klein-Lambda Matrix repräsentiert werden, welche im Vergleich zur Groß-Lambda Matrix mit $\Theta(n^3)$ Einträgen nur $\Theta(n^2)$ Einträge besitzt. Im Folgenden wird die Klein-Lambda Matrix $\lambda$ einfach *Lambda Matrix* genannt.

Eine Lambda Matrix, deren Punkte um den ersten Punkt (welcher auf der konvexen Hülle liegt) geordnet sind, wird als *natürlich* bezeichnet und die Nummerierung der Punkte wird als *natürliches Labeling* bezeichnet. Da sich durch Permutation der $n$ gegebenen Punkte die Lambda Matrix verändern kann, wird die *lexikographisch kleinste Lambda Matrix* $\lambda_{min}$ berechnet, um Gleichheit oder Unterschiedlichkeit zweier Punktmengen festzustellen. Die Bestimmung der lexikographisch kleinsten Lambda Matrix zu einer gegebenen Punktmenge ist in $O(n^3)$ Zeit möglich, da diese immer natürlich sein muss und es höchstens $2n$ natürliche Labelings bzw. Klein-Lambda Matrizen mit natürlichen Labelings gibt.

Zu einer gegebenen Punktmenge mit $n$ Punkten kann in $\Theta(n^3)$ Zeit der Order Type bestimmt werden. Das Problem, zu einem gegebenen Order Type eine Punktmenge zu finden oder zu beweisen, dass keine solche existieren kann, wird auch *Realisierungs-Problem* genannt und ist NP-schwer. Ist die Realisierbarkeit eines Order Types nicht bekannt, so spricht man von einem *abstrakten Order Type*. Aufgrund der NP-Schwere sind in den weiteren Kapiteln die Ergebnisse der abstrakten Order Types zusätzlich zu den Ergebnissen der realisierbaren Order Types aufgelistet.

## 2.2 Abstrakte Erweiterung

Um alle Order Types mit $n+1$ Punkte aufzuzählen, können die Order Types mit $n$ Punkten bzw. deren Lambda Matrizen um einen Punkt erweitert werden. Da in jedem Order Type zumindest ein Punkt auf der konvexen Hülle liegt und das Entfernen eines Punktes aus der konvexen Hülle der Umkehrung des Einfügens eines Punktes *im Äußeren* (außerhalb der konvexen Hülle) entspricht, können durch das *Erweitern im Äußeren* sämtliche Order Types mit $n+1$ Punkten aus den Order Types mit $n$ Punkten aufgezählt werden. Eine vollständige Erweiterung ist somit durch das Erweitern nur im Äußeren möglich.

Jeder Order Type mit $n+1$ Punkten bis zu $n+1$ (nicht notwendigerweise unterschiedliche) *Vorgänger-Order Types* mit $n$ Punkten und kann somit beim Erweitern mehrmals aufgezählt werden. Da der *letzte Punkt* (der Punkt mit dem größten Index) in einer natürlichen Lambda Matrix immer auf der konvexen Hülle liegt, kann ohne Beschränkung der Allgemeinheit angenommen werden, dass der zuletzt (auf der konvexen Hülle) eingefügte Punkt der letzte Punkt ist.

Durch die Erweiterung im Äußeren werden u.a. sämtliche natürliche Lambda Matrizen generiert, und da jeder Order Type durch genau eine lexikographisch kleinste Lambda Matrix repräsentiert wird, müssen nur jene Lambda Matrizen nach der Erweiterung aufgezählt bzw. ausgegeben werden, welche lexikographisch minimal sind. Da die lexikographisch kleinste Lambda Matrix eines Order Types natürlich ist, müssen beim Erweitern nur noch natürliche Lambda Matrizen generiert werden. Somit kann auf diese Weise eine vollständige Erweiterung ohne Duplikate durchgeführt werden.

Allerdings können durch das Erweitern von Order Types nur abstrakte Order Types aufgezählt werden, da nach dem Erweitern aufgrund der NP-Schwere im Allgemeinen keine Aussage über Realisierbarkeit möglich ist. Man spricht hier auch von *abstrakter Erweiterung* [13].

## 2.3 Das Zählen leerer konvexer 5-Ecke

Wird ein Order Type mit $n$ Punkten um einen Punkt außerhalb der konvexen Hülle erweitert, so wird keines der *alten convex 5-holes* (convex 5-holes des Vorgängers) zerstört, da der neue Punkt im Inneren eines convex 5-holes platziert werden müsste, um dieses zu zerstören. Somit gibt es nach dem Erweitern um einen Punkt im Äußeren immer gleich viele oder mehr convex 5-holes als im Vorgänger. Werden beispielsweise $n+1$ Order Types mit $\leq C$ convex 5-holes gesucht, so müssen beim Erweitern von $n$ auf $n+1$ Punkte nur noch jene (abstrakten) Order Types betrachtet werden, welche $\leq C$ convex 5-holes aufweisen.

Wird eine Erweiterung von $n$ Punkte Order Types auf $n+k$ Punkte durchgeführt, so kann schrittweise erweitert werden ($n \to n+1 \to n+2 \to \ldots \to n+k$). Liegen nach einem Schritt zuviele convex 5-holes in einem (abstrakten) Order Type vor, so kann die Erweiterung dieses (abstrakten) Order Types bereits an dieser Stelle abgebrochen werden, da die Anzahl in den weiteren Schritten nicht mehr verringert werden kann.

Desweiteren können neue convex 5-holes nur am zuletzt eingefügten Punkt

entstehen, da ein neues convex 5-hole andernfalls bereits im Vorgänger existieren würde (was ein Widerspruch wäre). Da der letzte Punkt also für die Berechnungen fixiert werden kann, lässt sich die Anzahl der notwendigen Rechenoperationen zum Zählen neuer convex 5-holes durch $O(n^4)$ beschränken.

Die Punkte $a, b, c, d, e$ bilden genau dann ein *konvexes 5-Eck ("convex 5-gon")*, wenn $c, d, e$ auf der linken Seite der gerichteten Geraden $a - b$ liegen und analog für $b - c$, $c - d$, $d - e$ und $e - a$. Gibt es einen Punkt $p$, sodass dieser bezüglich der fünf gerichteten Geraden $a - b$, $b - c$, $c - d$, $d - e$ und $e - a$ links liegt, so liegt der Punkt $p$ im Inneren des konvexen 5-Ecks. Andernfalls handelt es sich um ein *leeres konvexes 5-Eck ("convex 5-hole")*. Die Orientierung von je 3 Punkten kann direkt aus der Groß-Lambda Matrix ausgelesen werden und deshalb erlaubt es diese Definition, convex 5-gons und convex 5-holes auch in abstrakten Order Types zu zählen.

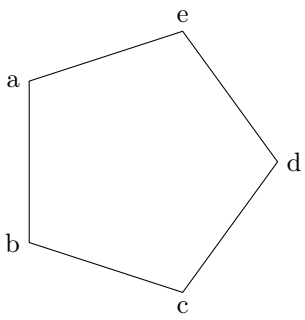


Abbildung 1: Ein (leeres) konvexes 5-Eck

# 3 Verlauf der Arbeit und Beschreibung der Tools

**Sep/Oct 2012:** Als erstes wurde das Tool `check5` erstellt, welches das Zählen von convex 5-holes in einer gegebenen Punktmenge ermöglichte. Implementiert wurde vorerst eine sehr einfache Zählmethode, bei der im Prinzip mit einer 5-fachen `for`-Schleife nach leeren konvexen 5-Ecken gesucht wurde.

Mit `check5` und der bekannten Order Type-Datenbank für $n \leq 10$ [5] konnten die Werte $h_5(n)$ für $n \leq 10$ verifiziert werden. In Kombination mit der `GeneralFunctionality` von `Merged Cape` [16, 15] konnten die relevanten $n = 10$ Order Types erweitert werden, sodass auch die Werte $h_5(11) = 2$ und $h_5(12) = 3$ verifiziert werden konnten. Auch eine Erweiterung auf $n = 13$ Punkte war möglich, mit welcher 168 abstrakte Order Types mit 3 convex 5-holes (Anzahl später bestätigt) gefunden wurden. Da nach kurzer Zeit bereits Realisierungen dieser Order Types gefunden werden konnten, war gezeigt, dass $h_5(13) = 3$ gilt.

**Dec 2012:** Die Erweiterung von `Merged Cape` um die `PentagonFunctionality` hat es ermöglicht, die Erweiterungs-Funktionalität von `Merged Cape` zu nutzen, ohne die Daten im Nachhinein händisch filtern zu müssen. Mit einer verbesserten Zähl-Methode konnte das Ergebnis (168 abstract Order Types mit 3 convex 5-holes) für $n = 13$ in weniger als 30 Minuten bestätigt werden.

Nach ca. einer Woche war die durchgeführte Berechnung zur vollständigen Erweiterung der abstrakten Order Types mit $\leq 6$ convex 5-holes bis $n = 14$ abgeschlossen. Nachdem alle so erzeugten $n = 14$ abstrakten Order Types 6 convex 5-holes aufwiesen und einige dieser abstrakten Order Types realisierbar waren, war gezeigt, dass $h_5(14) = 6$ gilt.

Einige Experimente ließen bereits vermuten, dass $h_5(15) = 9$ gelten könnte. Diese Vermutung wurde später durch intensive Berechnung auch bestätigt.

Um die Erweiterungen lokal effizient durchführen zu können, wurde ein kleines Tool mit dem Namen `paracape` erstellt, welches das automatische Aufteilen des Inputs in kleinere Parts und das parallele Abarbeiten dieser Parts ermöglichte.

**Jan 2013:** Da die vollständige Erweiterung für größere Punktmengen nur schwer durchführbar ist, habe ich aus dem Parallelisierungs-Tool `paracape` eine Modifizierung mit dem Namen `randcape` erstellt, mit der nun eine randomisierte Erweiterung möglich war. Wie sich allerdings herausgestellt hat, konnten kaum Order Types für $n > 20$ gefunden werden, da wegen dem *isFather-Test* (Überprüfung auf lexikographisches Minimum) in `Merged Cape` der Großteil der intern berechneten abstrakten Order Types verworfen wurde.

**Feb 2013:** Analog zu `check5` wurden auch `check6` zum Zählen von konvexen 6-Ecken und `check4` zum Zählen von konvexen Vierecken implementiert. Des weiteren wurde ein *split-Feature* implementiert, welches ermöglichte, (abstrakte) Order Types bezüglich ihrer Anzahl an convex 5-holes (oder einer anderen Eigenschaft) zu gruppieren und in verschiedene Ausgabedateien aufzuteilen. Da

`check*` sehr einfache Tools waren und die Parameter gleich wie in vielen anderen Tools gewählt waren, wurde kein zusätzlicher optionaler Parameter hinzugefügt. Für optionale Features in `check*` wurden Use-Flags definiert, welche beim Compilieren aktiviert oder deaktiviert sein können. Da das split-Feature genau so oft unerwünscht wie erwünscht war, wurden weitere Executables `check*_split` im `Makefile` angegeben und erstellt.

Aufgrund des isFather-Test-Problems wurde `Merged Cape` mit einem weiteren Parameter `ignoreFatherTest` ausgestattet, sodass der isFather-Test umgangen werden konnte. Somit konnten alle aus der Erweiterung entstandenen abstrakten Order Types aufgezählt werden.

Um die randomisierte Suche nach einer besseren oberen Schranke von $h_5(n)$ effizienter durchführen zu können, wurde ein Tool mit dem Namen `meta_randcape` implementiert. `meta_randcape` ermöglichte es, einen zufälligen gewählten Teil einer gegebenen Startmenge an abstrakten Order Types solange mit `randcape` zu erweitern, bis bessere Schranken gefunden wurden. Wird eine bessere Schranke gefunden, so werden diese (sofern gewünscht) automatisch übernommen und für spätere Suchvorgänge verwendet.

Mit `meta_randcape` und dem `ignoreFatherTest` Parameter konnte die obere Schranke von $h_5(n)$ für $n > 15$ verbessert werden und für $n > 20$ konnten nun überhaupt erst (abstrakte) Order Types mit wenigen convex 5-holes gefunden werden.

**Feb/Mar 2013:** Um die bisherigen Ergebnisse visualisieren und ästhetischer machen zu können, wurde ein weiteres Tool mit dem Namen `qtvis` implementiert. Mit `qtvis` konnten Realisierungen von Order Types (keine abstrakten) angezeigt und beliebig verändert werden. Dadurch konnte die obere Schranke von $h_5(n)$ für $n > 15$ einfach durch Probieren und etwas Glück verbessert werden.

In `qtvis` finden sich auch Algorithmen zum Zentrieren der Punkte im Inneren der jeweiligen Zelle, zum Skalieren der gesamten Punktmenge und ein einfaches Optimierungsverfahren, um die Anzahl der convex 5-holes (oder auch convex 6-holes) falls möglich zu reduzieren.

Mit Hilfe eines bereits bekannten $n = 29$ Order Type ohne convex 6-holes und 151 convex 5-holes [14] konnten die obere Schranke von $h_5(n)$ für $n \geq 30$ durch Einfügen neuer Punkte und durch Optimieren mit `qtvis` ebenfalls um einiges verbessert werden. In Kombination mit einem weiteren neuen Tool namens `reduce` war es ebenfalls möglich, die Schranken für $20 \leq n < 30$ deutlich zu verbessern. Das Tool `reduce` ermöglichte es, alle $(n-k)$ Punkte Vorgänger-Order Types durch Entfernen von $k$ Punkten aus einem gegebenen $n$ Punkte Order Type zu erzeugen.

**May/Jun 2013:** Anfang Mai war die von Prof. Aichholzer durchgeführte Berechnung zur vollständigen abstrakten Erweiterung der (abstrakten) Order Types mit $\leq 9$ convex 5-holes bis $n = 15$ abgeschlossen. Dabei hat sich ergeben, dass alle (abstrakten) Order Types mit $n = 15$ Punkten mindestens 9 convex

5-holes haben. Da auch ein $n = 15$ Order Type mit 9 convex 5-holes realisiert werden konnte, war $h_5(15) = 9$ gezeigt.

Aus den Ergebnissen dieser vollständigen Erweiterung konnte auch gezeigt werden, dass es bei $n = 16$ immer $\geq 10$ convex 5-holes gibt, da kein (abstrakter) Order Type mit nur 9 convex 5-holes gefunden werden konnte. Da ein $n = 16$ Order Type mit 11 convex 5-holes realisiert werden konnte, war gezeigt, dass entweder $h_5(16) = 10$ oder $h_5(16) = 11$ gelten muss.

Desweiteren konnten aus diesen Ergebnissen ein $n = 17$ Order Type mit nur 16 convex 5-holes und ein $n = 18$ Order Type mit nur 21 convex 5-holes generiert werden. Beide konnten realisiert werden.

# 4 Ergebnisse

In Tabelle 1 ist die minimale Anzahl $h_5(n)$ an convex 5-holes in Order Types mit $n$ Punkten für $n \leq 16$ gelistet. Die mit * markierten Einträge sind bereits bekannte Resultate [13, 12, 11, 9]. Bisher ist für $n = 16$ kein (abstrakter) Order Type mit nur 10 convex 5-holes bekannt - die Existenz eines solchen ist bisher jedoch auch nicht ausgeschlossen.

| $n$ | $h_5(n)$ |
|---|---|
| $\leq 9$ | 0* |
| 10 | 1* |
| 11 | 2* |
| 12 | 3* |
| 13 | 3 |
| 14 | 6 |
| 15 | 9 |
| 16 | $\in \{10, 11\}$ |

Tabelle 1: Minimale Anzahl $h_5(n)$ an convex 5-holes in Order Types mit $n$ Punkten

$n = 14$ und $n = 15$ Order Types mit 6 bzw. 9 convex 5-holes waren bereits vor meiner Arbeit bekannt [6], allerdings war die Minimalität dieser noch ungewiss. Nun wurde gezeigt, dass diese tatsächlich minimal sind bzw. dass $h_5(14) = 6$ und $h_5(15) = 9$ gilt.

Für $n = 13$ konnten 168 neue Order Types mit nur 3 convex 5-holes gefunden werden (siehe Abbildung 2). Ein $n = 13$ Order Type mit 3 convex 5-holes ist im Point Set Zoo aufgelistet [7].

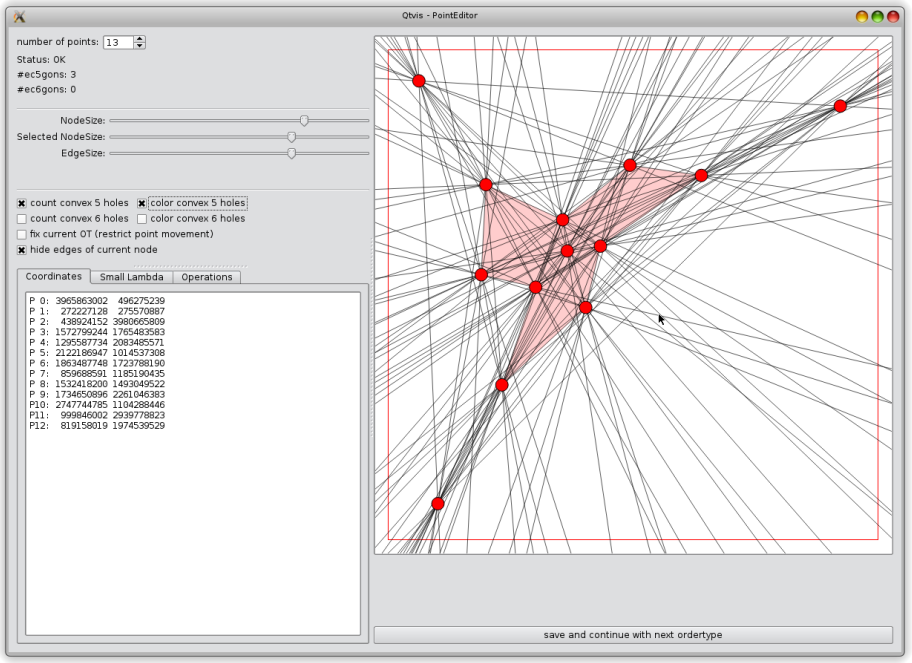


Abbildung 2: $n = 13$ Order Type mit nur 3 convex 5-holes

In Tabelle 2 sind die bisher besten Funde für realisierbare Order Types mit $n = 17..36$ Punkten gelistet, welche die bisher kleinste Anzahl an convex 5-holes aufweisen. Diese liefern nur eine obere Schranke für $h_5(n)$, die exakten Werte von $h_5(n)$ für $n \geq 17$ sind bisher noch unbekannt. Desweiteren sind in Tabelle 2 die Werte jener abstrakten Order Types gelistet, welche bisher die kleinste Anzahl an convex 5-holes aufweisen aber nicht notwendigerweise realisierbar sein müssen.

| n | $h_5(n)$ = min #c5h (realisierbar) | min #c5h (abstrakt) |
|---|---|---|
| 17 | $\leq$16 | |
| 18 | $\leq$21 | |
| 19 | $\leq$28 | $\leq$27 |
| 20 | $\leq$36 | $\leq$34 |
| 21 | $\leq$43 | $\leq$40 |
| 22 | $\leq$52 | $\leq$48 |
| 23 | $\leq$63 | $\leq$55 |
| 24 | $\leq$71 | |
| 25 | $\leq$82 | |
| 26 | $\leq$90 | |
| 27 | $\leq$103 | |
| 28 | $\leq$117 | |
| 29 | $\leq$134 | |
| 30 | $\leq$155 | |
| 31 | $\leq$177 | |
| 32 | $\leq$200 | |
| 33 | $\leq$223 | |
| 34 | $\leq$255 | |
| 35 | $\leq$290 | |
| 36 | $\leq$330 | |

Tabelle 2: Bisher beste Werte für $n = 17..36$ Punkte

Für die aufgelisteten Order Types mit $n = 17$ und $n = 18$ konnten Realisierungen gefunden werden. Diese Order Types wurden durch Erweiterung der $n = 15$ Order Types mit nur 9 convex 5-holes gefunden, müssen aber nicht notwendigerweise minimal sein!

Die aufgelisteten Order Types für $n \geq 19$ konnten mit dem Tool `qtvis` (siehe Kapitel 9) und etwas Glück gefunden werden.

Die aufgelisteten (abstrakten) Order Types für $n = 19..23$ konnten mit dem Tool `meta_randcape` (siehe Kapitel 9) durch randomisierte Erweiterung gefunden werden. Wie sich in der Tabelle erkennen lässt, weichen die Werte zwischen den abstrakten Order Types bereits bei $n = 23$ weit von denen der Realisierungen ab. Somit liegt die Vermutung nahe, dass die Werte noch weit von den tatsächlichen Minima entfernt sind.

# 5 check5

## 5.1 Beschreibung

`check5` ist ein sehr einfach gehaltenes Tool zum Abzählen von leeren konvexen 5-Ecken ("convex 5-holes") in abstrakten Order Types und in Realisierungen von Order Types. Der Output des Programms beinhaltet alle interessanten Order Types bzw. abstract Order Types, in denen
*count* $\leq$ `maximum_interested_count` gilt. Des weiteren ist mit dem `split`-Feature anstatt einer einzelnen Ausgabe-Datei wie etwa `till3.txt` ein Splitten in einzelne Dateien für eine bestimmte Anzahl möglich wie etwa `count0.txt`, `count1.txt`, ...

Nach dem Abarbeiten einer Datei wird eine kleine Statistik bezüglich der Anzahl der gefunden Order Types pro convex 5-hole-Anzahl aufgelistet.

Mit diesem Tool können Order Types und abstract Order Types mit der selben Anzahl an convex 5-holes gruppiert werden.

**Bemerkung zum Quellcode:** In der Implementierung werden convex 5-holes "ec5gons" oder auch "empty convex 5-gons" genannt.

## 5.2 Parameterbeschreibung

`./check5 n bytes filepath maximum_interested_count`

- `n`: Anzahl der Punkte des Input-Order Types bzw. abstract Order Types. n< `MAX_N`
- `bytes`: Als Input/Output-Datentypen für Order Types bzw. abstract Order Types wurden nur Points-Binary- (pb) und Lambda-Text-Dateien (lt) implementiert. (siehe `Merged Cape` Dokumentation für weitere Details [15]).

  Für 1, 2 oder 4 Bytes pro Koordinate des (Points-Binary) Order Types muss hier 1, 2 oder 4 als Parameter angegeben werden. Wird 0 angegeben, so handelt es sich um eine Lambda-Text-Datei (abstract Order Types).
- `filepath`: Der Pfad zur Input-Datei.
- `maximum_interested_count`: Anzahl der maximalen zu zählenden convex 5-holes. Wird diese Anzahl an convex 5-holes überschritten, so wird das Abzählen für den jeweiligen Order Type bzw. abstract Order Type beendet und für den aktuellen Order Type bzw. abstract Order Type wird der Zähler für die Statistik auf `maximum_interested_count` + 1 gesetzt (unabhängig von der tatsächlichen Anzahl).

**Beispiele:**

```
./check5 8 0 ../_data/aOT/lambda_abstr_8.txt 3
./check5 9 2 ../_data/OT/otypes09.b16 1
```

## 5.3 Build

Mittels `make check5` bzw. `make check5_split` kann die gewünschte Executable erstellt werden. Mittels `make build` können sämtliche checks erzeugt werden.

Standardmäßig werden die boost libraries [1] verwendet, um Ordner zu erstellen. Diese können aber entfernt werden, indem
`USEBOOST = -DBOOST_LIB -lboost_filesystem` im Makefile auskommentiert wird. Um `check*` (sämtliche `check`-Tools) ohne boost libraries auszuführen zu können, müssen die Output-Pfade (`out4`, `out5`, `out6` und `outgp`) allerdings existieren oder händisch erstellt werden.

## 5.4 Splitten des Outputs

Wird `check5_split` mittels `make check5_split` erstellt, so kann mit dieser Executable und den selben Parametern wie beim gewöhnlichen `check5` der Output in mehrere Dateien aufgeteilt werden. Bei `check5_split` handelt es sich um die selbe Implementation wie bei `check5`, nur eben ohne die Präprozessor-Definition `-DONLY_ONE_OUTPUT_FILE`.

## 5.5 Zähl-Methoden

### 5.5.1 SafetyCount

Bei der trivialen Zähl-Methode SafetyCount
(`Check5::ALLempty5GonsSafetyCount`) wird ein Punkt a ausgewählt, welcher OBdA der Punkt mit dem kleinsten Index im convex 5-hole sein soll. Danach werden Punkte $b, c, d, e$ mit größerem Index als $a$ gewählt und überprüft, ob es sich bei dem geschlossenen Polygon $a-b-c-d-e$ um ein convex 5-hole handelt (die Umlaufrichtung wird dabei berücksichtigt). Somit wird jedes convex 5-hole genau einmal gezählt.

### 5.5.2 FastCount

Neben der trivialen Methode (`Check5::ALLempty5GonsSafetyCount`) wurde auch eine Methode FastCount (`Check5::ALLempty5GonsFastCount`) zum schnelleren Abzählen implementiert. Diese verwendet eine Methode
(`Check5::countEmptyPentagonsAtLastPointFaster`), die die Anzahl der convex 5-holes am letzten Punkt $p$ zählt (genauer: nur Punkte mit Indizes $\leq p$), und summiert deren Ergebnisse für die Punkte 5 bis $n$ auf. (1 bis 4 Punkte können noch kein 5-Eck ergeben).

**Achtung:** Damit die FastCount-Methode (siehe Kapitel 5.5.2) korrekt arbeitet, muss der gegebene abstract Order Type ein natürliches Labeling haben! Der zuletzt eingefügte Punkt darf somit nicht im Inneren der zuvor eingefügten Punkte liegen!

Da bei `Merged Cape` (sowohl bei `Pentagon-` als auch `GeneralFunctionality`) die Erweiterungen so durchgeführt werden, dass der eingefügte Punkt immer auf der konvexen Hülle liegt, liegt immer ein natürliches Labeling vor, sofern die Eingabedaten ein natürliches Labeling haben.

Die Idee hinter der Zählmethode von
`Check5::countEmptyPentagonsAtLastPointFaster` für $n$ Punkte ist die Folgende:

- Jedes gezählte convex 5-holes $(h, i, j, k, l)$ beinhaltet den letzten Punkt, somit gilt $h = n - 1$ OBdA (Indizierung der Punkte von 0 beginnend).
- Wird $i$ gewählt, so müssen $j$, $k$ und $l$ auf der linken Seite der gerichteten Geraden $h - i$ liegen. Somit werden sämtliche Punkte, die nicht in der linken Halbebene liegen für $j$, $k$ und $l$ als ungültig maskiert.
- Wird $j$ unmaskiert gewählt (sofern ein solches gültiges bzw. unmaskiertes $j$ überhaupt existiert), so handelt es sich bei bei den Punkten $h-i-j$ um ein leeres konvexes 3-Eck. Die Punkte $k$ und $l$ müssen einerseits außerhalb des 3-Ecks $h - i - j$ und andererseits links von der gerichteten Geraden $i - j$ (da konvex) liegen. Dadurch werden weitere Kandidaten für $k$ und $l$ maskiert, sofern sie nicht bereits maskiert sind.
- Analog dazu wird $k$ gewählt (leeres konvexes 4-Eck $h - i - j - k$)
- Analog dazu wird $l$ gewählt (leeres konvexes 5-Eck $h - i - j - k - l$)
- Konnten zum gegebenen Punkt $h$ und dem ausgewählten Punkt $i$ gültige Punkte $j$, $k$, $l$ gefunden werden, so handelt es sich hierbei aufgrund der Konstruktion um gültiges leeres konvexes 5-Eck.
- Da sämtliche gültige Kandidaten für $i$, für $j$, für $k$ und für $l$ probiert werden, werden durch diese Methode alle leeren konvexen 5-Ecke (welche den letzten Punkt beinhalten) gefunden.

**Bemerkung:** In der Implementierung wurden die Rollen von $h$ und $i$ vertauscht, $i$ wird somit als der zuletzt eingefügte Punkt (der Punkt mit dem größten Index) fixiert. Diese Änderung hat sich als Optimierung herausgestellt, da somit anscheinend mehr Punkte bereits früher maskiert werden.
In `Merged Cape` kann die Vertauschung von $h$ und $i$ beliebig aktiviert oder deaktiviert werden.

## 5.6 qtvis-Erweiterung

Um mit dem Programm `qtvis` convex 5-holes in Punktmengen finden und anzeigen zu können, wurde in der Klasse Check5 eine Schnittstelle implementiert. Diese Schnittstelle ist durch das Präprozessor-Define
`CHECK5REPORTRESULTS` aktivierbar, da das Aufzählen der convex 5-holes außerhalb von `qtvis` nicht benötigt wird.

## 5.7 Mögliche Optimierungen und Erweiterungen

- FastCount: Liste anstatt Bitmask für gültige Punkte (erwartungsgemäß sehr wenige gültige Punkte für $k$ und $l$)
- FastCount: $j$ könnte als der letzte Punkt fixiert werden, eventuell bessere durchschnittliche Laufzeiten.
- Verallgemeinerung von FastCount auf nicht-leere 5-Ecke
- Verbesserung der Schnittstelle zu `qtvis` (erstens nur neue 5-Ecke auflisten statt alle o.ä. und zweitens nur 5-Ecke um den bewegten Punkt zählen und auflisten, da andere 5-Ecke nicht verändert werden)
- Laufzeit der trivialen Methode auf $\binom{n}{5}$ verbessern: durch Auswählen der Punkte, Ordnen der Punkte und Überprüfung auf innere Punkte.

# 6 Merged Cape-Erweiterung

## 6.1 Änderungen am ursprünglichen Merged Cape

- neuer Parameter `Pentagon` für `Merged Cape`:
  Ermöglicht das Zählen von convex 5-holes während dem abstrakten Erweitern.
- neuer Parameter `IgnoreFatherTest` für Cape:
  In cape ist der `FatherTest` standardmäßig aktiviert und somit werden eventuell "gute" Resultate mit einer hohen Wahrscheinlichkeit verworfen, und das nur um Duplikate zu vermeiden. Da dies bei der randomisierten Suche nach "guten" Order Types bzw. abstract Order Types allerdings nicht gewünscht ist, wurde der `IgnoreFatherTest` Parameter zu `Merged Cape` hinzugefügt. Somit werden "gute" Resultate nicht verworfen, was eine breitere Suche ermöglicht.

  Des weiteren muss der Parent eines "guten" Resultats für $n+1$ Punkte nicht unbedingt ein "gutes" Resultat für $n$ Punkte gewesen sein. Für $n \geq 20$ ist es bereits sehr schwer, überhaupt noch abstract Order Types zu finden (nicht unbedingt gute), welche den `isFatherTest` bestehen, da die Wahrscheinlichkeit den `isFatherTest` zu bestehen indirekt proportional zu n ist - sofern man von einer Gleichverteilung ausgeht.

  Einbettung in `VariableFunctionality::recAbstractExtension` (für allgemeine Erweiterungen) und
  `PentagonFunctionality::recAbstractExtension` (convex 5-holes)
- neuer Parameter für die Funktion
  `Functionality::calcLittleLambdaToBigLambda`:
  Es ist zwar möglich `MAX_N` kleiner zu setzen, um Zeit beim Ändern der Einträge in der (Groß-) Lambda Matrix zu sparen. Allerdings lässt sich nun auch mit dem hinzugefügten `bool dirty_initialisation` Parameter angegeben, ob Einträge in der Groß-Lambda Matrix mit Indizes $\geq$ `PNo` uninitialisiert bleiben sollen.
- Fix von `Functionality::getMinimalLambdaAndIsFatherLikeHappyEnd`:
  Überprüfung auf $j \geq 0$, da andernfalls ungültige Speicherzugriffe passieren, sofern kein gültiges Labeling gefunden werden kann.
- Fix von `Functionality::getMinimalLambdaAndIsFatherSimple`:
  Überprüfung auf `lastpoint`$ge$0, da andernfalls ungültige Speicherzugriffe passieren, sofern kein gültiges Labeling gefunden werden kann.
- Fix von Destruktoren (allgemein):
  `throw ()` von allen Destruktoren entfernt, da dies in den meisten Fällen eine schlechte Idee ist.
- Aufteilung von `TreeNode::insertNode`:
  Anstelle der "Search and Insert"-Methode eine eigene Search- und eine eigene Insert-Methode.
- Fix von Member `valid_cuts_` @ `class ValidCuts`:

`int32 valid_cuts_[MAX_N][21][3];` ersetzt durch
`int32 valid_cuts_[MAX_N][MAX_N][3];`

- Fix von Member
`standard_validation_set_` @ `class ArgumentReader`:
Initialisierung `standard_validation_set_ = true;` und
`standard_validation_set_ = false;` falls der Parameter
`-alternativevalidation` angegeben wird.
- Fix von `LittleLambdaTextFileReader::getNextLittleLambda`:
Bugfix für Lambda Textfiles mit Leerzeilen am Ende
- Fix von Parameter-Debug @ `main`:
`getTimeoutScreenEnabled` und `getTimeoutFileEnabled` vertauscht
- `Catch WrongArgumentException` @ `main`
- fehlende `<cstdio>` Includes ergänzt

### 6.1.1 Mögliche Optimierungen und Erweiterungen

- std::map (o.ä.) anstelle von `struct TreeNode` oder zumindest Such- und Einfügeoperationen verbessern

## 6.2 PentagonFunctionality

### 6.2.1 Implementierung

Die Funktionen zum Zählen von convex 5-holes wurden aus `check5` (siehe Kapitel 5) übernommen und angepasst. Die restliche Logik für die Erweiterung wurde aus der `General Functionality` von `Merged Cape`[16, 15] übernommen. Neu implementiert wurde eine Umgehung des `isFatherTests`, welche eigentlich ausschließlich für die randomisierte Erweiterung (siehe `randcape` Dokumentation, Kapitel 8) verwendet wird.

### 6.2.2 Optimierung

Wie bereits in der Dokumentation zu `check5` erwähnt wurde, kann beim Zählen von convex 5-holes $h$ mit $i$ vertauscht werden - anstatt den ersten Punkt zu fixieren kann man den zweiten Punkt des convex 5-holes als den lexikographisch Größten fixieren. Mittels einem Präprozessor-Define `FIX_H_AND_LOOP_OVER_I` kann $h$ anstelle von $i$ fixiert werden.

# 7 paracape

## 7.1 Beschreibung

`paracape` ist ein python script, welches den Input für `Merged Cape` in mehrere Parts aufteilt, die Abarbeitung der Parts parallelisiert, mit `Merged Cape` abarbeitet, und die einzelnen Teilergebnisse nach erfolgreicher Abarbeitung aller Parts wieder vereinigt.

## 7.2 Ablauf- und Parameterbeschreibung

In der main-Funktion sind sämtliche Parameter setzbar.

### 7.2.1 Teil 1: Aufteilung der Daten

Die als gegebenen Order Types bzw. abstract Order Types werden in `max_part_count` Teile aufgeteilt um diese später parallel abzuarbeiten.

Durch den Parameter `max_part_count` kann die maximale Anzahl an Parts festgesetzt werden. Da einzelne Daten (einzelne Order Types bzw. abstract Order Types) atomar sind und nicht aufgetrennt werden können, gibt `max_part_count` nur eine obere Schranke für die tatsächliche Anzahl der Parts an.

Der Parameter `max_parallel_processes` gibt an, wie viele der Parts zur gleichen Zeit (parallel) abgearbeitet werden dürfen. Beispielsweise wäre #Cores + 1 keine schlechte Wahl, da somit alle Cores ausgelastet werden können. Bei zu vielen gleichzeitigen Parts kann durch das ständige Neuzuordnen der Threads zur CPU ein zeitlicher Overhead entstehen und die Gesamtlaufzeit kann sich somit deutlich verschlechtern.

### 7.2.2 Teil 2: Parallele Verarbeitung der Parts

Mit den in der main-Funktion definierten Parametern wie etwa capetype, inputfile, inputtype, etc. wird `Merged Cape` für den jeweiligen Part des Inputs als eigener Prozess gestartet. (siehe `Merged Cape` Dokumentation für cape-spezifische Parameter [15])

Alle Zwischenergebnisse von `Merged Cape` werden in Pfad `TMP_DIR+"/tmp"` geschrieben. Sobald die Berechnung für den aktuellen Part erfolgreich beendet wurde, wird das Ergebnis des Parts von `TMP_DIR+"/tmp"` nach `TMP_DIR+"/done"` verschoben.

Durch ein `random.shuffle(parts)` in der main-Funktion (ca. Zeile 159) wird die Abarbeitungsreihenfolge der Parts randomisiert. Es werden aber Parts und somit alle abstract Order Types abgearbeitet, und die Teilergebnisse nach der parallelen Verarbeitung in der ursprünglichen Reihenfolge (Part1, Part2, ...) zusammengehängt. Dieses Feature kann natürlich auch ohne Bedenken deaktiviert werden.

### 7.2.3 Teil 3: Vereinigung der Teilergebnisse

Wurden alle Parts erfolgreich abgearbeitet, so werden die Dateien der einzelnen Teilergebnisse (1 Datei pro Part) zu einer Datei zusammengehängt.

**Korrektheit:** Es können durch diese Vorgehensweise keine Duplikate im Ergebnis auftreten, da ein abstract Order Type nur dann den isFather-Test besteht, sofern er aus einem eindeutig bestimmten Parent abstract Order Type erzeugt wurde. Da die einzelnen Parts disjunkt sind (beinhalten keine gemeinsamen abstract Order Types), kann kein abstract Order Type in den Ergebnissen zweier verschiedener Parts auftreten!

**Achtung:** TMP_DIR (globale Variable, ca. Zeile 21) muss auf den Pfad bzw. das Pfad-Präfix gesetzt werden, der für die Speicherung der Zwischenergebnisse verwendet werden soll. Bei einem Abbruch des Tools werden keine temporären Daten gelöscht und dieser Pfad muss selbstständig bereinigt werden! Dies ermöglicht die manuelle Wiederverwertung und Auswertung von Teilergebnissen.

**Beispiel:**

- Beispiel-Beschreibung: Erweiterung von $n = 8$ Order Types auf $n = 13$ abstract Order Types mit $\leq 3$ convex 5-holes
- Parameter für `Merged Cape` (Teil 2):

```
config = dict()
config["capetype"] = "p"
config["inputfile"] =
  "/home/manfred/TUG_misc/bak/ot/points/otypes08.b08"
config["inputtype"] = "pb"
config["byte"] = "1"
config["n"]    = "8"
config["nmax"] = "13"
config["boundsfile"] = "../../bound/3.txt"
config["setmatrixoutputstart"] = config["nmax"]
config["outputtype"] = "lt"
```

- Parameter für `paracape` (Teil 1):

```
max_part_count = 160
max_parallel_processes = 8
```

- Ausführung mittels `"python paracape.py"` oder `"./paracape.py"` (ohne zusätzliche Parameter)

## 7.3 Konstanten und Settings

* `TMP\_DIR`: Der Pfad-Prefix, an dem temporäre Dateien gespeichert werden. Zu diesem wird `"_tmp/"` und `"_done/"` angehängt und diese beiden Ordner werden während der Ausführung zur Speicherung von Zwischenergebnissen benötigt.

## 7.4 Funktionen

- `prepare_folders`: Diese Funktion erstellt den Zwischenspeicherungs-Pfad oder bereinigt diesen.
- `cleanup_folders`: Falls der Zwischenspeicherungs-Pfad bereits existiert, werden alte Dateien entfernt.
- `merge_results(result_files,outfile)`: Diese Funktion vereinigt die einzelnen Zwischenergebnisse (Dateinamen übergeben in der Liste `result_files`) und schreibt diese in eine Datei (outfile) zusammen.
- `job(i,j,config)`: Die Funktion job wird für jeden Part aufgerufen. Der Parameter i gibt den Start- und j den End-Index für die abzuarbeitenden Order Types bzw. abstract Order Types an. In `config` sind spezifische Parameter für cape eingetragen.
- `file_len(fname)`: Diese Funktion gibt die Anzahl der Zeilen der Datei zurück. Dabei wird momentan der Unix Befehl `wc -l` verwendet, um schnelleres Zählen zu ermöglichen.
- `countOTs(config)`: Diese Funktion gibt die Anzahl der Order Types bzw. abstract Order Types zurück, die durch die gegebene Konfiguration (config) als Input angegeben sind.
- `main`: In der main Funktion können Parameter für die Abarbeitung der Daten und für Cape gesetzt werden.

## 7.5 Mögliche Optimierungen und Erweiterungen

- Ähnlich wie bei `randcape` könnte man die Parameter via Kommandozeile übergeben
- Da im `config`-Hash momentan nur `Merged Cape` Parameter eingetragen werden, könnte man weitere Parameter zu `config` hinzufügen und die Funktion `job`, welche für jeden der einzelnen Parts ausgeführt wird, so anpassen, dass auch diese neuen Parameter an cape übergeben werden. Man könnte in der `job` Funktion auch mit einer for-Schleife durch alle Einträge des `config`-Hashes iterieren und sämtliche Parameter inklusive Wert zum Ausführungs-Befehl hinzufügen. (`config["parts"]` müsste dafür aus Hash entfernt und als eigener Parameter an die Funktion `job` übergeben werden, da dieser Parameter nicht für `Merged Cape` gedacht ist)

# 8 randcape

## 8.1 Beschreibung

Das Tool `randcape` war ursprünglich als eine leichte Modifizierung von `paracape` gedacht. Anstatt alle Order Types bzw. abstract Order Types zu erweitern, werden nur eine bestimmte Anzahl an Parts (`partcount`) zufällig ausgewählt und diese dann mittels `Merged Cape` erweitert, sodass die Anzahl der convex 5-holes kleiner gleich einer gegebenen oberen Schranke (`bound`) bleibt. Wie bereits in der Dokumentation zu `Merged Cape` erwähnt, gibt es den Parameter `ignoreFatherTest`. Wird dieser gesetzt, so werden keine durch Erweiterung gewonnenen abstract Order Types verworfen. Dies ermöglichte auch eine zufällige Erweiterung auf Punktmengen mit $n \geq 20$, welche, wie sich herausstellte, ohne ignoreFatherTest kaum Ergebnisse lieferte, da in den meisten Fällen (Erwartungsgemäß 1:20) der isFather-Test fehlgeschlagen ist.

Sind während der Abarbeitung der Parts bereits genügend "gute" Erweiterungen vorhanden, so wird die aktuelle obere Schranke für die Verarbeitung weitere Parts reduziert.

## 8.2 Verwendung

Struktur:
```
./randcape.py [n] [bytes] [infile] [outfile] [bound] [partcount]
[partsize] [maxparallel]
```

Beispiel:
```
./randcape.py 8 0 ../_data/aOT/lambda_abstr_8.txt n9.txt 0 20
10 5
```

## 8.3 Parameterbeschreibung

- `n`: Anzahl der Punkte
- `bytes`: Anzahl der Bytes (0 entspricht Lambda Text abstract Order Types, siehe `count` Dokumentation, Kapitel 5.2)
- `infile`: Die zu analysierende Inputdatei
- `outfile`: Der Pfad für die resultierende Datei mit Erweiterungen
- `bound`: Obere Schranke für die Anzahl der convex 5-holes (Ganzzahliger Wert)
- `partcount`: Anzahl der zu erweiternden Parts
- `partsize`: Größe eines einzelnen Parts (Anzahl der abstract Order Types pro Part)
- `maxparallel`: maximale Anzahl parallel zu verarbeitenden Parts

## 8.4 Konstanten und Settings

- `UPDATE_BOUNDS`: Wenn `UPDATE_BOUNDS` auf `True` gesetzt ist, so werden die Bounds (inklusive Boundfiles) automatisch upgedated.
- `BACKUP_PATH`: Der Pfad, an dem die Backup-Files abgespeichert werden sollen.
- `BACKUP_LZO_COMPRESSION`: Wenn `BACKUP_LZO_COMPRESSION` auf `True` gesetzt ist, so werden Backupfiles LZO [2] komprimiert.

## 8.5 Funktionen

- `file_len`, `countOTs`, `prepare_folders`, `cleanup_folders`, `merge_results`, `job`: (siehe `paracape` Dokumentation, Kapitel 7.4)
- `prepare_bound`: Erstellt eine `bound.txt` Datei, die von cape benötigt wird.
- `reduce_bound`: Vermindert die obere Schranke (bound.txt) für convex 5-holes für die aktuelle Erweiterung um 1.

# 9 meta_randcape

## 9.1 Beschreibung

`meta_randcape` ist eine Art Steuerungs-Script für `randcape` mit ein paar weiteren Funktionen:

Zuerst wird die zu erweiternde abstract Order Type-Datei auf Duplikate überprüft. Dann wird ein Backup der Datei angelegt (sofern eingestellt) und die Datei wird mittels `count5_split` gesplittet, wobei sich die obere Schranke für convex 5-holes aus der aktuell besten Schranke, einem gegebenen `epsilon` und einem Faktor `alpha` berechnet.

Danach wird eine neue abstract Order Type-Datei aus diesen geteilten Dateien erstellt (greedy - so wenig convex 5-holes wie möglich), wobei sich die Anzahl der benötigten abstract Order Types für diese neue Datei aus `parts` × `partsize` errechnet. Von dieser neuen Datei wird ebenfalls ein Backup erstellt.

Wurde diese Greedy-abstract Order Type-Datei erstellt, so wird diese mittels `randcape` erweitert.

Wird beim Erweitern mit `randcape` eine neue Schranke gefunden, so setzt `meta_randcape` die Schranke für weitere Suchvorgänge ebenfalls nach unten. Dadurch ist es möglich, die Suche beispielsweise mit einer Schranke von 50 convex 5-holes für $n = 1..20$ zu beginnen und `meta_randcape` die Schranken selbstständig finden und verbessern zu lassen.

**Bemerkung 1:** Es wird jeweils von $k$ Punkten auf $k + 1$ Punkte erweitert.

**Bemerkung 2:** Der Dateiname des $k$-Punkte abstract Order Types lautet `k.txt` und muss für Input-Dateien ebenfalls so lauten. Der Name der erstellten Greedy-Datei lautet `k.txt.part`.

**Bemerkung 3:** Die vielen Backups der Dateien sind notwendig, da mehrere Iterationen automatisch durchgeführt werden.

**Bemerkung 4:** Um die Backups klein zu halten wird hier `lzop` (Befehl für LZO-Kompression [2]) verwendet. Es wäre aber möglich, ein komprimiertes Dateisystem wie etwa Btrfs für die Backups zu verwenden. Mittels `compress=True` als Default-Parameter für die Funktion `calcBound` lässt sich die Kompression deaktivieren.

Im Ordner `best_bound` werden die aktuellen besten Schranken für alle $n$ gespeichert. In den Textdateien `best_bound/n.txt` finden sich die einzelnen Werte. Wird eine neue (bessere) Schranke gefunden, so wird der Fund des neuen Ergebnisses in der Datei `_NEW_BOUND` protokolliert und diese neue Schranke in `best_bound/n.txt` eingetragen.

## 9.2 Parameterbeschreibung

- `start`: Anzahl der Punkte, bei der die Erweiterung gestartet werden soll
- `end`: Anzahl der Punkte, bei der die Erweiterung gestoppt werden soll
- `parts`: Anzahl der zu verwendenten Parts
- `partsize`: Größe eines einzelnen Parts (Anzahl der abstract Order Types pro Part)
- `eps`: ganzzahliger Offset $\geq 0$ für die Toleranzgrenze oberhalb der aktuellen besten Schranke.
- `alpha`: Skalierungsfaktor $\geq 1$ für die Toleranzgrenze oberhalb der aktuellen besten Schranke. Beispiel: $alpha = 1.1$ bedeutet 10% mehr convex 5-holes sind erlaubt.
- `maxparallel`: gibt die maximale Anzahl an gleichzeitig zu verarbeiteten Parts an und wird an `randcape` übergeben.
- `maxtrials`: gibt die maximale Anzahl an Iterationen an, nach denen `meta_randcape` die Suche nach neuen Schranken beendet werden soll, sofern noch keine gefunden wurde.

## 9.3 Funktionen

- `exec_cmd(cmd)`:
  Wird zum Ausführen eines anderen Befehles (`cmd`) verwendet (nicht parallel).
- `file_len`, `ot_count`:
  (siehe `paracape` Dokumentation, Kapitel 7.4)
- `getDateTimeString`:
  gibt das aktuelle Datum und Uhrzeit als formatierten String zurück
- `myprint(s)`:
  Der Text `s` wird mit einer bestimmten Formatierung (inklusive Uhrzeit und Datum) ausgegeben.
- `split_till(file_to_check,n,bound)`:
  Die übergebene Datei (`file_to_check`) wird mittels `check5_split` analysiert und aufgeteilt. `bound` dient dabei als obere Schranke.
- `merge_parts_to_file(n,bound,bounds,dest,needed_input_ots)`:
  Erstellt eine neue Datei mit genügend abstract Order Types ($\geq$`needed_input_ots`) für die Erweiterung, sodass die Anzahl der convex5gons in dieser Datei minimal bleibt (greedy).
  Bemerkung: Ist `needed_input_ots` zu groß, so wird die Inputdatei gesplittet und wieder als Ganzes zusammengefügt (mit einer aufsteigenden Anzahl an abstract Order Types).

- `calcBound(bound,bound_alpha,bound_eps)`:
  berechnet aus der aktuellen Schranke(`bound`), alpha (`bound_alpha`) und epsilon (`bound_eps`) die Toleranzgrenze für weitere convex 5-holes.
- `backupFile(f_path,compress=True)`:
  Erstellt eine (komprimierte) Kopie der übergebenen Datei.
  Falls `compress=True` wird `lzop` (LZO-Kompression [2]) verwendet.
  Falls `compress=False` wird `cp` verwendet.
- `writeBestBound(n,b)`:
  Schreibt die gegebene Bound für weitere Ausführungen in die Bound-Datei.

## 9.4 Mögliche Optimierungen und Erweiterungen

- Zusammenziehen der einzelnen Funktionen aus `paracape`, `randcape` und `meta_randcape`.
- Abbrechen der parallel laufenden Erweiterungen, falls genügend gute abstract Order Types vorhanden sind.

# 10 qtvis

## 10.1 Beschreibung

`qtvis` ist ein interaktives Visualisierungstool für Punktmengen. Ursprünglich war es nur als Erweiterung von `vis.py` gedacht, jedoch wurden im Laufe der Zeit auch einige zusätzliche Features implementiert und viele weitere könnten bei Bedarf noch hinzugefügt werden. Als Framework wurde Qt verwendet, welches auch zur Namensgebung beigetragen hat.

## 10.2 Features

### 10.2.1 Punkte anzeigen

Wird `qtvis` gestartet und eine Order Type-Datei angegeben, so werden die Punkte der aktuellen Punktmenge (gegeben durch $x$- und $y$-Koordinaten) und sämtliche Geraden durch je 2 Punkte gezeichnet. Die Größe der Punkte und die Dicke der Geraden kann jederzeit mittels Slider verändert werden. Durch das Bewegen der Maus kann der Sichtbereich verschoben und mittels Mausrad kann im Sichtbereich gezoomt werden.

### 10.2.2 Punkte bewegen

Wird ein Punkt ausgewählt (angeklickt), so ändert sich dessen Farbe von rot (nicht selektiert) in blau (selektiert). Dieser kann mit Hilfe der Maus in der Ebene beliebig bewegt werden. Wird der Punkt bewegt, so wird die Lambda Matrix der neuen Punktmenge berechnet. Ändert sich dabei der Order Type, so werden Eigenschaften (wie etwa convex 5-holes) für diesen neuen Order Type berechnet und die Ansicht aktualisiert. Wird der Order Type allerdings nicht verändert, so werden die Eigenschaften nicht neu berechnet.

Es ist auch möglich, den aktuellen Order Type zu fixieren, sodass Punkte an nur bestimmten Positionen platziert werden können (`fix current OT`). Des weiteren kann auch eingestellt werden, dass sämtliche Geraden durch den selektierten Punkt nicht gezeichnet werden, was die aktuelle Zelle des ausgewählten Punktes besser sichtbar macht und das Verschieben somit erleichtert.

### 10.2.3 Convex 5-Holes anzeigen

Mithilfe der Schnittstelle von `check5` können die leeren konvexen 5-Ecke in der aktuellen Punktmenge gezählt und visualisiert werden. Um die CPU-Auslastung gering zu halten, wird nur dann ein Zählvorgang gestartet, wenn sich der Order Type verändert. In der `defines.h` von `check5` kann und muss eine obere Schranke für die Anzahl der convex 5-holes angegeben werden.

convex 5-holes werden gezählt, wenn `count convex 5 holes` aktiviert ist. Falls convex 5-holes gefunden werden, werden diese als rote 5-Ecke gezeichnet,

sofern `color convex 5 holes` aktiviert ist. Falls ein Punkt selektiert ist, werden angrenzende convex 5-holes blau gezeichnet.

### 10.2.4 Convex 6-Holes anzeigen

Analog zu convex 5-holes wurde das Zählen und Visualisieren von leeren konvexen 6-Ecken mithilfe der Schnittstelle von `check6` implementiert.

### 10.2.5 Zentrieren von Punkten in Zellen

Um die Darstellung von Order Types optisch etwas zu verschönern, können die Punkte in das Zentrum der jeweiligen Zelle bewegt werden. Dabei war geplant `LineSearch` zu verwenden, um die Ränder der Zelle zu finden, allerdings wurde nur ein einfaches Suchverfahren implementiert. Die Anzahl der Suchrichtungen wird durch `DIRECTION_COUNT` (`graphwidget.cpp`) angegeben.

### 10.2.6 Optimieren des Order Types bzgl. einer Eigenschaft

Wird die Optimierung gestartet, so wird versucht, die Punkte so zu verschieben, dass sich eine Eigenschaft (z.B. Anzahl convex 5-holes) verbessert. Dabei wird versucht, einen Punkt in eine benachbarte Zelle zu verschieben.

Es kann auch nur der letzte Punkt verschoben werden, sodass andere Punkte nicht bewegt werden.

## 10.3 Datenangabe

`qtvis` kann ohne zusätzliche Parameter gestartet werden. Beim Start erscheint ein Kontrollfenster (`Qtvis - ControlWindow`), in dem die zu öffnende Order Type-Datei angegeben werden kann. Dazu muss auch angegeben werden, wie viele Punkte die gegebenen Order Types haben und um welchen Dateityp es sich handelt (1, 2 oder 4 Bytes). Optional kann ein Output-Pfad angegeben werden, in dem das Ergebnis gespeichert werden soll. Wird kein Output-Pfad angegeben, so wird standardmäßig `Inputpfad.out` zum Speichern verwendet.

Es ist auch möglich, diese Parameter via Kommandozeile zu übergeben, dann sind die Dialogfelder nach dem Start der Anwendung mit den in der Kommandozeile angegebenen Werten ausgefüllt.

## 10.4 Öffnen, Speichern und Schließen

Sind die Angaben korrekt und kann die Datei geöffnet werden, so öffnet sich das Hauptfenster (`Qtvis - PointEditor`), in dem die Punkte eines Order Types angezeigt werden. Mit `save and continue with next ordertype` (Button rechts unten) können die aktuellen Punktpositionen gespeichert werden und es wird mit dem nächsten Order Type in der Datei fortgefahren (sofern vorhanden).

Wurden alle Order Types angezeigt bzw. bearbeitet, so erscheint erneut das Kontrollfenster. Andernfalls kann auch ohne Speichern zum Kontrollfenster zurückgekehrt werden, indem das Fenster einfach geschlossen wird.

## 10.5 Mögliche Optimierungen und Erweiterungen

- Punktkoordinaten im Textfeld veränderbar machen
- LineSearch Algorithmus (Bracketing and Sectioning Phase) implementieren
- `Speichern Als`- oder `Nicht speichern und fortfahren`-Feature
- Zählen und Visualisieren weiterer Eigenschaften
- Usability für Minimization verbessern
- andere (interaktive) Minimierungs-Verfahren implementieren
- Undo-Command oder Backup-Feature
- Anstatt dem ständigen Neuberechnen der Klein-Lambda Matrix könnte für größere $n$ auch die aktuelle Zelle berechnet und gespeichert werden, sodass nur beim Verschieben eines Punktes in eine andere Zelle die Lambda Matrix berechnet werden muss. Es könnte auch die Groß-Lambda Matrix gespeichert werden und nur die relevanten Einträge (Trippel mit dem verschobenen Punkt) neu berechnet werden.
- dynamische Fenstergröße
- byte=0 für Points-Text Dateien

# 11 Weitere Tools

## 11.1 unify_ot.py

### 11.1.1 Beschreibung

Das `unify_ot` Script überprüft die abstract Order Types aus der Inputdatei auf Duplikate und schreibt die Unikate in die Outputdatei.

### 11.1.2 Parameter

- `n`: Anzahl der Punkte
- `bytes`: Anzahl der Bytes pro Koordinate. Die aktuelle Implementation von `unify_ot` unterstützt allerdings nur Lambda-Text abstract Order Types als Input/Output, somit muss hier immer 0 (für Lambda-Text-Datei) übergeben werden.
- `inputfile`: Der Pfad der Inputdatei. Das lexikographische Minimum der abstract Order Types wird dabei nicht berechnet, muss bereits gewährleistet sein!
- `outfile`: Der Pfad der Outputdatei.

### 11.1.3 Mögliche Optimierungen und Erweiterungen

- Natürliches Labeling berechnen
- Lexikographisches Minimum berechnen
- Erweiterung auf Bytes $> 0$, um auch als Punktmengen gegebene Order Types (nicht-abstrakte Order Types) auf Duplikate überprüfen zu können

## 11.2 vis.py

### 11.2.1 Beschreibung

`vis.py` kann verwendet werden, um einen gegeben Order Type (Binary- oder Text-Datei) zu visualisieren. Dabei wird für jeden Order Type in der Inputdatei eine Grafik erstellt, als png-Datei gespeichert und, sofern der "Show Plot"-Kommandozeilen-Parameter (`"-SP"`) angegeben wird, auch direkt angezeigt.

### 11.2.2 Verwendete Libraries

- NetworkX [4]
- Matplotlib [3]

### 11.2.3 Parameter

- `number of points`: Anzahl der Punkte pro Order Type.
- `bytes`: Anzahl der Bytes. 1, 2 oder 4 Byte wie üblich. Wird hier 0 angegeben, so können die Order Types als Point-Text-Files abgespeichert werden. In jeder Zeile stehen $x$- und $y$-Koordinate, getrennt durch Whitespaces (Tabulatoren und Leerzeichen). Es können auch mehrere Order Types in einer einzelnen Datei gespeichert sein.
- `ordertype point file`: Pfad der Inputdatei.
- `"-SP"-Flag` (optional): wird `"-SP"` ("Show Plot") als zusätzlicher Parameter (an letzter Stelle) angegeben, so werden die erzeugten Grafiken direkt hintereinander angezeigt.

## 11.3 text2bin

### 11.3.1 Beschreibung

Aus einer Point-Text-Datei kann mit diesem Tool eine Point-Binary-Datei erstellt werden.

### 11.3.2 Parameter

- `n`, `bytes`, `in`, `out` (wie üblich, siehe `unify_ot.py` Dokumentation, Kapitel 11.1 oder andere Programme)
- `"-n"-Flag` (optional): Wird `-n` angegeben, so kann die erste Zeile aus der Text-Datei (in der die Anzahl der Punkte zusätzlich eingetragen sein kann) entfernt werden.

## 11.4 reduce

### 11.4.1 Beschreibung

Mit diesem Tool können sämtliche $m$ Punkte Teilmengen bzw. deren Order Types aus einem gegebenen Order Type mit $n$ Punkten generiert werden. Die Koordinaten der Punkte werden eingelesen und für alle Kombinationen von $m$ aus den gegebenen $n$ Punkten wird diese Teil-Punktmenge aufgezählt. Somit entstehen $\binom{n}{m}$ neue Order Types mit $m$ Punkten.

### 11.4.2 Parameter

- `n`, `bytes`, `in`, `out` (wie üblich, siehe `unify_ot.py` Dokumentation, Kapitel 11.1 oder andere Programme)

- `newn`: Anzahl der Punkte, die nach dem Entfernen von $k \geq 0$ Punkten übrig bleiben sollen. $k$ wird einfach berechnet, sodass $newn + k = n$ erfüllt ist.

## 11.5 check6 (experimentell)

### 11.5.1 Beschreibung

`check6` hat nichts mit dem Zählen von convex 5-holes zu tun und wurde nur für `qtvis` implementiert! Analog zu `check5` ist `check6` für das Zählen von convex 6-holes zuständig. Auch hier wurden Zählmethoden nach den selben Ideen implementiert. (siehe Kapitel 5.5)

## 11.6 check4 (experimentell)

### 11.6.1 Beschreibung

`check4` hat nichts mit dem Zählen von convex 5-holes zu tun und wurde nur aus eigener Interesse implementiert! Analog zu `check5` ist `check4` für das Zählen von convex 4-holes zuständig. Auch hier wurden Zählmethoden nach den selben Ideen implementiert. (siehe Kapitel 5.5) Anders als bei `check5` und `check6` ist es für `check4` bereits möglich nicht-leere konvexe 4-Ecke (Anzahl entspricht den Crossing Numbers [10]) zu zählen. Dazu muss das Präprozessor-Define `NON_EMPTY` gesetzt werden (beispielsweise in `check4.cpp` oder im `Makefile` als Parameter für `g++`).

### 11.6.2 Mögliche Optimierungen und Erweiterungen

- Schnittstelle zu `qtvis`

# 12 Literatur


[1] Boost C++ Libraries. http://www.boost.org/, June 2013.

[2] LZO Homepage. http://www.oberhumer.com/opensource/lzo/, June 2013.

[3] Matplotlib Homepage. http://matplotlib.org/, June 2013.

[4] NetworkX Homepage. http://networkx.github.io/, June 2013.

[5] Order Type Database. http://www.ist.tugraz.at/aichholzer/research/rp/triangulations/ordertypes/, June 2013.

[6] PoSeZo - The Point Set Zoo. http://www.eurogiga-compose.eu/posezo.php, June 2013.

[7] PoSeZo - The Point Set Zoo - Set of 13 points minimizing the numbers of convex 3-, 4-, and 5-holes. http://www.eurogiga-compose.eu/posezo/n13_c1_min_convex_3_4_5_holes/n13_c1_min_convex_3_4_5_holes.php, June 2013.

[8] O. Aichholzer, F. Aurenhammer, and H. Krasser. Enumerating Order Types for Small Point Sets with Applications. In *Proc. $17^{th}$ Ann. ACM Symp. Computational Geometry*, pages 11–18, Medford, Massachusetts, USA, 2001.

[9] O. Aichholzer, T. Hackl, and B. Vogtenhuber. On 5-Gons and 5-Holes. In A. Marquez, P. Ramos, and J. Urrutia, editors, *Special issue: XIV Encuentros de Geometría Computacional ECG2011*, volume 7579 of *Lecture Notes in Computer Science (LNCS)*, pages 1–13. Springer, 2012.

[10] O. Aichholzer and H. Krasser. Abstract Order Type Extension and New Results on the Rectilinear Crossing Number. *Computational Geometry: Theory and Applications, Special Issue on the 21st European Workshop on Computational Geometry*, 36(1):2–15, 2006.

[11] K. Dehnhardt. *Leere konvexe Vielecke in ebenen Punktmengen*. PhD thesis, TU Braunschweig, Germany, 1987.

[12] H. Harborth. Konvexe Fünfecke in ebenen Punktmengen. *Elemente der Mathematik*, 33:116–118, 1978.

[13] H. Krasser. *Order Types of Point Sets in the Plane*. PhD thesis, Institute for Theoretical Computer Science, Graz University of Technology, Austria, October 2003.

[14] M. H. Overmars. Finding sets of points without empty convex 6-gons. *Discrete & Computational Geometry*, Volume 29, Issue 1 , pp 153-158, 12 2001.

[15] M. Zöhrer. *merged cape - Software Documentation*, 2008. Projektbericht, TU Graz.

[16] M. Zöhrer. *merged cape - User Manual*, 2008. Projektbericht, TU Graz.