Programmierpraktikum Diskrete Optimierung (Modul P2C1)

Thema: Darstellungen von Rechteckpackungen

**Veranstaltungshomepage:**
`http://www.or.uni-bonn.de/lectures/ss18/praktikum_ss18.html`

## 1 Programmiersprache

Die Implementierungen müssen in C/C++ ausgeführt werden und unter Linux mit dem gcc/g++ kompilierbar sein, d.h. sie müssen dem ANSI C11/C++11-Standard genügen. Es wird empfohlen, C++ zu verwenden. Außer den Standard-Bibliotheken dürfen keine Hilfsbibliotheken verwendet werden, es sei denn es wird in der Aufgabenstellung ausdrücklich erwähnt. Der Code muss verständlich programmiert und kommentiert werden. Selbstverständlich müssen die Programme fehlerfrei funktionieren. Insbesondere werden sie mit dem Programm valgrind (http://valgrind.org) überprüft, und valgrind-Fehler führen zu Abzügen. Weitere Hinweise zu gutem Programmierstil finden Sie auch auf der oben angegebenen Internetseite.

## 2 Rechteckpackungen

Alle Rechtecke, die hier betrachtet werden, sind achsenparallel, haben also die Form $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$, wobei $x_{\min} \leq x_{\max}$ und $y_{\min} \leq y_{\max}$. Ziel ist es, eine gegebene Menge von Rechtecken überlappungsfrei zu platzieren, wobei überlappungsfrei bedeutet, dass sich das Innere von je zwei Rechtecken nicht überlappt (die Ränder von verschiedenen Rechtecken dürfen also aufeinender liegen). Wir wollen entweder alle Rechtecke in einer gegebenen rechteckigen Fläche $A$ unterbringen oder selbst ein flächenmäßig kleinstes Rechteck finden, in dem die Rechtecke platziert werden können. Zum Teil sind auch (ebenfalls rechteckige) Blockaden gegeben, mit denen sich die zu platzierenden Rechtecke ebenfalls nicht überlappen dürfen. Eine Platzierung, die alle diese Nebenbedingungen einhält, heißt zulässig.
Ein mögliches Optimierungsziel ist die Minimierung von Verbindungslängen. Dazu sind sogenannte Netze gegeben, die jeweils aus einer Menge von Anschlusspunkten (sogenannten Pins) bestehen. Ein Pins ist entweder in einer bestimmten Position fixiert, oder er liegt auf einem beweglichen Rechteck und wird mit diesem Rechteck mitbewegt. Wenn die Rechtecke platziert sind, haben also auch alle Pins eine Position. Die Länge eines Netzes sei dann die Bounding-Box-Netzlänge, das heißt der halbe Umfang eines kleinsten achsenparallelen Rechtecks, in das alle Pins passen.

## 3 Einführungsaufgabe

Als Teil der Einführungsaufgabe sollen zunächst Routinen geschrieben werden, mit denen sich eine Rechteck-Instanz und eine Platzierung einlesen lassen. Das zugehörige Format wird weiter unten angegeben. Anschließend soll die Platzierung auf Überlappungsfreiheit überprüft werden (in dem oben angegeben Sinn). Dafür soll ein Sweepline-Verfahren implementiert werden. Dabei betrachtet man (bei einem horizontalen Sweepline-Verfahren) nacheinander alle minimalen und maximalen $x$-Koordinaten von Rechtecken und untersucht für die aktuelle Koordinate $x^*$, welche Rechtecke von der Geraden $\{x^*\} \times \mathbb{R}$ geschnitten werden. Damit lässt sich leicht die Überlappungsfeiheit überprüfen. Das Verfahren soll Laufzeit $O(n(k + \log n))$ haben, wobei $n$ die Gesamtzahl der Rechtecke und $k$ die maximale Anzahl von Rechtecken sei, die sich mit einer Gerade der Form $\{x^*\} \times \mathbb{R}$ schneiden (eine Implementierung mit Laufzeit $O(n \log n)$ ist ebenfalls möglich, wird aber nicht gefordert).
Eine Beschreibung, wie das Programm aufgerufen werden soll, steht in Abschnitt 7.

**Spätester Abgabetermin für die Einführungsaufgabe ist der 1.5.2018.** Es wird jedoch empfohlen, die Implementierung der Gesamtaufgabe schon während der Semesterferien weitestgehend abzuschließen, da in der Vorlesungszeit häufig wenig Zeit bleibt.

## 4 Eingabe-Format:

Alle Zahlen, die in der Eingabe vorkommen sind ganze Zahlen, deren Absolutbeträge kleiner als $2^{31}$ sind. Eine gültige Eingabe sieht wie folgt aus:

- Die erste Zeile enthält genau vier Zahlen "$x_{\min}$ $x_{\max}$ $y_{\min}$ $y_{\max}$" mit $x_{\min} < x_{\max}$ und $y_{\min} < y_{\max}$, die eine Rechteckfläche $A = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ definieren.
- Es folgen Zeilen, die aus genau zwei positiven Zahlen bestehen. Jede diese Zeilen kodiert genau ein Rechteck, das platziert werden muss, wobei die erste Zahl die Breite und die zweite Zahl die Höhe des Rechteckes angibt. Die Rechtecke seien in der Reihenfolge, in der sie in der Datei stehen, von 0 bis $n - 1$ nummeriert.
- Dann folgen Zeilen, die mit dem Buchstaben "B" beginnen. Diese enthalten danach genau vier Zahlen "$x_{\min}$ $x_{\max}$ $y_{\min}$ $y_{\max}$" mit $x_{\min} \leq x_{\max}$ und $y_{\min} \leq y_{\max}$, die eine Blockade $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ definieren.
- Die folgenden Zeilen definieren Netze:
  - Die jeweils erste Zeile beginnt mit dem String "Net" und enthält außerdem genau eine nichtnegative Zahl, die das Gewicht des Netzes definiert.

– All weiteren Zeilen bis zum Ende der Datei oder, bis wieder eine Zeile mit dem String "Net" beginnt, definieren die Pins dieses Netzes und enthalten genau drei Zahlen "$i \quad x \quad y$". Es gilt $i \in \{-1, \ldots, n-1\}$, wobei $n$ die Zahl der zu platzierenden Rechtecke sei. Falls $i = -1$ ist, dann definiert der Eintrag einen fixierten Pin an der Position $(x, y)$. Andernfalls definiert der Eintrag einen (beweglichen) Pin auf dem Rechteck $i$. Die Zahlen $x$ und $y$ geben die Koordinaten des Pins relativ zur linken unteren Ecke des Rechtecks an. Wenn also die linke untere Ecke des Rechtecks an die Position $(x^*, y^*)$ gesetzt wird, liegt der Pins an Position $(x^* + x, y^* + y)$.

Beachten Sie, dass die Blockaden und die fixierten Pins nicht im Rechteck $A$ enthalten sein müssen. Ebensowenig müssen bewegliche Pins auf ihrem Rechteck liegen. Außerdem kann ein Netz mehrere fixierte Pins bzw. mehrere Pins auf demselben beweglichen Rechteck enthalten.

**Beispiel:**

```
0 5 1 7
3 2
2 4
B 1 3 5 8
Net 5
0 1 2
-1 3 6
Net 1
1 1 2
0 1 1
```

Diese Eingabe definiert die Grundfläche $A = [0, 5] \times [1, 7]$ und die Rechtecke $r_0$ mit Breite 3 und Höhe 2 und $r_1$ mit Breite 2 und Höhe 4. Das Gebiet $[1, 3] \times [5, 8]$ ist blockiert. Es gibt zwei Netze, eins mit Gewicht 5, das einen Pin auf $r_0$ mit einem fixierten Pin auf Position $(3, 6)$ verbindet, und eins mit Gewicht 1, das einen Pin auf $r_0$ mit einem Pin auf $r_1$ verbindet.

## 5 Ausgabe-Format:

Alle Einträge einer gültigen Ausgabe-Datei sind ganze Zahlen. Wenn $n$ die Zahl der zu platzierenden Rechtecke ist, enthält eine Datei, die eine Platzierung definiert, $n$ Zeilen, die von 0 bis $n-1$ nummeriert sind, wobei die $i$-te Zeile die Platzierung des $i$-ten Rechteckes kodiert. Jede Zeile ist von der Form "$x_{\min} \quad x_{\max} \quad y_{\min} \quad y_{\max} \quad F \quad D$". Das zugehörige Rechteck nimm dann die Fläche $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ ein. Die beiden letzten Zahlen definieren, ob die Orientierung des Rechtecks geändert wurde. Dabei ist $F \in \{0, 1\}$,

wobei $F = 1$ bedeutet, dass das Rechteck entlang einer vertikalen Linie gespiegelt wurde, während $F = 0$ bedeutet, dass das nicht passiert ist. Der letzte Eintrag $D \in \{0, 1, 2, 3\}$ legt fest, dass das Rechteck im Uhrzeigersinn um $D \cdot 90$ gedreht wurde. Bei diesen beiden Einträgen ist zu beachten, dass das Spiegeln vor dem Drehen ausgeführt wird. Setzt man also zum Beispiel $F = 1$ und $D = 1$, dann entspricht das einer Spiegelung entlang der Diagonalen $y = x$. Man beachte, dass sich beim Spiegeln und Drehen eines Rechtecks die Pins auf den Bauteilen mitbewegen.
Die folgenden beiden Zeilen definieren eine Ausgabe, die zu obiger Eingabe passt:

```
0 2 1 4 0 3
3 5 3 7 0 0
```

Diese Platzierung ist zulässig, und die Bounding-Box-Netzlänge beträgt 13. Die Drehung des 0-ten Rechtecks führt dazu, dass sich der bewegliche Pin des Netzes mit Gewicht 5 an der Position $(0, 2)$ befindet. Damit hat das erste Netz Länge 7 und das zweite Länge 6. Die gewichtete Bounding-Box-Netzlänge der Platzierung ist $5 \times 7 + 1 \times 6 = 41$.

# 6 Aufgaben:

Im Folgenden sollen bei den meisten Aufgabenstellungen lokal optimale Platzierungen bestimmt werden. Eine solche Platzierung soll mindestens garantieren, dass es (für eine geeignete Konstante $k$) keine Gruppe von $k$ Rechtecken gibt, deren Verdrehung oder Vertauschung innerhalb der Datenstrukturen eine Verbesserung ergibt. Dabei soll $k$ natürlich möglichst groß gewählt sein, so dass noch eine vernünftige Laufzeit entsteht. Bei einzelnen Problemen werden zusätzliche Bedingungen an lokal optimale Platzierungen gestellt.
Wenn nicht anders angegeben, soll immer das folgende Packungsproblem betrachtet werden: Das Optimierungsziel ist, eine zulässige Platzierung in einem dem Flächeninhalt nach kleinstem Rechteck zu finden (die Netzliste kann dann also ignoriert werden). Blockaden sollen ignoriert werden, und das Drehen von Rechtecken soll erlaubt sein. Das Programm soll auch so aufgerufen werden können, dass es eine Platzierung innerhalb von $A$ berechnet oder entscheidet, dass es keine solche gibt.
Die Zahl der zu platzierenden Rechtecke sei immer mit $n$ bezeichnet.
Wenn statt dessen das Netzlängen-Minimierungs-Problem zu lösen ist, dann ist, wenn die relativen Positionen der Rechtecke festliegen, das Duale eines Min-Cost-Flow-Problems zu lösen (siehe Funke et al. [2012]), um eine zu den Nebenbedingungen passende Platzierung mit kleinster Netzlänge zu bestimmen. Wendet man dazu den Sukzessive-Kürzeste-Wege-Algorithmus an, kann man dieses Flussproblem in Laufzeit $O(n^3 + Wn^2)$ lösen, wobei $W$ die Summe aller Netzgewichte sei und wir annehmen, dass die Zahl der Netze und der Pins ebenfalls $O(n)$ ist.

**Darstellungen kompaktifizierter Platzierungen:**

1. **O-Trees** Jan Polster
   Implementieren Sie die O-Tree-Datenstruktur (siehe Guo et al. [1999] und Kapitel 11.2 aus Alpert et al. [2009]). Ihr Programm soll in der Lage sein, zu einer gegebenen kompaktifizierten Rechteck-Platzierung einen O-Tree zu berechnen und umgekehrt zu einem O-Tree eine kompaktifizierte Rechteck-Platzierung zu bestimmen. Beide Schritte sollen nur lineare Laufzeit haben. Schreiben Sie ein Programm, das das Packungsproblem durch vollständige Enumeration von O-Trees optimal löst. Schreiben Sie außerdem ein Programm, das eine lokal optimale Lösung liefert. Zu dieser sollen auch keine Verbesserungen dadurch möglich sein, dass ein Knoten entfernt und (mit der Notation aus Kapitel 11.2 aus Alpert et al. [2009]) auf eine externe Position verschoben wird.

2. **B*-Trees** Kuangda Konrad Zou
   Implementieren Sie die B*-Tree-Datenstruktur (siehe Chang et al. [2000] und Kapitel 11.3 aus Alpert et al. [2009]). Ihr Programm soll in der Lage sein, zu einer gegebenen kompaktifizierten Rechteck-Platzierung einen B*-Tree zu berechnen und umgekehrt zu einem B*-Tree eine kompaktifizierte Rechteck-Platzierung zu bestimmen. Beide Schritte sollen nur lineare Laufzeit haben. Schreiben Sie ein Programm, das das Packungsproblem durch vollständige Enumeration von B*-Trees optimal löst. Schreiben Sie außerdem ein Programm, das eine lokal optimale Lösung liefert. Zu dieser sollen es auch keine Verbesserungen geben, die dadurch entstehen, dass ein Knoten an eine andere Baumposition verschoben wird.

3. **Corner Sequence** Alexander Holz
   Implementieren Sie die Corner-Sequence-Datenstruktur (siehe Lin et al. [2003] und Kapitel 11.4 aus Alpert et al. [2009]). Ihr Programm soll in der Lage sein, zu einer gegebenen kompaktifizierten Rechteck-Platzierung eine Corner Sequence zu berechnen und umgekehrt zu einer Corner Sequence eine kompaktifizierte Rechteck-Platzierung zu bestimmen. Beide Schritte sollen nur lineare Laufzeit haben. Schreiben Sie ein Programm, das das Packungsproblem durch vollständige Enumeration von Corner Sequences optimal löst. Schreiben Sie außerdem ein Programm, das eine lokal optimale Lösung liefert. Für diese soll auch keine Verbesserung mehr möglich sein, die dadurch entsteht, dass ein Term der Sequenz an eine andere Stelle verschoben wird.

**Darstellungen allgemeiner Platzierungen:**

4. **Sequence Pairs** (Rechteckpackung) Philipp Holzmann
   Implementieren Sie die Sequence-Pair-Datenstruktur (siehe Murata et al. [1995] und

Kapitel 11.5 aus Alpert et al. [2009]). Zu einer gegebenen Sequence-Pair-Darstellung sollen Sie in Laufzeit $O(n^2)$ eine kompaktifizierte Platzierung finden. Umgekehrt sollen Sie zu einer zulässigen Platzierung in Laufzeit $O(n \log n)$ eine Sequence-Pair-Darstellung finden. Schreiben Sie ein Programm, das das Packungsproblem durch vollständige Enumeration von Sequence Pairs optimal löst. Schreiben Sie außerdem ein Programm, das eine lokal optimale Lösung liefert. Auch eine Vertauschung innerhalb einer der beiden Reihenfolgen soll dabei keine Verbesserung erzeugen können.

5. **Sequence Pairs** (Netzlänge) Tobias Bork
   Implementieren Sie die Sequence-Pair-Datenstruktur (siehe Murata et al. [1995] und Kapitel 11.5 aus Alpert et al. [2009]). Das Ziel ist es in dieser Aufgabe, eine Platzierung mit möglichst kurzer Netzlänge zu finden.

   Benutzen Sie ihre Darstellung, um mittels vollständiger Enumeration eine optimale Lösung für das Netzlängen-Minimierungs-Problem zu suchen. Für die Umrechung von einer Sequence-Pair-Darstellung in eine dazu passende Platzierung mit kleinster Netzlänge reicht eine Laufzeit von $O(n^3 + Wn^2)$ aus (siehe oben).

6. **Sequence Pairs** (Zweiergruppe) Nicholas Schwab und Lukas Kempf
   Implementieren Sie die Sequence-Pair-Datenstruktur (siehe Murata et al. [1995] und Kapitel 11.5 aus Alpert et al. [2009]). Hier soll sowohl das Packungs- als auch das Netzlängen-Minimierungs-Problem gelöst werden.

   Zu einer gegebenen Sequence-Pair-Darstellung sollen Sie in Laufzeit $O(n \log n)$ (siehe Tang und Wong [2001] für diese schnellere Umrechnung) eine kompaktifizierte Platzierung finden. Umgekehrt sollen Sie zu einer zulässigen Platzierung in Laufzeit $O(n \log n)$ eine Sequence-Pair-Darstellung finden. Schreiben Sie ein Programm, das das Packungsproblem durch vollständige Enumeration von Sequence Pairs optimal löst. Schreiben Sie außerdem ein Programm, das eine lokal optimale Lösung liefert. Auch hier soll eine Vertauschung innerhalb einer der beiden Reihenfolgen dabei keine Verbesserung mehr erzeugen können.

   Benutzen Sie ihre Sequence-Pair-Datenstruktur auch, um mittels vollständiger Enumeration eine optimale Lösung für das Netzlängen-Minimierungs-Problem zu suchen. Für die Umrechung von einer Sequence-Pair-Darstellung in eine dazu passende Platzierung mit kleinster Netzlänge reicht eine Laufzeit von $O(n^3 + Wn^2)$ aus (siehe oben).

7. **Bounded-Sliceline Grid** (Rechteckpackung) Eva Gebertz
   Implementieren Sie die Bounded-Sliceline-Grid-Datenstruktur (siehe Nakatake et al. [1996] Kapitel 11.6 aus Alpert et al. [2009]). Zu einer gegebenen Bounded-Sliceline-Grid-Darstellung sollen Sie in Laufzeit $O(n^2)$ eine kompaktifizierte Platzierung

finden. Schreiben Sie ein Programm, das das Packungsproblem durch vollständige Enumeration von Bounded-Sliceline Grids optimal löst. Schreiben Sie außerdem ein Programm, das eine lokal optimale Lösung liefert.

8. **Bounded-Sliceline Grid** (Netzlänge) Paul Degenhardt
Implementieren Sie die Bounded-Sliceline-Grid-Datenstruktur (siehe Nakatake et al. [1996] Kapitel 11.6 aus Alpert et al. [2009]). Benutzen Sie ihre Darstellung, um mittels vollständiger Enumeration eine optimale Lösung für das Netzlängen-Minimierungs-Problem zu suchen. Für die Umrechung von einer Bounded-Sliceline-Grid-Darstellung in eine dazu passende Platzierung mit kleinster Netzlänge reicht eine Laufzeit von $O(n^3 + Wn^2)$ aus (siehe oben).

9. **Transitive Closure Graph** (Rechteckpackung) Maxim Shevchishin
Implementieren Sie die Transitive-Closure-Graph-Datenstruktur (siehe Lin und Chang [2005] und Kapitel 11.7 aus Alpert et al. [2009]). Zu einer gegebenen Transitive-Closure-Graph-Darstellung sollen Sie in Laufzeit $O(n^2)$ eine kompaktifizierte Platzierung finden. Umgekehrt sollen Sie zu einer kompaktifizierten Platzierung in Zeit $O(n^2)$ eine Transitive-Closure-Graph-Darstellung berechnen. Schreiben Sie ein Programm, das das Packungsproblem durch vollständige Enumeration von Transitive-Closure-Graphs optimal löst. Schreiben Sie außerdem ein Programm, das eine lokal optimale Lösung liefert. Diese soll so gestaltet sein, dass das Umkehren eine "reduction edge" keine Verbesserung mehr erzielen kann.

10. **Transitive Closure Graph** (Netzlänge)
Implementieren Sie die Transitive-Closure-Graph-Datenstruktur (siehe Lin und Chang [2005] und Kapitel 11.7 aus Alpert et al. [2009]). Benutzen Sie ihre Darstellung, um mittels vollständiger Enumeration eine optimale Lösung für das Netzlängen-Minimierungs-Problem zu suchen. Für die Umrechung von einer Transitive-Closure-Graph-Darstellung in eine dazu passende Platzierung mit kleinster Netzlänge reicht eine Laufzeit von $O(n^3 + Wn^2)$ aus (siehe oben).

11. **Adjacent Constraint Graph** (Rechteckpackung)
Implementieren Sie die Adjacent-Constraint-Graph (siehe Zhou und Wang [2004] und Kapitel 11.9 aus Alpert et al. [2009]). Zu einer gegebenen Adjacent-Constraint-Graph-Darstellung sollen Sie in Laufzeit $O(n^2)$ eine kompaktifizierte Platzierung finden. Schreiben Sie ein Programm, das das Packungsproblem durch vollständige Enumeration von Adjacent-Constraint-Graphen optimal löst. Schreiben Sie außerdem ein Programm, das eine lokal optimale Lösung liefert.

12. **Meta-CSP** (Rechteckpackung) Josefine Foos
Implementieren Sie den Backtracking-Algorithmus aus Moffitt und Pollack [2006] unter Verwendung der dort angegebenen Pruning-Methoden. Bei dieser Aufgabe

soll nur entschieden werden, ob die gegeben Rechteck in $A$ zulässig platziert werden können (und falls ja, soll natürlich auch so eine Platzierung berechnet werden).

13. **Meta-CSP** (Netzlänge) Lars Friederichs
    Modifizieren Sie den Backtracking-Algorithmus aus Moffitt und Pollack [2006] so dass er alle möglichen Platzierungen innerhalb von $A$ berechnet (siehe Funke et al. [2012]). Berechnen Sie auf diesem Wege eine Platzierung mit minimaler Netzlänge.

14. **Meta-CSP** (Zweiergruppe)
    Modifizieren Sie den Backtracking-Algorithmus aus Moffitt und Pollack [2006] so dass er alle möglichen Platzierungen innerhalb von $A$ berechnet (siehe Funke et al. [2012]). Berechnen Sie auf diesem Wege eine Platzierung mit minimaler Netzlänge. Zusätzlich soll das Programm in der Lage sein, Blockaden zu berücksichtigen. Außerdem soll es eine lokale Suchfunktion enthalten, bei der (in einer bereits zulässigen Platzierung) kleine Gruppen von Rechtecken beweglich gemacht werden und optimal plaziert werden (wobei die verbleibenden Rechtecke als Blockaden betrachtet werden).

**Das späteste Abgabedatum für die Programmieraufgabe ist der 15.7.2018.**

# 7 Programmaufruf

Ihr Programm soll per Kommandozeile aufgerufen werden. Dem Programm wird als Parameter der Name einer Instanz übergeben. Bei der Einarbeitungsaufgabe (und nur bei dieser) gibt es außerdem noch einen zweiten Parameter, welcher der Name einer Datei ist, die eine Platzierung (in dem Format aus Abschnitt 5) kodiert. Man beachte, dass für die Einführungaufgabe eigentlich nur die zweite Datei gelesen werden müsste, trotzdem empfiehlt es sich, auch schon die erste Datei zu lesen.
Ein Programmaufruf für die Einführungsaufgabe hat also die Form

```
<programmname> <name_eingabe_datei> <name_platzierungs_datei>
```

Für alle anderen Aufgaben lautet der Aufruf:

```
<programmname> <name_eingabe_datei>
```

# Literatur

Alpert, C., Mehta, D. und Sapatnekar, S. [2009]: *Handbook of Algorithms for Physical Design Automation.* CRC Press, 2009.

Chang, Y.-C., Chang, Y.-W., Wu, G.-M. und Wu, S.-W. [2000]: *$B^*$-trees: a new representation for non-slicing floorplans* DAC, 2000, 268–273.

Funke, J., Hougardy, S. und Schneider, J. [2012]: *Wirelength optimal rectangle packings* BPPC, 2012.

Guo, P.-N., Cheng, C.-K. und Yoshimora, T. [1999]: *An O-tree representation of non-slicing floorplan and its applications* DAC, 1999, 268–273.

Lin, J.-M., Chang, Y.-W. und Lin, S.-P. [2003]: *Corner sequence: A P-admissible floorplan representation with a worst case linear-time packing scheme* Transactions on VLSI Systems, 2003, 11, 4, 679–686.

Lin, J.-M. und Chang, Y.-W. [2005]: *TCG: a transitive closure graph-based representation for non-slicing floorplans* Transactions on VLSI Systems, 2005, 13, 2, 288–292.

Moffitt, M., und Pollack, M. [2006]: *Optimal rectangle packing: a meta-csp approach.* 2006.

Murata, H., Fujiyoshi, K., Nakatake, S. und Kajatani, Y. [1995]: *Rectangle-packing-based module placement* ICCAD, 1995, 472–479.

Nakatake, S., Fujiyoshi, K., Murata, H. und Kajitami, Y. [1996]: *Module placement on BSG-structure and IC layout applications* ICCAD, 1996, 484–491.

Tang, X. und Wong, D. [2001]: *FAST-SP: a fast algorithm for block placement based on sequence pair* ASP-DAC, 2001, 521–526.

Zhou, H. und Wang, J. [2004]: *ACG–Adjacent constraint graph for general floorplans* ICCAD, 2004, 572–575.