

DIPLOMARBEIT
Layout chemischer Strukturformeln

Jasper Möller

moellerj@inf.uni-konstanz.de

Universität Konstanz
FB Mathematik und Statistik

Betreuer: Prof Dr. U. Brandes

Konstanz, den 5. Dezember 2004

Urhebervermerk

Ich versichere, dass ich die vorliegende Arbeit selbständig angefertigt habe und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Konstanz, den 5. Dezember 2004

Jasper Möller

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	1
1.1.1	Wozu überhaupt <i>automatische Erzeugung von Layouts</i> ?	1
1.1.2	Wieso verwendet man nicht ein etabliertes Verfahren zum Zeichnen von Graphen?	1
1.1.3	Damit haben sich doch bestimmt schon viele befasst?	2
1.2	Ziele der Arbeit	2
2	Problemstellung	3
2.1	Definitionen	3
2.1.1	Strukturanalyse von Ringsystemen	3
2.1.2	Layouts	5
2.2	Präzisierung der Problemstellung	10
2.2.1	Allgemeine Anforderungen an das Layout	10
2.2.2	Anforderungen an den Layouter	10
2.2.3	Weitere Features	12
2.2.4	Ziele dieser Arbeit	13
2.3	Zur Komplexität uniformer Layoutalgorithmen	13
3	Bisherige Ansätze	15
3.1	Verfahren von Shelley	15
3.1.1	Strukturanalyse der Ringsysteme	15
3.1.2	Layout der Ringsysteme	16
3.1.3	Generierung absoluter Koordinaten	17
3.1.4	Feinlayout	17
3.1.5	Weitere Feature-Erkennung und Ausgabe	18
3.1.6	Bewertung des Verfahrens	18
3.2	Das Verfahren von Boissonnat et al.	18
3.2.1	Idee	18
3.2.2	Layoutalgorithmus	18
3.2.3	Bewertung	19

4	Untersuchte Teilaspekte	21
4.1	Eigenschaften der untersuchten Graphen	21
4.1.1	Einfache Kenngrößen	21
4.1.2	Zusammenhangseigenschaften	22
4.1.3	Planaritätseigenschaften	22
4.1.4	Komplexität der Ringsysteme	22
4.2	Fazit	22
5	Blocklayout	27
5.1	Eigenschaften außenplanarer Graphen	27
5.1.1	Test auf Außenplanarität	29
5.1.2	Zykelbasen außenplanarer Graphen	29
5.1.3	Bestimmung der minimalen Zykelbasis	31
5.2	Layoutalgorithmen für außenplanare Blöcke	34
5.2.1	Uniformes Layout	34
5.3	Nichtuniforme Layouts und Winkelmaximierung	35
5.3.1	Exkurs: Winkelmaximierung in planaren Graphen	35
5.3.2	Kreuzungserkennung bei außenplanaren Graphen	38
5.3.3	Einfache Verfahren zur Maximierung der Winkelauflösung	40
5.3.4	Einbettungserhaltender Spring Embedder	40
5.4	Fazit und Auswertung	45
5.5	Optimierungen und Ausblicke	46
6	Baumlayout	49
6.1	Definitionen und allgemeine Designkriterien	49
6.2	Hierarchische Layoutalgorithmen	50
6.2.1	Lagenlayout	50
6.2.2	Radiallayout	51
6.2.3	Bewertung	52
6.3	Ballon-Layout	53
6.4	Neuer Ansatz: Sektoroptimierungen	56
6.4.1	Initialisierung	60
6.4.2	Sektoroptimierung	61
6.4.3	Koordinatengenerierung und gesamter Algorithmus	66
6.4.4	Optimierungen	67
6.5	Experimente	72
6.6	Fazit	74

7	Gesamtlayout	83
7.1	Grundgerüst	83
7.2	Bestimmung der Sektoren für Schnittknoten und Blöcke	87
7.2.1	Nur eine Aus-/Eintrittskante, streng monoton wachsende Rotationswinkel, jeder Schnittknoten ist eine Ecke der konvexen Hülle	88
7.2.2	Andere Fälle	94
7.2.3	Blöcke in Spiro-Anordnung	97
7.3	Fazit	97
8	Zusammenfassung und Ausblicke	99
	Die beiliegende CD	101

(liegt nur der Papierausgabe in der Universitätsbibliothek Konstanz bei)

Kapitel 1

Einleitung

1.1 Motivation

Diese Arbeit befasst sich mit Problemen, die bei der automatischen Erzeugung von Layout chemischer Strukturformeln auftreten. Natürlich ist die Frage berechtigt, was an diesem Thema so speziell ist, dass man darüber soviel Worte verlieren muss. Vor dem eigentlichen Einstieg in die Materie soll daher eine kleine informelle Übersicht über den Themenkreis stehen.

1.1.1 Wozu überhaupt *automatische Erzeugung von Layouts*?

Steht der Chemiker nicht sowieso den ganzen Tag im Labor?

Ich persönlich vermute eher, dass ein Chemiker heutzutage mehr Zeit vor dem Computer als vor dem Abzug verbringt. Wie in den meisten Naturwissenschaften, so ist auch in der Chemie der Forscher oft mit einer Unmenge an Daten konfrontiert, die ausgewertet, insbesondere auch klassifiziert werden müssen. Meistens haben die Substanzen noch nicht einmal einen systematischen Namen, sondern liegen lediglich als *connection tables* (das ist nichts anderes als eine Repräsentation eines Graphen durch Adjazenzmatrizen) vor. Viele Substanzen unterscheiden sich nur an wenigen Stellen, und man ist oft daran interessiert, wie sich die Eigenschaften einer Substanz durch gezielte Manipulation an einer einzigen Stelle verändern. Daher kommt einer konsistenten Darstellung der zugrundeliegenden Graphen immer noch eine entscheidende Rolle zu, da man derartige Klassifikationen möglichst schnell durch Ansehen der Struktur erledigen möchte. Daneben wird ein Großteil der Zeichnungen in wissenschaftlichen Veröffentlichungen immer noch von Hand oder bestenfalls halbautomatisch erstellt, was viel Zeit bei oft nicht zufriedenstellenden Ergebnissen beansprucht.

1.1.2 Wieso verwendet man nicht ein etabliertes Verfahren zum Zeichnen von Graphen?

Die zweidimensionale Repräsentation von Strukturformeln ist sozusagen ein halbabstraktes Verfahren. Abstrakt deshalb, weil in Wirklichkeit die Positionen der Atome eben dreidimensional sind. Es handelt sich hier um eine formale Darstellung, die allerdings noch deutliche Bezüge zur Realität aufweist. So werden beispielsweise Ringstrukturen als Vereinigung möglichst regulärer Polygone dargestellt, was der Realität deutlich eher entspricht als die Darstellung durch ein Kreislayout. Wegen der oben kurz angesprochenen Anforderungen hat sich allerdings ein ausgefeilter Formalismus für die Darstellung herausgebildet - einige Beispiele für die Probleme, die dadurch entstehen können, finden sich in den folgenden Kapiteln. Insbesondere sind hier gleichmäßige Winkelaufteilung und einheitliche Kantenlängen, aber auch konsistente Anordnung ähnlicher Strukturen von Bedeutung. Daneben gibt es eine Vielzahl isolierter Regeln, von denen ein kleiner Teil im nächsten

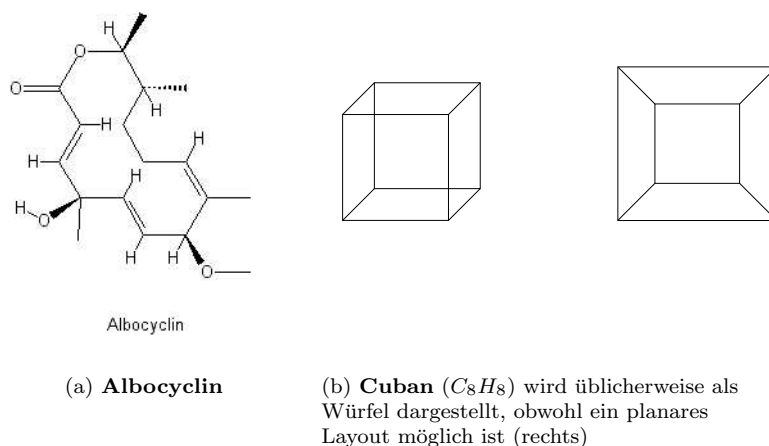


Abbildung 1.1: Unterschiedliche Darstellungen von unterschiedlichen Molekülklassen

Kapitel vorgestellt wird. Oft weichen die Designkriterien von denen ab, die sonst üblicherweise für das Layout von Graphen verwendet werden, oder werden zumindest anders gewichtet.

1.1.3 Damit haben sich doch bestimmt schon viele befasst?

Obwohl die auftretenden Graphen nicht besonders groß und aus graphentheoretischer Sicht meist auch nicht sehr komplex sind, scheint es doch nur wenige Veröffentlichungen und noch weniger reale Implementationen zu diesem Thema zu geben. Dies mag überraschen, da sich sonst Chemie und Graphentheorie durchaus kreativ beeinflussen.¹ Tatsächlich scheinen die meisten Veröffentlichungen zu diesem Thema auch aus der Chemie und nicht der Mathematik oder Informatik zu stammen, woran sicherlich der oben angesprochene Umfang an Detailregeln, die oft einem Nichtchemiker nicht einmal bekannt sind, nicht unschuldig sein dürfte. Zusätzlich decken die auftretenden Substanzen auch eine große Klasse von Graphen ab, für die durchaus unterschiedliche Darstellungen üblich sind, man vergleiche die Darstellungen für Albocyclin (Abb. 1.1(a)) und Cuban (Abb. 1.1(b)), die beide planar einbettbar sind.

1.2 Ziele der Arbeit

In dieser Arbeit sollen konkret zwei verschiedene Teilaspekte des Layouts von chemischen Strukturformeln genauer untersucht werden. Zum einen werden in Kapitel 5 Layoutalgorithmen für zweifache Zusammenhangskomponenten untersucht, hierbei liegt der Schwerpunkt im wesentlichen darauf, für Blöcke, die ein uniformes Layout zulassen, dieses möglichst effizient (idealerweise in Linearzeit) zu bestimmen, und anderenfalls wenigstens ein planares ringtreues Layout mit möglichst guter Winkelauflösung zu finden. In Kapitel 6 wird dann das Layout von Bäumen genauer untersucht, wobei im Gegensatz zu den sonst üblichen Layoutalgorithmen gezielt versucht wird, die Winkeluniformität zu gewährleisten. In Kapitel 7 wird schließlich kurz darauf eingegangen, wie man beide Teile kombinieren kann, um einen Layoutalgorithmus für das gesamte Molekül zu erhalten. Zuerst sollen jedoch die verwendeten Begriffe eingeführt und die Problemstellung anhand dieser Definitionen präzisiert werden.

¹So haben viele Untersuchungen zur Graphisomorphie oder zur Analyse von Zykelbasen ihren Ausgangspunkt in chemischen Fragestellungen.

Kapitel 2

Problemstellung

2.1 Definitionen

Definition 2.1 (Molekül). Ein **Molekül** ist ein zusammenhängender schlichter ungerichteter Graph $G = (V, E)$ mit Knoten- und Kantenlabels. Die Knoten heißen auch **Atome**, die Kanten **Bindungen**.

Konvention. Im folgenden haben alle Graphen diese Eigenschaften, sind also insbesondere stets zusammenhängend, und die Begriffe Molekül/Graph usw. sind beliebig austauschbar.

Bemerkung 2.2. Obwohl in einem Molekül auch Mehrfachbindungen auftreten können, wird der Graph trotzdem nicht als Multigraph betrachtet, da diese Mehrfachbindungen für das Layout wie Einfachbindungen behandelt werden.

Bemerkung 2.3. Da in den meisten Strukturdatenbanken auch Substanzen vorkommen, deren Graph nicht zusammenhängend ist, müsste man für eine allgemeinere Definition eigentlich auf den Graphzusammenhang verzichten. Da aber im folgenden stets die Komponenten getrennt behandelt werden können, wurde hier diese Eigenschaft gleich mit verlangt.

Es folgen einige Definitionen und Folgerungen, um die Problemstellung später präzisieren zu können. Die Beweise für die meisten Sätze, sofern nicht offensichtlich, kann man beispielsweise [Har74], [Vol96], [LS97] entnehmen, so dass hier auf eine Wiederholung verzichtet wurde. Wir beginnen mit Definitionen, die lediglich allgemeine strukturelle Eigenschaften beschreiben:

2.1.1 Strukturanalyse von Ringsystemen

Ein zentrales Thema der chemischen Strukturanalyse ist die Analyse und Beschreibung von Ringstrukturen, da viele organische Substanzen solche Komponenten enthalten.

Definition 2.4 (Zykel). Ein Teilgraph $C = (V_C, E_C)$ von $G = (V, E)$ heißt **Zykel**, falls alle Knoten geraden Grad haben. Ein Zykel heißt **einfach** oder **elementar**, wenn er zusammenhängend ist und alle Knoten Grad zwei haben. $e \in E \setminus E_C$ heißt **Sehne**, wenn e zu zwei Knoten aus C inzident ist. $|C| := |E_C|$ heißt **Länge** von C .

Den Zusammenhang zum Wegebegriff stellt folgender Satz her:

Satz 2.5. *Jeder einfache geschlossene Weg in G ist ein einfacher Zykel, und jeder einfache Zykel liefert bei entsprechender Ordnung der Knoten einen einfachen geschlossenen Weg in G .*

Satz und Definition 2.6 (Zykelraum). Sei \mathcal{C}_G die Menge aller Zyklen von $G = (V, E)$. Dann wird \mathcal{C}_G mit der Addition

$$\oplus : \mathcal{C}_G \times \mathcal{C}_G \rightarrow \mathcal{C}_G, \quad X \oplus Y := (X \cup Y) \setminus (X \cap Y) \quad (2.1a)$$

und der Skalarmultiplikation

$$\cdot : \{0, 1\} \times \mathcal{C}_G \rightarrow \mathcal{C}_G, \quad 1 \cdot X = X, 0 \cdot X = \emptyset \quad (2.1b)$$

zu einem Vektorraum über \mathbb{Z}_2 , dem sogenannten **Zykelraum** $(\mathcal{C}_G, \oplus, \cdot)$. Eine Basis \mathcal{B}_G von \mathcal{C}_G heißt **Zykelbasis**. Es gilt:

$$\mu(G) := \dim \mathcal{C}_G = |V| - |E| + 1 \quad (2.1c)$$

$\mu(G)$ heißt auch **zyklomatische Zahl** von G .

Definition 2.7 (minimale Zykelbasis). Die Länge $l(\mathcal{B}_G)$ einer Zykelbasis ist definiert als:

$$l(\mathcal{B}_G) := \sum_{C \in \mathcal{B}_G} |C| \quad (2.2)$$

Eine Zykelbasis mit minimaler Länge heißt **minimale Zykelbasis**.

Satz 2.8. Jeder Zykel in einer minimalen Zykelbasis ist einfach.

Definition 2.9 (Ring). Ein Zykel heißt **relevant** oder **Ring**, wenn er in einer minimalen Zykelbasis vorkommt. Die Menge aller Ringe von G wird mit \mathcal{R}_G bezeichnet.

Da die Längen aller Zykelbasen nichtnegative ganze Zahlen sind, und jeder Graph mindestens eine Zykelbasis besitzt, ist die Existenz einer minimalen Zykelbasis stets gesichert. Damit folgt unmittelbar:

Korollar 2.10. Ein zusammenhängender Graph besitzt mindestens $\mu(G)$ Ringe.

Beweis. Klar mit Satz 2.6 und Def. 2.9. □

Der folgende Satz liefert eine insbesondere im Zusammenhang mit chemischen Strukturformeln nützliche Charakterisierung von Ringen:

Satz 2.11. Ein Zykel ist genau dann relevant, wenn er sich nicht als Summe von strikt kürzeren Zykeln schreiben lässt.

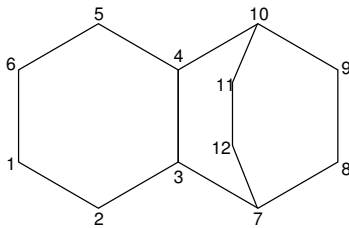


Abbildung 2.1: Nicht eindeutige minimale Zykelbasis

Im Allgemeinen ist eine minimale Zykelbasis nicht eindeutig bestimmt, wie man in Abb. 2.1 sehen kann: Hier ist $\mu(G) = 14 - 12 + 1 = 3$, aber es gilt $\mathcal{R}_G = \{(1, 2, 3, 4, 5, 6), (3, 7, 8, 9, 10, 4), (3, 7, 12, 11, 10, 4), (7, 8, 9, 10, 11, 12)\}$, also $|\mathcal{R}_G| = 4$, so dass nicht alle vier Ringe gleichzeitig in einer minimalen Zykelbasis vorkommen können.

Im Allgemeinen ist also \mathcal{R}_G keine Zykelbasis mehr. Allerdings gilt offenbar

$$\mathcal{R}_G = \bigcup_{\mathcal{B}_G \text{ minimal}} \mathcal{B}_G \quad (2.3)$$

Definition 2.12 (Ringzusammenhangsgraph). Zwei Zyklen heißen **adjacent**, wenn sie mindestens eine Kante gemeinsam haben. Der **Ringzusammenhangsgraph** $R(G) = (V_R, E_R)$ von G ist ein Multigraph, definiert durch:

$$V_R := \mathcal{R}_G \quad (2.4a)$$

$$\{R_1, R_2\} \in E_R \text{ mit Vielfachheit } k \Leftrightarrow R_1 \text{ und } R_2 \text{ haben } k \text{ Kanten gemeinsam.} \quad (2.4b)$$

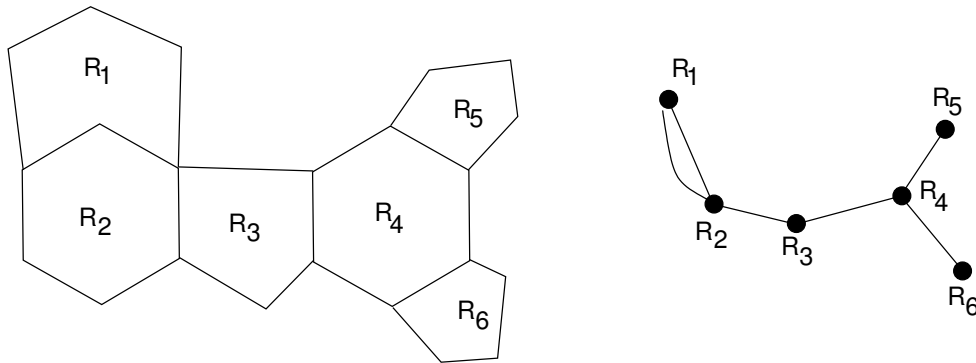


Abbildung 2.2: Beispiel für einen Graph und seinen Ringzusammenhangsgraph. Hier ist die minimale Zykelbasis eindeutig bestimmt

2.1.2 Layouts

Planarität

Da in dieser Arbeit hauptsächlich Zeichnungen von Molekülgraphen behandelt werden sollen, werden jetzt einige Eigenschaften zusammengestellt, die solch eine Zeichnung besitzen kann, und die für die Problemstellung von besonderem Interesse sind. Wir betrachten ausschließlich die Standardrepräsentation, die im folgenden auch einfach Layout genannt wird.¹Ebenso unterscheiden wir, sofern nicht anders erwähnt, nicht zwischen den Elementen eines Graphen und deren Repräsentation in der Ebene.

Definition 2.13 (Planare Einbettung, planarer Graph). Ein Layout eines Graphen heißt **planare Einbettung**, wenn sich keine zwei Kanten (abgesehen von den gemeinsamen Knoten inzidenter Kanten) kreuzen. Ein Graph heißt **planar**, wenn er eine planare Einbettung besitzt. Eine planare Einbettung zerlegt die Ebene in Gebiete, sogenannte **Facetten** F_i . Das (einzige) unbeschränkte Gebiet heißt **äußere Facette** F_0 , die anderen Facetten heißen **innere Facetten**. Jede innere Facette ist einfach zusammenhängend. Der Rand einer Facette F wird mit ∂F bezeichnet. Die Menge aller Facetten in einer planaren Einbettung wird mit \mathcal{F} bezeichnet.

Die durch die planare Einbettung vorgegebene zyklische Ordnung der Kanten um jeden Knoten nennt man **kombinatorische Einbettung**².

Notation:

$$uv \prec_v vw \quad :\Leftrightarrow \quad uv \text{ liegt in der zyklischen Ordnung um } v \text{ direkt vor } vw$$

In diesem Fall sollen uv und vw **direkt benachbart** (bzgl. v) heißen. Ein planarer Graph, bei dem sich keine Kante zwischen (nicht-adjazenten) Knoten einfügen läßt, so dass er planar bleibt, heißt **maximaler planarer Graph**.

Planare Einbettungen sind im Allgemeinen nicht eindeutig bestimmt, wie folgender Satz zeigt:

Satz 2.14. *Jeder zweifach zusammenhängende planare Graph kann so in die Ebene eingebettet werden, dass jede beliebige Facette zur äußeren wird.*

Etwas überraschend ist die folgende Tatsache:

Satz 2.15 (Fáry). *Zu jedem planaren Graphen gibt es eine **geradlinige** planare Einbettung.*

¹Insbesondere werden Kanten stets als Jordan-Kurven zwischen ihren Endpunkten repräsentiert, können sich also nicht selbst kreuzen o.ä.

²Diese ist damit unabhängig vom Verlauf der Kanten, der Größe der Facetten etc.

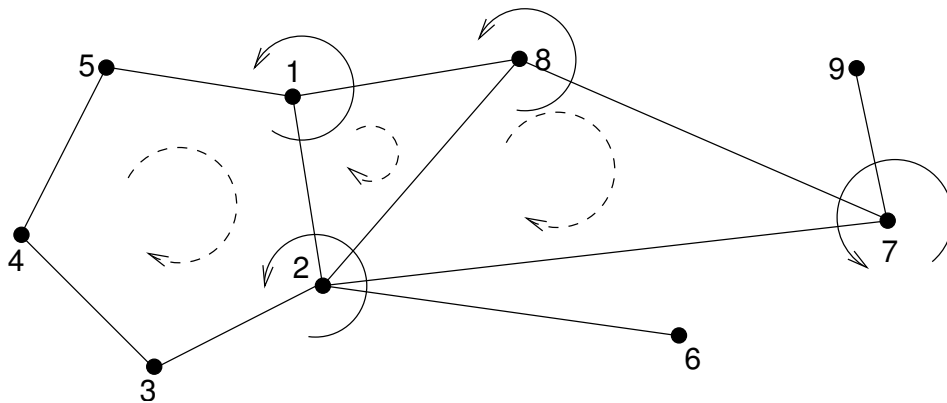


Abbildung 2.3: Beispiel für die gewählte Orientierung, für alle inneren Facetten und höhergradigen Knoten markiert

Aus der Definition der planaren Einbettung und den Eigenschaften der Standardrepräsentation folgt direkt:

Satz 2.16. [Har74] *Jede innere Facette ist einfach zusammenhängend, und ihr Rand bildet einen einfachen Zykel in G . Die Ränder aller inneren Gebiete bilden eine Zykelbasis von G , und damit folgt die **Eulersche Polyederformel** für jede planare Einbettung eines zusammenhängenden planaren Graphen:*

$$|V| - |E| + |\mathcal{F}| = 2 \quad (2.5)$$

Für spätere Anwendungen ist es wichtig, eine einheitliche Durchlaufrichtung für die Ränder der Facetten festzulegen.

Definition 2.17 (Orientierung). Für einen planaren Graphen G sei eine planare Einbettung gegeben. Eine Festlegung der Durchlaufrichtung aller Knoten des Randes einer Facette F heißt **Orientierung** von F . F heißt links (rechts) orientiert, wenn beim Durchlaufen von ∂F das Innere von F stets links (rechts) liegt.

Notation für Knoten $v, w \in \partial F$:

$$v \prec_{\partial F} w \quad :\Leftrightarrow \quad v, w \text{ adjazent und } v \text{ wird vor } w \text{ durchlaufen}$$

Konvention. Damit Orientierung, kombinatorische Einbettung und Winkelmessung im mathematisch positiven Sinne (d.h. gegen den Uhrzeigersinn) zueinander passen, wird im Folgenden stets davon ausgegangen, dass alle Facetten rechts orientiert (evtl. mit Ausnahme der äußeren Facette) sind und die kombinatorische Einbettung durch Umlauf gegen den Uhrzeigersinn gegeben ist (s. Abb. 2.3).

Definition 2.18 ((Maximal) außenplanarer Graph). Ein Graph, der eine geradlinige planare Einbettung besitzt, bei der alle Knoten eine ausgezeichnete Facette beranden, heißt **außenplanar**. Üblicherweise wird diese Facette als äußere Facette dargestellt. Ein außenplanarer Graph, bei dem keine Kante mehr hinzugefügt werden kann, ohne die Außenplanarität zu zerstören, heißt **maximal außenplanar**.

Bemerkung 2.19. Nicht jeder planare Graph ist außenplanar, z.B. ist der K_4 zwar planar, aber nicht außenplanar.

Eine wichtige Eigenschaft (nicht nur) von planaren Layouts ist eine möglichst große **Winkelauflösung**:

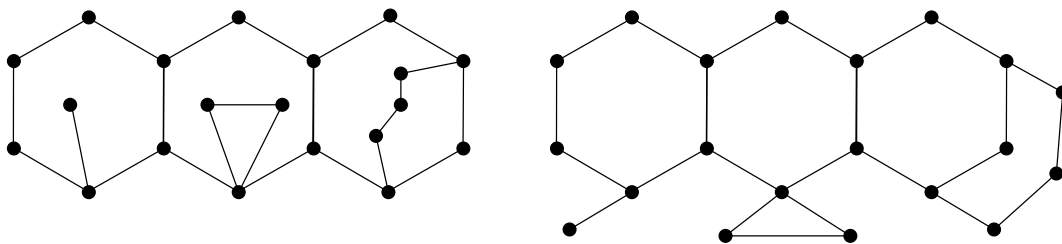


Abbildung 2.4: Links eine nicht ringtreue, rechts eine ringtreue Einbettung desselben Graphen

Definition 2.20 (Winkelauflösung). Gegeben sei ein planarer Graph $G = (V, E)$ mit planarer Einbettung mit der Facettenmenge \mathcal{F} . Sei $x(v, F)$ für $v \in V$, $F \in \mathcal{F}$ der Winkel an Knoten v zur Facette F , dann heißt

$$x_{min} := \min_{\substack{v \in V, F \in \mathcal{F} \\ v \text{ inzident zu } F}} \{x(v, F)\} \quad (2.6)$$

Winkelauflösung der planaren Einbettung.

Bemerkung 2.21. Da je zwei direkt benachbarte Kanten dieselbe Facette beranden, ist dies gleichbedeutend mit:

$$x_{min} := \min_{\substack{v \in V, uv, vw \in E \\ uv \prec_v vw}} \{\angle(uv, vw)\} \quad (2.7)$$

Spezielle Layouts

Wir wollen nun besondere Klassen von Layouts definieren, die prototypisch für Layouts von Molekülgraphen sind. Da wir im folgenden vor allem Ringsysteme zufriedenstellend layouten wollen, erweist sich folgende Definition als nützlich:

Definition 2.22 (Ringtreu). Eine planare Einbettung heißt **ringtreu**, wenn jede innere Facette von einem Ring berandet wird. Eine ringtreue Einbettung heißt **stark ringtreu**, wenn auch jeder Ring eine innere Facette berandet (s. auch Abb. 2.4).

In diesem Fall ist der Ringzusammenhangsgraph von G im wesentlichen dasselbe wie der geometrische Dualgraph (es fehlen nur die Kanten und Knoten für die äußere Facette).

Da die Ränder der inneren Facetten eines planar eingebetteten Graphen eine Zykelbasis bilden, bilden diese im Fall einer ringtreuen Einbettung sogar eine minimale Zykelbasis von G , und es gilt offenbar:

Bemerkung 2.23. Eine ringtreue Einbettung ist genau dann stark ringtreu, wenn die minimale Zykelbasis von G eindeutig bestimmt ist.

Da man im Allgemeinen beim Layout einer Strukturformel die Ringe jeweils als innere Facetten darstellen möchte, ist diese Aussage natürlich erst einmal nicht sehr befriedigend. Es wird sich jedoch (in Kap. 5.1.2) zeigen, dass für eine große Klasse von Molekülen die minimale Zykelbasis tatsächlich eindeutig bestimmt ist.

Man kann sich allgemeiner fragen, ob sich jeder planare Graph so einbetten läßt, dass die Ränder der inneren Facetten einer *vorgegebenen* Zykelbasis entsprechen (dann wäre insbesondere jeder planare Graph ringtreu einbettbar). Dies ist leider nicht der Fall, insbesondere auch nicht für minimale Zykelbasen (s. beispielsweise [LS97]). Daher ist folgende Definition nicht überflüssig:

Definition 2.24 (planare Zykelbasis). Eine Zykelbasis \mathcal{C}_G eines planaren Graphen G , für die es eine Einbettung mit Facettenmenge \mathcal{F} gibt, so dass $\{\partial F \mid F \in \mathcal{F} \setminus \{F_0\}\} = \mathcal{C}$ gilt, heißt **planare Zykelbasis**.

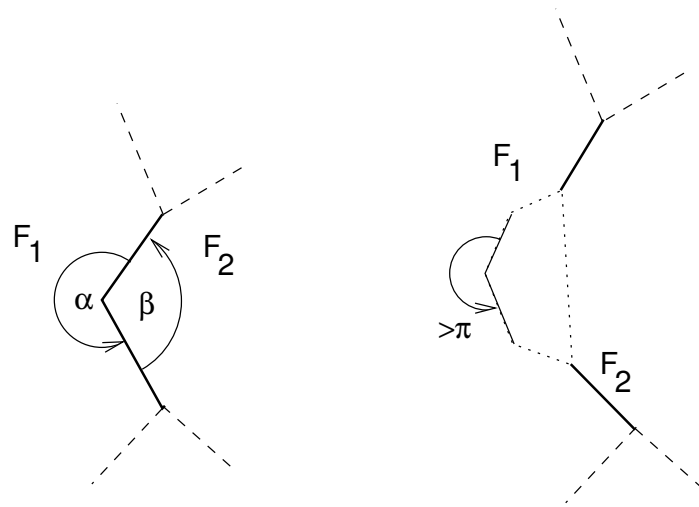


Abbildung 2.6: Konvexität und Facettenadjazenz

Eine erste konkrete Aussage über die *Form* der inneren Facetten enthält die folgende Definition:

Definition 2.25 (Konvexe Einbettung). Eine geradlinige planare Einbettung heißt **konvex**, falls alle inneren Facetten streng konvexe Polygone sind, d.h. an jedem Knoten des Randes der Innenwinkel $< \pi$ ist.

Bemerkung 2.26. Eine üblichere Definition von Konvexität fordert auch Konvexität für den Rand der Einbettung von G , dies soll im Folgenden aber stets explizit gefordert werden.

Bemerkung 2.27. Konvexität von Facetten überträgt sich nicht auf deren Vereinigung (s. beispielsweise zwei adjazente regelmäßige Sechsecke).

Bemerkung 2.28. Eine konvexe Einbettung ist im Allgemeinen nicht ringtreu, wie Abb. 2.5 zeigt:

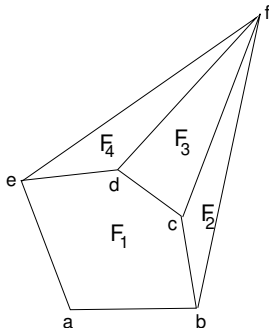


Abbildung 2.5: Nicht ringtreue konvexe Einbettung

Eine notwendige Bedingung für die Konvexität einer Einbettung ist leicht zu sehen:

Lemma 2.29. *In einer konvexen Einbettung haben zwei innere Facetten stets höchstens eine Kante gemeinsam.*

Beweis. s. dazu Abb. 2.6, es können nicht beide Winkel α und β kleiner als π sein. Wegen der Konvexität ist es auch nicht möglich, dass zwei adjazente Facetten zwei nicht aufeinander folgende Kanten gemeinsam haben, da sonst auf dem Rand einer Facette mindestens ein Innenwinkel $> \pi$ auftreten müsste. \square

Da die Ränder aller inneren Facetten einfache Zyklen sind, kann es demnach insbesondere keine konvexe Einbettung geben, wenn je zwei beliebige verschiedene einfache Zyklen eines Graphen mehr als eine Kante paarweise gemeinsam haben. Dies kann man wieder in Abb. 2.6 sehen, wenn man die Kanten cf und df weglässt.

Ein hinreichendes Kriterium liefert:

Satz 2.30 (Tutte). *Jeder dreifach zusammenhängende planare Graph besitzt eine konvexe planare Einbettung.*

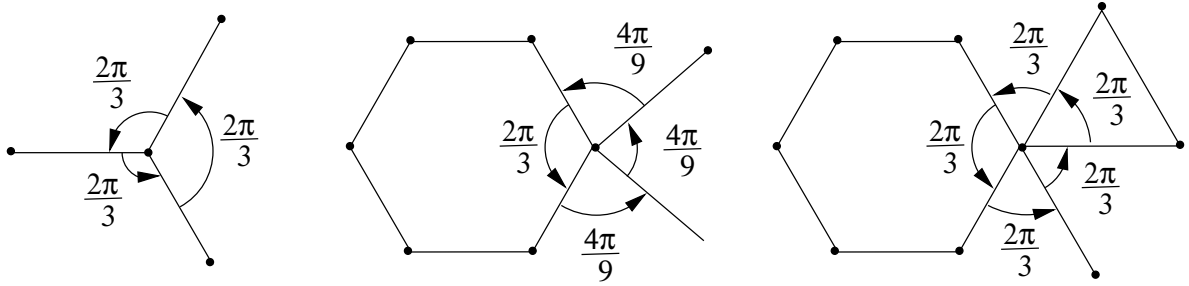


Abbildung 2.7: Winkelzuweisung bei winkeluniformer Einbettung

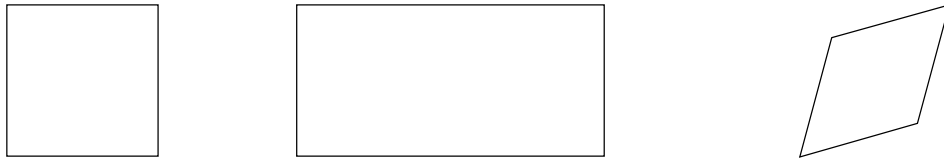


Abbildung 2.8: Uniformes, nur winkeluniformes, nur kantenuniformes Layout einer Facette

Eine solche konvexe Einbettung kann man mittels Schwerpunktlayouts finden, wenn man Knoten des Randes der äußeren Facette in irgendeiner Einbettung auf dem Rand eines konvexen Polygons fixiert.

Definition 2.31 ((Kanten-/Winkel)uniforme Einbettung). Eine geradlinige planare Einbettung heißt **kantenuniform**, falls alle Kanten dieselbe Länge haben.

Eine geradlinige planare Einbettung heißt **winkeluniform**, falls gilt (s. Abb. 2.7):

1. Die Einbettung ist ringtreu
2. Für alle $F \in \mathcal{F} \setminus \{F_0\}$, $v \in V$ gilt:

$$x(v, F) = \pi - \frac{2\pi}{|\partial F|} \quad (2.8a)$$

3. Für alle $v \in V$ gilt: Der Winkel zwischen zwei direkt benachbarten Kanten an v , die nicht zu derselben inneren Facette gehören, ist

$$\delta = \frac{2\pi - \sum_{F \in \mathcal{F}(v)} x(v, F)}{\deg(v) - |\mathcal{R}(v)|} \quad (2.8b)$$

wobei $\mathcal{F}(v) := \{F \in \mathcal{F} \mid v \in \partial F\} \setminus \{F_0\}$.

Eine geradlinige planare Einbettung heißt **uniform**, wenn sie kanten- und winkeluniform ist.

Bemerkung 2.32. Winkeluniforme Einbettungen sind offenbar konvex. Bei uniformen Einbettungen bilden die inneren Facetten sogar regelmäßige Polygone.

Bemerkung 2.33. Kanten- und Winkeluniformität implizieren sich im Allgemeinen nicht gegenseitig (s. Abb. 2.7).

2.2 Präzisierung der Problemstellung

Mit den Begriffen aus dem vorherigen Abschnitt kann nun die Aufgabenstellung präzisiert werden.³

2.2.1 Allgemeine Anforderungen an das Layout

Zuerst soll festgelegt werden, welche Eigenschaften das Layout von Molekülgraphen auf jeden Fall haben sollte:

Problem 2.34 (Layout eines Moleküls). Gegeben sei ein Molekül M . Dann soll ein Layout von M folgende Eigenschaften haben:

1. Das Layout soll uniform sein.
2. Falls dies nicht möglich ist, soll das Layout konvex und ringtreu sein.
3. Falls auch dies nicht möglich ist, soll das Layout wenigstens planar sein.
4. Gibt es keine planare Einbettung, soll wenigstens die Anzahl der Kantenkreuzungen minimal sein.
5. In den Fällen 2. und 3. soll jeweils die Winkelauflösung maximal und die mittlere Abweichung der Kantenlängen minimal sein.
6. Einige spezielle Indikatoren, wie cis-/trans-Anordnungen bei Doppelbindungen müssen korrekt behandelt werden (s. Abb. 2.9(a)).
7. Ketten von C-Atomen sollten gewinkelt mit einem Winkel von 120° dargestellt werden (s. Abb. 2.9(b)).
8. H-Atome werden üblicherweise nicht gezeichnet, wenn sie an ein C-Atom gebunden sind. Falls die Ausgangsdaten solche Atome enthalten⁴, müssen diese für den Zeichenvorgang erkannt und ignoriert werden (s. Abb. 2.9(b)).
9. Das Layout sollte weiteren allgemeinen Anforderungen an ein gutes Layout genügen, insbesondere sollten Knoten nicht zu nahe beieinander liegen, Überlappungen von graphischen Elementen generell vermieden werden, der Platzbedarf gering sein etc..

2.2.2 Anforderungen an den Layouter

Neben den Anforderungen an das Layout gibt es natürlich auch Anforderungen daran, wie ein Zeichenverfahren tatsächlich arbeiten soll:

1. Man wird nicht erwarten können, für jedes beliebige Molekül effizient ein optimales Layout finden zu können, da einige der zugrundeliegenden Probleme \mathcal{NP} -schwer sind. Wünschenswert ist jedoch: Die Klasse von Graphen, für die ein optimales Layout effizient gefunden werden kann, soll klar charakterisiert sein und einen signifikanten Teil der tatsächlich auftretenden Moleküle abdecken.
2. Alternativ kann für kleine Probleminstanzen ein genaues, schlecht skalierendes Verfahren, für große Moleküle dagegen ein weniger genaues, aber dafür schnelles Verfahren verwendet werden.

³Eine Aufzählung weiterer Aspekte sowie deren Begründung aus Sicht des Chemikers kann man z.B. in [IE98] finden, wo auch auf 3D-Repräsentationen und Aspekte der Interaktivität eingegangen wird

⁴In vielen Datenbanken fehlen solche **impliziten** H-Atome komplett, weil ihre Anzahl aus der Restvalenz der anderen beteiligten Atome herleitbar ist

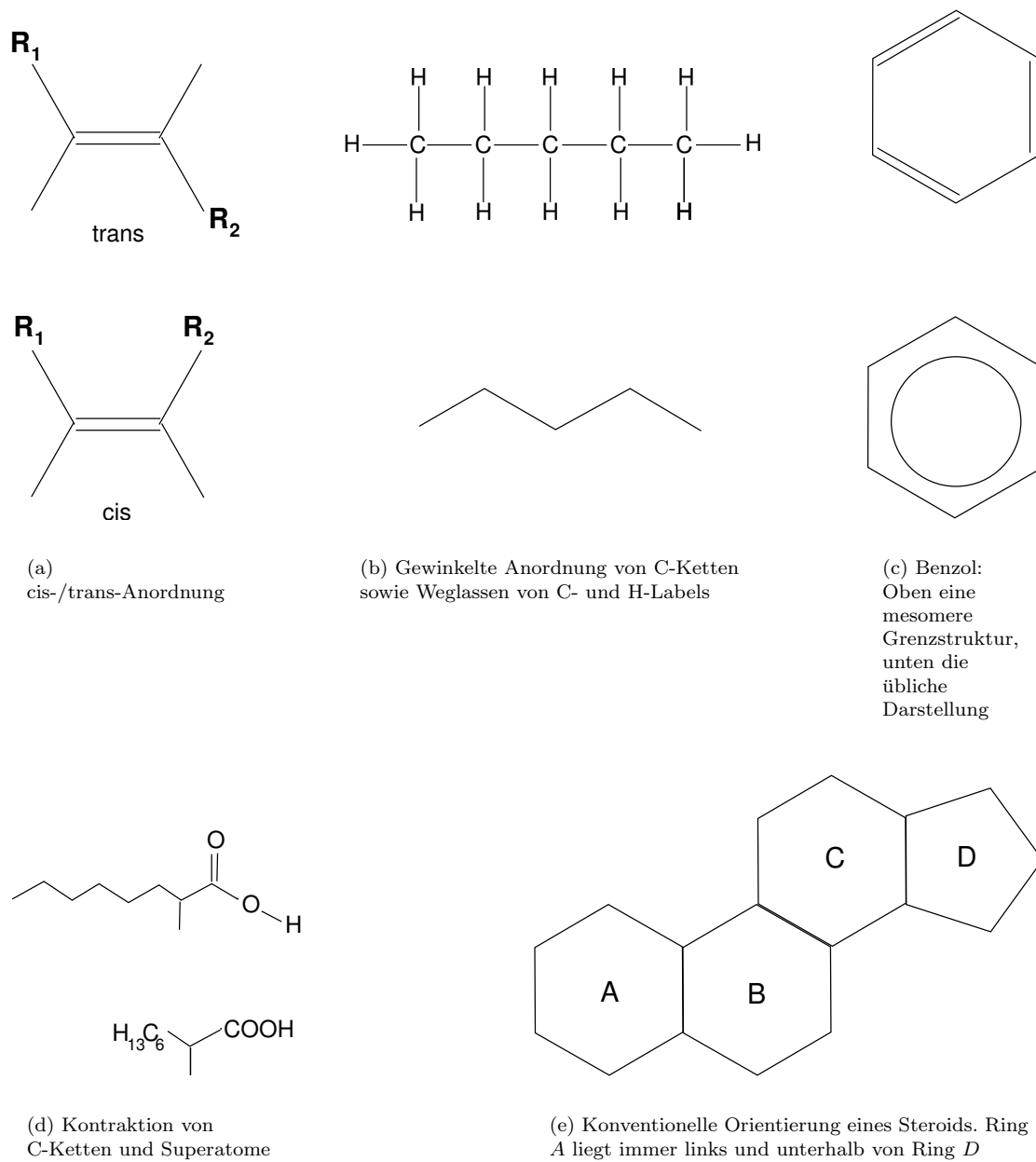


Abbildung 2.9: Verschiedene Teilaspekte beim Layout chemischer Strukturformeln

3. Das Verfahren soll keine Benutzerinteraktion erfordern.
4. Das Verfahren soll möglichst wenig Heuristiken verwenden, die zudem ausschließlich auf strukturellen Eigenschaften der zugrundeliegenden Graphen beruhen.
5. Es sollte leicht möglich sein, weitere Anforderungen (s.u.) in das Verfahren einbauen zu können.

Notwendigerweise sind einige der Anforderungen etwas unpräzise formuliert. Eine der ersten Aufgaben wird es daher sein, eine geeignete Klasse an Graphen zu finden, die viele Moleküle abdeckt. Dies wird in Kapitel 4 geschehen. Zusätzlich sollte es möglich sein, ein darauf optimal arbeitendes Verfahren leicht auf eine größere Klasse von Graphen zu erweitern, ohne zu viel an Effizienz zu verlieren. Einige Ansätze dazu werden in 5 und 6 vorgestellt. Punkt 2. läßt sich vor allem damit begründen, dass es sich bei diesen großen Molekülen typischerweise um Proteine und ähnliche Strukturen handeln wird, für die sowieso die 3-dimensionale Struktur entscheidend ist, so dass man sich mit einem ebenen Layout lediglich einen groben Überblick über die Struktur des Moleküls verschaffen möchte.

2.2.3 Weitere Features

Neben den oben genannten Bedingungen sollten für ein Layout auch noch weitere Bedingungen beachtet werden. Teilweise lassen sich diese Anforderungen jedoch schlecht formalisieren, oder sie müssen nur für bestimmte Anwendungen wirklich beachtet werden. Es ist daher wünschenswert, die Struktur des Layouters so zu gestalten, dass diese Features bei Bedarf einfach mit eingebunden werden können.

1. C-Atome werden im Allgemeinen nicht gezeichnet (s. Abb. 2.9(b)).
2. Zusätzliche Labels wie Atomladungen sollten ebenfalls korrekt gezeichnet werden.
3. Mehrfachbindungen und stereochemische Indikatoren sollten korrekt erkannt und gezeichnet werden.
4. Aromatische Strukturen werden auf eine spezielle Weise dargestellt. Daher sollten solche Strukturen ebenfalls erkannt und korrekt dargestellt werden.(s. Abb. 2.9(c))
5. Manche *funktionalen Gruppen* von Molekülen werden zu sogenannten **Superatomen** zusammengefasst, die ebenfalls wie ein einzelnes Atom zu behandeln sind. Ebenso werden gelegentlich lange C-Ketten zu Superatomen zusammengefasst, was natürlich das Layout unter Umständen stark verändern kann (s. Abb. 2.9(d)).
6. Es sollte eine möglichst große Anzahl an Ein- und Ausgabeformaten beherrscht oder jeweils leicht konvertierbare Formate verwendet werden.
7. Viele Strukturen werden anhand gewisser (gewachsener) Konventionen angeordnet, was ebenfalls berücksichtigt werden sollte(s. Abb. 2.9(e)).
8. Das Verfahren sollte lernfähig sein, d.h. wenn der Benutzer explizit ein anderes Layout vorgibt, sollte dies respektiert werden, und das neue Layout für ähnliche Fälle auch verwendet werden können.

Für viele dieser Anforderungen gibt es schon geeignete Tools, die den tatsächlichen Zeichenvorgang gemäß dieser Konventionen durchführen. Daher wird der Schwerpunkt hier auf die eigentliche Koordinatengenerierung für die Atome gelegt.

2.2.4 Ziele dieser Arbeit

Konkret sollen zwei verschiedene Teilaspekte des Layouts von chemischen Strukturformeln genauer untersucht werden. Zum einen werden in Kapitel 5 Layoutalgorithmen für zweifache Zusammenhangskomponenten untersucht. Hierbei liegt der Schwerpunkt im Wesentlichen darauf, für Blöcke, die ein uniformes Layout zulassen, dieses möglichst effizient (idealerweise in Linearzeit) zu bestimmen, und anderenfalls wenigstens ein planares ringtreues Layout mit möglichst guter Winkelauflösung zu finden. In Kapitel 6 wird dann das Layout von Bäumen genauer untersucht, wobei im Gegensatz zu den sonst üblichen Layoutalgorithmen gezielt versucht wird, die Winkeluniformität zu gewährleisten. In Kapitel 7 werden schließlich beide Teile kombiniert, um einen Layoutalgorithmus für das gesamte Molekül zu erhalten.

2.3 Zur Komplexität uniformer Layoutalgorithmen

Bevor ein Algorithmus entworfen werden soll, ist es sinnvoll, sich über die zu erwartende Komplexität Gedanken zu machen. In unserem Fall ist folgendes Entscheidungsproblem zentral:

Problem 2.35 (Uniform Planar Drawing, UPD).

Instanz: Ein planarer Graph G

Frage: Gibt es ein *uniformes* planares Layout für G ?

Satz 2.36. *UPD ist \mathcal{NP} -schwer.*

Der Verzicht auf Winkeluniformität bringt keine Erleichterung:

Problem 2.37 (Unit Length Planar Straight-line Drawing, ULPGD).

Instanz: Ein planarer Graph G

Frage: Gibt es ein *kantenuniformes* planares Layout für G ?

Satz 2.38 (Battista et al.). *ULPGD ist \mathcal{NP} -schwer.*

Beide Beweise können mit der Methode der sogenannten *logic engine* geführt werden und verlaufen im wesentlichen analog, zu Details s. [BCF01], [BCF00], [EW96] für UPD bzw. [BETT99], 335ff für ULPGD.

Als Konsequenz wird man vermutlich keinen Algorithmus erwarten können, der auf allen Probleminstanzen ein uniformes Layout effizient bestimmt. Es ist daher nötig, die Klasse der untersuchten Graphen einzuschränken und/oder die Problemstellung abzuschwächen. Beides wird ab Kapitel 4 untersucht. Zunächst sollen jedoch einige bekannte Ansätze diskutiert werden.

Kapitel 3

Bisherige Ansätze

In diesem Kapitel sollen die wichtigsten bisher bekannten Verfahren für das Layout chemischer Strukturformeln kurz vorgestellt und bewertet werden. Dabei werden nur Verfahren vorgestellt, die tatsächlich implementiert und in (kommerziellen oder freien) Produkten verwendet werden, und die ein Layout im wesentlichen automatisch erzeugen. Nicht betrachtet werden z.B. die diversen Pakete für \LaTeX (z.B. \PPCHTeX [PPC95]), mit denen nach wie vor ein Großteil der Formeln in wissenschaftlichen Publikationen erstellt wird.

3.1 Verfahren von Shelley

Dieses Verfahren wurde 1983 von Craig A. Shelley entwickelt [She83] und wird (mit einigen Verbesserungen) beispielsweise im Programmpaket *IMAGE* oder im *GIF-Creator* verwendet. Hierbei handelt es sich um ein sechsstufiges Verfahren, bei dem in jedem Schritt verschiedene Heuristiken verwendet werden. Die einzelnen Schritte werden nun im Überblick dargestellt.

3.1.1 Strukturanalyse der Ringsysteme

Im ersten Schritt werden alle Ringe gesucht und der zugehörige Ringzusammenhangsgraph konstruiert, in dem jeder Knoten einem Zykel entspricht und zwischen zwei Knoten genau dann eine Kante besteht, wenn die entsprechenden Zykel mindestens einen **Knoten** gemeinsam haben.¹

Danach werden Codes für alle

- Atome (ATCD)
- Zyklischen Atome (CYATCD)
- Zykel (CYCD)
- Ringsysteme (RSCD)

generiert, wobei diese Codes nicht von der Nummerierung in der Adjazenzmatrix abhängen dürfen. Für den ATCD-Code wird dabei eine Variante der *Morgan-Codierung* [Mor65] verwendet, die in der Chemieinformatik dazu verwendet wird, Ringsysteme eindeutig zu identifizieren. Dieser Code wächst bei Annäherung an das Zentrum, was später bei der Layoutverfeinerung benutzt wird. Die anderen Codes werden dann sukzessive aus dem ATCD erzeugt. Der Sinn dieser Codierung besteht hauptsächlich darin, die Ringsysteme gemäß den üblichen chemischen Konventionen anzuordnen.

¹Beachte: Dies entspricht nicht der (üblichen) Definition aus Kap. 2. Insbesondere sind hier die Ringsysteme, d.h. die Komponenten, die einer Zusammenhangskomponente im Ringzusammenhangsgraph entsprechen, nicht notwendig **zweifach** zusammenhängend.

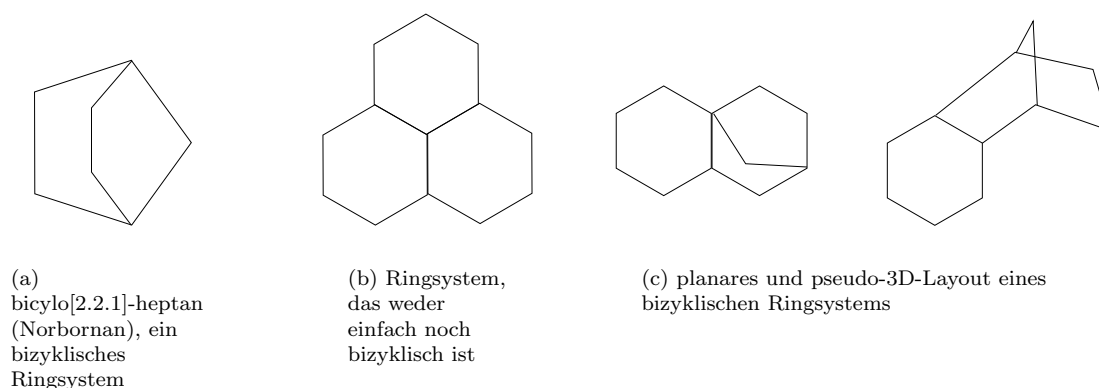


Abbildung 3.1: Zum Verfahren von Shelley

3.1.2 Layout der Ringsysteme

Bestimmung der Komplexität

Um relative Koordinaten für jedes Ringsystem zu generieren, werden diese zunächst nach ihrer Komplexität klassifiziert. Die Definition wurde hier gleich in die Notation aus Kapitel 2 übersetzt:

Definition 3.1 (Komplexität eines Ringsystems). Für ein Ringsystem $G = (V, E)$

$$n = |\mathcal{R}_G| - |V| - |E| - 1 = |\mathcal{R}_G| - \mu(G) \quad (3.1)$$

die **Komplexität** dieses Ringsystems.

Ein Ringsystem mit Komplexität 0 heißt **einfach**.

Definition 3.2 (Bicyklische Ringsysteme). Ein Ringsystem heißt **bicyklisch**, falls gilt:

1. Die Komplexität ist 1.
2. Zwei Zyklen haben mindestens drei aufeinander folgende Atome gemeinsam (eine sogenannte **Brücke**).
3. Das Entfernen einer Brücke liefert einen neuen Zyklus.

Das klassische Beispiel für ein bicyklisches System ist Norbornan C_7H_{12} (s. Abb. 3.1(a))

Komplexere Ringsysteme (s. z.B. Abb. 3.1(b)) werden nach Möglichkeit zu einfachen oder bicyklischen Systemen vereinfacht, indem man sogenannte **strategische Zyklen** schrittweise entfernt. Dabei werden die entfernten Zyklen auf einem Stack abgelegt, um später wieder im Layoutprozeß verwendet werden zu können.²

Generierung relativer Koordinaten

Bei einfachen Ringsystemen ist der Ringzusammenhangsgraph sogar ein Baum. Als Wurzel wird hier der Knoten mit dem höchsten CYCD-Code gewählt, und die Kinder eines Knotens werden nach absteigender CYCD-Codierung geordnet. Es gibt hier zwei für das Layout relevante mögliche Verbindungen von Zykeln:

²Es gibt Systeme, die nicht auf einfache oder bicyklische Systeme reduziert werden können

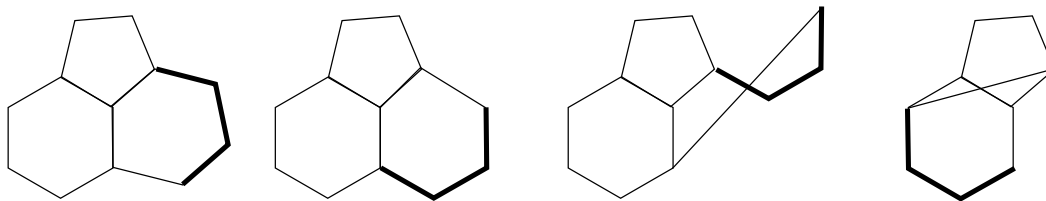


Abbildung 3.2: Layout komplexerer Ringsysteme. Der uniform gezeichnete Teil des dritten Rings ist hervorgehoben. Die beiden ersten Layouts sind beide akzeptabel.

- Fused, d.h. zwei Zyklen teilen sich eine Kante
- Spiro, d.h. zwei Zyklen haben nur genau einen Knoten gemeinsam

Jeder Zyklus wird als reguläres Polygon gelayoutet, wobei der Ringzusammenhangsbaum vom Wurzelknoten ausgehend mittels Breitensuche abgearbeitet wird. Je nach Verbindungstyp werden für das Layout eines neuen Zyklus unterschiedliche Verfahren verwendet.

Bei bicyklischen Systemen verwendet man abhängig von der Konnektivität der Brückenatome ein 2D- oder ein Pseudo-3D-Layout (s. Abb. 3.1(c)).

Bei komplexeren Systemen wird erst der einfache oder bicyklische Kern gelayoutet. Dann werden die entfernten Zyklen der Reihe nach vom Stack entfernt und für deren Atome Koordinaten generiert, und zwar ebenfalls mit einem regulären Polygon als Template. Da es hier im Allgemeinen mehrere Möglichkeiten gibt, den neuen Ring anzufügen, wird für jedes Layout die Energie

$$E = \sum_{v,w \in V} \frac{1}{d(v,w)}$$

berechnet und das Layout mit der geringsten Energie gewählt (s. Abb. 3.2).

3.1.3 Generierung absoluter Koordinaten

Zuerst werden einem Kern absolute Koordinaten zugewiesen (das Atom mit dem höchsten ATCD-Code bzw. das Ringsystem mit dem höchsten RSCD-Code), wobei das Kernsystem evtl. noch rotiert und gespiegelt werden muss, um eine konventionelle Orientierung zu erhalten (dies muss vom Benutzer angegeben werden). Danach werden den übrigen Atomen absolute Koordinaten mittels eines komplexen Energiemodells zugewiesen. Im Unterschied zu einem gewöhnlichen Spring Embedder ist die (anders als gewöhnlich definierte) Potentialfunktion hier diskret, da in jedem Schritt nur diejenigen Positionen betrachtet werden, die in einem uniformen Layout möglich sind. Bei der Abarbeitung werden außerdem stets zuerst die zyklischen Atome und dann erst die azyklischen Atome jeweils in absteigender ATCD-Ordnung betrachtet.

3.1.4 Feinlayout

Da im vorherigen Schritt unter Umständen das Layout noch nicht optimal ist, wird jetzt versucht, die Energie des Layouts zu minimieren (wobei hier eine andere Potentialfunktion verwendet wird). Hierbei wird eine Abwandlung eines Spring Embedders verwendet, wobei nur versucht wird, Gruppen nahe beieinander liegender Atome aufzulösen. Außerdem besteht hier auch die Möglichkeit, die zyklische Ordnung um jedes Atom noch zu ändern.

3.1.5 Weitere Feature-Erkennung und Ausgabe

Bevor das Strukturdiagramm schließlich ausgegeben wird, werden noch zusätzliche Features erkannt und mit in den Zeichenprozeß integriert. Beispielsweise wird versucht, aromatische Strukturen zu erkennen oder Ketten zu kontrahieren.

3.1.6 Bewertung des Verfahrens

Das Verfahren arbeitet relativ schnell und liefert bei den meisten kleinen bis mittleren Strukturen eine ordentliche Ausgabequalität, insbesondere wird meistens die Substanz auch korrekt orientiert.

Allerdings skaliert das Verfahren schlecht mit der Größe der Strukturen. Störend ist die Vielzahl an (graphentheoretisch) schwer zu begründenden Heuristiken, durch die es auch nur schwer möglich ist, Garantien zur Qualität des Layouts zu machen. So ist z.B. nicht klar, ob jede Struktur, die ein zulässiges Layout besitzt, auch so gezeichnet werden kann. Außerdem kann trotz des beträchtlichen Aufwands nicht jede Struktur überhaupt gezeichnet werden. Zudem erfordert das Verfahren an einer Stelle Benutzerinteraktion³.

3.2 Das Verfahren von Boissonnat et al.

3.2.1 Idee

Dieses in [BCF01] beschriebene Verfahren ist eigentlich darauf spezialisiert, Familien von Molekülen konsistent anzuordnen. Allerdings beinhaltet es auch einen kompletten Layouter für einzelne Moleküle, und in Verbindung mit einer Template-Datenbank kann man das Verfahren dazu verwenden, einzelne Moleküle gemäß den üblichen Konventionen anzuordnen. Der Layouter selbst basiert auf einem anderen Ansatz als die bisher beschriebenen Verfahren. Vorausgesetzt wird hier, dass der Molekülgraph planar ist, eine planare Einbettung muss aber nicht schon vorgegeben sein. Grundlegend ist die einfache Beobachtung, dass für ein gegebenes Molekül die Menge aller Layouts mit starren Winkeln und Kantenlängen endlich ist. Man kann also im wesentlichen den Raum der möglichen Layouts mit starren Winkeln und Kantenlängen einfach absuchen, bis ein zulässiges Layout gefunden worden ist.

3.2.2 Layoutalgorithmus

Beginnend bei einem Knoten⁴ wird der Graph in Breitensuche abgearbeitet, indem den jeweils adjazenten Knoten uniforme Koordinaten zugewiesen werden. Blöcke kann man hierbei als Ganzes verarbeiten, weil durch die Zuweisung von Koordinaten an einen weiteren Knoten bereits das gesamte Layout des Blocks festgelegt ist. Wird bei einer Koordinatenzuweisung festgestellt, dass das Layout nicht mehr zulässig ist, wird ein Backtracking zur letzten zulässigen Koordinatenzuweisung durchgeführt, und es werden alternative Koordinaten zugewiesen. Ebenso wird ein Backtracking durchgeführt, wenn für einen Knoten überhaupt kein uniformes Layout der Nachbarn existiert. Für die Auswahl, welche Knoten als nächstes betrachtet werden sollen und welches Layout als nächstes getestet wird, werden zwei Heuristiken eingesetzt:

- Für adjazente Knoten: Größter oder längster Teilbaum
- Für mögliche Layouts: Entwicklung möglichst langer Ketten

³Dieses Problem wird etwas entschärft, indem die Orientierungen gespeichert und wiederverwendet werden können

⁴Dieser ist je nach Anforderung geeignet zu wählen, z.B. als Templateknoten, s. dazu [BCF01]

3.2.3 Bewertung

Der Algorithmus arbeitet meistens schnell und erfüllt die spezielle Aufgabe gut. Durch die intelligente Suchstrategie werden bei den meisten Molekülen wenig Backtracking-Schritte benötigt. Außerdem findet der Algorithmus stets ein zulässiges Layout, wenn ein solches überhaupt existiert.

Nachteilig ist hingegen, dass die Laufzeit im schlimmsten Fall exponentiell mit der Größe des Moleküls wächst. Außerdem gibt es für den Fall, dass überhaupt kein zulässiges Layout existiert, keine alternative Strategie.

Kapitel 4

Untersuchte Teilaspekte

4.1 Eigenschaften der untersuchten Graphen

Bevor Algorithmen für die in 2.2.4 angesprochenen Teilprobleme untersucht werden, wurde versucht, möglichst viel über die strukturellen Eigenschaften möglicher Eingabedaten herauszufinden. Die Definition eines Moleküls (S. 3, 2.1) ist sehr weit gehalten, insbesondere wird Planarität **nicht** vorausgesetzt. Oftmals ist für Sonderfälle aber sowieso eine andere Darstellung üblich als für die *normalen* Strukturformeln.

Was aber ist *normal*? Zur Klärung dieser Frage wurde ein Datensatz vom *National Cancer Institute (NCI)* der USA, der über 250 000 Substanzen enthält, auf verschiedene Kenngrößen hin untersucht¹. Dort sind vor allem Substanzen enthalten, die vom NCI auf Karzinogenität untersucht wurden. Der Nachteil dieses Sets ist, dass bestimmte Klassen von Molekülen dort unterrepräsentiert sind. Es fehlen beispielsweise Proteine und Enzyme; da bei diesen Substanzen aber im allgemeinen die räumliche Anordnung für den visuellen Eindruck entscheidend ist, kann man den Datensatz des NCI als durchaus repräsentativ für die üblicherweise im 2D-Layout dargestellten Substanzen ansehen. Einige Stichproben wurden zusätzlich mit einem Satz von Proteinen durchgeführt.

Der Datensatz des NCI liegt im SDF-Format vor und wurde mit `OpenBabel`[ope] und XSLT nach GraphML[gra] gewandelt, wobei evtl. vorhandene Koordinaten erhalten blieben. Die Proteindaten wurden ebenfalls nach GraphML konvertiert.

4.1.1 Einfache Kenngrößen

Zuerst wurden die Verteilung einiger einfacher Kenngrößen der Graphen bestimmt, nämlich der Knoten- und Kantenanzahlen sowie der minimalen, maximalen und mittleren Grade. Die Verteilungen sind in Abb. 4.1 zu sehen. Man erkennt insbesondere, dass die meisten Graphen relativ klein sind, der Mittelwert beider Größen liegt bei ca. 20, und über 3/4 aller Substanzen haben weniger als 30 Knoten oder Kanten. Zudem sind die maximalen Knotengrade ebenfalls stark beschränkt, und die mittleren Knotengrade liegen größtenteils im Intervall [1.5, 3]. Dies entspricht den Erwartungen, da die Strukturen stark vom Kohlenstoffgerüst dominiert werden, wobei jedes C-Atom maximal Grad 4 haben kann. Die wenigen Ausreißer beim Maximalgrad werden fast ausschließlich durch metallorganische Substanzen verursacht.

¹Der komplette Datensatz kann unter der Adresse http://cactus.nci.nih.gov/Download/NCI_aug00_2D.sdz heruntergeladen werden. Achtung: Die Downloadgröße beträgt 90MB, unkomprimiert werden über 900MB benötigt!

4.1.2 Zusammenhangseigenschaften

Des Weiteren wurden verschiedene Zusammenhangseigenschaften bestimmt: die zyklomatische Zahl und die Anzahlen der Zusammenhangskomponenten, 2-fachen Zusammenhangskomponenten und nicht trivialen 2-fachen Zusammenhangskomponenten. Die Verteilungen sind in Abb. 4.2 zu sehen. Hier fällt einerseits auf, dass es tatsächlich unzusammenhängende Moleküle gibt (wenngleich der Anteil gering ist), und andererseits die Anzahl nichttrivialer Blöcke meistens gering ist. Außerdem wurde die Anzahl der Bäume bestimmt, wobei lediglich 8,298% der Substanzen tatsächlich Bäume sind.

4.1.3 Planaritätseigenschaften

Zusätzlich wurde untersucht, wieviel Graphen planar oder sogar außenplanar sind. Die Planarität wurde mit dem in der Bibliothek `yFiles`[yfi] implementierten Algorithmus, die Außenplanarität mit dem in Kapitel 5 genauer vorgestellten Algorithmus 1 getestet. Hierbei wurde festgestellt, dass lediglich 20 Substanzen im Datensatz des NCI nicht planar einbettbar sind, also über 99,9% planar sind. Außerdem sind auch fast 95% der Substanzen sogar außenplanar.

4.1.4 Komplexität der Ringsysteme

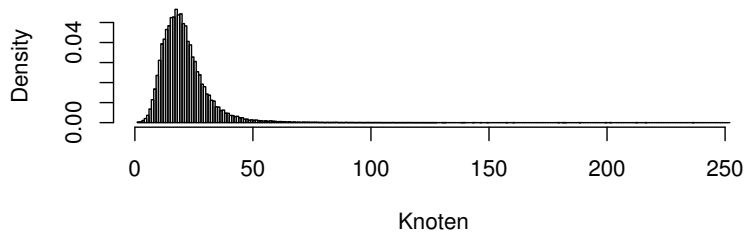
Zum Abschluss wurde für alle außenplanaren Graphen mit den Algorithmen aus Kapitel 5 bestimmt, wie groß die Zykelbasen der nichttrivialen Blöcke sind, sowie die minimale und maximale Größe der Ringe in einem Block ermittelt. Insgesamt wurden dabei 399 404 Blöcke analysiert. Die Verteilungen sind in Abb. 4.3 zu sehen.

Auch hier fällt die im Allgemeinen geringe Komplexität der Ringsysteme auf, die zum einen normalerweise wenig Ringe enthalten, und andererseits meistens Ringe der Größe 5-6 enthalten.

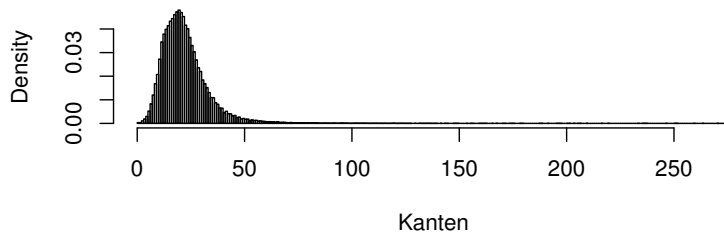
4.2 Fazit

Es treten offenbar (im Gegensatz zur Definition) auch unzusammenhängende Graphen auf. Da man aber bei diesen die Einzelkomponenten getrennt zeichnen kann, und in den meisten Fällen alle Komponenten bis auf eine trivial sind, stellt dies nicht wirklich ein Problem dar. Weiter erkennt man, dass die Komplexität der Graphen recht gering ist: Der Knotengrad ist typischerweise stark begrenzt und der Durchschnittsgrad liegt im allgemeinen zwischen 2 und 4. Graphen, die keine Bäume sind, bestehen aus wenig nichttrivialen Blöcken, die zudem meistens nicht sehr groß sind, und deren Zykelbasis ebenfalls klein ist - die Ringsysteme sind also nicht übermäßig komplex. Bezüglich der Einbettbarkeit sieht man zuerst, dass fast alle Graphen planar sind, was in Anbetracht der geringen Komplexität nicht besonders überraschend ist. Interessanter ist jedoch, dass ca. 95% der Graphen sogar außenplanar sind. Wie man im folgenden Kapitel sieht, haben außenplanare Blöcke einige sehr angenehme strukturelle Eigenschaften. Zur Vereinfachung der Fragestellung wird daher in den folgenden Kapiteln angenommen, dass die Graphen außenplanar sind. Auch bei dieser Klasse von Graphen sind im allgemeinen die Ringsysteme wenig komplex.

Relative Häufigkeiten für Knotenanzahl

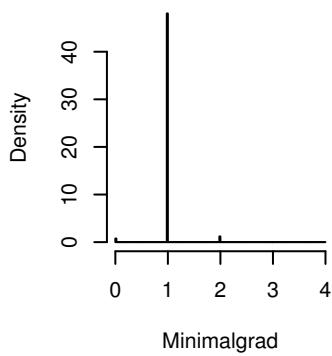


Relative Häufigkeiten für Kantenanzahl

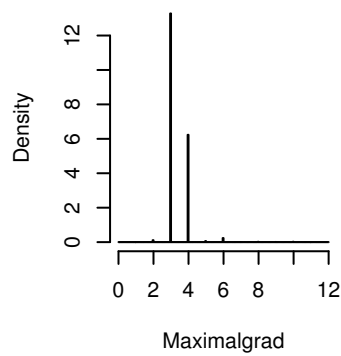


(a) Relative Häufigkeiten für die Größen der Graphen

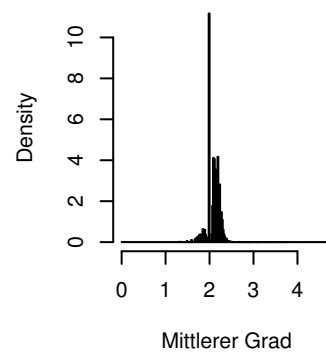
Minimalgrad



Maximalgrad



Mittlerer Grad

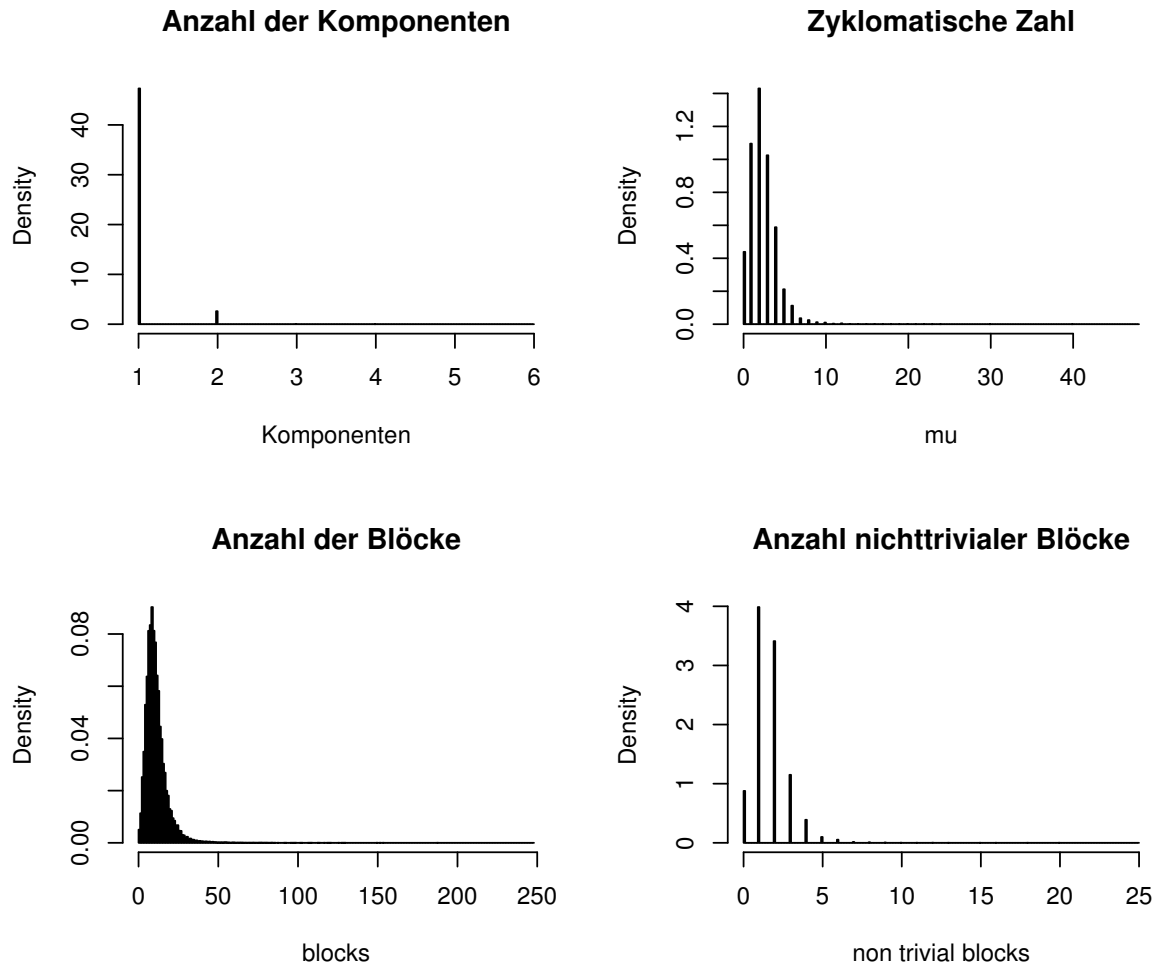


(b) Knotengrade

	Minimum	1. Quartil	Median	Mittelwert	3. Quartil	Maximum
Knoten	1	15	20	21,48	25	252
Kanten	0	15	21	22,82	27	276
min. Grad	0	1	1	1,009	1	4
max. Grad	0	3	3	3,367	4	12
mittl. Grad	0	2	2,114	2,101	2,190	4,769

(c) Übersicht

Abbildung 4.1: Einfache Kenngrößen

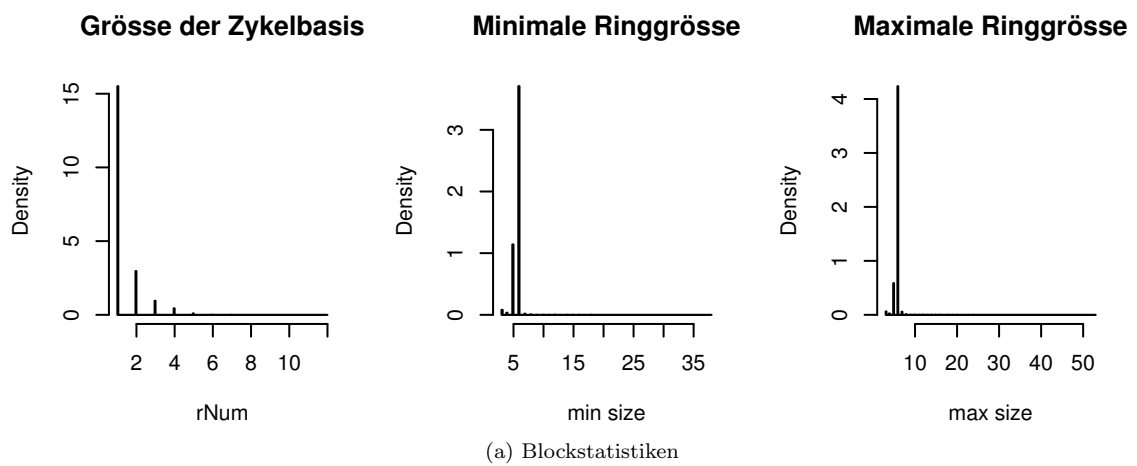


(a) Anzahl der Komponenten und Blöcke

	Minimum	1. Quartil	Median	Mittelwert	3. Quartil	Maximum
Komponenten	1	1	1	1,055	1	6
$\mu(G)$	0	1	2	2,398	3	48
Anzahl Blöcke	0	7	10	11,48	14	248
Anzahl nichttrivialer Blöcke	0	1	2	1,692	2	25

(b) Übersicht

Abbildung 4.2: Zusammenhangseigenschaften



	Minimum	1. Quartil	Median	Mittelwert	3. Quartil	Maximum
Größe der Zykelbasis	1	1	1	1	1	12
Minimale Ringgröße	3	5	6	5,729	6	38
Maximale Ringgröße	3	6	6	5,878	6	53

(b) Übersicht

Abbildung 4.3: Blockstatistiken

Kapitel 5

Blocklayout

In diesem Kapitel soll zuerst gezeigt werden, wie für einen außenplanaren Block ein uniformes Layout bestimmt werden kann, sofern ein solches überhaupt existiert. Der eigentliche Layout-Algorithmus ist linear in der Größe des Blocks, die Entscheidung, ob ein uniformes Layout existiert, hat dagegen im allgemeinen eine Komplexität $O(|E| \log |E|)$. Es stellt sich heraus, dass auf dem Testdatensatz nur ein verschwindend geringer Prozentsatz der Substanzen kein uniformes Layout zulässt. Für diese Fälle wird danach als Fallback ein Algorithmus vorgestellt, der ausgehend von einem außenplanaren Layout versucht, die Winkelauflösung zu maximieren. Für diesen Schritt lassen sich allerdings keine Laufzeitgarantien mehr geben.

Zuerst sollen jedoch wie angekündigt einige der angenehmen Eigenschaften vorgestellt werden, die außenplanare Blöcke zusätzlich zu planaren Graphen besitzen. Wie üblich ist die äußere Facette so gewählt, dass alle Knoten auf ihrem Rand liegen.

5.1 Eigenschaften außenplanarer Graphen

Im Folgenden sei $H(G)$ stets ein Hamiltonkreis des Graphen G .

Lemma 5.1. *[LS97] Ein außenplanarer Graph $G(V, E)$ ist genau dann hamiltonsch, wenn er 2-fach zusammenhängend ist. In diesem Fall ist der Hamiltonkreis $H(G)$ sogar eindeutig bestimmt.*

Für 2-fach zusammenhängende außenplanare Graphen gibt es damit eine eindeutige Partition von E in $H(G)$ und die Menge der Sehnen $S(G)$. In 5.1.1 wird ein Algorithmus vorgestellt, der 2-fach zusammenhängende Graphen in Linearzeit auf Außenplanarität testet und dabei gleichzeitig $S(G)$ und damit dann auch $H(G)$ bestimmt.

Im Folgenden sind alle Graphen stets als 2-fach zusammenhängend vorausgesetzt. Durch die Wahl der äußeren Facette ist dann offenbar die Einbettung eines außenplanaren Graphen schon eindeutig festgelegt. Ein Layout eines Graphen, bei dem die Knoten von G der Reihe nach die Ecken eines regulären $|V|$ -Ecks besetzen, heißt **Kreislayout** (s. Abb. 5.1). Nach Definition besitzt damit jeder außenplanare Graph ein kreuzungsfreies Kreislayout, und da solche Kreislayouts offenbar immer konvex sind, folgt:

Lemma 5.2. *Jeder außenplanare Graph besitzt eine konvexe Einbettung.*

Will man außenplanare Graphen charakterisieren, so hat man zunächst folgendes notwendiges und hinreichendes Kriterium (in Analogie zu planaren Graphen) zur Verfügung:

Satz 5.3. *[Mit79] Ein Graph ist genau dann außenplanar, wenn er keinen zum K_4 oder dem $K_{3,2}$ homöomorphen Teilgraphen besitzt.*

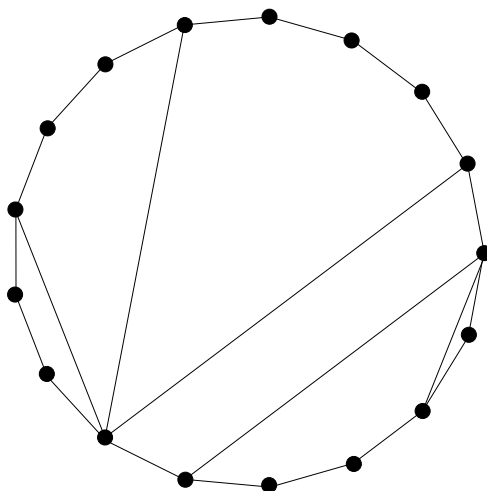


Abbildung 5.1: Kreislayout eines außenplanaren Graphen mit 17 Knoten

Praktikabler in der Anwendung ist allerdings folgendes Kriterium:

Satz 5.4. *Jeder maximale außenplanare Graph mit n Knoten besitzt genau $2n - 3$ Kanten und $n - 1$ Facetten sowie mindestens zwei Knoten vom Grad 2, sogenannte 2-Knoten.*

Ein Charakterisierung außenplanarer Graphen liefert dann:

Satz 5.5. [Mit79] *Ein Graph G mit n Knoten ist genau dann maximal außenplanar, wenn er entweder ein K_3 ist, oder die folgenden vier Eigenschaften erfüllt:*

1. G hat genau $2n - 3$ Kanten
2. G hat mindestens zwei 2-Knoten
3. keine Kante begrenzt mehr als zwei Facetten
4. durch Löschen eines beliebigen 2-Knotens erhält man wieder einen maximal außenplanaren Graphen

Korollar 5.6. [Mit79] *In einem maximal außenplanaren Graphen sind die beiden adjazenten Knoten eines 2-Knotens adjazent.*

Für nicht maximal außenplanare Graphen gilt:

Lemma 5.7. [Mit79] *Ein Graph G ist genau dann außenplanar, wenn er durch Triangulierung maximal außenplanar wird.*

Damit erhält man auch folgenden Satz, der später für die Verarbeitung außenplanarer Blöcke sehr hilfreich sein wird:

Satz 5.8. *Der geometrische Dualgraph $D(G)$ der inneren Facetten eines eingebetteten außenplanaren Graphen G ist ein Baum.*

Beweis. Dass $D(G)$ zusammenhängend ist, ist für einen 2-fach zusammenhängenden planaren Graphen stets erfüllt. Ein maximal außenplanarer Graph hat nach 5.5 $n - 2$ innere Facetten und $2n - 3$ Kanten, also $n - 3$ Sehnen. Zwei adjazente innere Facetten haben genau eine Sehne gemeinsam, damit ist $D(G)$ schlicht, woraus die Aussage für maximale außenplanare Graphen folgt. Für nicht maximale Graphen ist nur zu beachten, dass durch Entfernen einer Sehne sich auch die Anzahl der Facetten um eins verringert. \square

5.1.1 Test auf Außenplanarität

Bekannt ist, dass man einen Graphen in Linearzeit auf Planarität testen kann (z.B. Hopcroft u. Tarjan[HT74]). Auch für Außenplanarität gibt es einen Test in Linearzeit, der auf Satz 5.5 beruht. Zusätzlich benötigt man noch die einfache Tatsache, dass ein Graph genau dann (maximal) außenplanar ist, wenn dies für seine zweifachen Zusammenhangskomponenten gilt. Für den Test wird die rekursive Struktur von Satz 5.5 ausgenutzt, indem sukzessive alle 2-Knoten entfernt werden. Jeder der dabei entstehenden Teilgraphen muss wieder maximal außenplanar sein, also die übrigen Bedingungen des Satzes erfüllen. Bleibt am Schluss kein K_3 übrig, oder waren die Bedingungen bei einem Zwischenschritt verletzt, ist der Graph nicht maximal außenplanar.

Zum Test auf Außenplanarität kann man obige Idee modifizieren, indem man einfach virtuelle Triangulierungskanten einfügt, falls dies nötig ist. Außerdem kann man dabei gleich alle Sehnen des Blocks bestimmen, denn jede (echte) Kante, die zwei Nachbarn eines 2-Knotens verbindet, ist eine Sehne außer evtl. im letzten Schritt, in dem nur noch ein Dreieck vorhanden ist. Daher wird hier auch nicht getestet, ob zum Schluss der K_3 übrig bleibt, sondern ob der Graph nur noch eine Kante hat. Der Algorithmus arbeitet bei geeigneter Implementation der Adjazenzlisten von G in Linearzeit, dies und die Korrektheit wird in [Mit79] gezeigt. Der Algorithmus mit den für die Sehnenbestimmung nötigen Ergänzungen ist in Algorithmus 1 gezeigt.

5.1.2 Zykelbasen außenplanarer Graphen

Bevor wir nun zum zentralen Satz dieses Kapitels kommen, benötigen wir noch etwas Vorarbeit. Im Folgenden seien die Knoten eines 2-fach zusammenhängenden außenplanaren Graphen G so nummeriert und die Kanten so orientiert, dass $H(G)$ aus den Kanten $(i, i+1)$ für $1 \leq n \leq n-1$ und $(1, n)$ bestehe, und für alle $e = (i, j) \in S(G)$ stets $i < j$ gelte. OBdA sei $\deg(n) = 2$. Dann kann man auf der Sehnenmenge $S(G)$ eine partielle Ordnung wie folgt festlegen:

$$a = (i_a, j_a) \prec b = (i_b, j_b) \iff i_b \leq i_a < j_a \leq j_b \text{ und } a \neq b \quad (5.1)$$

Gibt es kein $c \in S(G)$ mit $a \prec c \prec b$, so heißt b **unmittelbarer Nachfolger** von a , $a \prec\prec b$. Für alle $a \in S(G)$ sei $Y_a := \{b \in S(G) \mid b \prec\prec a\}$. Y_* sei die Menge aller \prec -maximalen Kanten in $S(G)$.

Sei weiter $T(G)$ ein aufspannender Baum irgendeines Graphen G , dann gibt es für jede Kante $a \notin T(G)$ einen eindeutigen Zykel in $T \cup a$, der als **fundamental** bezeichnet wird. Die Menge aller Fundamentalzylinder zu einem aufspannenden Baum $T(G)$ bildet eine Zykelbasis, die sogenannte **fundamentale Zykelbasis bzgl. $T(G)$** .

Für außenplanare Blöcke gilt nun folgender wichtiger Satz:

Satz 5.9. [LS97] *Sei $G = (V, E)$ ein zweifach zusammenhängender außenplanarer Graph. Dann besitzt G eine eindeutig bestimmte minimale Zykelbasis \mathcal{M} .*

Im weiteren Verlauf (5.1.3) werden wir sogar einen Algorithmus vorstellen, der diese Zykelbasis in Linearzeit bestimmt. Einen Beweis dieses Satzes, wie auch der meisten anderen Sätze dieses Abschnitts findet man wieder in [LS97]. Hier sei lediglich \mathcal{M} explizit angegeben:

Für G ist $T(G) = H(G) \setminus \{(1, n)\}$ ein aufspannender Baum. Die Fundamentalbasis (F) bzgl. $T(G)$ besteht damit aus den eindeutig bestimmten Zykeln F_a in $T(G) \cup \{a\}$ für $a \in S(G)$ sowie $H(G)$ selbst. Definiere nun:

$$C_a := \left(\bigoplus_{b \in Y_a} F_b \right) \oplus F_a \text{ und } C_* := \left(\bigoplus_{b \in Y_*} F_b \right) \oplus H(G) \quad (5.2)$$

Dann ist $\mathcal{M} := \{C_a \mid a \in S(G)\} \cup \{C_*\}$ die gesuchte minimale Zykelbasis.

In 2.1.2 hatten wir gesehen, dass im Allgemeinen nicht jede Zykelbasis planar ist, insbesondere auch nicht jede minimale. Daher ist es sehr angenehm, dass dies im Falle außenplanarer Graphen doch gilt:

```

Input : Ein Graph  $G = (V, E)$  mit  $n$  Knoten
Output : Ist  $G$  außenplanar oder maximal außenplanar, welches sind die Sehnen?
1 begin
    Data : Liste von Knotenpaaren  $pairs \leftarrow \emptyset$ 
    Data : Stack von Knoten  $2\text{-verts} \leftarrow$  Alle 2-Knoten von  $G$ 
    Data : Liste von Kanten  $S(G) \leftarrow \emptyset$ 
    Data : Liste von Kanten  $triang \leftarrow \emptyset$ 
    Data : Liste von Kanten  $edges \leftarrow E$ 
2   if  $(|2\text{-verts}| < 2)$  oder  $(|E| > 2|V| - 3)$  then return Nicht außenplanar
3   while  $|V| > 2$  do
4      $v \leftarrow 2\text{-verts.pop}()$ 
5      $n_1, n_2 \leftarrow$  Nachbarn von  $v$ 
6     if  $|V| \neq 3$  then  $pairs.add(n_1, n_2)$ 
7     if  $\{n_1, n_2\} \in E$  then
8       if  $(\{n_1, n_2\} \notin triang)$  und  $(|V| \neq 3)$  then  $S(G).add(\{n_1, n_2\})$ 
9     else
10      /*Virtuelle Triangulierungskante gefunden */
11       $triang.add(\{n_1, n_2\})$ 
12       $E \leftarrow E \cup \{n_1, n_2\}$ 
13      if  $\{v, n_1\} \in triang$  then  $edges.add(\{v, n_1\})$ 
14      if  $\{v, n_2\} \in triang$  then  $edges.add(\{v, n_2\})$ 
15      Lösche  $v$  aus  $G$ 
16      if  $n_1$  ist neuer 2-Knoten then  $2\text{-verts.add}(n_1)$ 
17      if  $n_2$  ist neuer 2-Knoten then  $2\text{-verts.add}(n_2)$ 
18      if  $(|N| = 2)$  and  $(\{n_1, n_2\} \in triang)$  then  $edges.add(\{n_1, n_2\})$ 
19      if  $|2\text{-verts}| < 2$  then return Nicht außenplanar
20   if  $|E| \neq 1$  then return Nicht außenplanar
21   Sortiere  $edges$  und  $pairs$  lexikographisch
22   if Ein Element von pairs kommt nicht in edges vor then
23     return nicht außenplanar
24   else
25     if  $triang = \emptyset$  then
26       return Maximal außenplanar
27     else
28       return Außenplanar
29
30
31 end

```

Algorithmus 1 : Test auf Außenplanarität

Korollar 5.10. [LS97] Für einen zweifachzusammenhängenden außenplanaren Graphen $G = (V, E)$ ist \mathcal{M} eine planare Zykelbasis der Länge $l(\mathcal{M}) = 2|E| - |V|$

5.1.3 Bestimmung der minimalen Zykelbasis

Die Ergebnisse des letzten Abschnittes kann man direkt in einen Algorithmus 2 zur Bestimmung der minimalen Zykelbasis eines außenplanaren Blocks umsetzen, indem man einfach die Konstruktion der einzelnen Zykel C_a und C_* nachvollzieht. Als Eingabe wird der Hamiltonkreis $H(G)$ benötigt, der sich mit Hilfe von 1 bestimmen lässt.

<p>Input : Ein außenplanarer Block $G = (V, E)$ mit Hamiltonkreis $H(G) = \{1, \dots, n\}$ Output : Minimale Zykelbasis \mathcal{M}</p> <pre style="margin: 0;"> 1 begin Data : Stack von Kanten $S \leftarrow \emptyset$ 2 $\mathcal{M} \leftarrow \emptyset$ for all Knoten $i, 1 \leq i \leq n$ do 3 while Es gibt eine Kante (i, k_j) oben auf S do 4 $(i, k_j) \leftarrow S.pop()$ 5 $P \leftarrow$ Weg von k_j nach i in $H(G)$ 6 $\mathcal{M} \leftarrow \mathcal{M}.add(P \cup \{(k_j, i)\})$ 7 Lösche die Knoten in $P \setminus \{i, k_j\}$ aus $H(G)$ 8 for all Kanten $(i, k_j), n \geq k_1 > k_2 > \dots > i + 1$ do 9 $S.push((i, k_j))$ 10 11 12 end </pre>
--

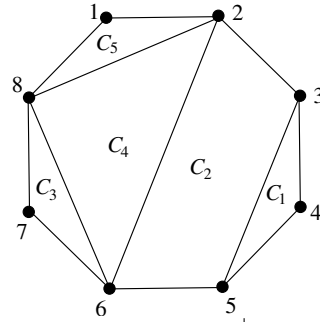
Algorithmus 2 : Bestimmung der minimalen Zykelbasis eines außenplanaren Blocks

Dieser Algorithmus legt in den Schritten 8 und 9 die Sehnen und die Kante $(1, n)$, die noch nicht verarbeitet wurden, auf S ab, und zwar für jeden Knoten i in aufsteigender Ordnung. Damit begrenzen diese Kanten im Kreislayout direkt nebeneinanderliegende Facetten, die in 5-6 erzeugten Zykel haben damit keine Sehnen, und alle Knoten außer i und j haben Grad 2. In Schritt 7 werden die durch P erzeugten Ohren dann entfernt. Implementiert man $H(G)$ beispielsweise als verkettete Liste, so sieht man leicht, dass dieser Algorithmus in $O(|V|)$ arbeitet: jeder Knoten wird einmal betrachtet, und jede Sehne wird höchstens einmal auf den Stack gelegt. Die Suche nach P kann einfach als Herausnehmen des Teilstücks $i \dots k_j$ aus $H(G)$ implementiert werden. Ein Beispiel für die Arbeitsweise des Algorithmus sieht man in Abb. 5.2.

Betrachtet man den Ablauf, in dem der Algorithmus die Zykel findet, so fällt auf, dass die Zykel resp. die entsprechenden Facetten im Kreislayout sozusagen von außen nach innen gefunden werden, und dass der Graph in jedem Reduktionsschritt 2-fach zusammenhängend und außenplanar bleibt. Im Verlaufe des Beweises von 5.9 wird genau dies gezeigt:

Lemma 5.11. *Algorithmus 2 liefert auf einem zweifach zusammenhängenden außenplanaren Graphen eine totale Ordnung C_1, \dots, C_k der minimalen Zykelbasis $\mathcal{M} = (C_1, \dots, C_k)$ derart, dass für alle $i = 1, \dots, k - 1$ C_i zu genau einem Zykel C_j mit $j > i$ adjazent ist. Sei $G_0 := G$, $G_i := G_{i-1} \setminus (C_i \setminus \{\text{Endknoten von } C_i\})$. Dann ist G_i für $i = 1, \dots, k - 1$ ebenfalls zweifach zusammenhängend und außenplanar.*

Will man zusätzlich noch den Dualgraphen $D(G)$ für die inneren Facetten bestimmen, so ist lediglich der Schritt 4 und der Schritt 5 zu erweitern. Für Schritt 4 nutzt man aus, dass die Facette, die durch $P \cup \{(i, k_j)\}$ berandet wird, neu gefunden wurde (weil (i, k_j) ja bisher noch gar nicht betrachtet wurde), also kann man für $P \cup \{(i, k_j)\}$ auf jeden Fall einen neuen Knoten in $D(G)$ einfügen. Zusätzlich merkt man sich noch die Kante, über die diese Facette betreten wurde (z.B. in einem Knotenattribut).



i	Aktion	S	\mathcal{M}	$H(G)$
-	Initialisierung	\emptyset	\emptyset	(1,2,3,4,5,6,7,8)
1	push (1,8)	(1,8)	\emptyset	(1,2,3,4,5,6,7,8)
2	push (2,8)	(1,8),(2,8)	\emptyset	(1,2,3,4,5,6,7,8)
2	push (2,6)	(1,8),(2,8),(2,6)	\emptyset	(1,2,3,4,5,6,7,8)
3	push (3,5)	(1,8),(2,8),(2,6),(3,5)	\emptyset	(1,2,3,4,5,6,7,8)
4	-	(1,8),(2,8),(2,6),(3,5)	\emptyset	(1,2,3,4,5,6,7,8)
5	pop (3,5), $C_1=(3,4,5)$	(1,8),(2,8),(2,6)	$\{C_1\}$	(1,2,3,5,6,7,8)
6	pop (2,6), $C_2=(2,3,5,6)$	(1,8),(2,8)	$\{C_1, C_2\}$	(1,2,6,7,8)
6	push (6,8)	(1,8),(2,8),(6,8)	$\{C_1, C_2\}$	(1,2,6,7,8)
7	-	(1,8),(2,8),(6,8)	$\{C_1, C_2\}$	(1,2,6,7,8)
8	pop (6,8), $C_3=(6,7,8)$	(1,8),(2,8)	$\{C_1, C_2, C_3\}$	(1,2,6,8)
8	pop (2,8), $C_4=(2,6,8)$	(1,8)	$\{C_1, C_2, C_3, C_4\}$	(1,2,8)
8	pop (1,8), $C_5=(1,2,8)$	\emptyset	$\{C_1, C_2, C_3, C_4, C_5\}$	

Abbildung 5.2: Beispiel für Algorithmus 2

```

Data : Map von Kanten nach Knoten  $connection \leftarrow \emptyset$ 
...
4  $(i, k_j) \leftarrow S.pop()$ 
4a Füge neuen Knoten  $n$  in  $D(G)$  ein
4b if  $(i, k_j) \in S(G)$  then  $connection.add((i, k_j), n)$ 

```

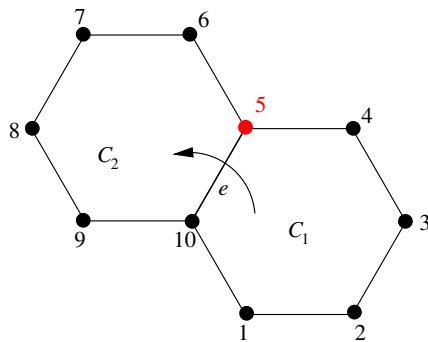
Um die andere Facette zu finden, mit der n verbunden wird, muss beim Suchen des Weges P nur überprüft werden, ob man eine Sehne e findet. Da e jetzt auf dem Rand von G liegt, muss die durch e auf der anderen Seite begrenzte Facette f bereits über e gefunden und entfernt worden sein, so dass man diese Facette ebenfalls über die $connection$ -Map finden kann:

```

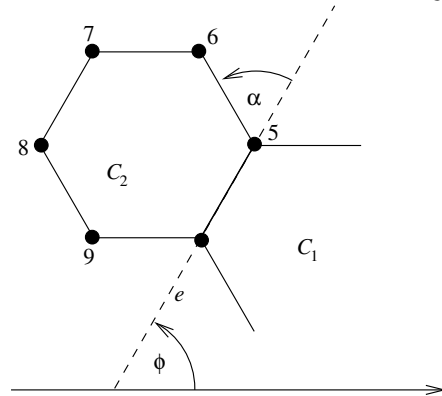
Data : Map von Kanten nach Knoten  $connection \leftarrow \emptyset$ 
...
5  $P \leftarrow$  Weg von  $k_j$  nach  $i$  in  $H(G)$ 
5a forall Kanten  $e \in P$  do
5b   if  $e \in S(G)$  then
5c      $f \leftarrow connection.get(e)$ 
5d     Füge Kante zwischen  $n$  und  $f$  in  $D(G)$  ein

6  $\mathcal{M} \leftarrow \mathcal{M}.add(P \cup \{(i, k_j)\})$ 
7 Lösche die Knoten in  $P \setminus \{(i, k_j)\}$  aus  $H(G)$ 
7a Füge  $(i, k_j)$  zusätzlich an der richtigen Stelle in  $H(G)$  ein

```



(a) Eintrittsknoten: Beim Betreten von C_2 über e ist Knoten 5 der Eintrittsknoten, der Knoten 6 wird damit als nächstes positioniert



(b) Übergang zur nächsten Facette: Als nächstes wird Knoten 6 gezeichnet, der Startwinkel ist um ϕ zu vergrößern,

Abbildung 5.3: Betreten der nächsten Facette

Diese Schritte erhöhen die Komplexität nicht, da zwar alle Kanten des Pfades P abgesucht werden, diese aber danach nicht mehr betrachtet werden, da das Ohr ja entfernt wird. Es wird also im schlimmsten Fall jede Kante einmal besucht, was wegen Satz 5.5 immer noch linear in der Anzahl der Knoten bleibt.

Eine weitere nützliche Eigenschaft von Algorithmus 2 ist die Orientierungserhaltung, genauer:

Lemma 5.12. *Algorithmus 2 orientiert alle Zyklen gleich wie $H(G)$*

Liegt also das Innere von G im Kreislayout beispielsweise immer links von $H(G)$, so sind auch die inneren Facetten links orientiert.

Beweis. Nach Konstruktion sind die einzelnen Zyklen bereits orientiert. Jeder Zyklus enthält außerdem eine Kante (i, j) mit $i < j$, entweder weil $(i, j) \in H(G)$ oder weil das komplette Stück $(i+1, j-1)$ aus $H(G)$ bereits entfernt wurde. Damit sind alle Zyklen gleich wie $H(G)$ orientiert. \square

Dies machen wir uns zunutze, um für jeden Zyklus C den **Eintrittsknoten** $\varepsilon(C, e)$ bzgl. einer Sehne $e \in C$ zu bestimmen. OBdA sei G dazu links orientiert, dann ist $\varepsilon(C, e)$ derjenige Knoten von e , der beim Betreten von C über die Kante e rechts liegt (s. Abb. 5.3(a)). Man sieht darin auch, dass für die durch e neu gefundene Facette F_1 der Eintrittsknoten gerade i ist, für die andere Facette F_2 dagegen k_j . Damit kann man in Schritt 4c und 5d zusätzlich noch diese Knoten speichern:

```

Data : Map von Kanten nach Knoten connection  $\leftarrow \emptyset$ 
Data : Map von Knoten nach Knoten entry  $\leftarrow \emptyset$ 
...
4c entry.add( $n, i$ )
5  $P \leftarrow$  Weg von  $k_j$  nach  $i$  in  $H(G)$ 
5a forall Kanten  $e \in P$  do
5b   | if  $e = (n_1, n_2) \in S(G)$  then
5c   |   |  $f \leftarrow$  connection.get( $e$ )
5d   |   | Füge Kante zwischen  $n$  und  $f$  in  $D(G)$  ein
5e   |   | entry.add( $f, n_2$ )

```

5.2 Layoutalgorithmen für außenplanare Blöcke

5.2.1 Uniformes Layout

Mit den Ergebnissen aus den letzten Abschnitten kann man nun einen einfachen uniformen Layoutalgorithmus für außenplanare Blöcke angeben. Im Dualgraphen sei dazu zu jedem Knoten der ursprüngliche Ring (im Attribut *rings*) gespeichert. Außerdem sei α der Außenwinkel des regulären Polygons, das gerade gezeichnet wird. Die Koordinaten von v seien mit v_x und v_y bezeichnet. Außerdem seien die Ringe R jeweils bei 0 beginnend indiziert (zum Beispiel als Arrays implementiert). Für den Eintrittsknoten sei jeweils nur die Position im Array R gespeichert.

Dann kann man $D(G)$ einfach mit Tiefen- oder Breitensuche abarbeiten:

```

Input : Außenplanarer zweifach zusammenhängender Graph  $G$ 
Input : Kantenlänge  $l$ 
Data : Graph  $D(G)$ 
1 begin
2    $D(G) \leftarrow$  Dualgraph von  $G$ , bestimmt mit Algorithmus 2
3    $v_0 \leftarrow$  Erster Knoten von  $D(G)$ 
4   /*Breitensuche auf  $D(G)$  bei  $v_0$  beginnend */
5   BFS( $D(G), v_0$ )
6 end

```

Algorithmus 3 : Uniformes Layout außenplanarer Blöcke

wobei **BFS_start**(Knoten v_0) die Knoten des Starttrings so auf den Ecken eines regulären n -Ecks platziert, dass der erste Knoten im Ursprung liegt:

```

/*Wird beim Start für  $v_0$  aufgerufen */
6 begin
7    $R \leftarrow$   $rings.get(v_0)$ 
8    $n \leftarrow R.length()$ 
9    $v \leftarrow R[0]$ 
10   $v_x \leftarrow 0, v_y \leftarrow 0$ 
11   $\alpha \leftarrow 2\pi/n$ 
12  /*Der gesamte bisherige Außenwinkel */
13   $\phi \leftarrow 0$ 
14  for  $i = 1$  to  $n - 1$  do
15     $v \leftarrow R[i]$ 
16     $v_x \leftarrow l \cos \phi + R[i - 1]_x$ 
17     $v_y \leftarrow l \sin \phi + R[i - 1]_y$ 
18     $\phi \leftarrow \phi + \alpha$ 
19 end

```

Prozedur BFS_start(Knoten v_0)

und **BFS_forward**(Knoten v_0 , Kante $e = (i, j)$) ganz analog die Knoten des neu gefundenen Rings platziert. Dabei ist zu beachten, dass i und j schon Koordinaten zugewiesen bekommen haben, und der Startwinkel nicht 0 ist, sondern $\alpha + \angle(x\text{-Achse}, e)$ (s. Abb. 5.3(b)). Da der Knoten das erste Mal besucht wird, ist auch nur die Eintrittskante e bisher gezeichnet worden, so dass den übrigen $n - 2$ Knoten noch Koordinaten zugewiesen werden müssen:

```

/*Wird jedesmal aufgerufen, wenn ein Knoten  $n$  das erstmal über die Kante  $e$  besucht
wird */
20 begin
21    $R \leftarrow rings.get(v_0)$ 
22    $n \leftarrow R.length()$ 
23    $i_0 \leftarrow \varepsilon(R, e)$ 
24    $\alpha \leftarrow 2\pi/n$ 
   /*Der gesamte bisherige Außenwinkel */
25    $\phi \leftarrow \alpha + \angle(x\text{-Achse}, e)$ 
26   for  $i = i_0 + 1$  to  $i_0 + n - 1$  do
27      $v' \leftarrow R[i \bmod n]$ 
28      $v'_x \leftarrow l \cos \phi + R[(i - 1) \bmod n]_x$ 
29      $v'_y \leftarrow l \sin \phi + R[(i - 1) \bmod n]_y$ 
30      $\phi \leftarrow \phi + \alpha$ 
31
32 end

```

Prozedur `BFS_forward(Knoten v_0 , Kante e)`

Der Algorithmus liefert offenbar ein uniformes Layout eines außenplanaren Blocks, sofern ein solches überhaupt möglich ist, und benötigt $O(|V|)$ Schritte. Der einzige kritische Punkt ist die Implementation der Attribute und Eintrittsknoten, was sich aber beispielsweise durch Zeiger auf die Position des Eintrittsknotens in R bewerkstelligen läßt.

5.3 Nichtuniforme Layouts und Winkelmaximierung

Leider besitzt nicht jeder außenplanare Block ein uniformes Layout (s. Abb. 5.4). Es stellt sich also einerseits die Frage, wie man solche Fälle erkennt, und andererseits, wie man für solche Fälle ein alternatives Layout bestimmt. Als Designziel sollte dabei mindestens eine möglichst große Winkelauflösung stehen, da diese für den optischen Gesamteindruck hier stark entscheidend ist.

5.3.1 Exkurs: Winkelmaximierung in planaren Graphen

Bevor unser spezielles Problem behandelt werden soll ist es nötig, kurz auf die allgemeine Problematik der Winkelauflösung bei planaren Graphen einzugehen.

Problem 5.13 (Maximale Winkelauflösung ohne vorgegebene kombinatorische Einbettung).

Gegeben sei ein planarer Graph.

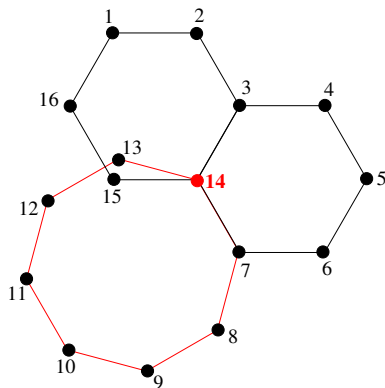
Finde eine geradlinige planare Einbettung mit maximalem minimalem Winkel, d.h. ist \mathcal{F} die Facettenmenge einer Einbettung von G , so dass

$$x_{min} := \min_{\substack{v \in V, F \in \mathcal{F} \\ v \text{ inzident zu } F}} \{x(v, F)\}$$

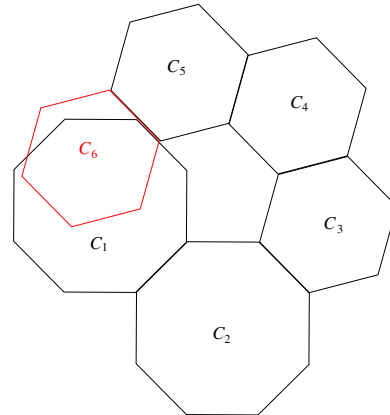
maximal über alle planaren Einbettungen ist. (s. dazu 2.20)

Satz 5.14 (Kant, 1996). *Problem 5.13 ist \mathcal{NP} -schwer.*

Es stellt sich die Frage, ob das Problem leichter wird, wenn eine kombinatorische Einbettung schon vorgegeben ist, wie das beispielsweise bei uns durch die Wahl der äußeren Facette schon geschehen ist:



(a) Kritischer Knoten: Die an Knoten 14 anliegenden Facetten haben zusammen einen Innenwinkel von 375°



(b) Hier ragt Facette C_6 in das innere von Facette C_1

Abbildung 5.4: Zwei Beispiele, bei denen uniformes Layout nicht möglich ist

Problem 5.15 (Maximale Winkelauflösung mit vorgegebener kombinatorischer Einbettung).

Gegeben sei ein planarer Graph mit vorgegebener kombinatorischer Einbettung mit Facettenmenge \mathcal{F} .

Finde eine geradlinige planare Einbettung, so dass x_{min} maximal wird.

In diesem Fall kann man leicht zwei notwendige Bedingungen für die einzelnen Winkel angeben, damit die Winkelzuweisung überhaupt realisierbar ist:

Satz und Definition 5.16 (Lokal konsistente Winkelzuweisung). Eine Winkelzuweisung $x : V \times \mathcal{F} \rightarrow \mathbb{R}$ heißt **lokal konsistent**, falls gilt:

1. **Knotenbedingung**

$$\forall v \in V : \sum_{F \in \mathcal{F} \text{ inzident zu } v} x((v, F)) = 2\pi \quad (5.3a)$$

2. **Facettenbedingung**

$$\forall F \in \mathcal{F} \setminus \{F_0\} : \sum_{v \in V \text{ inzident zu } F} x((v, F)) = (d_G(F) - 2)\pi \quad (5.3b)$$

$$\sum_{v \in V \text{ inzident zu } F_0} x((v, F_0)) = (d_G(F_0) + 2)\pi \quad (5.3c)$$

wobei $d_G(F) = |\{v \in V \mid v \text{ inzident zu } F\}|$.

Lokale Konsistenz einer Winkelzuweisung ist eine notwendige Bedingung für die Realisierbarkeit durch eine geradlinige planare Einbettung.

Die Bedingungen aus obigem Satz lassen sich leicht in ein LP umformulieren, das sich sogar effizient lösen ließe.

Leider ist lokale Konsistenz aber nicht hinreichend für die Realisierbarkeit durch eine Einbettung, wie man in der nächsten Abbildung sieht

Definition 5.17 (Dreiecksgraph, Rad). Ein planarer dreifach zusammenhängender Graph $G = (V, E)$, heißt **Dreiecksgraph**, falls alle Facetten bis auf eine, die als äußere gewählt wird, Dreiecke sind. Ein **Rad** R_d mit $d \geq 3$ ist ein planar eingebetteter Dreiecksgraph mit $d+1$ Knoten v_1, v_2, \dots, v_d und $2d$ Kanten, wobei $\deg(v_i) = d$, v_i mit allen $v_j, j = 1, \dots, d$ verbunden ist, sowie v_i jeweils mit $v_{i+1 \pmod d}$ verbunden ist (s. Abb. 5.3.1). In diesem Layout heißen die Winkel α_i **linke Winkel**, β_i **rechte Winkel**.

Es folgt sofort aus den Definitionen:

Lemma 5.18. *Ein eingebetteter außenplanarer Graph besitzt keine Räder.*

Eine hinreichende Bedingung für eine Klasse von Graphen liefert nun der folgende Satz:

Satz 5.19 (Di Battista u. Vismara). *Sei $G = (V, E)$ ein planarer Dreiecksgraph mit einer kombinatorischen Einbettung mit Facettenmenge \mathcal{F} . Für eine Winkelzuweisung x gibt es eine geradlinige Realisierung dieser Einbettung genau dann, wenn gilt:*

$$\forall v \in V : \sum_{F \in \mathcal{F} \text{ inzident zu } v} x((v, F)) = 2\pi \tag{5.4a}$$

$$\forall F \in \mathcal{F} \setminus \{F_0\} : \sum_{v \in V \text{ inzident zu } F} x((v, F)) = (d_G(F) - 2)\pi \tag{5.4b}$$

Ceva-Bedingung: $\forall v \in V, v \notin \partial F_0$ sei R_d Rad mit Zentrum v und $\alpha_1 \dots \alpha_d, \beta_1 \dots \beta_d$ linke und rechte Winkel zu v . Dann ist

$$\prod_{i=1}^d \frac{\sin \alpha_i}{\sin \beta_i} = 1 \tag{5.4c}$$

$$\forall v \in V, v \in \partial F_0 : \sum_{F \in \mathcal{F} \setminus \{F_0\} \text{ inzident zu } v} x((v, F)) \leq \pi \tag{5.4d}$$

Die ersten beiden Bedingungen sind nichts anderes als die lokale Konsistenz für innere Facetten. Wegen Bedingung 5.4d, die die Facettenbedingung für die äußere Facette in Def. 5.16 ersetzt, hat eine solche Einbettung G notwendig einen konvexen Rand. Bedingung 5.4c ist hochgradig nichtlinear, so dass sich zwar wieder ein Optimierungsproblem formulieren lässt, dies aber im Allgemeinen nicht mehr effizient lösbar ist.

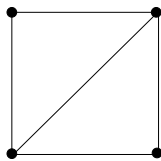
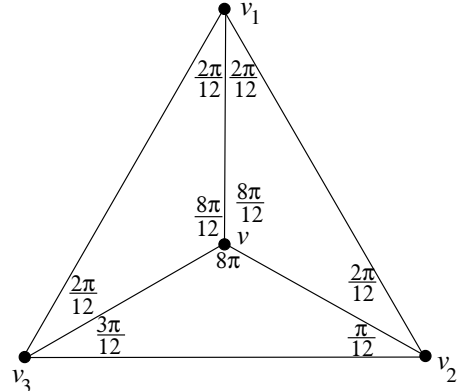


Abbildung 5.5: Maximal außenplanar, aber nicht dreifach zusammenhängend



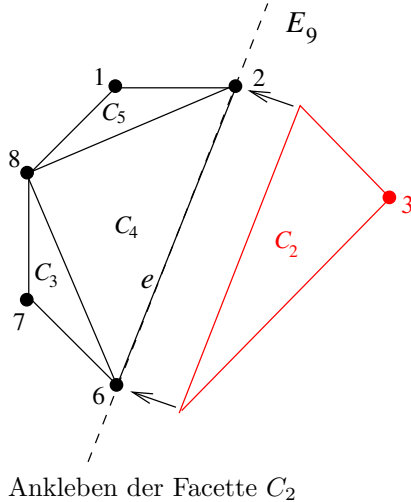
Beispiel für ein Rad mit nicht realisierbarer lokal konsistenter Winkelzuweisung der inneren Facetten

Da außenplanare Graphen keine Räder besitzen, könnte man hoffen, dass auf die störende Ceva-Bedingung verzichtet werden kann, somit lokale Konsistenz auch hinreichend für Realisierbarkeit ist. Allerdings sind außenplanare Graphen im Allgemeinen *nicht* dreifach zusammenhängend und lassen sich auch nicht dreifach zusammenhängend machen, ohne dass die Außenplanarität verloren geht, wie man schon für den einfachen maximal außenplanaren Graphen in Abb. 5.5 sieht.

Fordert man zusätzlich noch die Konvexität des Randes von G , so erhält man wenigstens noch:

Lemma 5.20. *Sei G ein maximaler außenplanarer Graph, mit der üblichen kombinatorischen Einbettung. Für eine Winkelzuweisung x gibt es genau dann eine geradlinige Realisierung mit konvexem Rand von G , wenn die Bedingungen 5.4a, 5.4b und 5.4d aus Satz 5.19 erfüllt sind.*

Beweis. Die Notwendigkeit der drei Bedingungen ist klar. Die Rückrichtung kann man durch Induktion über die inneren Facetten zeigen, wobei die Reihenfolge der Facetten durch die umgekehrte Ordnung aus Lemma 5.11 gegeben ist. Dazu beachte man, dass in einem maximalen außenplanaren Graphen alle inneren Facetten Dreiecke sind, somit bis auf einen Skalierungsfaktor durch die drei Innenwinkel festgelegt sind. Außerdem ist auf den minimalen Zykeln und damit auf den inneren Facetten durch 5.11 eine Ordnung definiert, bei der adjacente Facetten unmittelbar aufeinander folgen. Sei G_i der Teilgraph von G mit den Facetten F_1, \dots, F_i , und $F_0(G_i)$ seine äußere Facette.



Induktionsanfang G_1 hat nur die innere Facette F_1 , diese ist ein Dreieck, wenn die Winkelzuweisung die obigen Bedingungen erfüllt, damit gibt es sicher eine konvexe Realisierung der Winkelzuweisung.

Induktionsschritt Sei G_i als konvex außenplanar eingebettet vorausgesetzt und realisiere die gegebene Winkelzuweisung. Wegen der Eigenschaften der Ordnung 5.11 ist die neue Facette F_{i+1} zu genau einer schon vorhandenen adjazent, und man kann sie so skalieren, dass sie auf die Kante e passt, ohne dass sich ihre Innenwinkel ändern. Damit läßt sich die Winkelzuweisung auch für G_{i+1} gradlinig realisieren. Da G_i als konvex vorausgesetzt war, ist die Halbebene E_i auf der äußeren Seite von e noch frei, sodass der Graph weiterhin planar bleibt. Durch Hinzufügen von F_{i+1} ändern sich in G_{i+1} gegenüber G_i lediglich die Winkel an den Knoten v und w . Da wegen der Außenplanarität von G v und w auch auf dem Rand von G liegen, gilt auch $\sum_{F \in \mathcal{F} \setminus \{F_0\}} \text{inzident zu } v \ x((v, F)) \leq \pi$ bzw. entsprechend für w , da gegenüber G ja höchstens innere Facetten an v bzw. w fehlen können, die zum Innenwinkel beitragen. Damit ist auch der Rand von G_{i+1} konvex \square .

Verzichtet man auch auf die Konvexität des Randes und fordert lediglich lokale Konsistenz im Sinne von Def. 5.16, so wird der Satz falsch, wie Abb. 5.6 zeigt. Verzichtet man auf die Triangulierung, so hat man zwar etwas Spielraum zur Realisierung, indem man evtl. die Kantenlängen verändern kann, ohne die Winkel zu verändern, allerdings hilft das auch nicht weiter bei der Entscheidung, ob eine beliebige Winkelzuweisung nun realisierbar ist, geschweige denn bei der Formulierung eines Optimierungsproblems.

5.3.2 Kreuzungserkennung bei außenplanaren Graphen

Im Allgemeinen kann man also für außenplanare Graphen nicht ohne Weiteres erkennen, ob eine gegebene Winkelzuweisung realisierbar ist. Insbesondere kann man nicht einfach durch Überprüfung der lokalen Konsistenzbedingungen entscheiden, ob ein uniformes Layout möglich ist. Die einfachste Strategie, um diese Frage trotzdem zu entscheiden, ist es sicherlich, einfach winkel- und kantenuniform zu zeichnen, ohne sich um Planarität zu kümmern und dann zu überprüfen, ob sich irgendwelche Kanten kreuzen. Da nach Konstruktion beim Beginn eines neuen Rings die neuen

Kanten nie in das Innere des gerade verlassenen Rings zeigen können, liegt nie ein Ring ganz im Inneren eines anderen, so dass es genügt, zu überprüfen, ob sich irgendwelche Kanten von $H(G)$ schneiden. Dies läßt sich beispielsweise mit einem *Sweep-Line*-Algorithmus in $O(|V| \log |V|)$ implementieren. Trotzdem gibt es zwei einfache Heuristiken, mit denen sich unter Umständen eine unnötige Bestimmung eines uniformen Layouts und/oder der Kantenkreuzungen vermeiden läßt.

Heuristik 1

Hier wird einfach für jeden Knoten v die Konsistenzbedingung 5.4a überprüft. Da für jeden Ring in einem uniformen Layout der Innenwinkel δ kleiner als π ist, genügt es offenbar, diejenigen Knoten zu überprüfen, zu denen mehr als zwei Facetten inzident sind. Dies sind genau diejenigen Knoten v mit $\deg(v) \geq 4$ (s. Abb. 5.4(a)). Ein solcher Knoten soll **kritisch** heißen. Gibt es einen kritischen Knoten, bei dem Bedingung 5.4a für die durch uniformes Layout gegebene Winkelzuweisung verletzt ist, kann es wegen der Eindeutigkeit der Einbettung überhaupt kein (winkel-)uniformes Layout geben.

Heuristik 2

Wenn es keine kritischen Knoten gibt, so liegt zwischen zwei nicht direkt benachbarten Ringen auf dem Rand von G mindestens eine weitere Kante. Der ungünstigste Fall tritt dann ein, wenn die Ringe wie in Abb. 5.4(b) gezeigt angeordnet sind, d.h. es gibt eine **Kette** von Ringen so, dass auf dem Rand von G direkt aufeinanderfolgende Kanten immer zu unterschiedlichen, aber adjazenten Ringen gehören. Beachtet man hierbei noch, dass der Außenwinkel γ um so kleiner wird, je größer die Ringe sind, kann man sich überlegen, wieviele Ringe von der maximal im Ringsystem auftretenden Größe vorhanden sein können, ohne dass Kantenkreuzungen im uniformen Layout auftreten:

Maximale Ringgröße	3	4	5	6	7	8	9	10	11	≥ 12
Maximale Ringanzahl	N/A	∞	9	5	4	3	3	3	3	2

Tabelle 5.1: Maximal mögliche Ringanzahlen im ungünstigsten Fall

Bemerkung 5.21. Wenn die maximale Ringgröße 3 ist, kann es nicht mehr als 2 Ringe geben, ohne dass es kritische Knoten gibt.

Man muss also lediglich die maximale Ringgröße und die Anzahl der Ringe in G bestimmen und mit Tabelle 5.1 überprüfen. Gibt es nicht mehr Ringe und keine kritischen Knoten, so ist ein uniformes Layout auf jeden Fall möglich.

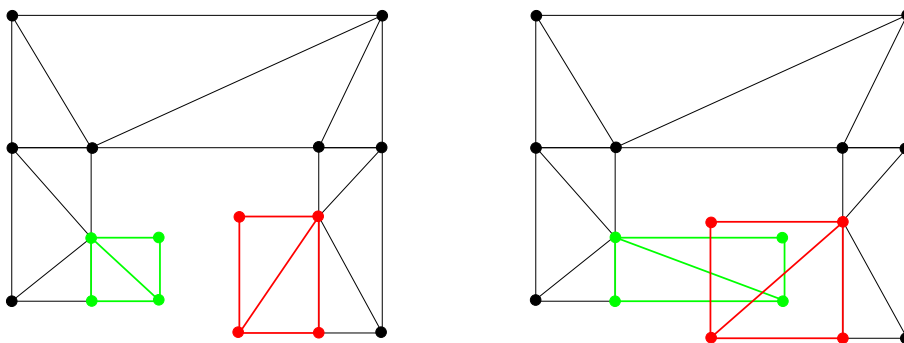


Abbildung 5.6: Ohne Konvexität ist lokale Konsistenz nicht hinreichend - hier führt eine lokal konsistente Winkelzuweisung auf der rechten Seite zum Überlappen der farbigen Facetten.

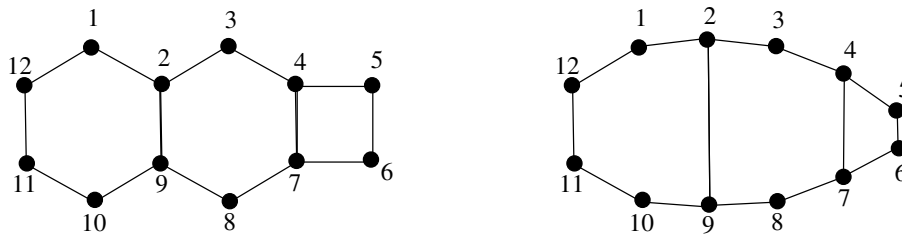


Abbildung 5.7: In einem rein konvexen Layout (rechts) ist die Ringstruktur deutlich schlechter zu erkennen

5.3.3 Einfache Verfahren zur Maximierung der Winkelauflösung

Falls kein uniformes Layout möglich ist möchte man natürlich das Ringsystem trotzdem zeichnen können. Hierbei ist man insbesondere an einer möglichst guten Winkelauflösung interessiert. Eine Möglichkeit, für die sich die Winkelauflösung effizient optimieren läßt, wäre es, ein Layout mit konvexem Rand von G zu konstruieren, für das die lokalen Konsistenzbedingungen ausreichen würden.

Allerdings sehen solche Layouts meist deutlich anders aus, als man es für chemische Strukturformeln erwarten würde, da insbesondere die Ringstruktur deutlich schlechter zu erkennen ist (s. Abb. 5.7). Insofern ist diese Methode für unser Problem bestenfalls von theoretischem Interesse.

Alternativ könnte man versuchen, solange uniform zu zeichnen wie es möglich ist. Stellt man beim Einfügen einer Kante fest, dass man eine Kantenkreuzung erhält, so wird die Kante entsprechend verkürzt oder der entsprechende Innenwinkel verändert. Der Nachteil dieser Methode ist, dass die Verzerrungen nicht gleichmäßig verteilt werden. Das Ergebnis hängt insbesondere stark vom Startknoten bei der Abarbeitung des Dualgraphen ab. Beginnt man hier beispielsweise im Zentrum, werden u.U. gerade am Rand die Ringe stark verzerrt, was meistens für den visuellen Eindruck eher ungünstig ist (s. Abb. 5.8).

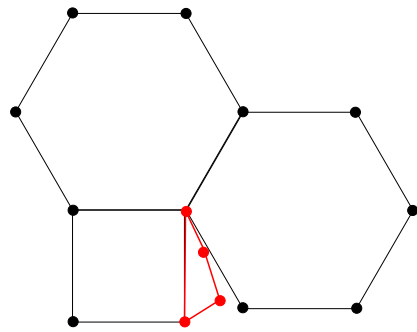


Abbildung 5.8: Bei ungünstiger Abarbeitungsreihenfolge werden Facetten am Rand u.U. stark verzerrt, wie es hier für die rote Facette der Fall ist

Wenn man auch weniger effiziente Verfahren in Betracht zieht, so kann man beispielsweise kräftebasierte Verfahren, sogenannte *Spring Embedder* anwenden. Ein Beispiel für einen solchen Embedder, der vorgegebene Einbettungen erhält, wird im nächsten Abschnitt vorgestellt.

5.3.4 Einbettungserhaltender Spring Embedder

Spring Embedder versuchen ein Layout mit Hilfe physikalischer Analogien (z.B. durch Simulation wirkender Kräfte) zu finden¹. Im Allgemeinen ist nicht garantiert, dass ein Spring Embedder eine gegebene Einbettung erhält. Will man dies sicherstellen, so muss man zusätzlichen Aufwand betreiben. In [Ber99] wird von F. Bertault ein Verfahren vorgestellt, wie man dies erreichen kann. Dazu werden jedem Knoten v acht Sektoren (s. Abb. 5.9) zugewiesen, und es wird berechnet, wie weit sich der Knoten maximal in jedem dieser Sektoren bewegen kann, ohne dass die Einbettung sich ändert, also Kantenkreuzungen verschwinden oder neu hinzukommen. Dabei genügt es, für jede Zone $Z_0(v), \dots, Z_7(v)$ den maximalen Radius $R_i(v)$ zu bestimmen, in dem sich der Knoten

¹s. auch [KW01],[BETT99]

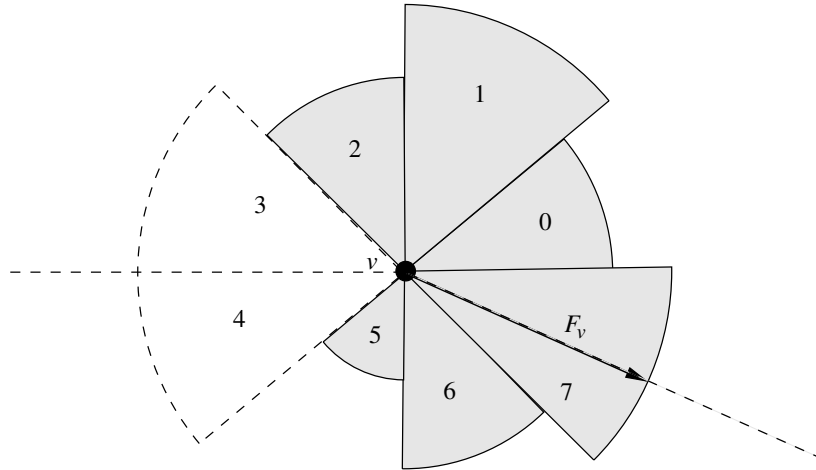


Abbildung 5.9: Die Zonen und Kraftbeschränkung im Algorithmus von Bertault

sicher bewegen kann. Beim Positionieren der Knoten wird dann bestimmt, in welche Richtung der Knoten durch die Kraft $F(v)$ bewegt wird, und ggf. wird $F(v)$ so skaliert, dass v die entsprechende Zone nicht verlässt. Zu Beginn eines Iterationsschrittes werden alle Zonen aller Knoten mit ∞ initialisiert, und dann die Zonen mit Algorithmus 6 bestimmt. Der Einfachheit halber sei die Position eines Knotens v ebenfalls mit v bezeichnet. Das Liniensegment zwischen a und b sei mit $[a, b]$ bezeichnet. $d(a, b)$ bezeichne den euklidischen Abstand zwischen a und b . Einen Korrektheitsbeweis für den Algorithmus findet man ebenfalls in [Ber99].

Die Lage der Zonen für die Fälle 1 und 2 ist in Abb. 5.10 gezeigt. Gegenüber [Ber99] wurden zwei Veränderungen vorgenommen: Zum einen wurde ein Mindestabstand zwischen Knoten und Kanten gefordert (Schritt 6), da sonst zwar dieser Abstand nie wirklich 0, aber betragsmäßig beliebig klein werden kann, was in der Praxis zu Rundungsfehlern und dann zu Änderungen der Einbettung führt. Außerdem wurde in Schritt 14-17 noch der Fall extra berücksichtigt, bei dem $v - i_v$ genau auf einer Sektorgrenze liegt. In diesem Fall wird der Vektor beiden anliegenden Sektoren zugerechnet, was zu Änderungen in zusätzlichen Zonen führt. Auf diese Weise werden wieder die Auswirkungen von Rundungsfehlern bei der Bestimmung der Projektion und des Winkels gemildert und die Symmetrie des Layouts erhöht.

Das Hauptproblem bei diesem Verfahren, wie allgemein bei Spring Embeddern, ist die schlechte Performance. Alleine für die Zonenbestimmung werden in jedem Iterationsschritt $O(|V||E|)$ Berechnungen benötigt, dazu kommt noch die Bestimmung der Kräfte selbst, die je nach Anwendung unterschiedlich aufwendig ist. Da bei den meisten Verfahren hierfür die Laufzeit auch quadratisch in der Größe des Graphen ist, ist das Verfahren relativ langsam. Hinzu kommt, dass das Verfahren hier stark zu Oszillationen neigt, vermutlich, weil die Bewegungsfreiheit der Knoten teilweise stark eingeschränkt wird. Hingegen ist es von Vorteil, dass die Zonenbestimmung vollkommen unabhängig von der Berechnung der Kräfte ist.

Für unser spezielles Problem müssen noch die Kräfte angegeben werden, die wirken sollen. Dies sind einerseits die üblichen Anziehungs- und Abstoßungskräfte für Knoten und Kanten, hier nach Fruchterman und Reingold[FR91]. Dabei wirken für jedes Paar von Knoten u und v Anziehungskräfte $F^a(u, v)$:

$$F^a(u, v) := \frac{d(u, v)}{\delta} (u - v) \quad (5.5a)$$

und Abstoßungskräfte $F^r(u, v)$:

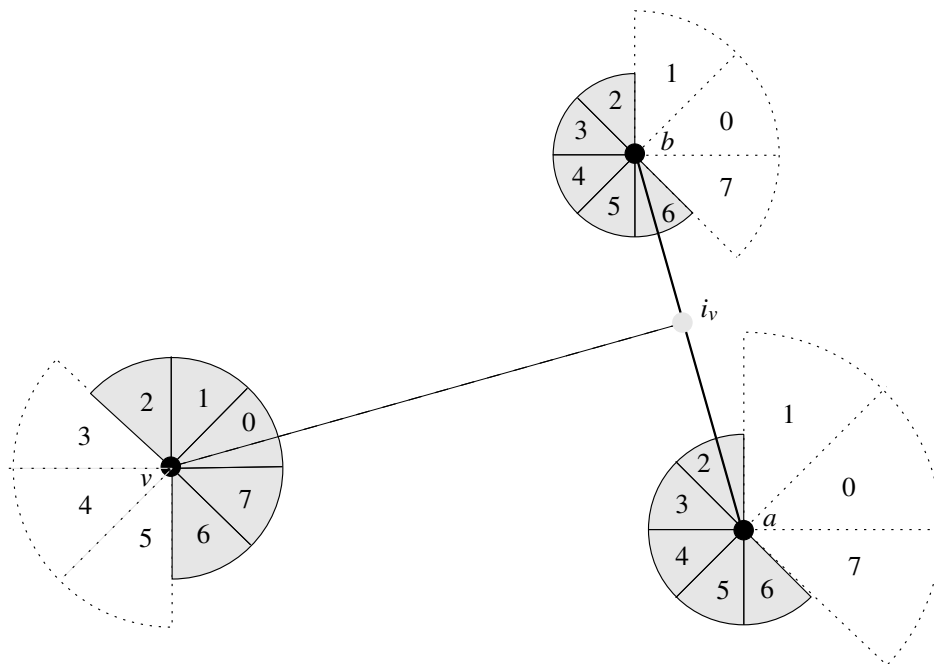
$$F^r(u, v) := \frac{\delta^2}{d(u, v)^2} (v - u) \quad (5.5b)$$

```

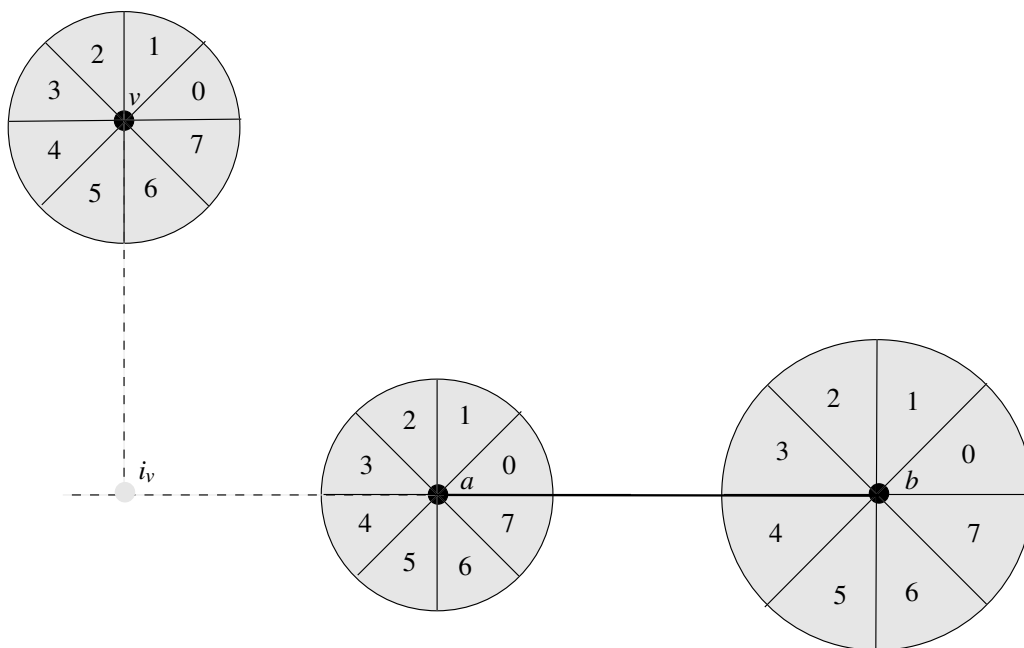
Input : Graph  $G = (V, E)$ , minimaler Abstand von Knoten zu Kante  $d_{min}$ 
Data : Für jeden Knoten 8 Zonen  $Z_0(v), \dots, Z_7(v)$ 
1 begin
2   Initialisiere alle Zonen mit  $\infty$ 
3   forall Knoten  $v \in V$  do
4     forall Kanten  $e = \{a, b\} \in E$  do
5       Bestimme die Projektion  $i_v$  von  $v$  auf die durch  $e$  bestimmte Gerade  $a + t(b - a)$ 
6        $d \leftarrow \max(d(v, i_v) - d_{min}, 0)$ 
7       /*Fall 1
8       if  $i_v \in [a, b]$  then
9         Bestimme die Zone  $Z_i(v)$ , in die  $i_v - v$  zeigt
10        for  $j = i + 6$  to  $s + 10$  do
11           $R_{j \bmod 8}(v) \leftarrow \min(R_{j \bmod 8}(v), d/3)$ 
12        for  $j = i + 2$  to  $s + 6$  do
13           $R_{j \bmod 8}(a) \leftarrow \min(R_{j \bmod 8}(a), d/3)$ 
14           $R_{j \bmod 8}(b) \leftarrow \min(R_{j \bmod 8}(b), d/3)$ 
15        if  $i_v - v$  liegt auf einer Sektorgrenze then
16           $R_{(i+11) \bmod 8}(v) \leftarrow \min(R_{(i+11) \bmod 8}(v), d/3)$ 
17           $R_{(i+7) \bmod 8}(a) \leftarrow \min(R_{(i+7) \bmod 8}(a), d/3)$ 
18           $R_{(i+1) \bmod 8}(b) \leftarrow \min(R_{(i+1) \bmod 8}(b), d/3)$ 
19        else
20          /*Fall 2
21          for  $j = 0$  to  $7$  do
22             $R_j(v) \leftarrow \min(R_j(v), \min(d(a, v), d(b, v))/3)$ 
23             $R_j(a) \leftarrow \min(R_j(a), d(a, v)/3)$ 
24             $R_j(b) \leftarrow \min(R_j(b), d(b, v)/3)$ 
25          end
26        end
27      end

```

Algorithmus 6 : Zonenbestimmung im Algorithmus von Bertault



(a) Zonenbestimmung: Fall 1



(b) Zonenbestimmung: Fall 2

Abbildung 5.10: Zur Zonenbestimmung im Algorithmus von Bertault

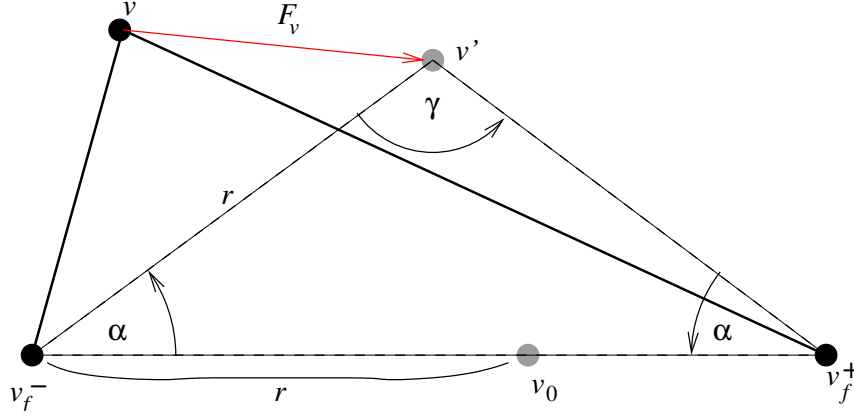


Abbildung 5.11: Die Bestimmung der Winkelausgleichskräfte

wobei δ die gewünschte Kantenlänge ist².

Außerdem wirken noch Kanten-Knoten-Abstoßungskräfte $F^e(v, \{a, b\})$ zwischen Paaren $v \in V$, $e = \{a, b\} \in E$:

$$F^e(v, \{a, b\}) := \begin{cases} -\frac{(\gamma - d(v, i_v))^2}{d(v, i_v)}(i_v - v) & \text{falls } i_v \in [a, b], d(i_v, v) < \gamma, v \neq a, v \neq b \\ 0 & \text{sonst} \end{cases} \quad (5.5c)$$

wobei γ ein geeignet zu wählender Parameter ist, und i_v wieder die Projektion von v auf $a + t(b - a)$ bezeichne. Für die Berechnung der Kräfte ist die Laufzeit in einem Iterationsschritt $O(|V|^2)$ für F^a bzw. F^r und $O(|V||E|)$ für F^e

Zusätzlich sollen für unser Problem noch Winkelausgleichskräfte wirken, durch die versucht wird, den Winkel zwischen zwei zur selben Facette gehörigen Kanten möglichst an den gewünschten Innenwinkel anzupassen. Sei dazu f eine Facette, v_f^- der unmittelbare Vorgänger, v_f^+ der unmittelbare Nachfolger von v in ∂f , so wird versucht, v so zu positionieren, dass v an der Spitze eines gleichschenkligen Dreiecks $v_f^- v v_f^+$ mit Spitzenwinkel γ liegt (s. Abb. 5.11). Gibt man γ vor, wirkt damit auf v die Kraft $F^w(v, v_f^-, v_f^+, \gamma)$:

$$F^w(v, v_f^-, v_f^+, \gamma) := v - \left(\frac{\sin \alpha}{\sin \gamma} \begin{pmatrix} \cos \alpha & \sin \alpha \\ -\sin \alpha & \cos \alpha \end{pmatrix} (v_f^+ - v_f^-) + v_f^- \right) \quad (5.5d)$$

Hierbei gibt der Ausdruck in der Klammer die neue Position v des Knotens v an, diese ergibt sich durch Rotation von $v_0 = v_f^- + r \frac{v_f^+ - v_f^-}{|v_f^+ - v_f^-|}$ um v_f^- um den Winkel $\alpha := (\pi - \gamma)/2$ (s. Abb. 5.11).

Für einen Iterationsschritt wird damit für einen 2-fach zusammenhängenden planaren Graphen $O(|E|)$ benötigt: Jede Kante tritt bei der Berechnung in einer Facette einmal als Vorgänger- und einmal als Nachfolgerkante auf, und jede Kante gehört maximal zu zwei inneren Facetten.

Insgesamt wirken damit auf einen Knoten in einem Iterationsschritt

$$F(v) = c_1 \left(\sum_{\{u, v\} \in E} F^a(u, v) + \sum_{u \in V} F^r(u, v) + \sum_{\{a, b\} \in E} F^e(v, \{a, b\}) - \sum_{\substack{v \in V, w \in V \\ \{u, w\} \in E}} F^e(u, \{v, w\}) \right) + c_2 \sum_{\substack{f \in \mathcal{F} \setminus \{f_0\} \\ f \text{ inzident zu } v}} F^w(v, v_f^-, v_f^+, \pi - \frac{2\pi}{|\partial f|}) \quad (5.5e)$$

²In [Ber99] sind die Vorzeichen bei beiden Kräften gerade vertauscht, wodurch die Kräfte nicht korrekt wirken

wobei man mit c_1 und c_2 das Verhältnis zwischen den beiden Arten von Kräften einstellen kann. Abschließend wird F noch mit einem konstanten Dämpfungsfaktor δ skaliert, um exzessive Bewegungen zu vermeiden. Bewährt hat sich hier $\delta \in [0.05, 0.1]$.

Im Falle von außenplanaren Blöcken kann man als kreuzungsfreie Anfangseinbettung einfach das Kreislayout für G wählen, welches sich leicht bestimmen läßt wenn man $H(G)$ kennt. Die entstehenden Layouts sind allerdings nicht mehr zwingend konvex. Dies läßt sich beheben, wenn man für jede innere Facette f und jeden Knoten $v \in \partial F$ zusätzlich noch virtuelle Kanten $\{v_f^-, v_f^+\}$ einfügt, die lediglich für die Zonenberechnung verwendet werden (s. ABB 5.12). Damit wird verhindert, dass eine Facette hinterher nicht mehr konvex ist, weil diese Kanten nicht überschritten werden können. Zum Abschluss wird der Block dann noch so skaliert, dass seine mittlere Kantenlänge der vorgegebenen Kantenlänge entspricht.

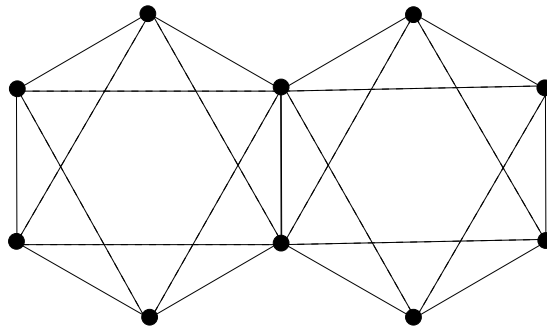
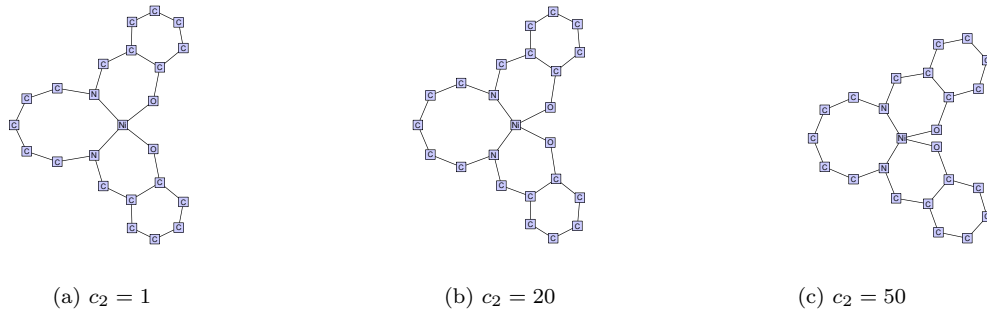


Abbildung 5.12: Virtuelle Kanten, um Konvexität sicherzustellen

5.4 Fazit und Auswertung

Zuerst wurde untersucht, wie viele der außenplanaren Graphen aus dem Testdatensatz des NCI keine uniforme Einbettung zulassen. Erstaunlicherweise ist dies lediglich bei 65 Substanzen der Fall, von denen 28 paarweise nichtisomorph sind. Insofern ist nachträglich die Überlegung gerechtfertigt, für diesen Fall einen weniger effizienten Algorithmus zu wählen. Außerdem wurde untersucht, wie oft sich bereits mit den Heuristiken 1 und 2 entscheiden läßt, ob ein uniformes Layout möglich ist. Im Testdatensatz treten insgesamt 399449 außenplanare Ringsysteme auf, davon ließ sich lediglich bei 1221 Ringsystemen diese Frage nicht mit den Heuristiken entscheiden. Dies ist nicht besonders überraschend, wenn man die Statistiken aus Kapitel 4 betrachtet, da die meisten Ringsysteme zu klein sind, um problematisch sein zu können.

Weiter wurde untersucht, wieviel Iterationen Algorithmus 6 mit den gewählten Kräften im Schnitt auf den Substanzen im Testdatensatz benötigt. Dies hängt natürlich stark von der Wahl der Parameter ab, wählt man $c_1 = 1$, $c_2 = 10$, werden im Schnitt zwischen 15 und 150 Iterationen benötigt, wobei die Ringsysteme zwischen 11 und 53 Knoten haben. Die Berechnung der größten Substanzen dauert damit auf dem Testsystem (AMD Athlon 2400) ca. 2 Sekunden, wobei der gesamte Algorithmus in Java implementiert wurde, und keinerlei weitere Optimierungen vorgenommen wurden. Interessanterweise benötigt das Verfahren ganz ohne Winkelausgleichskräfte meistens deutlich mehr Iterationen bei (subjektiv) schlechterer Layoutqualität. Erhöht man c_2 stark, zum Beispiel auf 50, so wird die Qualität der Layouts etwas besser, gleichzeitig nimmt die Anzahl der Iterationen aber auch stark zu. Derselbe Effekt tritt ein, wenn man nur die Winkelausgleichskräfte verwendet, also $c_1 = 0$ setzt, was den potentiellen Laufzeitvorteil durch die bessere Laufzeit für die Kraftberechnung wieder zunichte macht. Einige Auswirkungen der Wahl von c_2 auf die Layoutqualität sieht man in Abb. 5.13. Es fällt insbesondere auf, dass bei größerem c_2 die Ringe deutlich weniger verzerrt sind.

Abbildung 5.13: Abhängigkeit von c_2

Insgesamt hat man also (zumindest für diesen Testdatensatz) einen Algorithmus, der auf dem weitaus größten Teil der außenplanaren Graphen effizient entscheidet, ob ein uniformes Layout existiert und dieses ggf. dann auch bestimmt. Für Substanzen ohne uniformes Layout existiert immerhin noch ein Algorithmus, der ein außenplanares (mit Modifikationen sogar konvexes) Layout mit akzeptabler Winkelauflösung bestimmt.

5.5 Optimierungen und Ausblicke

Am eigentlichen Zeichenalgorithmus für den uniformen Fall kann man sicherlich wenig weiter optimieren. Wenn man (auch im Hinblick auf andere Testdatensätze) die Effizienz verbessern will, so wird dies sicherlich auf dem Gebiet der Winkelmaximierung geschehen müssen. Es bleibt eine entscheidende Frage, ob es für nichtkonvexe außenplanare Graphen evtl. hinreichende Kriterien für die Realisierbarkeit von Winkelzuweisungen gibt, die sich zudem noch in ein effizient lösbares Optimierungsproblem umsetzen lassen (dies ist nicht einmal für die oben betrachtete Klasse von 3-fach zusammenhängenden Dreiecksgraphen der Fall). Nicht ausgenutzt wurde beispielsweise, das man aus dem Dualgraphen, der hier ein Baum ist, eventuell noch Bedingungen gewinnen kann.

Zur Verbesserung des Algorithmus von Bertault kann man primär die üblichen Techniken für Spring Embedder einsetzen, wobei hier insbesondere an die Implementation einer Oszillationserkennung zu denken ist. Die Anzahl der Zonenberechnungen zu minimieren lohnt sich wenig, solange man nicht auch die Effizienz der Kräfteberechnungen erhöht. Nachteilig wie bei vielen Spring Embeddern ist, dass die Anzahl der Iterationen und die Qualität des Layouts durch mehrere Parameter bestimmt ist, die sich nicht immer allgemeingültig für alle Probleminstanzen einstellen lassen. Die Kombination aus Winkelausgleichskräften und Kreuzungsvermeidung hat sich jedoch als recht stabil auch gegenüber größeren Änderungen der frei wählbaren Parameter erwiesen, selbst bei extremen Einstellungen von c_2 gegenüber c_1 ist das Layout noch akzeptabel.

Zur Übertragung auf nicht außenplanare Graphen sei hier noch kurz herausgestellt, wo die Außenplanarität oben überhaupt eingegangen ist:

- Die Außenplanarität ermöglichte eine effiziente Bestimmung einer minimalen Zykelbasis, also der Elemente, die die Bausteine des Layouts bilden. Insbesondere musste man sich um Fragen der Eindeutigkeit keine Gedanken machen, weil diese automatisch erfüllt ist, und die auftretende minimale Zykelbasis ist planar, was die Existenz eines ringtreuen Layouts sicherstellt. Das Verfahren läßt sich im Wesentlichen direkt auf diejenigen Fälle planarer Graphen verallgemeinern, bei denen \mathcal{M} eindeutig bestimmt und planar ist. Für nicht außenplanare, aber planare Graphen kann eine minimale Zykelbasis in $(O(|E|^2 + |V| \log |V|))$ bestimmt werden [Vis97]. Will man alle relevanten Zyklen, also $\mathcal{R}(\mathcal{G})$ bestimmen, so besitzt der beste bekannte Algorithmus eine Laufzeit von $O(\Delta^4 |V|^4) + O(|V| |\mathcal{R}(\mathcal{G})|)$, wobei Δ der Maximalgrad in G ist.

- Der Dualgraph außenplanarer Graphen ist ein Baum. Dies erleichtert insbesondere die Arbeit bei der inkrementellen Erstellung des Layouts.
- Für den Algorithmus von Bertault konnte das Kreislayout als initiales Layout gewählt werden. Zwar gibt es effiziente Algorithmen, um für beliebige planare Graphen eine planare Einbettung zu bestimmen, allerdings ist diese dann im Allgemeinen nicht mehr eindeutig durch die Wahl der äußeren Facette bestimmt.

Kapitel 6

Baumlayout

Als zweites Teilproblem soll jetzt das Layout der in Molekülen auftretenden Baumstrukturen behandelt werden. Da man nicht jeden Baum gleichzeitig längen- und winkeluniform zeichnen kann (wie man leicht an der Familie vollständiger ternärer Bäume wachsender Höhe sieht), muss man also im allgemeinen einen Kompromiss zwischen beiden Kriterien eingehen. Wie man in Kap. 4 gesehen hat, ist der Knotengrad typischerweise gering (meist kleiner als 6, und im Schnitt nicht größer als 4), und die Graphen und damit die auftretenden Baumstrukturen sind nicht sehr groß. Im Folgenden wird zuerst untersucht, inwieweit sich die bekannten Layout-Algorithmen für Bäume zur Lösung der typischerweise bei Strukturformeln auftretenden Anforderungen eignen. Im Anschluss daran wird ein alternativer Ansatz vorgestellt, der versucht, gezielt winkeluniforme Layouts zu erstellen.

6.1 Definitionen und allgemeine Designkriterien

Neben den üblichen Kriterien für einen Layoutalgorithmus (hier insbesondere Minimierung des Platzbedarfs, möglichst gleiche Kantenlängen, gute Winkelauflösung) ist bei Bäumen insbesondere eine hohe Effizienz entscheidend. Ziel fast aller Algorithmen ist eine Laufzeit, die linear in der Größe des Baumes ist. Wünschenswert ist insbesondere, die Layoutberechnung während eines geordneten Durchlaufs durch den Baum vornehmen zu können. Dies und die rekursive Definition von Bäumen (s. Def. 6.3) führt üblicherweise zu einem *Divide-and-Conquer*-Ansatz für den Layoutalgorithmus.

Im Folgenden sei kurz auf die übliche Terminologie in diesem Zusammenhang eingegangen:

Definition 6.1 (Baum). Ein **Baum** ist ein zusammenhängender kreisfreier Graph. Ein **Wurzelbaum** T ist ein Baum mit einem ausgezeichneten Knoten $r \in T$, der sogenannten **Wurzel**. Ein **freier Baum** ist ein Baum ohne eine ausgezeichnete Wurzel.

Die folgenden Definitionen sind nur für Wurzelbäume sinnvoll:

Definition 6.2. In einem Wurzelbaum lassen sich alle Kanten so ordnen, dass die Wurzel der einzige Knoten ohne eingehende Kanten ist, und alle anderen Knoten genau eine eingehende Kante haben. In diesem Fall heißt für jede gerichtete Kante $(u, v) \in T$ u **Elternknoten** von v und v **Kind(knoten)** von u . Ein Knoten v heißt **Nachfolger** von u , wenn es einen gerichteten Weg von u nach v gibt. Dies liefert eine partielle Ordnung auf den Knoten von T . Der transitive Abschluss der Nachfolgerrelation bzgl. u heißt **Teilbaum** $T(u)$ an/unter u . Ein Knoten ohne Kinder heißt **Blatt**. In einem Wurzelbaum ist die **Tiefe** $t(v)$ eines Knotens v definiert als die Länge des (eindeutig bestimmten) Weges von der Wurzel zu v . Die **Höhe** $h(T)$ eines Wurzelbaumes ist die maximale Tiefe aller Blätter. Ein **n -ärer Baum** (oder **Baum der Ordnung n**) ist ein Wurzelbaum, in dem jeder Knoten höchstens n Kinder hat. Er ist **vollständig n -är**, wenn jeder Knoten bis auf die Blätter genau n Kinder hat.

Wie schon angedeutet, lassen sich n -äre Bäume auch rekursiv definieren:

Definition 6.3 (rekursive Definition von Bäumen). Ein n -ärer Baum ist entweder leer oder besteht aus einer Wurzel r mit n Teilbäumen T_1, \dots, T_n .

Im Folgenden sei für einen Graphen $G = (V, E)$ der **Abstand** $d(u, v)$ zweier Knoten u und v die Länge eines kürzesten Weges zwischen u und v , wenn ein solcher existiert, sonst ∞ .

Definition 6.4 (Radius, Zentrum). Die **Exzentrizität** $e(u)$ eines Knotens u ist definiert als

$$e(u) := \max_{v \in V} d(u, v) \quad (6.1a)$$

Der **Radius** $r(G)$ von G ist definiert als

$$r(G) := \min_{v \in V} e(v) \quad (6.1b)$$

Das **Zentrum** $Z(G)$ von G besteht aus allen Knoten v mit $e(v) = r(G)$.

Satz 6.5 (Jordan/Sylvester). *Das Zentrum jedes Baumes besteht aus einem Knoten oder zwei adjazenten Knoten.*

Wählt man in einem Baum einen Knoten aus dem Zentrum als Wurzel, so hat der entstehende Wurzelbaum offenbar minimale Höhe. In den folgenden Layoutalgorithmen, die auf Wurzelbäumen arbeiten, wählt man daher häufig einen Zentrumsknoten als Wurzel, wenn lediglich einen freien Baum gegeben hat.

6.2 Hierarchische Layoutalgorithmen

6.2.1 Lagenlayout

Da Bäume häufig hierarchische Strukturen modellieren, ist es üblich, ein sogenanntes **Abwärts-layout** zu verwenden, d.h. dass jeder Knoten oberhalb seiner Nachfolger platziert wird. Fordert man zusätzlich für die Koordinaten $x(v), y(v)$ der Knoten, dass $y(v) = -t(v)$ für alle $v \in V$ gilt, so heißt das entstehende Layout **tiefengeschichtet** oder **Lagenlayout**. Insbesondere liegen dann alle Knoten mit derselben Tiefe auf einer horizontalen Linie. Der Einfachheit halber betrachten wir im Folgenden nur Binärbäume, wobei sich die Resultate aber in naheliegender Weise auf beliebige n -äre Bäume verallgemeinern lassen (s. dazu beispielsweise [BETT99]).

Der folgende Satz enthält einen einfachen Layoutalgorithmus für Binärbäume:

Satz 6.6. *Sei T ein Binärbaum und v_1, \dots, v_n eine Ordnung seiner Knoten in inorder-Reihenfolge. Dann liefert die Koordinatenzuweisung*

$$x(v_i) := i, \quad y(v_i) := -t(v_i) \quad (6.2)$$

ein kreuzungsfreies Lagenlayout. Dieses kann in Linearzeit bestimmt werden.

Bemerkung 6.7. Dieses Layout hat die zusätzliche Eigenschaft, dass alle Knoten ganzzahlige Koordinaten haben, es handelt sich also um ein sog. **Gitterlayout**.

Dieses einfache Layout hat jedoch einige Nachteile:

- Die Layouts sind im Allgemeinen unnötig breit, genauer: haben stets die Breite n .
- Kanten können die Länge $\Theta(n)$ haben.
- Knoten sind im Allgemeinen nicht über ihren Nachfolgern zentriert.

Der **Algorithmus von Tilford und Reingold**[RT81] versucht diese Nachteile zu beheben, indem einerseits für die Teilbäume unter einem Knoten jeweils nur relative Koordinaten berechnet werden, und andererseits versucht wird, die Teilbäume anhand ihrer **Konturen** möglichst platzsparend auszulegen.

Definition 6.8. Die **linke Kontur** eines Binärbaums der Höhe h ist die Folge von Knoten v_0, \dots, v_h mit der Eigenschaft, dass v_i der am weitesten links liegende Knoten der Tiefe i ist. Die rechte Kontur ist analog definiert.

Der Algorithmus von Tilford und Reingold durchläuft den Baum T zweimal, einmal in *postorder*, um die Konturen und die x -Offsets zu bestimmen, dann in *preorder*, um aus den x -Offsets die absolute Koordinate zu bestimmen. Eine detaillierte Beschreibung kann man beispielsweise wieder in [BETT99] finden. Der Algorithmus läuft in Linearzeit ab und liefert ein Layout Γ mit folgenden Eigenschaften:

- Γ ist ein kreuzungsfreies Lagenlayout.
- Γ ist einbettungserhaltend und legt identische Teilbäume gleich aus.
- Je zwei Knoten von T haben horizontal und vertikal mindestens Abstand 1.
- Knoten sind über ihren Nachfolgern zentriert.
- Der Flächenbedarf von Γ ist $O(n^2)$.

Das Layout ist allerdings nicht notwendigerweise breitenminimal. Für rationale Koeffizienten gibt es Algorithmen, die ein Layout mit obigen Eigenschaften und minimaler Breite in polynomialer Laufzeit liefern. Fordert man jedoch zusätzlich ein Gitterlayout, ist das Problem \mathcal{NP} -schwer.

Der Algorithmus lässt sich leicht auf beliebige n -äre Bäume verallgemeinern, indem man mit obigem Verfahren alle nebeneinanderliegenden Teilbäume platziert und zum Schluss die Wurzel mittig anordnet.

6.2.2 Radiallayout

Eine Alternative, die auf dem vorherigen Abschnitt basiert und häufig für freie Bäume verwendet wird, ist das **Radiallayout**. Hierbei wird jeder Knoten eines Baumes auf einem Kreis platziert, dessen Radius proportional zur Tiefe des Knotens ist. Jeden Teilbaum von v könnte man nun komplett in einem Kreissektor W_v zeichnen, dessen Winkel proportional zur Größe des

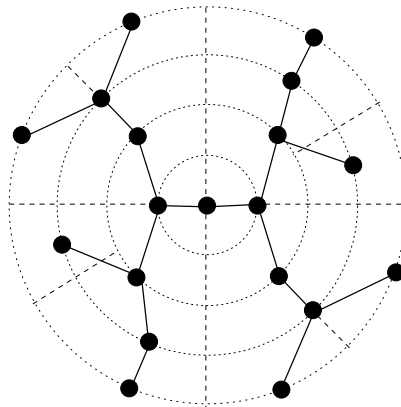
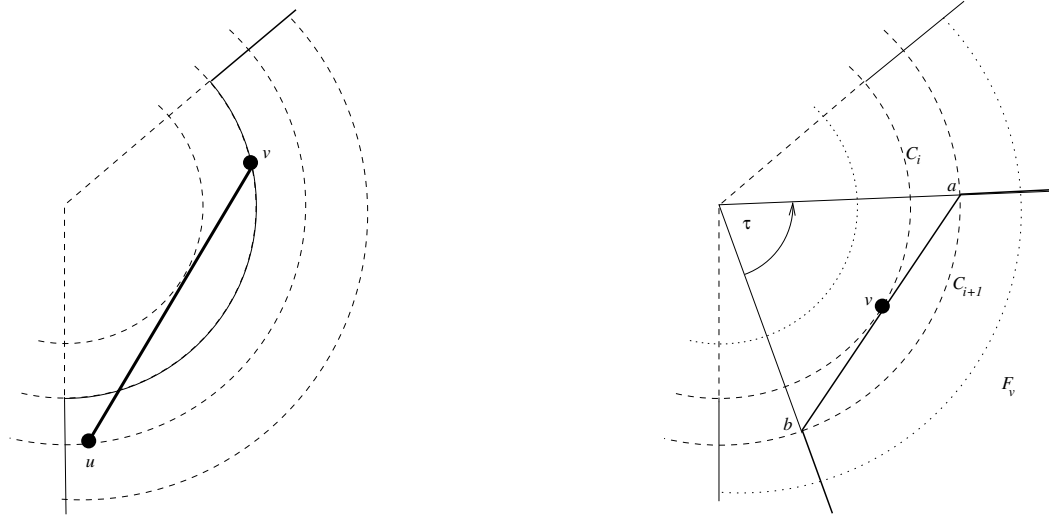


Abbildung 6.1: Radiallayout eines Binärbaumes



(a) Eine Kante kann den ihr zugewiesenen Sektor verlassen, obwohl beide Endknoten darin liegen

(b) Konvexe Teilmenge eines Sektors

Abbildung 6.2: Winkelbestimmung im Radiallayout

Teilbaums ist (s. Abb. 6.1). Dies kann allerdings zu Kantenkreuzungen führen, da eine Kante den Sektor verlassen kann, obwohl beide Endknoten in dem Sektor liegen (s. Abb. 6.2(a)). Um dies zu verhindern, wird die verfügbare Region für jeden Teilbaum auf eine *konvexe Teilmenge* dieses Sektors beschränkt. Liege dazu v auf dem Kreis C_i , und die Tangente an C_i durch v schneide C_{i+1} in a und b , dann beschränken wir $T(v)$ auf die durch die Strecke ab und die beiden Radien durch a und b eingefasste Region F_v (s. Abb. 6.2(b)). Der Winkel γ für F_v lässt sich einfach berechnen durch $\cos(\gamma/2) = \frac{\rho_i}{\rho_{i+1}}$.

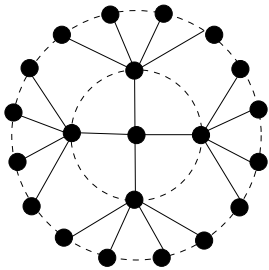


Abbildung 6.3: Radiallayout eines vollständigen Baums der Ordnung 4

Mit den Verfahren aus dem letzten Abschnitt lässt sich dann leicht ein Algorithmus für ein Radiallayout entwerfen, der ebenfalls in Linearzeit abläuft. Das entstehende Layout ist in dem Sinne hierarchisch, dass alle Kanten von der Wurzel aus strikt nach außen zeigen (s. Abb. 6.1). Durch Veränderung der Winkelzuteilungsstrategie für die Teilbäume erhält man verschiedenen Variationen des Algorithmus (z.B. kann man statt der Größe der Teilbäume die Anzahl der Blätter wählen etc., s. [BETT99]). Bei festem Lagenabstand beträgt der Platzbedarf für N -äre Bäume offenbar einfach $O(h(T)^2)$. Allerdings liegt in diesem Fall die Winkelauflösung im worst-case Fall in $O(1/N^{h(T)})$, da bei vollständigen Bäumen der Ordnung N die Anzahl der Kinder exponentiell mit der Höhe zunimmt, der verfügbare Radius für jede Tiefe aber nur linear wächst (s. Abb. 6.3).

6.2.3 Bewertung

Ein großer Vorteil beider Verfahren ist ihre Effizienz sowie die relativ unkomplizierte Implementation. Zudem liefern beide Verfahren ein relativ kompaktes (wenn auch nicht notwendig platzminimales) Layout. Nachteilig für unser Problem ist hingegen die Ausrichtung auf eine hierarchische

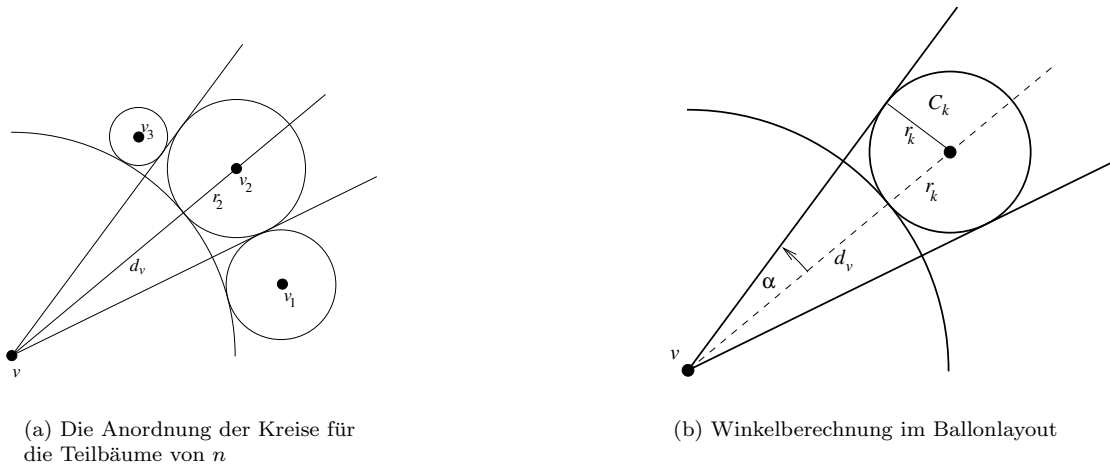


Abbildung 6.4: Bestimmung der relevanten Größen im Ballonlayout

Struktur, was besonders im Lagenlayout auffällt. Dies entspricht nicht den üblichen Konventionen für die Darstellung chemischer Strukturformeln. Das Radiallayout eignet sich hierfür prinzipiell besser, allerdings ist auch hier die hierarchische Struktur deutlich erkennbar. Die entstehenden Layouts sind im Allgemeinen auch nicht einmal für Binärbäume winkeluniform. Bei festem Lagenabstand läßt sich die Winkelauflösung bei n -ären Bäumen offenbar beliebig klein machen, was besonders nachteilig ist.

6.3 Ballon-Layout

Ein anderes Verfahren, das keine Abwärtslayouts erzeugt, ist das sogenannte **Ballonlayout**. Dieses Verfahren besitzt im Allgemeinen eine relativ gute Winkelauflösung, und mit Modifikationen kann sogar Winkeluniformität erreicht werden. Allerdings wird diese Eigenschaft mit einem (im worst-case) exponentiellen Platzverbrauch erkauft.

Die wesentliche Eigenschaft dieses Layouts ist, dass alle Knoten eines Teilbaums innerhalb eines Kreises liegen, und die unmittelbaren Nachfolger von v so platziert werden, dass die zugehörigen Kreise konstanten Abstand von v haben. Ein Implementation, die das Layout ohne Umwege über 3D-Repräsentationen berechnet, wird beispielsweise in [MH98] beschrieben. Sei dazu $v \in T$ ein Knoten mit Kindern v_1, \dots, v_n . Außerdem soll jeder Teilbaum $T(v_i)$, $i = 1, \dots, n$ komplett in einem Kreis C_i mit Radius r_i liegen, wobei v_i gerade im Mittelpunkt von C_i liegt. Ferner sei $d_v := \max\{r_1, \dots, r_n\}$ der Abstand der Kreisränder von v , d.h. es gilt $d(v, v_i) = d_v + r_i$ (s. Abb. 6.4(a)). Wie man in Abb. 6.4(b) sieht, benötigt C_i dann den Winkel $2\alpha_i$, wobei $\alpha_i = \arcsin\left(\frac{r_i}{d_v + r_i}\right)$. Bei diesem Vorgehen kann es allerdings, abhängig von d_v und den einzelnen r_i , geschehen, dass der zur Verfügung stehende Platz nicht ausreicht, dass also $\sum_{i=1}^n \alpha_i > \pi$ gilt. In diesem Fall werden die Kreise C_i mit einem gemeinsamen Faktor c skaliert, wählt man $c = \frac{\pi}{\sum \alpha_i}$, so kann man die neuen Radien r'_i , $i = 1, \dots, n$ folgendermaßen berechnen:

$$r'_i = \frac{\sin c\alpha_i}{1 - \sin c\alpha_i} d_v$$

Im anderen Fall, wenn also noch Platz zur Verfügung steht, wird dieser gleichmäßig zwischen alle Kindkreise verteilt.

Damit kann man einen rekursiven Algorithmus zur Berechnung des Ballonlayouts eines Wurzelbaumes $T(r)$ angeben. Wie die vorherigen Algorithmen benötigt auch dieses Verfahren zwei

Durchläufe: Zuerst werden in **postorder** die Radii, Skalierungsfaktoren, Abstände und relativen Koordinaten berechnet, wobei die eigentliche Arbeit in den Routinen **adjustChildren** und **setRadius** geschieht. Danach werden in einem **preorder**-Durchlauf die absoluten Koordinaten bestimmt. Für jeden Knoten v werden insgesamt gespeichert:

- d_v : Die Distanz der Kindkreise
- r_v : Der Radius des Kreises C_v , in dem der Teilbaum $T(v)$ liegt
- α_v : Der von dem Kreis C_v belegte Winkel am Elternknoten
- c_v : Der evtl. nötige Skalierungsfaktor für die Kindkreise von v
- f_v : Der noch zur Verfügung stehende freie Platz an v
- x_v, y_v : Koordinaten von v .

```

postorder(Knoten  $v$ )
begin
  |  $d_v \leftarrow 0$ 
  | forall Kinder  $k$  von  $v$  do
  |   | postorder( $k$ )
  |   |  $d_v \leftarrow \max(d_v, r_k)$ 
  |   | adjustChildren( $v$ )
  |   | setRadius( $v$ )
end

```

Prozedur **postorder**(*Knoten* v)

```

adjustChildren(Knoten  $v$ )
begin
  |  $s \leftarrow 0$ 
  | forall Kinder  $k$  von  $v$  do
  |   |  $\alpha_k \leftarrow \arcsin \frac{r_k}{r_k + d_v}$ 
  |   |  $s \leftarrow s + \alpha_k$ 
  |   | if  $s > \pi$  then
  |   |   | /*Zu viel Platz belegt */
  |   |   |  $c_v \leftarrow \pi/s$ 
  |   |   |  $f_v \leftarrow 0$ 
  |   | else
  |   |   | /*Platz übrig, es muss nichts skaliert werden */
  |   |   |  $c_v \leftarrow 1$ 
  |   |   |  $f_v \leftarrow \pi - s$ 
end

```

Prozedur **adjustChildren**(*Knoten* v)

```

setRadius(Knoten  $v$ )
Data :  $m$ : Minimaler Radius, den ein Kreis belegen soll
begin
  |  $r_v \leftarrow \max(d_v, m) + 2d_v$ 
end

```

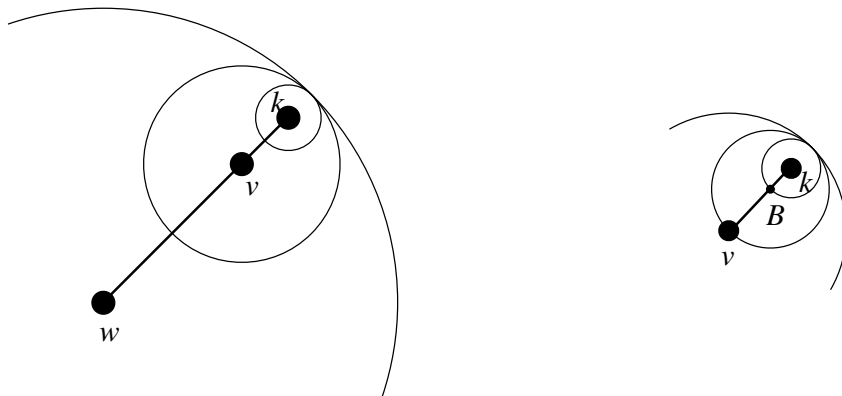
Prozedur **setRadius**(*Knoten* v)

```

preorder(Knoten  $v$ , double  $x$ , double  $y$ , double  $\lambda$ , double  $\theta$ )
begin
  /*Weise  $v$  absolute Koordinaten zu */
   $x_v \leftarrow x$ 
   $y_v \leftarrow y$ 
  /*Skaliere die Distanz, falls nötig; wird von den Eltern aus akkumuliert */
   $d' \leftarrow \lambda \cdot d_v$ 
  /*Verteile den freien Platz auf alle Kindknoten, und bestimme den absoluten
  Startwinkel, von dem aus die Rotationen gemessen werden */
  if  $v$  ist Wurzel von  $T$  then
     $\phi \leftarrow 0$ 
     $free \leftarrow \frac{f_v}{\deg(v)}$ 
  else
    /*Die Nullrichtung ist entgegengesetzt der Richtung der Eingangskante */
     $\phi \leftarrow \theta - \pi$ 
    /*Der Knoten hat  $\deg(v) - 1$  Kinder */
     $free \leftarrow \frac{f_v}{\deg(v)-1}$ 
    boolean  $firstVisit \leftarrow \text{TRUE}$ 
  forall Kinder  $k$  von  $v$  do
    /*Skaliere, falls nötig */
     $\alpha' \leftarrow c_v \cdot \alpha_v$ 
     $r' \leftarrow d_v \frac{\sin \alpha'}{1 - \sin \alpha'}$ 
    if  $firstVisit$  then
      /*Beim ersten Kind fällt freier Platz nur einmal an */
       $\phi \leftarrow \alpha' + free$ 
       $firstVisit \leftarrow \text{FALSE}$ 
    else
      /*sonst zweimal */
       $\phi \leftarrow \alpha' + 2free$ 
    /*Bestimme relative Koordinaten von  $k$  bzgl.  $v$  durch Rotation */
     $x_k \leftarrow (\lambda r' + d') \cos \phi$ 
     $y_k \leftarrow (\lambda r' + d') \sin \phi$ 
    /*Absolute Koordinaten, akkumulierten Skalierungsfaktor und absoluten
    Eingangswinkel weitergeben */
    preorder( $k, x_k + x_v, y_k + y_v, \lambda \frac{r'}{r_k}, \phi$ )
    /*Nochmal um die zweite Hälfte des belegten Sektors weiterdrehen */
     $\phi \leftarrow \phi + a'$ 
  end

```

Prozedur `preorder(Knoten v , double x , double y , double λ , double θ)`



(a) Der exzessive Platzverbrauch im Ballonlayout fällt besonders bei langen Ketten auf

(b) Platzeinsparung durch Schwerpunktlayout, der Kreis für v wird um B zentriert

Abbildung 6.5: Platzverbrauch im Ballonlayout

Das Gesamtlayout von $T(r)$ wird dann einfach berechnet durch Algorithmus 11:

```

Input : Ein Wurzelbaum  $T(r)$  mit  $n$  Knoten
1 begin
2   postorder( $r$ )
3   preorder( $r$ , 0, 0, 1, 0)
4 end

```

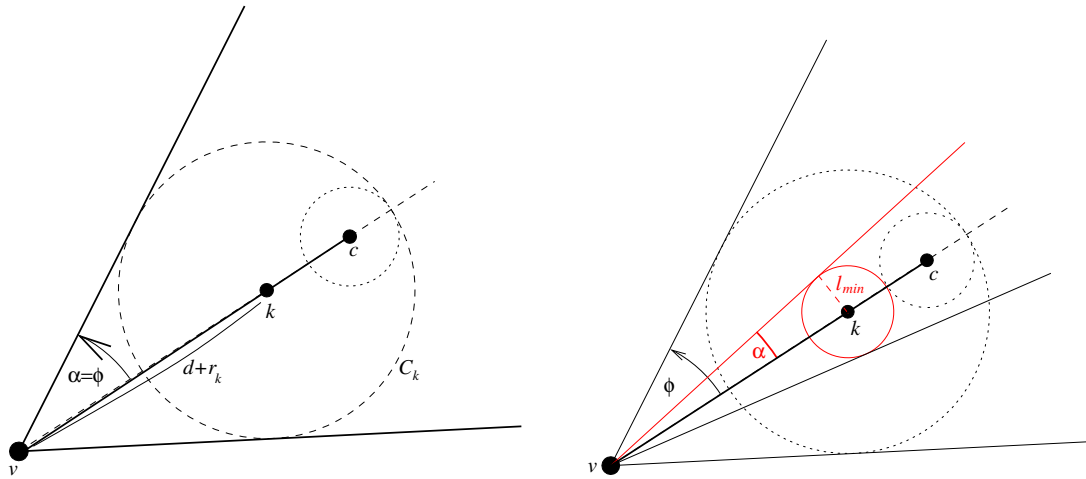
Algorithmus 11 : Ballonlayout

Der Algorithmus arbeitet offenbar in $O(|V|)$, da bei beiden Durchläufen zu jedem Knoten lediglich die unmittelbaren Kinder betrachtet werden. Ebenfalls weist der Algorithmus eine relativ gute Winkelauflösung auf: sofern die jeweiligen Teilbäume an den Knoten ähnliche Höhen haben, weichen die belegten Winkel auch nicht sehr voneinander ab. Allerdings kann keine Winkeluniformität garantiert werden.

Sehr nachteilig ist hingegen, dass die Kantenlänge exponentiell mit der aktuellen Tiefe des Baums abnimmt, was an der Konstruktion der Radii liegt (dort wird in jedem Schritt mit dem Faktor 3 multipliziert). Dies ist nicht überraschend, da die Anzahl der Knoten (und damit die belegte Fläche) im schlimmsten Fall exponentiell mit der Höhe des Baumes wächst. Allerdings wird bei der hier angewandten Strategie viel Platz verschwendet, indem die Kreise oft unnötig groß gewählt werden. Besonders eklatant fällt dies bei Bäumen niedriger Ordnung und/oder mit langen Ketten auf, wie sie in chemischen Strukturformeln oft vorkommen (s. Abb. 6.5(a)). In [MH98] wird dazu eine alternative Platzierungsstrategie vorgeschlagen, bei der das Zentrum des Kreises C_v im *Schwerpunkt* aller Kinder von v liegt, und der Radius minimal möglich gewählt wird (s. Abb. 6.5(b)). Dies ist, wenn man Kreise als geometrische Container beibehalten will, und zusätzlich die Laufzeit linear bleiben soll, eine gute Strategie, allerdings ist der Platzverbrauch oft immer noch zu groß, was sich wieder besonders bei langen Ketten bemerkbar macht.

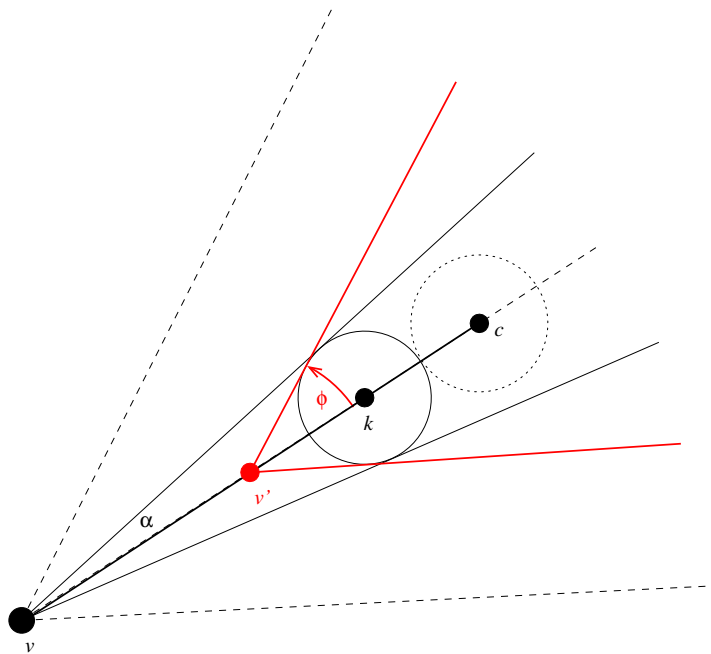
6.4 Neuer Ansatz: Sektoroptimierungen

Eines der Hauptprobleme des im vorherigen Abschnittes beschriebenen Algorithmus ist der exponentiell mit der Höhe wachsende Platzverbrauch. Wenn man die minimale gewünschte Kantenlänge



(a) Wenn für k maximal der Winkel ϕ reserviert ist, der Kreis C_k aber tatsächlich diesen Winkel $\alpha = \phi$ benötigt, kann die Distanz von k im Ballonlayout nicht verringert werden.

(b) Tatsächlich belegt die Kette c, k an v aber lediglich den Winkel $\alpha = \arcsin \frac{l_{min}}{d(v,k)} < \phi$



(c) k und der daran hängende Teilbaum $T(k)$ kann damit problemlos so verschoben werden, dass v relativ zu k die Position v' erhält.

Abbildung 6.6: Distanzminimierung durch genauere Winkelbestimmung

mit l_{min} bezeichnet, so wird zum Beispiel für eine Kette von n Knoten immer ein kompletter Kreis mit Radius $> n \cdot l_{min}$ verwendet, und dafür ein entsprechend großer Winkel am Elternknoten reserviert. Tatsächlich benötigt wird aber nur ein Winkel von $\arcsin \frac{l_{min}}{d(v,k)}$. In einem Fall wie in Abb. 6.6 kann man also den gesamten Teilbaum $T(k)$ problemlos bis zur minimalmöglichen Kantenlänge an v heranschieben, und hat unter Umständen sogar noch Freiraum übrig, den man für andere Teilbäume verwenden kann. Ziel der folgenden Abschnitte soll es daher sein, das Layout zu kompaktieren. Das verwendete Verfahren hat im worst-case (vollständige n -äre Bäume) zwar immer noch einen exponentiellen Platzverbrauch in der Höhe des Baumes, allerdings tritt dieser Fall bei Strukturformeln nur selten ein.

In den Algorithmen in 6.2 wurde zur Breitenminimierung versucht, anhand der Konturen der Teilbäume zu bestimmen, wie nah man diese aneinanderlegen kann. Mitentscheidend für die Effizienz dieser Berechnung war jedoch, dass das Layout abwärts ist, also alle Kinder unter- bzw. außerhalb des Elternknotens liegen. Dadurch konnte man einfach anhand der Höhe der betrachteten Teilbäume entscheiden, bis wann eine Abarbeitung überhaupt nötig ist. Verzichtet man auf die Tiefenschichtung, geht dies im Allgemeinen nicht mehr, da die Kindknoten auch weiter oben/innen zu liegen kommen können. Im schlimmsten Fall muss man also immer die Teilbäume komplett absteigen, um die Konturen bestimmen zu können, wodurch im allgemeinen keine Bearbeitung des Baumes in Linearzeit mehr möglich ist. Im Ballonlayout wird die Linearität dadurch gewährleistet, dass keine Informationen verwendet werden, für deren Berechnung unter Umständen der komplette Teilbaum abgelaufen werden muss. Stattdessen wird jedem Kindknoten k von v lediglich ein Container C_k zugeordnet, so dass stets garantiert ist, dass $T(k)$ komplett in C_k liegt. Eine mögliche Modifikation besteht also darin, statt Kreisen andere Containerobjekte zu verwenden. Diese sollten folgende Eigenschaften haben:

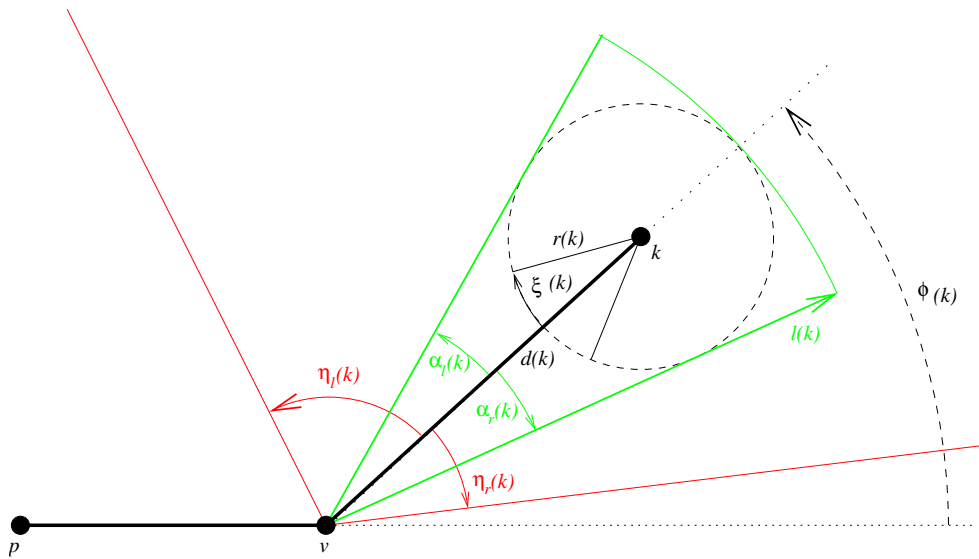
Invarianzbedingung: Zu jedem Zeitpunkt sollen alle Knoten und Kanten von $T(k)$ (inkl. k) komplett in C_k liegen.

Konturierungsbedingung: C_k soll die Form von $T(k)$ möglichst gut beschreiben. In unserem Fall heißt das insbesondere, dass der von C_k belegte Winkel an v möglichst wenig vom für $T(k)$ tatsächlich benötigten Winkel abweicht.

Effizienzbedingung: Die Bearbeitung soll während geordneter Durchläufe durch den Baum geschehen. Ist es zur Berechnung irgendwelcher Werte nötig, in einem Schritt weiter auf- oder abzusteigen, so soll dies mit einer konstanten Tiefe geschehen. Ziel wird es sein, zur Berechnung von C_v alle C_k sowie eventuell weitere Daten von allen Kindern k zu kennen.

Eine Möglichkeit für Containerobjekte C_k besteht darin, statt kompletter Kreise Kreissektoren zu verwenden, deren Spitze in v liegt, und deren Winkel so gewählt wird, dass obige Bedingungen erfüllt sind. Sektoren sollten sich deshalb gut als Containerobjekte eignen, da man bei Bäumen erwarten wird, dass sie mit zunehmender Tiefe breiter werden, ihre Form durch Kreissektoren also relativ gut beschrieben wird. Dies soll im folgenden präzisiert werden. Wie immer sei $T(r)$ ein Wurzelbaum mit Wurzel r , so dass insbesondere Eltern- und Kindknoten stets wohldefiniert sind. Für jeden Knoten k werden im Verlaufe des Algorithmus folgende Daten gespeichert:

- $d(k)$: der aktuelle Abstand $d(v, k)$ zum Elternknoten v
- $r(k)$: Radius des Kreises, der den Knoten mit allen Kindern garantiert enthält. Dies wird nur in der Initialisierungsphase benötigt, s.u.
- $\alpha_r(k)$, $\alpha_l(k)$: Der aktuelle Winkel nach rechts bzw. links, den der Sektor $S(k)$ an v gerade besitzt.
- $\eta_r(k)$, $\eta_l(k)$: Der Winkel nach rechts bzw. links, der für den Sektor für $S(k)$ an v aktuell zur Verfügung steht.
- $\phi(k)$: Der relative Rotationswinkel von k , gemessen bzgl. der Elternkante $\{p, v\}$ von v

Abbildung 6.7: Die Daten die für einen Knoten k gespeichert werden

- $x(k), y(k)$: Koordinaten von k .
- $l(k)$: Die Länge des Sektors $S(k)$, d.h. der Radius des Kreises, aus dem $S(k)$ ausgeschnitten wurde.
- $\xi(k)$: Winkelbereich, der für die Elternkante $\{v, k\}$ auf jeden Fall freizuhalten ist.

Die Bedeutung der einzelnen Daten kann man in Abb. 6.7 sehen. Im Gegensatz zum Ballonlayout ist es nicht nötig, Skalierungsfaktoren oder nicht belegten Platz zu berechnen, da dies dynamisch im Verlauf des Algorithmus geschieht resp. unnötig ist. Außerdem werden als globale Parameter eingeführt:

- r_{min} : der minimale Radius, der für einen Teilbaum freigehalten wird. Damit werden Knotenüberlappungen vermieden.
- l_{min} : die minimale Kantenlänge

Der Algorithmus arbeitet in drei Stufen, die wieder mit drei Durchläufen durch den Baum realisiert werden können:

Initialisierung: In diesem Schritt werden die Teilbäume ähnlich wie im Ballonlayout ausgelegt, so dass ein kreuzungsfreies, in diesem Fall aber auch noch uniformes Winkelayout entsteht.

Sektoroptimierung: Da im Initialisierungsschritt viel zu viel Platz verschwendet wird, wird jetzt versucht, den Platzbedarf zu optimieren

Koordinatengenerierung: Hier werden wieder rekursiv für jeden Knoten aus den vorberechneten Kantenlängen und relativen Rotationswinkeln die absoluten Koordinaten berechnet.

Im Folgenden werden die einzelnen Schritte genauer vorgestellt:

6.4.1 Initialisierung

Dieser Schritt bestimmt ein initiales Layout fast genau so wie im Ballonlayout. Im Unterschied dazu werden allerdings die Kindkreise von v nicht so positioniert, dass ihre Ränder einen einheitlichen Abstand d_{max} von v haben und davon abhängig die benötigten Winkel berechnet. Stattdessen wird für jeden Kindkreis derselbe Winkel res bereitgestellt, und daraus dann die nötige Kantenlänge $d(k)$ berechnet. Es gilt:

$$d(k) = \max\left(\frac{r(k)}{\sin res}, l_{min}\right) \quad (6.3)$$

und $r(k)$ wurde bereits vorher bestimmt. Die Berechnung des Radius $r(v)$ von C_v verändert sich ebenfalls, es gilt einfach:

$$r(v) = \max\{d(k) + r(k) \mid k \text{ Kind von } v\} \cup \{r_{min}\} \quad (6.4)$$

Hierbei werden dann Blätter mit r_{min} und Kantenlänge l_{min} initialisiert. Außerdem werden die relativen Rotationswinkel $\phi(k)$ so bestimmt, dass sich ein winkelnormales Layout ergibt. Diese Winkel werden im weiteren Verlauf des Algorithmus nicht mehr verändert. All dies wird von der Prozedur **firstWalk** erledigt, wobei rekursiv erst alle nötigen Werte für die Kinder bestimmt werden, bevor v selbst initialisiert wird.

```

firstWalk(Knoten  $v$ )
begin
  /*Reserviere Platz für die Elternkante, außer wir sind an der Wurzel */
  if  $v$  ist Wurzel then
     $\xi(v) \leftarrow 0$ 
    /*Stelle jedem Kind maximalen Platz zur Verfügung */
     $res \leftarrow \frac{\pi}{\deg(v)}$ 
  else
    /*Reserviere für die Elternkante ein gleich großes Stück wie für die Kinder */
     $\xi(v) \leftarrow \frac{\pi}{\deg(v)}$ 
    /*Teile den Rest gleichmäßig auf */
     $res \leftarrow \frac{\pi - \xi(v)}{\deg(v) - 1}$ 
   $r_{max} \leftarrow r_{min}$ 
  /*Der akkumulierte Rotationswinkel, zu Anfang steht er auf dem ersten Kind */
   $\phi \leftarrow -\pi + \xi(v) + res$ 
  forall Kinder  $k$  von  $v$  do
    /*Bearbeite erst Kind  $k$  */
    firstWalk( $k$ )
    /*Berechne den Radius von  $C_v$  und passe die Distanzen der Kinder an */
     $d(k) \leftarrow \max\left(\frac{r(k)}{\sin res}, l_{min}\right)$ 
     $r_{max} \leftarrow \max(r_{max}, d(k) + r(k))$ 
     $\alpha_l(k) \leftarrow \eta_l(k) \leftarrow res$ 
     $\alpha_r(k) \leftarrow \eta_r(k) \leftarrow -res$ 
     $\phi(k) \leftarrow \phi$ 
    /*Setze den Rotationswinkel weiter */
     $\phi \leftarrow \phi + 2res$ 
   $r(v) \leftarrow r_{max}$ 
end

```

Prozedur firstWalk(Knoten v)

Dieser Schritt läuft insgesamt in $O(|V|)$, da jeder Knoten genau einmal als Kindknoten k besucht wird. Das Layout ist kreuzungsfrei, und für jeden Sektor $\alpha_l(k), \alpha_r(k)$ ist die Invarianzbedingung erfüllt, da der gesamte Teilbaum an k sogar in dem Kreis C_k liegt, der komplett innerhalb von

$\alpha_l(k), \alpha_r(k)$ liegt. Zum Zeitpunkt der Berechnung von $d(k)$ ist der Radius von $r(k)$ bereits bekannt, und für Blätter sind alle Werte bis auf den belegten Winkel richtig initialisiert.

Außerdem ist das Layout winkelnorm, da der Rotationswinkel konstant weiter gesetzt wird und die nötige Rotation passend bestimmt wurde. Nach Konstruktion wird eine gegebene zyklische Ordnung der Kindknoten respektiert.

6.4.2 Sektoroptimierung

Leider verbraucht das im ersten Schritt erzeugte Layout im Allgemeinen noch mehr Platz als das Ballonlayout, da die Distanzen meistens noch größer sind. Daher werden jetzt die Größen und Abstände der Sektoren genauer bestimmt. Dies geschieht wieder rekursiv, so dass erst die jeweiligen Teilbäume an v komplett ausgelegt werden, bevor v selbst bearbeitet wird. Dabei wird in jedem Schritt die Invarianzbedingung beibehalten. Im Überblick passiert dabei Folgendes:

1. Bearbeite alle Teilbäume $T(k)$ an den Kindern k von v komplett. Dabei werden die Distanzen zu k noch nicht geändert.
2. Bestimme die tatsächlich belegten Winkel und Längen jedes Kindsektors $S(k)$
3. Verteile die zur Verfügung stehenden Winkel neu
4. Optimierte die Distanzen bzgl. der tatsächlich belegten und zur Verfügung stehenden Winkel neu
5. Bestimme erneut die tatsächlich belegten Winkel und Längen jedes Kindsektors $S(k)$

Das Gerüst für dieses Vorgehen steckt in der Prozedur **secondWalk**:

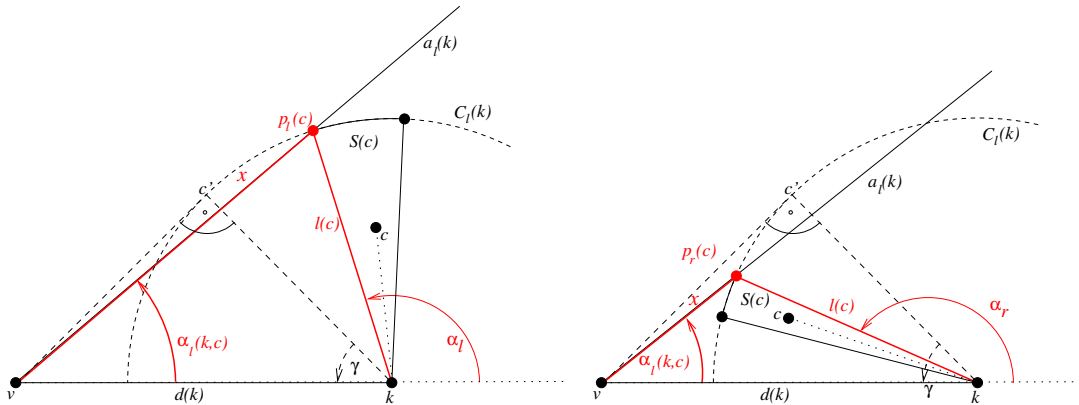
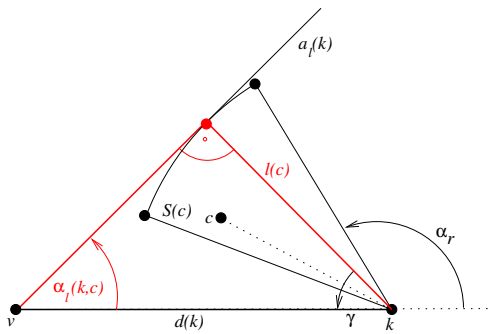
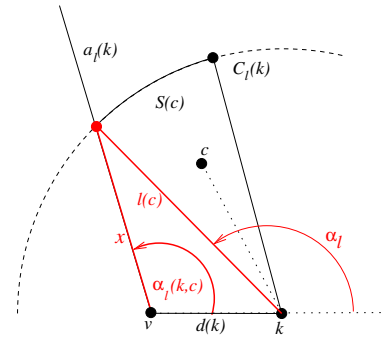
```

secondWalk(Knoten  $v$ )
begin
  forall Kinder  $k$  von  $v$  do
    /*Bearbeite erst Teilbaum  $T(k)$  */
    secondWalk( $k$ )
    /*Bestimme die aktuelle Länge und den belegten Winkel von  $S(k)$  */
    recalcLengths( $k$ )
    recalcAngles( $k$ )
    /*Verteile die freien Winkel neu */
    redistributeAngles( $v$ )
  forall Kinder  $k$  von  $v$  do
    /*Berechne optimierte Distanzen */
     $d \leftarrow$  acquireOptimalDistance( $k$ )
    if  $d < d(k)$  then
       $d(k) \leftarrow d$ 
      /*Bestimme die aktuelle Länge und den belegten Winkel von  $S(k)$  neu */
      recalcLengths( $k$ )
      recalcAngles( $k$ )
    end
  end

```

Prozedur secondWalk(Knoten v)

Für die Bestimmung der tatsächlich belegten Winkel und Längen sowie der optimierten Distanzen des Sektors $S(k)$ seien die aktuelle Distanz $d(k)$ sowie die kompletten Sektoren $S(c)$ aller Kinder c von k bereits gegeben, sowie für die Distanzberechnung der für $S(k)$ verfügbare Winkel.

(a) Winkelbestimmung, der linke Endpunkt von $S(c)$ ist relevant(b) Winkelbestimmung, der rechte Endpunkt von $S(c)$ ist relevant(c) Winkelbestimmung, $a_l(k)$ berührt $S(c)$ am oberen Kreisrand(d) Winkelbestimmung für den Fall $d(k) \leq l(c)$ Abbildung 6.8: Winkelberechnung in Abhängigkeit von der Lage von $S(c)$

Winkelberechnung

Sei ein Sektor $S(c)$ ausgewählt, der komplett auf der linken Seite in $S(k)$ liegt, so wie in Abb. 6.8 abgebildet. Dann kann man bei gegebener Distanz $d(k)$ den Winkel $\alpha_l(k, c) := \alpha_l(k, a_l(c), a_r(c))$ in Abhängigkeit von c so bestimmen, dass $S(c)$ noch komplett in $S(k)$ liegt. Für diese Berechnung ist es entscheidend, an welcher Stelle die linke Sektorbegrenzung $a_l(k)$ den Sektor $S(c)$ trifft. Sei zuerst $d(k) > l(c)$. Dann wird der Winkel γ im Dreieck v, k, c' bestimmt, wobei c' der Berührungspunkt der Tangente durch v an den linken Halbkreis $C_l(k)$ um k mit Radius $l(c)$ ist. Es sind verschiedene Fälle zu betrachten, abhängig von der Lage der Ränder von $S(c)$ (s. Abb. 6.8(a)-6.8(c)), wobei stets $\alpha_{l/r} := \phi(c) + \alpha_{l/r}(c)$:

1. $\alpha_l < \pi - \gamma$: Hier berührt $a_l(k)$ den Sektor $S(c)$ am linken Rand im Endpunkt $p_l(c)$. Dann ist $\theta = \pi - \alpha_l$, und damit berechnet man $x = d(k)^2 + r(c)^2 - 2d(k)l(c) \cos \theta$, $\delta = \arccos \frac{l(c)^2 + x - d(k)^2}{2l(c)\sqrt{x}}$, und damit $\alpha_l(k, c) = \pi - \delta - (\pi - \alpha_l) = \alpha_l - \delta$.
2. $\alpha_r > \pi - \gamma$: Hier berührt $a_l(k)$ den Sektor $S(c)$ am rechten Rand im Endpunkt $p_r(c)$. Entsprechend ist dann $\theta = \pi - \alpha_r$, und damit berechnet man $x = d(k)^2 + l(c)^2 - 2d(k)l(c) \cos \theta$, $\delta = \arccos \frac{r(c)^2 + x - d(k)^2}{2l(c)\sqrt{x}}$, und damit $\alpha_l(k, c) = \pi - \delta - (\pi - \alpha_r) = \alpha_r - \delta$.

3. Ansonsten berührt $a_l(k, c)$ den Sektor $S(c)$ im abschließenden Kreisbogen, und es gilt einfach $\alpha_l(k, c) = \pi/2 - \gamma$

Ist $d(k) \leq l(c)$, so kann es keine Tangente an den Kreis mehr geben, da v auch schon innerhalb von $C_l(k)$ liegt. In diesem Fall berührt allerdings in jedem Fall $a_l(k)$ den Sektor $S(c)$ am linken Rand, so dass wieder Fall 1 von oben eintritt. (s. Abb. 6.8(d))

In all diesen Fällen gilt $\alpha_r(k, c) = \arcsin \frac{r_{min}}{d(k)}$, da auf der rechten Seite lediglich der für den Knoten k selbst belegte Radius zu berücksichtigen ist.

Liegt der Sektor $S(c)$ komplett auf der rechten Seite in $S(k)$, kann man dies auf die schon behandelte Situation durch Spiegelung an der Mittelachse von $S(k)$ zurückführen, man muss also lediglich $a_l(c)$ und $a_r(c)$ vertauschen und alle Vorzeichen umkehren. Falls keiner der beiden Fälle eintritt, der Sektor also mittig liegt, kann man einfach die beiden Hälften rechts und links der Mittelachse getrennt betrachten, also jeweils einen der beiden Winkel auf 0 setzen.

Die Prozedur **recalcAngles** bestimmt dann einfach die jeweils maximalen bzw. minimalen Winkel über alle Sektoren $S(c)$.

```

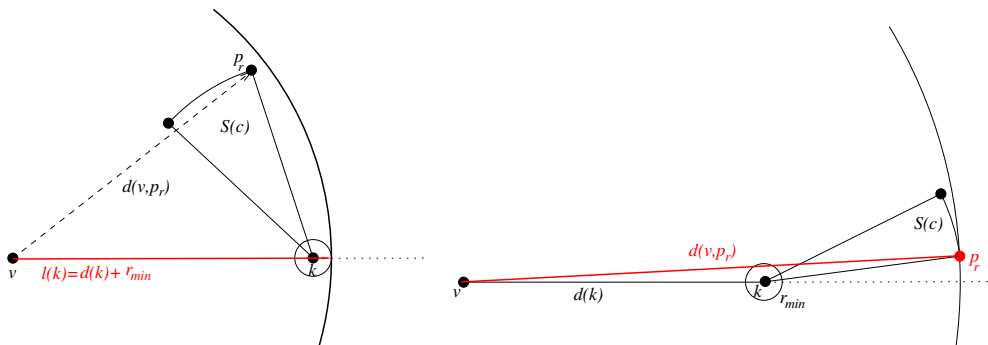
recalcAngles(Knoten  $k$ )
begin
  /*Stelle sicher, dass der Winkel für Blätter richtig initialisiert ist          */
   $\alpha_l(k) \leftarrow \arcsin \frac{r_{min}}{d(k)}$ 
   $\alpha_r(k) \leftarrow -\arcsin \frac{r_{min}}{d(k)}$ 
  forall Kinder  $c$  von  $k$  do
    |  $\alpha_l(k) \leftarrow \max(\alpha_l(k), \alpha_l(k, c))$ 
  end

```

Prozedur **recalcAngles**(Knoten k)

Längenberechnung

Für die Berechnung der maximalen Sektorlänge sei wieder ein Sektor $S(c)$ gegeben, der komplett auf der linken Seite in $S(k)$ liegt. Dann ist der Endpunkt p_r von $a_r(c)$ immer derjenige Punkt des oberen abschließenden Kreisbogens zwischen p_l und p_r , der zu v den größten Abstand besitzt. Damit gibt es dann zwei für die Länge des Sektors relevante Lagen (s. Abb. 6.9(a),6.9(b)):



(a) Längenbestimmung, Fall 1

(b) Längenbestimmung, Fall 2

Abbildung 6.9: Längenberechnung in Abhängigkeit von der Lage von $S(c)$

1. $d(v, k) \geq d(v, p_r)$: In diesem Fall haben alle Punkte in $S(c)$ ebenfalls einen kleineren Abstand zu v , und es gilt $l(k, c) = d(v, k) + r_{min}$. Hierbei ist zu berücksichtigen, dass für die Längenberechnung noch zusätzlich der Minimalradius für den Knoten k zu berücksichtigen ist.
2. $d(v, k) < d(v, p_r)$: In diesem Fall gilt $l(k, c) = d(v, p_r)$.

Für Sektoren $S(c)$, die nicht links in $S(k)$ liegen, kann man die Situation wieder auf den gerade besprochenen Fall zurückführen, so dass man für die Gesamtlänge des Sektors wieder einfach über alle Kinder c von k maximieren kann (s. Prozedur **recalcLengths**).

```

recalcLengths(Knoten k)
begin
  /*Stelle sicher, dass die Länge für Blätter richtig initialisiert ist          */
   $l(k) \leftarrow d(k) + r_{min}$ 
  forall Kinder  $c$  von  $k$  do
    |  $l(k) \leftarrow \max(l(k), l(k, c))$ 
  end

```

Prozedur recalcLengths(Knoten k)

Distanzberechnung

Die Bestimmung der minimal möglichen Distanz $d(k)$ schließlich ist im wesentlich dual zur Winkelbestimmung: Sei also der Winkel $\eta_l(k)$ vorgegeben. Entscheidend ist dann wieder, wo der linke Rand a_l von $S(k)$ einen ganz links liegenden Sektor $S(c)$ berührt. Dazu kann man wieder die Lage von $\alpha_l = \phi(c) + \alpha_l(c)$ und $\alpha_r = \phi(c) + \alpha_r(c)$ bzgl. $\eta_l(k) + \pi/2$ betrachten (s. Abb. 6.10(a)-6.10(d)):

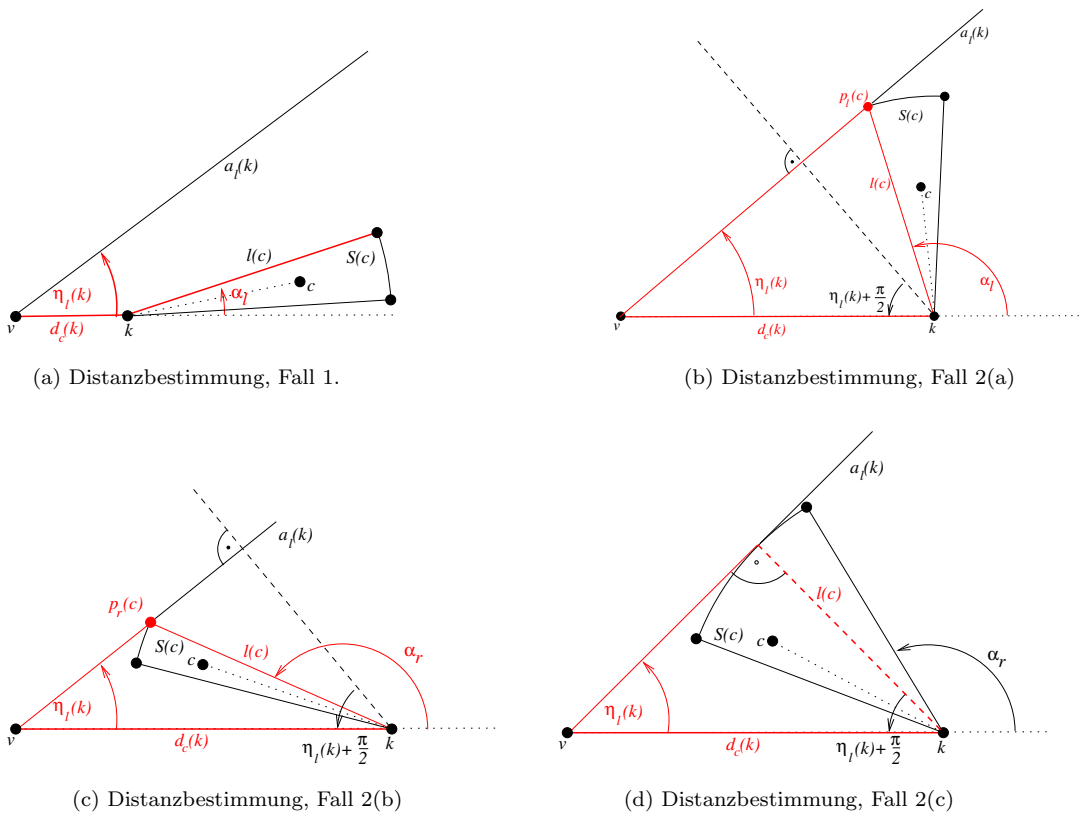
1. $\alpha_l \leq \eta_l(k)$: Hier kann man die Distanz zwischen v und k beliebig verkleinern, ohne dass $S(c)$ den Sektor $S(k)$ verlässt. Also kann man $d_c(k) = l_{min}$ setzen.
2. $\alpha_l > \eta_l(k)$: In diesem Fall ist die minimal mögliche Distanz von k durch die Lage von $S(c)$ tatsächlich begrenzt:
 - (a) $\alpha_l < \eta_l(k) + \pi/2$: Dann berührt $a_l S(c)$ am linken Rand, und mit dem Sinussatz findet man $d_c(k) = \frac{r(c) \sin \gamma}{\sin \eta_l(k)}$, wobei $\gamma = \pi - \alpha_l(c)$.
 - (b) $\alpha_r > \eta_l(k) + \pi/2$: Dann berührt $a_l(k) S(c)$ am rechten Rand, und mit dem Sinussatz findet man $d_c(k) = \frac{r(c) \sin \gamma}{\sin \eta_l(k)}$, wobei $\gamma = \pi - \alpha_r(c)$.
 - (c) $\alpha_r \leq \eta_l(k) + \pi/2 \leq \alpha_l$: Dann berührt $a_l(k) S(c)$ tangential am oberen Kreisbogen, und man findet $d_c(k) = \frac{r(c)}{\sin \eta_l(k)}$.

```

recalcDistance(Knoten k)
begin
   $d(k) \leftarrow l_{min}$ 
  forall Kinder  $c$  von  $k$  do
    |  $d(k) \leftarrow \max(d(k), d_c(k))$ 
  end

```

Prozedur recalcDistance(Knoten k)

Abbildung 6.10: Bestimmung der minimalen Distanz $d_c(k)$ in Abhängigkeit von der Lage von $S(c)$

Wie bei den anderen Berechnungen lassen sich die Fälle, in denen $S(c)$ rechts oder mittig in $S(k)$ liegt, auf diesen Fall zurückführen. In der Prozedur **recalcDistance** wird wieder über alle Kinder c von k das Maximum der minimal möglichen Distanzen bestimmt, also $d(k) = \max_{c \text{ Kind von } k} d_c(k)$.

Neuverteilung der verfügbaren Winkel

Es bleibt noch, eventuell übrige freie Winkel zu verteilen. Die einfachste Strategie ist es, eventuell freien Winkel zwischen zwei Sektoren je zur Hälfte den beiden Sektoren zusätzlich zur Verfügung zu stellen. Dies geschieht in der Prozedur **redistributeAngles**.

```

redistributeAngles(Knoten  $v$ )
begin
  forall Kinder  $k$  von  $v$  do
    if  $k$  hat einen rechten Nachbarn  $k'$  then
       $excess \leftarrow (\phi(k) + \alpha_r(k)) - (\phi(k') + \alpha_l(k'))$ 
       $\eta_r(k) \leftarrow \alpha_r(k) - excess/2$ 
       $\eta_l(k') \leftarrow \alpha_l(k') + excess/2$ 
    end
  end

```

Prozedur **redistributeAngles**(Knoten v)

Laufzeit

Insgesamt hat ein kompletter Durchlauf von **secondWalk**(r) wieder eine Laufzeit von $O(|V|)$, da für jede der einzelnen Berechnungen jeder Knoten k höchstens einmal angesehen wird, und nicht weiter in den Baum abgestiegen wird. Hierbei wird $T(r)$ wieder in postorder-Reihenfolge durchlaufen. Nach Konstruktion ist zu jedem Zeitpunkt der Abarbeitung die Invarianzbedingung an die Sektoren erfüllt, und damit die Planarität sichergestellt.

6.4.3 Koordinatengenerierung und gesamter Algorithmus

Die Generierung absoluter Koordinaten läuft ganz analog zum Ballonlayout ab, wobei hier allerdings keine Skalierungsfaktoren berücksichtigt werden müssen, und die relativen Rotationswinkel ϕ ebenfalls schon vorher in Schritt 1 bestimmt wurden. Dies geschieht in der Prozedur **preorder**.

```

preorder(Knoten  $v$ , double  $x$ , double  $y$ , double  $\theta$ )
begin
  /*Weise  $v$  absolute Koordinaten zu */
   $x(v) \leftarrow x$ 
   $y(v) \leftarrow y$ 
  /*Skaliere die Distanz, falls nötig; wird von den Eltern aus akkumuliert */
   $d' \leftarrow \lambda \cdot v.d$ 
  /*Verteile den freien Platz auf alle Kinder, und bestimme den absoluten Startwinkel,
  von dem aus die Rotationen gemessen werden */
  if  $v$  ist Wurzel von  $T$  then
     $\phi \leftarrow 0$ 
  else
     $\phi \leftarrow \theta$ 
  forall Kinder  $k$  von  $v$  do
     $x \leftarrow x(v) + d(k) \cos(\phi + \text{phi}(k))$ 
     $y \leftarrow y(v) + d(k) \sin(\phi + \text{phi}(k))$ 
    preorder( $k, x, y, \phi + \text{phi}(k)$ )
  end

```

Prozedur **preorder**(Knoten v , double x , double y , double θ)

Auch dieser Schritt hat insgesamt eine Laufzeit von $O(|V|)$.

Insgesamt ergibt sich dann für die Erzeugung des Sektorlayouts folgender Ablauf:

```

Input : Ein Wurzelbaum  $T(r)$  mit  $n$  Knoten
1 begin
2   firstWalk( $r$ )
3   secondWalk( $r$ )
4   preorder( $r, 0, 0, 0$ )
5 end

```

Algorithmus 19 : Sektorlayout

Die Laufzeit für Algorithmus 19 beträgt insgesamt $O(|V|)$, wie für die einzelnen Schritte bereits gezeigt. Der Algorithmus berechnet nach Konstruktion ein winkeluniformes kreuzungsfreies Layout von $T(r)$ und erhält eine gegebene zyklische Ordnung der Kindknoten.

Im allgemeinen wird der Platzbedarf geringer sein als im Ballonlayout. Allerdings tritt auch hier im worst-case eine exponentielle Vergrößerung der Kantenlängen zur Wurzel hin auf, falls die Sektoren $S(k)$ der Kinder von v nämlich den Kreis um v in jede Richtung bzgl. der Ränder von $S(v)$ komplett ausfüllen, was beispielsweise bei vollständigen Bäumen der Ordnung ≥ 3 eintritt.

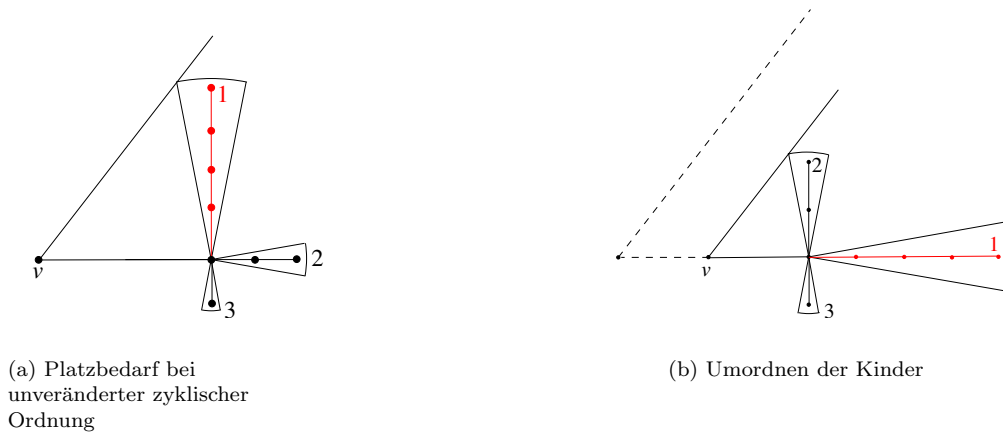


Abbildung 6.11: Änderung der zyklischen Ordnung kann den Platzbedarf deutlich verringern

6.4.4 Optimierungen

In diesem Abschnitt sollen verschiedene Methoden vorgestellt werden, den Platzbedarf weiter zu reduzieren und evtl. andere visuelle Parameter zu optimieren.

Knotenneuordnung

Versucht man, einige größere Strukturformeln mit dem Sektorlayout zu zeichnen, so fällt auf, dass oft die gegebene, meist zufällige zyklische Ordnung der Kindknoten ein platzeffizienteres Layout verhindert. So zeigt in Abb. 6.11(a) eine lange Kette zur Seite und verhindert damit, dass die Distanz zwischen v und k weiter reduziert werden kann, weil der Sektor $S(v)$ dafür zu schmal ist. Unabhängig sollten auch sonst lange Ketten möglichst gradlinig dargestellt werden. Da im allgemeinen die zyklische Ordnung der Kinder nicht zwingend erhalten bleiben muss, hat man hier noch Optimierungspotential, indem man versucht, lange Sektoren $S(k)$ so zu legen, dass sich der belegte Winkel des Elternsektors $S(v)$ verringert. Im Beispiel kann man die lange Kette $T(1)$ so platzieren, dass sie nach außen zeigt, d.h. lange Ketten in der Mitte der zyklischen Ordnung auftauchen (s. Abb 6.11(b)). Damit wird gleichzeitig das zweite angesprochene Problem erledigt.

Im Folgenden wird gezeigt, wie man diese einfachste Neuordnungsstrategie, bei der die höchsten Teilbäume in der Mitte der neuen zyklischen Ordnung auftreten, in das bestehende Framework einbinden kann. Daneben sind viele andere Strategien denkbar, zum Beispiel eine Ausrichtung nach der absoluten Außenrichtung (hierfür kann man beispielsweise bei der rekursiven Abarbeitung eine Vorzugsrichtung mitpropagieren). Die Prozedur **firstWalk** wird hierzu so modifiziert, dass dabei die Höhen der Teilbäume berechnet werden. Jeder Knoten v erhält dazu ein neues Attribut $h(v)$, in dem die Höhe des Teilbaums $T(v)$ gespeichert ist.

In der Prozedur **reorderChildren** werden dann die Kindknoten umsortiert. Um bei der Umordnung der Kinder eine gleichmäßigere Ausrichtung der Ketten zu erreichen, wurde bei jedem Knoten die Vorzugsrichtung zwischen links und rechts gewechselt.

Bei geeigneter Implementation lassen sich die Listenoperationen in konstanter Zeit durchführen, so dass für die **for**-Schleife nur Laufzeit $O(|children(v)|)$ benötigt wird. Da die möglichen auftretenden Werte für $h(c)$ ganzzahlig sind und im Intervall $[0, |V|]$ liegen, kann das Sortieren der Liste ebenfalls in $O(|children(v)|)$ erledigt werden, beispielsweise mit *Bucketsort*. Da beim Abarbeiten des Baums jede Liste von Kindern nur einmal bearbeitet wird, bleibt dadurch die Gesamtlaufzeit für den ersten Schritt linear in der Größe des Baums.

```

firstWalk(Knoten  $v$ )
begin
  ...
  /*Neu: Initialisiere die Teilbaumhöhe */
   $h(v) \leftarrow 1$ 
   $h_{max} \leftarrow 0$ 
  forall Kinder  $k$  von  $v$  do
    ...
    |  $h_{max} \leftarrow \max(h_{max}, h(k))$ 
    |  $h(v) \leftarrow h(v) + h_{max}$ 
    if  $v$  hat mindestens 2 Kinder then reorderChildren( $v$ )
  ...
end

```

Prozedur **firstWalk**(Knoten v)

```

firstWalk(Knoten  $v$ )
begin
  Sortiere  $children(v) = c_1, \dots, c_k$  aufsteigend nach  $h(c_i)$ 
  List  $l1 \leftarrow \emptyset$ 
  List  $l2 \leftarrow \emptyset$ 
  boolean  $flag \leftarrow preferredDirection(v)$ 
  /*Verteile die Kinder abwechselnd auf beide Hilfslisten */
  forall  $c$  in  $children(v)$  do
    if  $flag$  then
      |  $l1.append(c)$ 
    else
      if  $v$  ist Wurzel then
        | /*Für die Wurzel liegen die beiden längsten Ketten einander gegenüber */
        |  $l2.append(c)$ 
      else
        | /*Sonst drehe die eine Liste um */
        |  $l2.putFirst(c)$ 
       $flag \leftarrow NOT\ flag$ 
       $preferredDirection(c) \leftarrow NOT\ preferredDirection(v)$ 
     $children(v) \leftarrow l1 + l2$ 
  ...
end

```

Prozedur **reorderChildren**(Knoten v)

Alternative Winkelneuverteilung

Die hier verwendete Methode, freie Winkel zu gleichen Teilen auf die Nachbarsektoren zu verteilen, ist sehr einfach. Daneben sind viele alternative Strategien denkbar:

1. Der weiter entfernte Sektor bekommt einen größeren Teil des freien Stücks zur Verfügung gestellt, also beispielsweise $\eta := \frac{d(k)}{d(k)+d(k')} excess$ zusätzlich für k und $\eta' := \frac{d(k')}{d(k)+d(k')} excess$ zusätzlich für k' . Damit werden stark unterschiedliche Distanzen, wie sie nach der Initialisierungsphase auftreten können, ausgeglichen.
2. Sektoren, die schon einen größeren Winkel belegen, bekommen auch mehr zusätzlichen Freiraum zur Verfügung gestellt, also z.B. $\eta := \frac{\alpha(k)}{\alpha(k)+\alpha(k')} excess$, $\eta' := \frac{\alpha(k')}{\alpha(k)+\alpha(k')} excess$. Hierbei ist die Überlegung, dass breite Sektoren vermutlich auch große Teilbäume beinhalten, so dass es sinnvoll ist, für diese auch mehr Platz zu reservieren.
3. Wie zuvor, nur bekommt jetzt der schmalere Sektor mehr Platz: $\eta := \frac{\alpha(k')}{\alpha(k)+\alpha(k')} excess$, $\eta' := \frac{\alpha(k)}{\alpha(k)+\alpha(k')} excess$. Hier ist die Überlegung, ungleich verteilte Winkel wieder auszugleichen.

Die Auswirkungen dieser Strategien und ob es sich lohnt, diese zu kombinieren, wurden weiter unten noch genauer untersucht.

Verwendung von Nachbarsektoren

Oft kommt es vor, dass ein Sektor $S(k)$ weit entfernt ist und viel Freiraum am unteren Ende besitzt, aber die Nachbarsektoren den ihnen zur Verfügung gestellten Platz gar nicht in voller Länge benötigen. Dann kann man versuchen, diese Sektoren zusätzlich für $T(k)$ zu verwenden, wobei man aber darauf achten muss, $T(k)$ nur so weit wie zulässig zu verschieben. Hierfür werden neue Attribute verwendet:

- $f(v)$: Gibt den Radius an, bis zu dem der Sektor $S(v)$ an der Spitze nicht belegt ist (s. Abb. 6.12(a)).
- $tainted(v)$: boolesches Attribut, wird gesetzt, wenn der Sektor $S(v)$ für einen Nachbarn mitverwendet wurde. Dies wird benötigt, um denselben Platz nicht zweimal freizugeben.

Außerdem ist es sinnvoll, die Distanzneubestimmung nicht in der normalen zyklischen Ordnung, sondern nach aufsteigender Teilbaumhöhe sortiert durchzuführen, um zuerst die kürzeren Sektoren auszulegen. Diese modifizierte Ordnung wird in **reorderChildren** sowieso bestimmt und kann für jeden Knoten zusätzlich gespeichert, ohne dass sich die Laufzeit ändert.

Für die Bestimmung von $f(k)$ kann man **recalcLengths** modifizieren. Für einen ganz links in $S(k)$ liegenden Sektor $S(c)$ gilt (s. Abb. 6.12(b)-6.12(d)):

1. $\alpha_l \leq \pi/2$: Dann kann der Rand des Minimalkreises den Sektor $S(c)$ nicht treffen, und es gilt $f(k) = d(k) - r_{min}$.
2. Sonst: hier kann der Minimalkreis den Sektor $S(c)$ entweder an der linken Begrenzung s_l berühren, oder er trifft lediglich den Endpunkt p_l dieser Begrenzung. Hierzu wird die Projektion i_l von v auf s_l bestimmt. Liegt i_l tatsächlich auf der Strecke s_l , so gilt $f(k) = \min\{d(k) - r_{min}, d(i_l, v), d(p_l, v)\}$, ansonsten $f(k) = \min\{d(k) - r_{min}, d(p_l, v)\}$.

Damit kann man dann die Prozedur **recalcDistance** modifizieren, indem man zusätzlich noch versucht die Nachbarsektoren k' , k'' mit zu verwenden, wenn diese noch frei sind. Ergibt sich bei der maximal möglichen Verschiebung keine Verbesserung, wird der Sektor nicht mit verwendet,

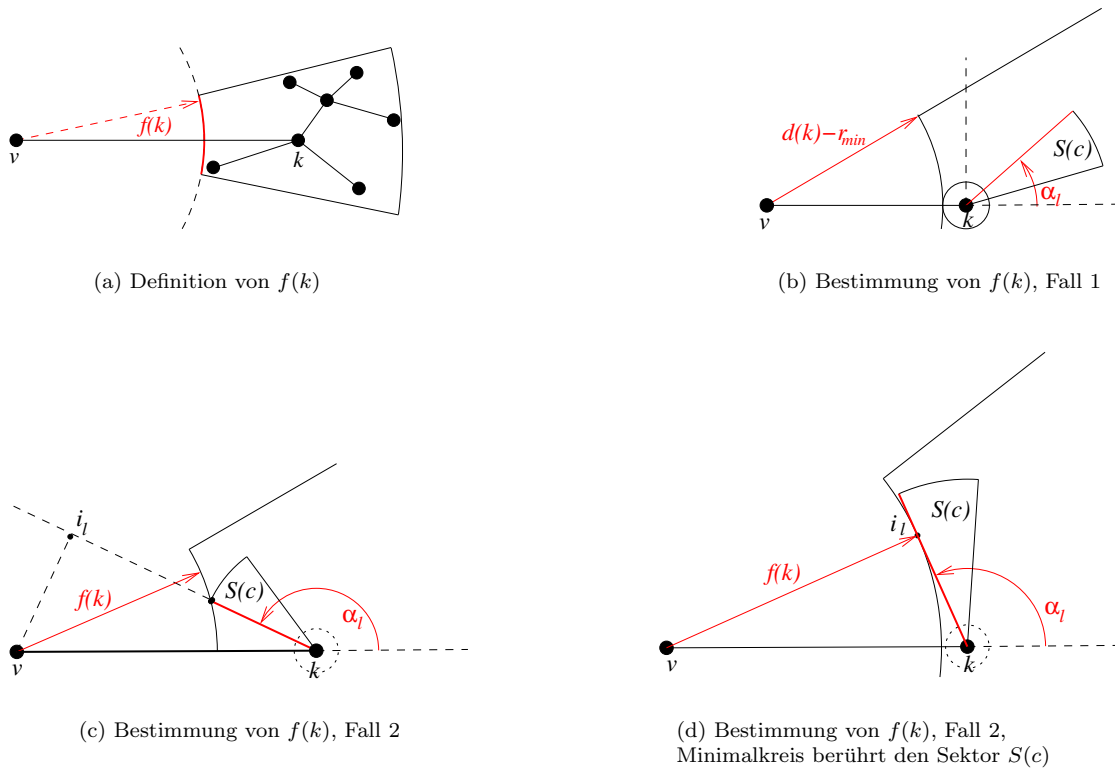


Abbildung 6.12: Bestimmung des Bereichs, der an der Spitze des Sektors frei ist

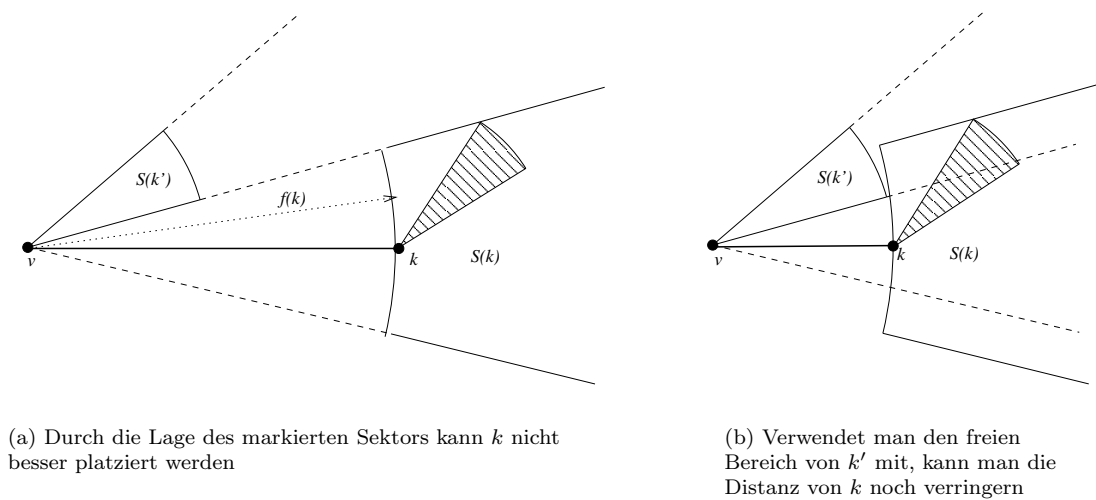


Abbildung 6.13: Verwendung von Nachbarsektoren

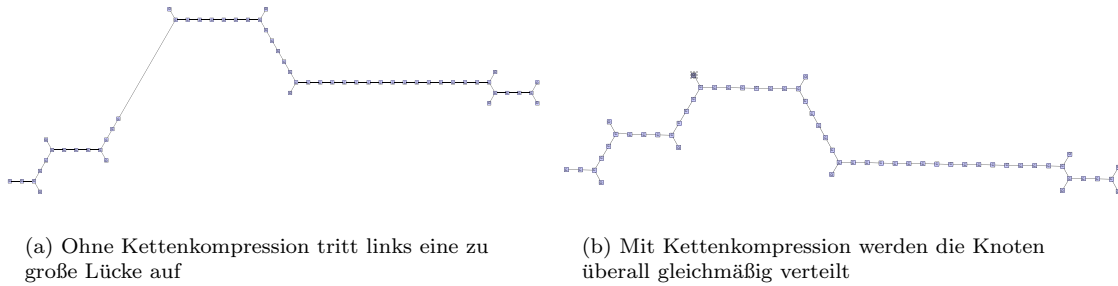


Abbildung 6.14: Kettenkompression

ansonsten wird der verwendete Winkel von k um den von k' bzw. k'' verwendeten Bereich vergrößert, und $tainted(k')$ bzw. $tainted(k'')$ gesetzt. Da die Krümmung des Minimalkreises von k geringer ist als diejenige des Maximalkreises von k' bzw. k'' , kann es bei dieser Distanzreduzierung nicht geschehen, dass belegte Regionen von $S(k)$ plötzlich in die belegten Bereiche der Nachbar-sektoren ragen. Die Prozedur zur Berechnung der einzelnen Werte $d(k, c, \eta_l, \eta_r)$ wird dahingehend modifiziert, dass sie einen zusätzlichen Parameter $minAllowed$ bekommt, der die minimal einzuhaltende Distanz angibt. Will man einen Nachbarsektor $S(k')$ mitverwenden, so ist offenbar $minAllowed = l(k') + d(k) - f(k)$ zu setzen, da $d(k) - f(k)$ gerade die Länge des belegten Bereichs zwischen dem Minimalkreis von k und k angibt.

Für die Mitverwendung lediglich des linken Nachbarn ist die Prozedur in `tryLeftNeighbor` angegeben, bei Verwendung beider Nachbarn gibt es entsprechend mehr Kombinationen, die man überprüfen kann.

```

tryLeftNeighbor(Knoten  $k$ )
begin
   $k' \leftarrow$  Linker Nachbar von  $k$ 
   $d_1 \leftarrow$  Bestimme minimale Distanz für  $\eta_l(k), \eta_r(k)$ ,  $minAllowed = d_{min}$ 
  if NOT  $tainted(k')$  AND  $l(k') < f(k)$  then
     $d_2 \leftarrow$  Bestimme minimale Distanz für  $\eta_l(k) - \eta_r(k') + \eta_l(k'), \eta_r(k)$ ,
     $minAllowed = l(k') + d(k) - f(k)$ 
    if  $d_2 < d_1$  then
      /*Durch Verwendung des linken Nachbarn Platzgewinn möglich          */
       $d(k) \leftarrow d_2$ 
       $tainted(k') \leftarrow TRUE$ 
    else
       $d(k) \leftarrow d_1$ 
    else
       $d(k) \leftarrow d_1$ 
  end

```

Prozedur `tryLeftNeighbor`(Knoten k)

Kettenkompression

Eine weitere Möglichkeit, das Layout zu verbessern, besteht darin, bei langen Ketten die Knoten mit Grad 2 nicht einzeln zu layouten. Dies verringert nicht so sehr den Platzbedarf, führt aber dazu, dass bei solchen Ketten alle Knoten einen gleichmäßigen Abstand haben - im unmodifizierten Algorithmus ist dies nicht der Fall (s. Abb. 6.14(a)).

Dazu werden in einem Vorverarbeitungsschritt alle Knoten mit Grad 2 temporär entfernt und die Kette durch eine Kante ersetzt. Dann wird der modifizierte Baum gezeichnet, und zum Schluss werden wieder die entfernten Knoten eingefügt. Damit dies funktioniert, muss man für die Endknoten der Kette die erlaubte Mindestdistanz d_{min} erhöhen, so dass beim Wiedereinfügen der entfernten Knoten diese wieder den Mindestabstand l_{min} besitzen. Im restlichen Algorithmus wird dann jeweils statt mit l_{min} mit $d_{min}(k)$ verglichen. Das Finden und Entfernen der Ketten geschieht während einer Tiefensuche auf T in der Prozedur **compressChains** und kann mit einem Durchlauf durch den Baum erledigt werden, beginnend mit **compressChains**($r, r, 0, \text{new List}$). Die Laufzeit ändert sich also nicht durch diesen Schritt.

```

Data : Liste von Listen chains
compressChains(Knoten k)
Data : Knoten v: Aktueller Knoten in der Tiefensuche
Data : Knoten chainStart: Anfangsknoten der aktuellen Kette
Data : Integer chainLength: Aktuelle Länge der Kette
Data : Liste currentChain: Aktuelle Kette
begin
  if (v ≠ Wurzel von T) AND (v hat genau ein Kind v') then
    /*Neuen Knoten einer Kette gefunden, füge ihn der Liste hinzu */
    currentChain.addLast(v)
    Entferne v aus T
    /*Steige die Kette weiter ab */
    compressChains(v', chainStart, chainLength+1, currentChain)
  else
    if chainLength > 0 then
      /*Wir haben das Ende einer Kette erreicht */
      /*Merke Start- und Endknoten zusätzlich */
      currentChain.addFirst(chainStart)
      currentChain.addLast(v)
      Füge Kante {chainStart, v} in T ein
      /*In der Kette kamen chainLength + 1 Kanten vor */
      Setze Mindestdistanz von v auf (chainLength + 1) $l_{min}$ 
      Füge currentChain zu chains hinzu
      /*Suche nach neuen Ketten */
    forall Kinder v' von v do
      | compressChains(v', v, 0, new List)
  end

```

Prozedur compressChains(Knoten *k*)

Den Unterschied zum unmodifizierten Algorithmus sieht man in Abb. 6.14(b).

6.5 Experimente

In Experimenten wurden verschiedene Varianten des Sektorlayouts untersucht. Dazu wurden aus dem Datensatz des NCI die Daten von 2000 Bäumen extrahiert. Um noch einige Beispiele von größeren Bäumen und von Bäumen mit höherer Ordnung zur Verfügung zu haben, wurden außerdem Bäume der Größe 10-160 Knoten, jeweils mit unterschiedlichen maximalen Ordnungen, generiert. Für beide Gruppen wurden bei jedem Lauf des Algorithmus die durchschnittliche Kantenlänge und die maximale Abweichung vom Mittelwert der Kantenlängen bestimmt, außerdem, falls die Algorithmen nicht winkeluniform sind, noch die Winkelauflösung und die maximale Ab-

weichung vom Idealwinkel. Für die synthetischen Graphen wurde zusätzlich noch die Abhängigkeit vom Maximalgrad der Bäume untersucht.¹

Konkret wurden untersucht:

- unmodifiziertes Ballonlayout
- Sektorlayout:
 - Die unterschiedlichen Strategien zur Winkelneuverteilung, jeweils mit und ohne Umordnen der Kindknoten, wie in 6.4.4 beschrieben.
 - Exemplarisch die Kombination der beiden Strategien 1 und 2 zur Winkelneuverteilung: Hierfür wurden nach einem Pass über die Kinder und der Bestimmung der Daten der Teilbäume die freien Winkel noch einmal mit der zweiten Strategie verteilt und versucht, die Teilbäume noch besser anzuordnen.
 - Die beste Strategie aus a) und b) kombiniert einzeln mit Verwendung der Nachbarsektoren, Kettenkompression, sowie beiden Optimierungen.
- Radiallayout, wie es in der Bibliothek `yFiles` implementiert ist. Hierbei handelt es sich nicht genau um den oben beschriebenen Algorithmus, sondern um eine Mischung aus Ballon- und Radiallayout. Der Layouter lässt sich allerdings so parametrisieren, dass er im wesentlichen wie ein reines Radiallayout arbeitet (durch Wahl eines freien Winkels von weniger als 90°). Dies bietet eine ungefähre Abschätzung, wie gut etablierte Algorithmen bei diesem Problem abschneiden. Leider gibt es zu den Interna sowie zur Laufzeit dieses Layouters keine Angaben, so dass auf eine ausführlichere Darstellung hier verzichtet wurde.
- Derselbe Layouter, aber mit möglichst großem freien Winkel.

Als Startknoten wurde jeweils einer der Zentrums-knoten des Baums gewählt. Alle Algorithmen wurden in Java unter Verwendung der Bibliothek `yFiles` implementiert. Wegen der Implementation in Java wurde auf genaue Laufzeitmessungen verzichtet, Stichproben haben jedoch ergeben, dass die Zeit für die Bearbeitung der größten Substanzen auf dem Testsystem (AMD Athlon 2400+, 512MB RAM) noch unter einer Sekunde lag, wobei die meiste Zeit für das Starten der JVM und das Einlesen und Parsen der GraphML-Daten verbraucht wurde.

Da die Werte für das Ballonlayout meist mehrere Größenordnungen über denen der anderen Algorithmen liegen, wurden sie in diesen Fällen nicht mit abgedruckt. Man erkennt, dass die unterschiedlichen Winkelverteilungsstrategien sich wenig unterscheiden, insbesondere ist auch die einfache Defaultstrategie durchaus akzeptabel. Dies gilt sowohl für die realen als auch für die synthetischen Daten, auch wenn hier tendenziell die aufwendigeren Strategien etwas besser arbeiten. Vergleicht man die Effizienz der Optimierungsstrategien, so fällt auf, dass insgesamt die Kompression langer Ketten den größten Gewinn bringt, und zwar sowohl bzgl. der durchschnittlichen Kantenlänge wie auch bzgl. der maximalen Abweichung. Insbesondere bei den NCI-Daten ist dieses Verfahren fast optimal: Sowohl bei der mittleren Kantenlänge wie auch bei der maximalen Streuung treten kaum Abweichungen auf. Interessanterweise ist dies auch kaum von der Ordnung der Bäume abhängig, so dass bei den synthetischen Daten, dieser Algorithmus bzgl. beider Kriterien sogar besser arbeitet als die Verfahren aus `yFiles`, bei denen Winkelgrößen und Kantenlängen beide optimiert werden können. Insgesamt arbeiten allerdings alle Verfahren sowohl mit zunehmender Größe als auch zunehmender Ordnung schlechter.

Die Werte für die Winkelauflösung lassen zwar keinen Trend erkennen, interessant ist hier aber, dass selbst die nicht winkelbeschränkte Variante des `yFiles`-Algorithmus stets eine beträchtliche Abweichung vom Idealwinkel erzeugt (bei den anderen beiden Algorithmen war dies ja durchaus zu erwarten).

¹Bei den Daten des NCI liegen die Durchschnittsgrade alle zwischen 3 und 4, so dass für diese Daten die Abhängigkeit vom Knotengrad nicht untersucht wurde

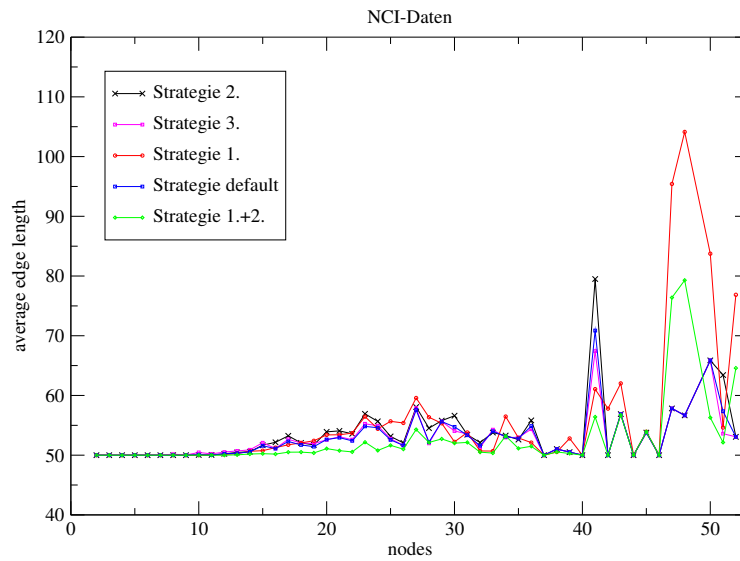
6.6 Fazit

In diesem Abschnitt wurden verschiedene Algorithmen für das Layout der bei chemischen Strukturformeln auftretenden Baumstrukturen untersucht. Da sich die hierarchischen Verfahren vom visuellen Eindruck her nicht gut eignen, wurde ein Layouter entworfen, der Winkeluniformität garantiert, aber zusätzlich versucht, den Platzbedarf zu minimieren. Es zeigt sich, dass mit dem Sektorlayout unter Verwendung der zusätzlichen Optimierungen für viele reale Strukturen eine fast optimale Darstellung erzielt wird. Hierbei bleibt die Laufzeit stets noch linear in der Größe des Graphen, allerdings kann bei kleinen Substanzen (bis maximal 12-13 Knoten) oft auf die erweiterten Optimierungen ganz verzichtet werden, was die Rechenzeit für viele Fälle nochmals deutlich verringern dürfte.

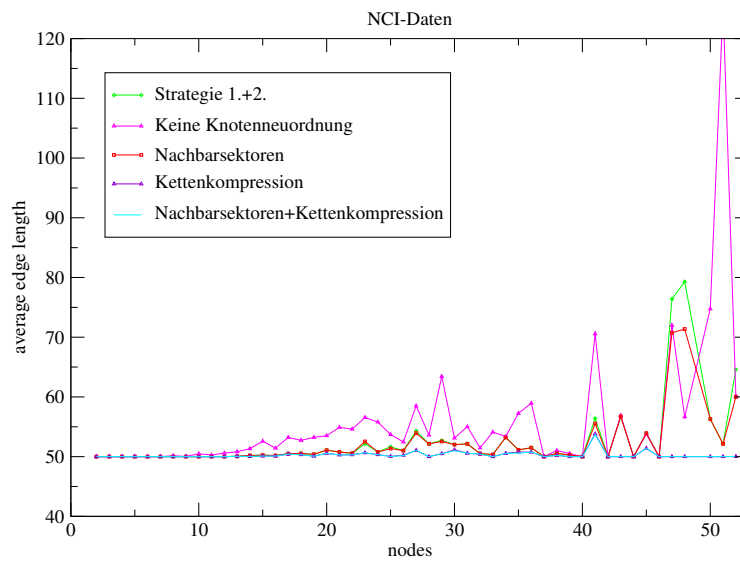
Bei größeren Bäumen gibt es allerdings noch Verbesserungspotential, wobei an erster Stelle hier eine intelligentere Umordnung der Kindknoten sowie besseres Recycling von freiem Platz in Nachbarsektoren steht. Nachteilig ist insbesondere, dass bei größeren Substanzen oft einzelne unnötig lange Kanten auftreten (s. Abb 6.15(a)). Außerdem ist es durch das Design des Algorithmus nicht möglich, den Baum mehrmals nacheinander mit demselben Verfahren abzarbeiten, was bestimmte Optimierungen ausschließt.

Vorteilhaft ist hingegen, dass der Layouter sich flexibel erweitern lässt, so kann man beispielsweise bestimmte Knoten von Verarbeitungsschritten ausnehmen oder schon vorher mit Daten belegen. So wurde hier darauf verzichtet, C-Ketten gewinkelt darzustellen, was sich aber leicht einbauen lässt. Einige Erweiterungen werden im nächsten Kapitel vorgestellt.

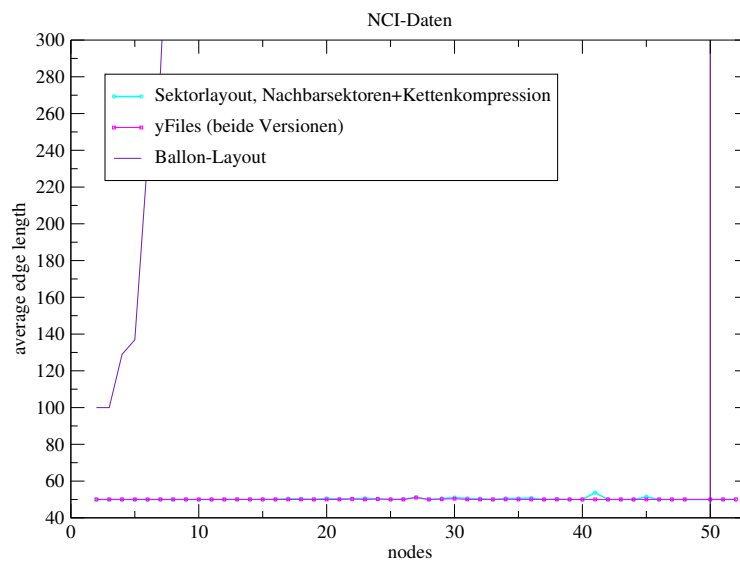
Durchschnittliche Kantenlängen, NCI-Daten Winkelneuverteilungsstrategien



Optimierungsstrategien

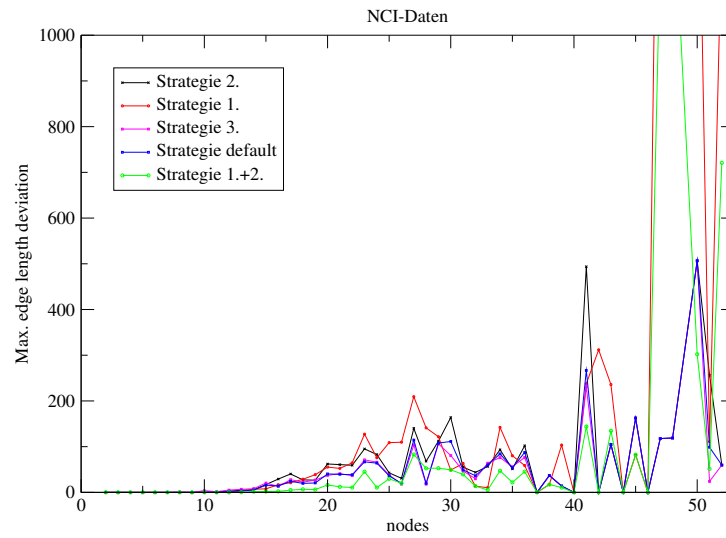


Vergleich der Algorithmen

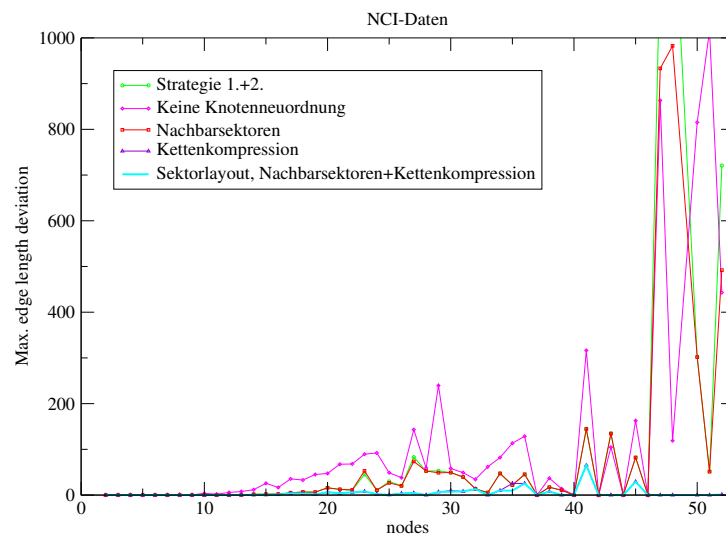


Maximale Abweichung der Kantenlänge, NCI-Daten

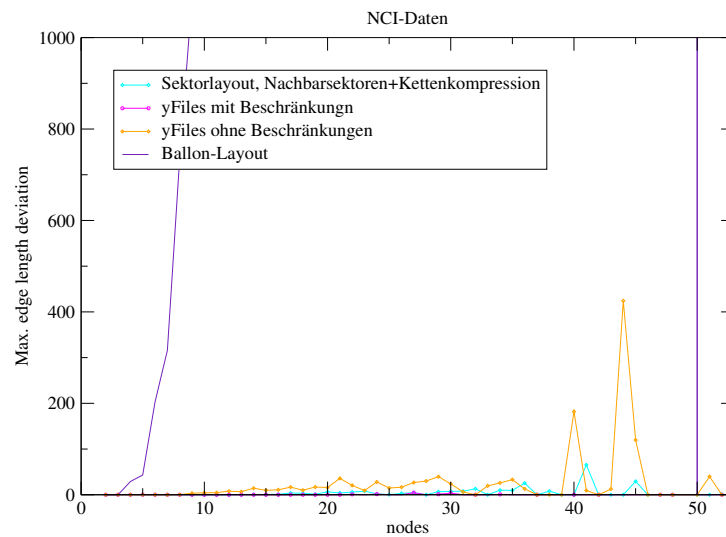
Winkelneuverteilungsstrategien



Optimierungsstrategien

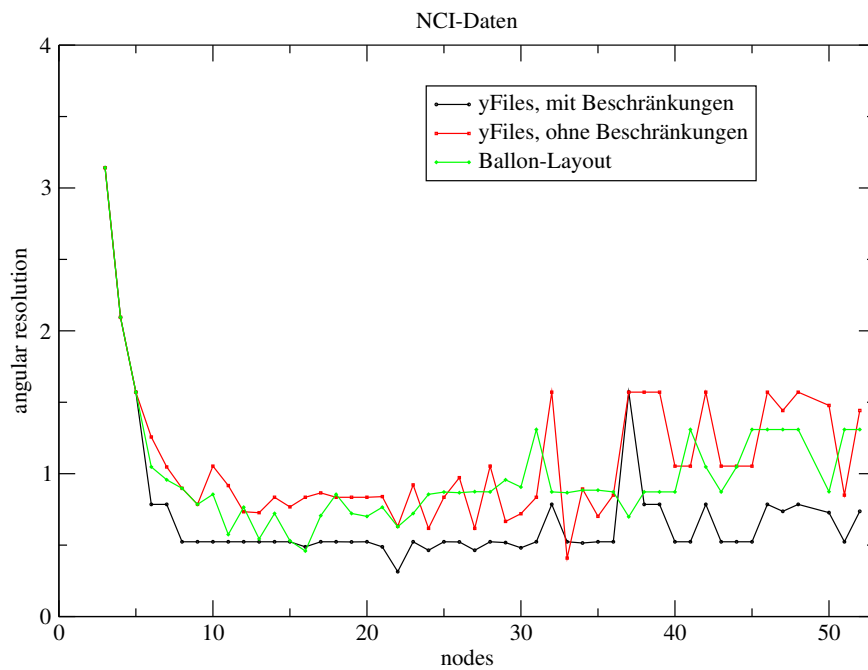


Vergleich der Algorithmen



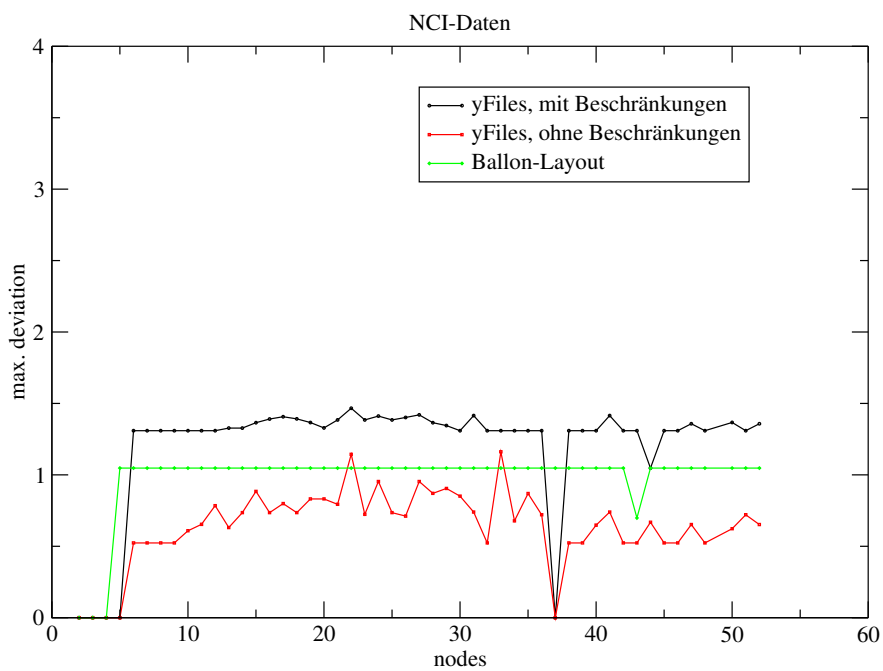
Winkelauflösung, NCI-Daten

Minimale Winkelauflösung

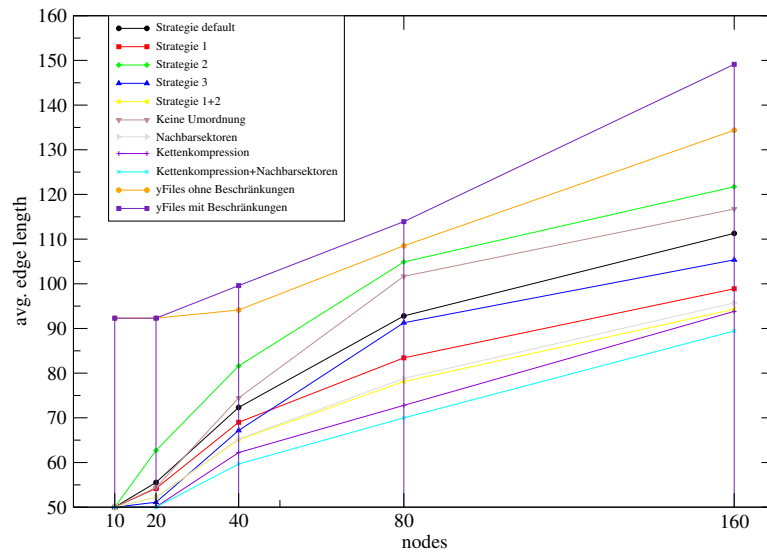


Abweichung vom Idealwinkel, NCI-Daten

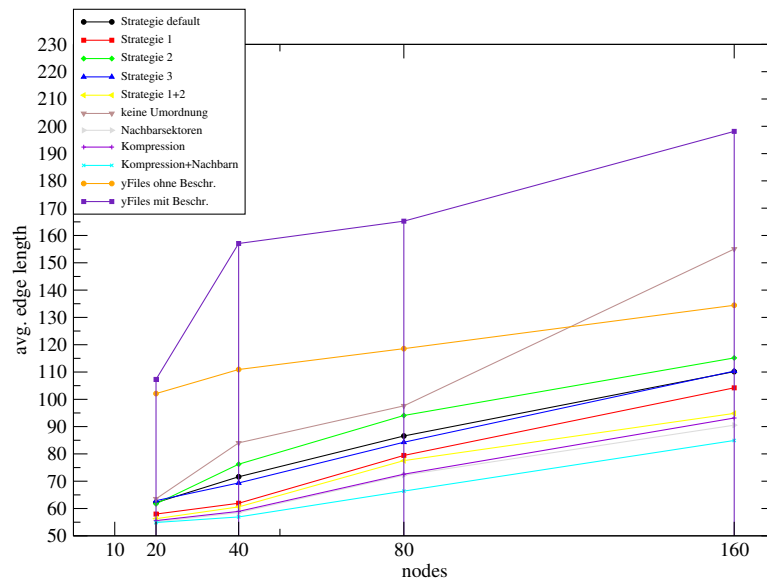
Maximale Abweichung vom Idealwinkel



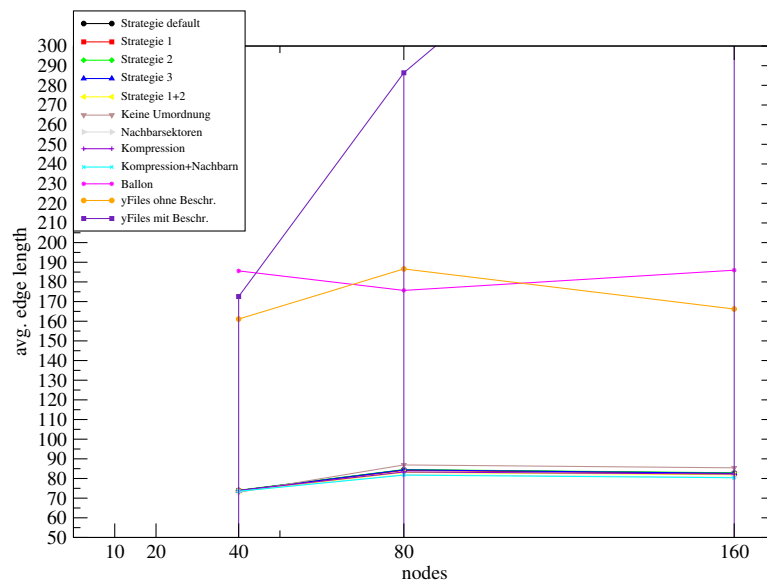
Durchschnittliche Kantenlängen, Synthetische Daten Maximalgrad 3



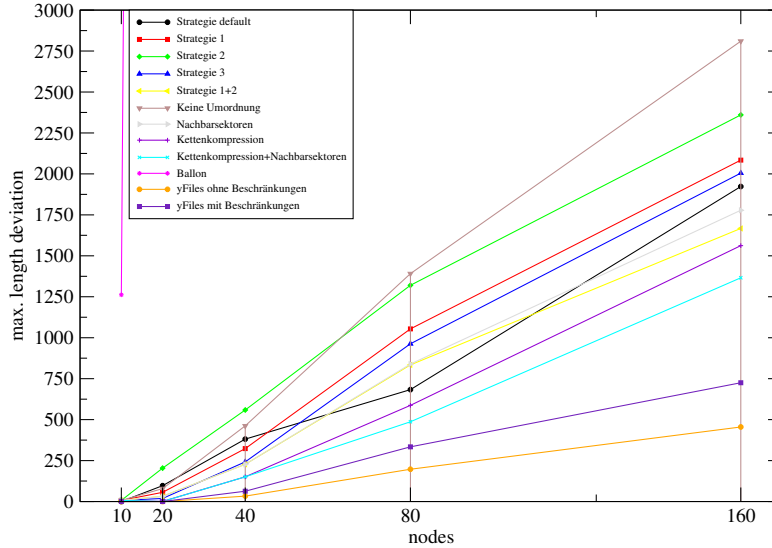
Maximalgrad 6



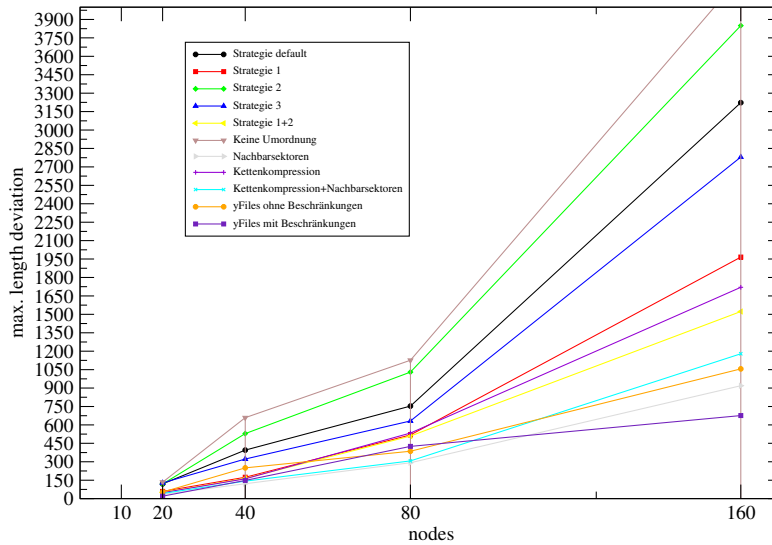
Maximalgrad > 10



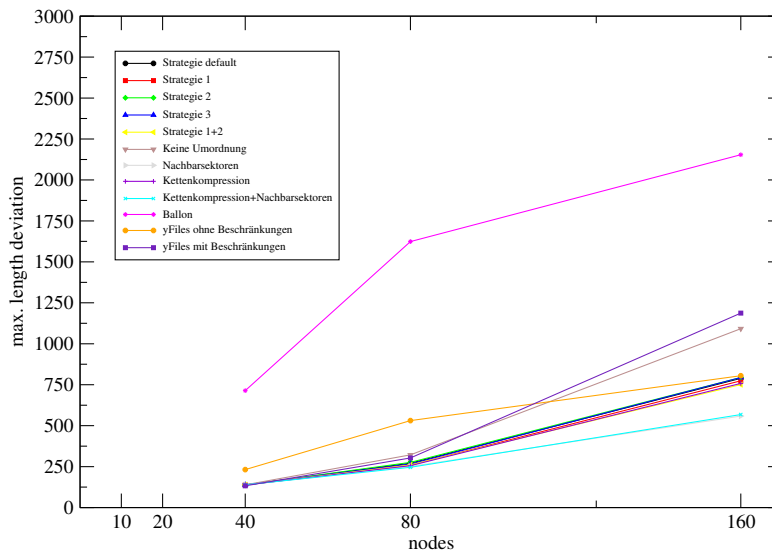
Maximale Abweichung der Kantenlängen, Synthetische Daten Maximalgrad 3



Maximalgrad 6

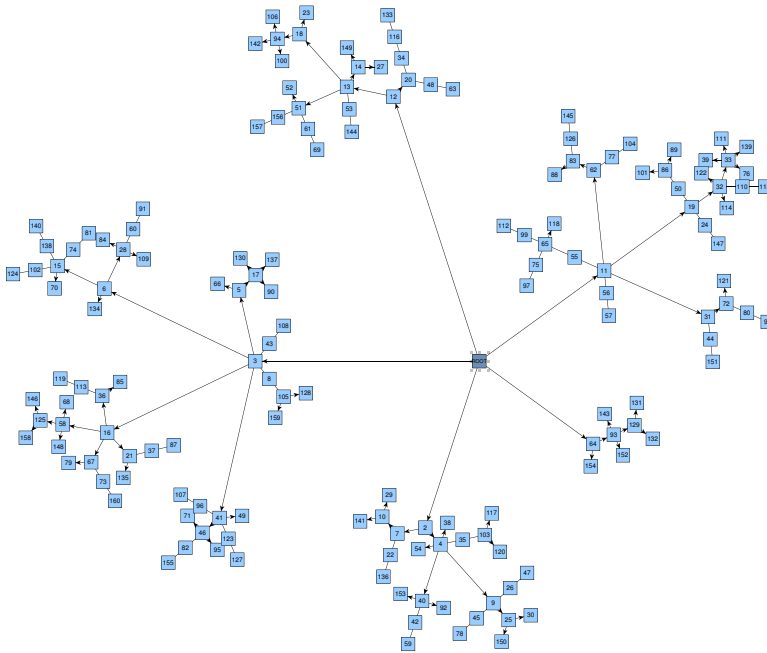


Maximalgrad > 10



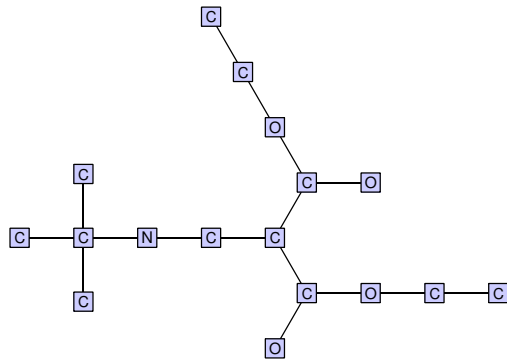


(a) Layout der Substanz Nr. 181462. Man erkennt die unnötig lange Kante, obwohl die einzelnen Teilbäume sehr gut aussehen. Hier hat sich die unnötig große Sektorbestimmung des rechten Teilbaums negativ ausgewirkt

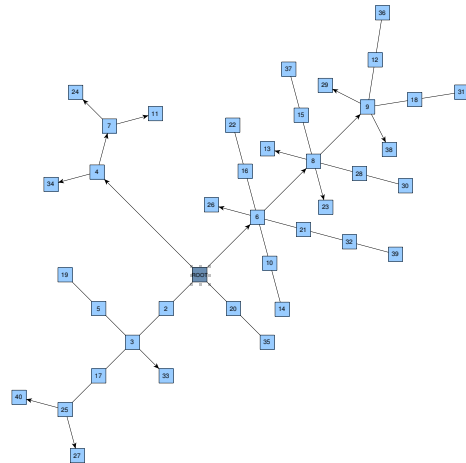


(b) Bei Bäumen höherer Ordnung ist das Ergebnis oft sehr gut, hier ein synthetischer Baum

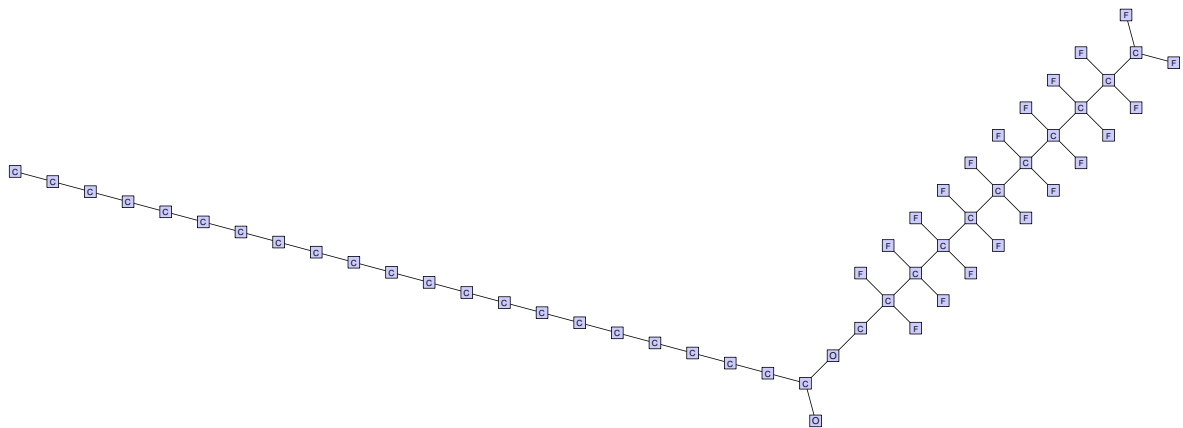
Abbildung 6.15: Beispiele 1



(a) Layout der Substanz Nr. 156896. Kleine Substanzen werden fast immer optimal gezeichnet



(b) Ein synthetischer Baum. Die entstehenden Layouts unterscheiden sich oft stark von den üblichen Layouts von Bäumen



(c) Layout der Substanz Nr. 65381. Überraschenderweise spielte hier die Wahl des Startknotens keine Rolle

Abbildung 6.16: Beispiele 2

Kapitel 7

Gesamtlayout

Zum Abschluss der Arbeit soll noch kurz darauf eingegangen werden, wie man die beiden in den vorigen Kapiteln beschriebenen Layouter verbinden kann, um das Gesamtlayout für ein Molekül zu erhalten. Für den einfachsten Fall wird dabei eine vollständige Beschreibung der Änderungen, die man gegenüber den in den vorigen Kapiteln beschriebenen Algorithmen vornehmen muss, angegeben. Danach wird kurz erläutert, wie man komplizierte Fälle darauf zurückführen kann, und welche Probleme dort auftreten können. Somit liefert dieses Kapitel auch einen Ausblick darauf, welche Erweiterungen mit den beschriebenen Verfahren noch möglich sind.

Im Folgenden sei dafür das Molekül stets wieder als zusammenhängend und außenplanar, aber nicht mehr notwendig 2-fach zusammenhängend vorausgesetzt.

7.1 Grundgerüst

Die Idee für das Gesamtlayout ist sehr einfach:

1. Ordne dem Graphen G eine Baumstruktur B zu, die alle wesentlichen strukturellen Aspekte beschreibt.
2. Finde ein Layout für B mit einem der im vorigen Kapitel beschriebenen Verfahren, wobei an geeigneter Stelle Modifikationen vorzunehmen sind.

Für Schritt 2. wird im Folgenden stets der beschriebene Sektorlayouter mit allen Optimierungen zum Einsatz kommen. Dies erleichtert das Layout von Teilbäumen, die an Blöcken hängen, da mit dem Sektorlayouter Bäume prinzipiell in beliebige Richtungen gezeichnet werden können. Für Schritt 1. wird man intuitiv versuchen, jedem Block von G einen Knoten von B zuzuordnen. Ein erster Ansatz führt auf den sogenannten **Block-** oder **Gliedergraphen** $B(G) = (V_B, E_B)$, wobei V_B die Menge aller Blöcke von G ist, und $\{a, b\} \in E_B$ genau dann, wenn a und b einen Schnittknoten gemeinsam haben (s. Abb. 7.1(b)). $B(G)$ ist allerdings im Allgemeinen kein Baum[Har74]. Besser geeignet ist ein Graph, dessen Knotenmenge aus den Blöcken von G und allen Schnittknoten besteht (s. Abb. 7.1(c)):

Definition 7.1 (Block-Artikulations-Baum). Sei G ein zusammenhängender Graph mit den Blöcken $\{B_i\}$ und den Schnittknoten (oder **Artikulationen**) $\{c_j\}$. Der **Block-Artikulations-Baum** $bc(G)$ hat die Knotenmenge $\{B_i\} \cup \{c_j\}$, und zwei Knoten aus $bc(G)$ sind genau dann adjazent, wenn gilt: $a \in \{B_i\}$ und $b \in \{c_j\}$, sowie $b \in a$.

Satz 7.2. [Har74] Ein Graph G ist genau dann der Block-Artikulations-Baum eines Graphen, wenn G ein Baum ist, in dem der Abstand je zweier Blätter gerade ist.

Damit ist die Bezeichnung Block-Artikulations-**Baum** nachträglich gerechtfertigt. Knoten aus $bc(G)$, die einem Block B_i in G entsprechen, heißen im folgenden einfach **Blockknoten**.

Nachteilig ist hier, dass auch für triviale Blöcke, die nur aus einer Kante bestehen, ein zusätzlicher Knoten in $bc(G)$ eingefügt wird, was das Layout der Baumkomponenten in G unnötig erschwert. Daher wird besser ein **komprimierter** Block-Artikulations-Baum $kbc(G)$ verwendet. In diesem werden Knoten, die einem trivialen Block entsprechen und keine Blätter in $kbc(G)$ sind, entfernt, und stattdessen eine Kante zwischen den beiden Nachbarn dieses Knotens eingefügt (s. Abb. 7.1(d)). Ein solcher Knoten v hat in $bc(G)$ immer Grad 2, da seine Nachbarn in $bc(G)$ gerade die beiden Enden der Kante sind, aus der der Block v besteht. Jeder Blockknoten für einen trivialen Block, der ein Blatt in $kbc(G)$ ist, entspricht dagegen einer Elternkante eines Blattes in G und wird durch dieses Blatt ersetzt. Zusätzlich wird von jedem Schnittpunkt, der zwei triviale Blöcke miteinander verbunden hat, dieses Attribut wieder entfernt, so dass man später beim Ablaufen von $kbc(G)$ leicht erkennt, ob man gerade einen nicht-trivialen Block betreten oder verlassen hat.

Wenn für G die Mengen $\{B_i\}$ und $\{c_j\}$ gegeben sind, so kann man $kbc(G)$ leicht bestimmen, wenn man beim Bestimmen der Blöcke für jeden Knoten gleich die Blöcke speichert, zu denen er gehört.

$kbc(G)$ enthält nun genügend strukturelle Informationen über G , um daraus ein Gesamtlayout für G konstruieren zu können. Der Einfachheit halber wird davon ausgegangen, dass zwischen den Elementen in G und in $kbc(G)$ geeignete Verknüpfungen existieren, so dass man beispielsweise durch Zugriff auf den Knoten b in $kbc(G)$ gleich auch Zugriff auf den korrespondierenden Block oder Schnittpunkt in G erhält.

Zur Vereinfachung benötigen wir noch folgende Definition:

Definition 7.3 (Eintritts/Austrittsknoten). Für einen Blockknoten b aus $kbc(G)$ heißt der Elternknoten r **Eintrittsknoten** bzgl. b , und alle Kinder c_i von b heißen **Austrittsknoten** bzgl. b . Die Elternkante eines Eintrittsknotens heißt **Eintrittskante**, alle Kindkanten von Austrittsknoten, sowie alle Kindkanten eines Eintrittsknotens, die nicht inzident zu b sind, heißen **Austrittskanten** bzgl. b . Nach Konstruktion von $kbc(G)$ sind Eintritts- und Austrittsknoten jeweils Schnittpunkte (s. Abb. 7.2(a)).

Damit erhält man für das Grundgerüst des Gesamtlayouts:

```

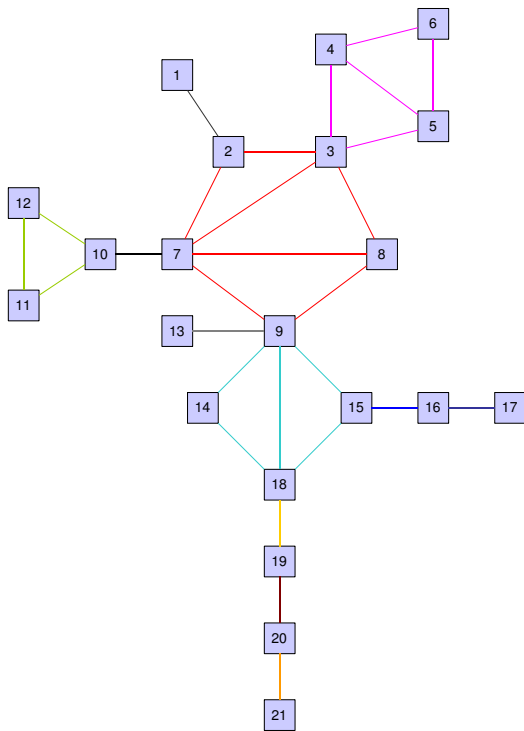
Input : Ein Molekül  $G$ 
1 begin
2   if  $G$  ist kein Baum then
3     Bestimme  $kbc(G)$  und einen Zentrumsknoten  $z$  von  $kbc(G)$ 
4     Finde (mit Breitensuche) einen nichttrivialen Blockknoten  $b$  mit minimaler Distanz
       zu  $z$ 
5     Wende Sektorlayout auf  $kbc(G)$  mit Wurzel  $b$  an
6     Übertrage die Layoutinformationen von  $kbc(G)$  auf  $G$ 
7   else
8     Wende Sektorlayout auf  $G$  einem Zentrumsknoten als Wurzel an
9
10 end

```

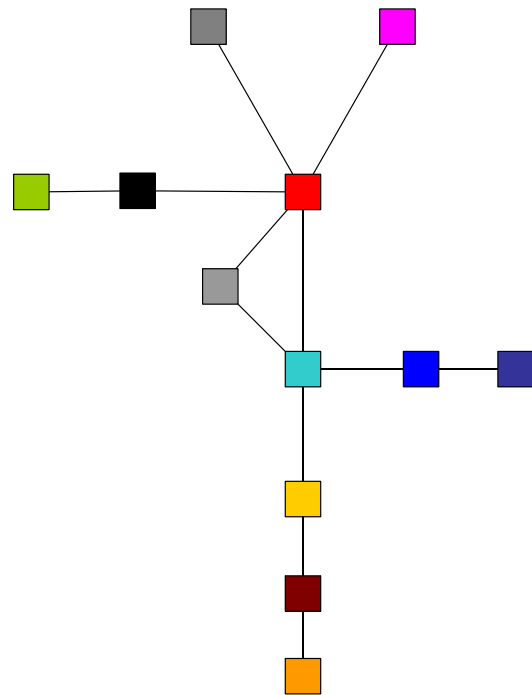
Algorithmus 24 : Gesamtlayout, Gerüst

Dies ist natürlich noch bei Weitem kein vollständiger Algorithmus:

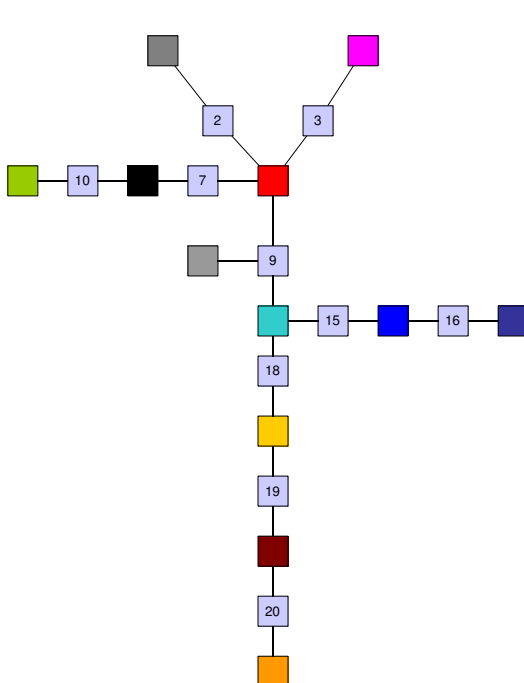
1. Es wird nicht berücksichtigt, dass Blockknoten einen anderen Platzbedarf haben als normale Knoten.
2. Ausgangskanten von Schnittpunkten müssen einen modifizierten relativen Rotationswinkel bekommen.



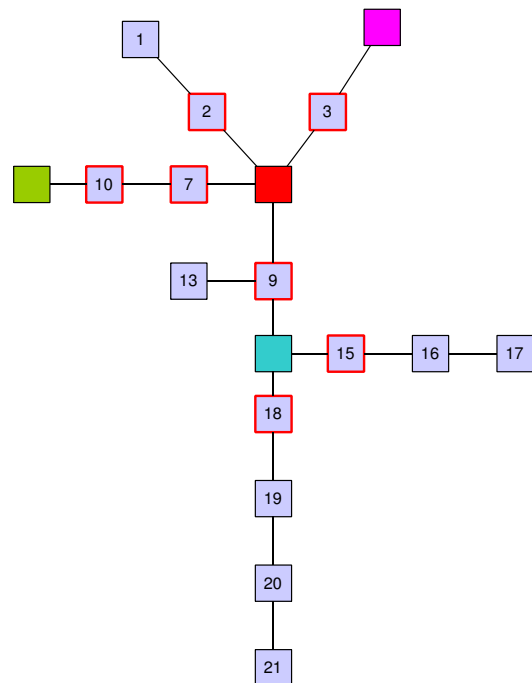
(a) Gegebener Graph, zu demselben Block gehörende Kanten sind farbig markiert



(b) Der zugeordnete Blockgraph. Man erkennt, dass dieser i.A. kein Baum ist



(c) Der zugehörige Block-Artikulations-Baum



(d) Der zugehörige kondensierte Block-Artikulations-Baum. Als Schnittknoten markierte Knoten sind rot umrandet

Abbildung 7.1: Die verschiedenen Strukturgraphen für einen Graphen

3. Die zyklische Ordnung von Schnittknoten um einen Blockknoten darf nicht geändert werden.
4. Bei der Generierung absoluter Koordinaten ist zu berücksichtigen, dass Blöcke anders gezeichnet werden müssen als normale Knoten.

Statt in einem Durchlauf ein Layout für den gesamten Baum $kbc(G)$ zu erzeugen, ist es einfacher, erst von außen für alle Teilbäume endgültige relative Koordinaten zu generieren. Dazu ist folgende Eigenschaft des Sektorlayouts (wie auch des Radiallayouts) sehr nützlich:

Bemerkung 7.4. Um für einen Teilbaum $T(k)$ eines Baumes vollständige relative Koordinaten (bzgl. k) generieren zu können, muss der für k am Elternknoten von k bereitgestellte Winkel oder die Distanz von k nicht bekannt sein.

Damit kann man den Algorithmus für das Gesamtlayout modifizieren:

```

Input : Ein Molekül  $G$ 
1 begin
2   if  $G$  ist kein Baum then
3     Bestimme  $kbc(G)$ 
4     Bestimme einen Zentrumsknoten  $z$  von  $kbc(G)$ 
5     Finde (mit Breitensuche) einen nichttrivialen Blockknoten  $b$  mit minimaler Distanz
      zu  $z$ 
6     firstPass( $kbc(G), b$ )
7   else
8     .
9   ..
10 end

```

Algorithmus 25 : Gesamtlayout, Gerüst

wobei **firstPass** die gerade beschriebene Idee umsetzt:

```

firstPass(Knoten  $r$ )
begin
  forall Kinder  $c$  von  $r$  do
  | firstPass( $c$ )
3  if  $r$  ist Blockknoten then
  | Zeichne den zu  $r$  gehörenden Block
  | Markiere Elternknoten von  $r$  als Eintrittsknoten von  $b$ 
  | Bestimme den relativen Rotationswinkel  $\phi'$  der ersten Kante auf dem Rand von  $b$ 
  | bzgl. der Eintrittskante
  | Bestimme die verfügbaren Winkel und relativen Rotationswinkel für alle
  | Austrittsknoten und -kanten
  | forall Austrittsknoten  $c$  zu  $r$  do
  | | Wende Sektorlayout auf  $T(c)$  mit Wurzel  $c$  und verfügbarem Winkel  $\alpha(c)$  an
12 if  $r$  ist Eintrittsknoten für  $b$  then
  | Bestimme den für alle Austrittskanten an  $r$  zur Verfügung stehenden Winkel und die
  | relativen Rotationswinkel
  | forall Kinder  $c \neq b$  von  $r$  do
  | | Wende Sektorlayout auf  $T(r)$  mit Wurzel  $r$  und verfügbarem Winkel  $\alpha$  an
  | | Bestimme den belegten Sektor für  $r$ 
end

```

Prozedur firstPass(Knoten r)

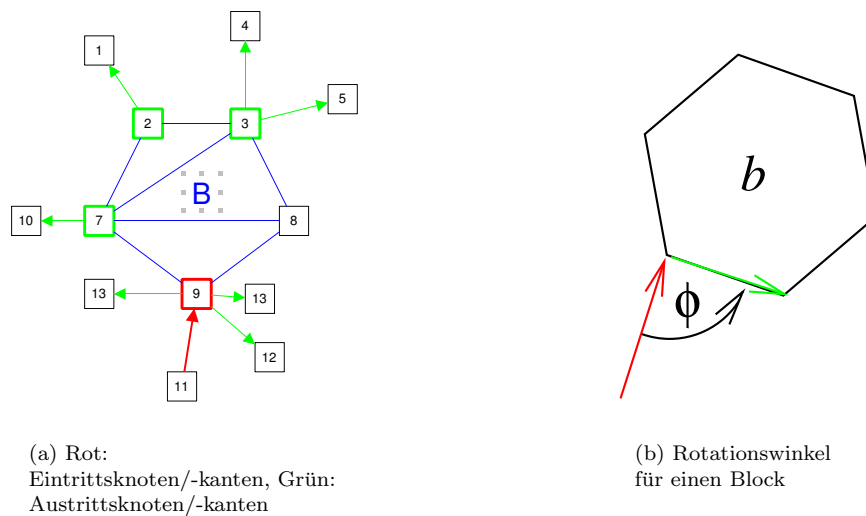


Abbildung 7.2: Definition und Behandlung von Eintrittsknoten

Die angegebenen Schritte bedürfen einiger Erläuterung:

- Ab Zeile 3 wird der zu einem Blockknoten gehörende Block gezeichnet, was für außenplanare Blöcke mit einem der Algorithmen aus Kapitel 5 geschehen kann. Damit kann man dann für alle Kinder c von r bestimmen, welchen Winkel α man den daran hängenden Teilbäumen zur Verfügung stellen kann, ohne dass es Überschneidungen mit dem Block gibt. Dies geschieht in der Prozedur **setAvailableAngle**, die weiter unten noch erläutert wird. Ebenso werden dort bereits die relativen Rotationswinkel für die Ausgangskanten aus c berechnet und zugewiesen. Danach kann man dann den Teilbaum $T(c)$, beginnend bei c , auslegen. Hierzu muss man den Algorithmus für das Sektorlayout an zwei Stellen modifizieren: Zum einen wird auch für die Wurzel der verfügbare Winkelbereich auf α eingeschränkt. Zum anderen wird im Sektorlayout nur soweit abgestiegen, bis ein Schnittknoten gefunden wird, und dann der komplette daran hängende Teilbaum am Stück an das bestehende Layout angehängt. Dies ist möglich, weil in **firstPass** dieser Teilbaum bereits berechnet wurde.
- Zeile 12ff wird ausgeführt, wenn ein Block b über einen Schnittknoten r verlassen wird. Damit ist das Layout für b und alle Kinder von b bereits bekannt. Da aber an r noch weitere Teilbäume hängen können (s. Abb. 7.2(a)), müssen diese zuerst auch ausgelegt werden, bevor der von r belegte Platz bestimmt werden kann.

Die Generierung absoluter Koordinaten für die Knoten kann im Wesentlichen analog zum normalen Sektorlayout erfolgen, es ist lediglich die Behandlung der Blöcke einzufügen. Dazu wird beim *preorder*-Durchlauf durch $kbc(G)$ jedesmal dann, wenn man einen Eintrittsknoten findet, für alle Kinder, die einen Block b repräsentieren, dieser als ganzes rotiert und verschoben, und diese Transformation an alle Kinder c_i von b weitergegeben. Hierzu wird der relative Rotationswinkel der ersten Kante in $H(b)$ nach dem Eintrittsknoten bestimmt, und zwar bzgl. der Eintrittskante (s. Abb. 7.2(b)).

7.2 Bestimmung der Sektoren für Schnittknoten und Blöcke

Im Folgenden soll nun die oben bereits angesprochene Berechnung der Sektoren für Blöcke bzw. für deren Ein- und Austrittsknoten genauer erläutert werden. Es zeigt sich, dass hier in vielen

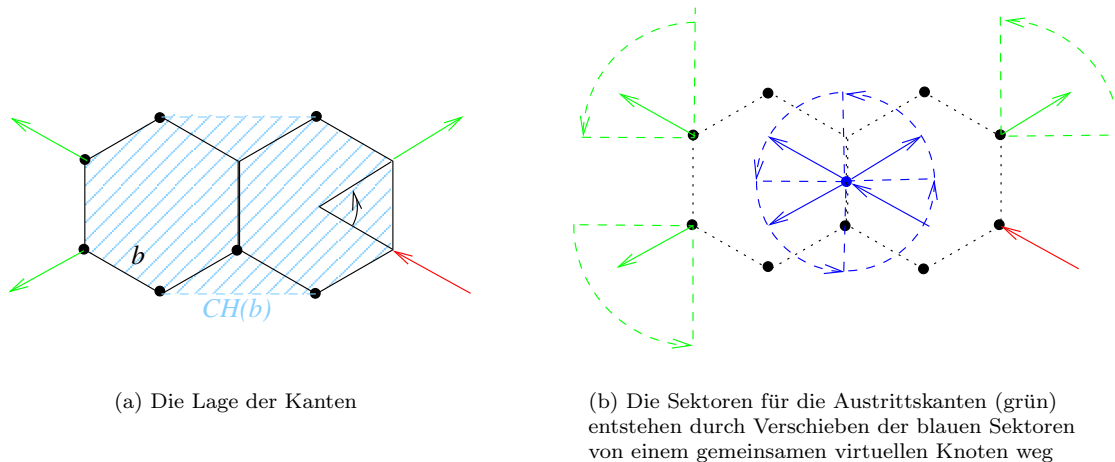


Abbildung 7.3: Die einfachste Situation bei Blöcken

Fällen Probleme auftreten, die es beim einfachen Sektorlayout für Bäume nicht gibt. Dies liegt im Wesentlichen daran, dass die Position der Schnittknoten an einem Block relativ zueinander bereits festliegt, so dass es beim Anhängen der Teilbäume zu Problemen kommen kann. Oft ist keine der möglichen Lösungen wirklich befriedigend. Die unterschiedlichen Situationen werden nun geordnet nach Komplexität vorgestellt, wobei zur Vereinfachung zuerst angenommen sei, dass sich keine zwei Blöcke in Spiro-Anordnung befinden, diese also nicht nur genau einen Knoten gemeinsam haben. Außerdem bezeichne $CH(b)$ stets die konvexe Hülle von b .

7.2.1 Nur eine Aus-/Eintrittskante, streng monoton wachsende Rotationswinkel, jeder Schnittknoten ist eine Ecke der konvexen Hülle

Die einfachste und auch häufigste Situation ist in 7.3(a) gezeigt. Hier sind folgende Bedingungen erfüllt:

1. Zu jedem Schnittknoten, der zum Block b gehört, ist genau eine Kante inzident, die nicht zu b gehört (dies kann eine Eintritts- oder Austrittskante sein).
2. Bestimmt man die Winkel jeder Aus- bzw. Eintrittskante, so dass an diesen Stellen Winkeluniformität (wenigstens für diese Kanten bzgl. der Blockkanten) gewährleistet ist, so ist der Winkel zwischen zwei auf dem Rand von b direkt aufeinanderfolgenden Kanten positiv.
3. Jeder Schnittknoten ist eine Ecke von $CH(b)$ und die beiden dazu inzidenten Kanten auf dem Rand von b sind Kanten von $CH(b)$.

Lemma 7.5. *Sind die Bedingungen 1.-3. für einen Block b erfüllt, so kann man stets Sektoren für die Schnittknoten so bestimmen, dass sich diese weder paarweise noch mit b überschneiden, und jede Ein- bzw. Austrittskante bei uniformer Winkelzuweisung in das Innere des entsprechenden Sektors zeigt.*

Beweis. Weist man den Aus- und Eintrittskanten wie in Bedingung 2. relative Rotationswinkel zu, so stellt diese Bedingung sicher, dass man immer die Sektoren für die Schnittknoten so wählen kann, dass sich diese paarweise nicht schneiden: Bestimmt man nämlich die Sektoren so, wie wenn alle diese Kanten denselben Elternknoten besäßen, so schneiden sich diese Sektoren auch dann nicht, wenn man jeden Schnittknoten auf den Rand von b verschiebt (s. Abb. 7.3(b)). Neben

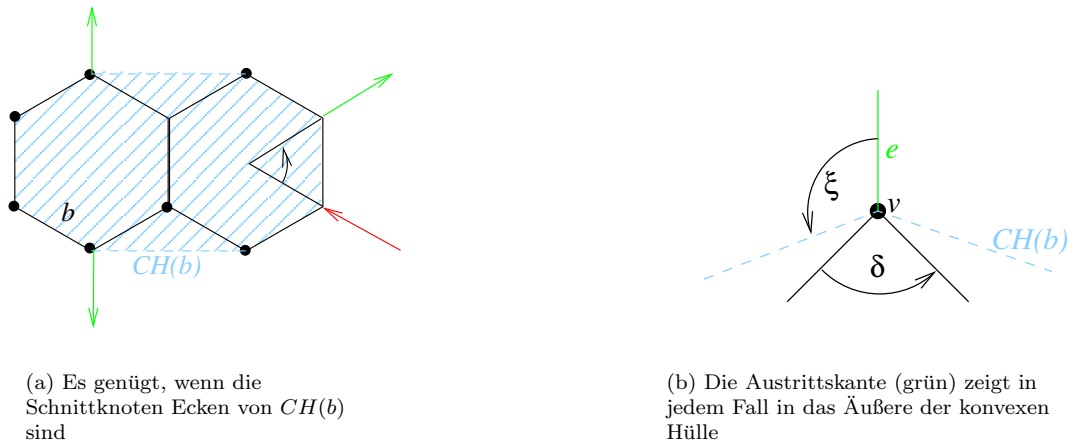


Abbildung 7.4: Schnittknoten auf der konvexen Hülle

dieser Einschränkung für nebeneinanderliegende Sektoren ist noch zu berücksichtigen, dass die Sektorgrößen so gewählt werden müssen, dass der Sektor zu keinen Überschneidungen mit b führt. Wegen Bedingung 3. kann man für die linke und rechte Begrenzung einfach die Richtungen der nach links bzw. nach rechts inzidenten Kanten auf dem Rand von b wählen, wobei die entsprechende Ein- oder Austrittskante wegen der Winkeluniformität dann ebenfalls in diesen eingeschränkten Bereich zeigt. \square

Bemerkung 7.6. Bedingung 3. lässt sich abschwächen: Es genügt, wenn jeder Schnittknoten eine Ecke von $CH(b)$ ist (Abb. 7.4(a)).

Beweis. Es genügt zu zeigen, dass auch hier immer noch jede Kante in das Äußere der konvexen Hülle zeigt. Sei v ein Schnittknoten auf der konvexen Hülle. Im allgemeinen hat die zu v inzidente Aus- oder Eintrittskante zur konvexen Hülle keine uniformen Winkel mehr, zeigt aber immer noch bzgl. $CH(b)$ nach außen, denn wenn ξ der Winkel zwischen der Kante e und einer der zu v inzidenten Kanten der konvexen Hülle ist, δ der Innenwinkel von b an v , so wird ξ minimal, wenn eine der zu v inzidenten Kanten aus ∂b auf $CH(b)$ liegt. In diesem Fall gilt $\xi \leq 2\pi - (\pi - \delta) - \delta - \frac{2\pi - \delta}{2} = \frac{\delta}{2} > 0$ (s. Abb. 7.4(b)). \square

Sektorbestimmung für Austrittsknoten

Damit kann man in `setAvailableAngle` die relativen Rotationswinkel und die Sektoren für alle Austrittsknoten bestimmen, wobei wieder für b eine linksorientierte außenplanare Einbettung mit Hamiltonkreis $H(b)$ vorausgesetzt wird. Ferner wurde der Eintrittsknoten von b bereits in `firstPass` bestimmt.

Bisher nicht eingegangen wurde auf die Bestimmung von $CH(b)$ von b bzw. hier von $H(b)$. Hierfür gibt es mehrere Möglichkeiten: zum einen kann man mit einem der Standardalgorithmen (z.B. Graham Scan) die konvexe Hülle von $H(b)$ in $O(|H(b)| \log |H(b)|)$ bestimmen. Allerdings wird hier nicht die ganze Hülle benötigt, sondern lediglich eine Entscheidung, ob v eine Ecke der Hülle ist, sowie gegebenenfalls die links und rechts an v inzidenten Kanten dieser Hülle. Nimmt man an, dass $H(b)$ links orientiert ist, wie dies die Algorithmen aus Kapitel 5 leisten, kann dies in $O(|H(b)|)$ geschehen, beispielsweise durch Algorithmus 28. Dieser verwendet die Beobachtung aus dem Beweis für Lemma 7.6 und bestimmt dabei gleich die Richtungen der Kanten von $CH(b)$.

```

setAvailableAngle(Knoten b)
Data : Liste cyclicOrder: Zyklische Ordnung der Schnittknoten um b in kbc(G)
Data : Eintrittsknoten v' bzgl. b
begin
  /*Bestimme zuerst die zyklische Ordnung der Knoten in kbc(G) konsistent mit b */
  forall Knoten v in H(b), beginnend bei v' do
    | if v ist Schnittknoten und  $\neq v'$  then Füge v an das Ende von cyclicOrder an
    | Setze die zyklische Ordnung der Kinder von b auf cyclicOrder
  if v' liegt nicht auf CH(b) oder v' hat mehr als ein Kind then return Kein Layout
  möglich
  /*Bestimme die Lage von b bzgl. v', indem die Winkel der Randkanten relativ zur
  Eintrittskante akkumuliert werden */
   $\delta \leftarrow$  Innenwinkel von b an v'
   $\phi_{rel} \leftarrow \pi - \delta/2$ 
  forall Knoten k  $\in H(b)$ , beginnend mit dem Nachfolger von v' do
    | if Zu k ist mehr als eine Kante  $\notin b$  inzident then return Kein Layout möglich
    |  $\delta \leftarrow$  Innenwinkel von b an k
    | if k ist Austrittsknoten then
      | if k liegt auf CH(b) und hat nur ein Kind k' then
        | /*Bestimme den Rotationswinkel */
        |  $\phi(child(k)) \leftarrow \phi_{rel} + \frac{2\pi-\delta}{2}$ 
        |  $\alpha_l(k) \leftarrow$  Winkel zwischen  $\{k, k'\}$  und der linken Kante an k der konvexen
        | Hülle von b
        |  $\alpha_r(k) \leftarrow$  Winkel zwischen  $\{k, k'\}$  und der rechten Kante an k der konvexen
        | Hülle von b
      | else
      | return Kein Layout möglich
    |  $\phi_{rel} \leftarrow \phi_{rel} + \pi - \delta$ 
  forall Knoten k  $\in \{v'\} \cup childrenOrder$  do
    | if Winkel  $\omega$  zwischen Austrittskante an k und der nächsten Austrittskante an k' ist
    | positiv then
    | |  $\alpha_l(k) \leftarrow \min(\alpha_l(k), \omega/2)$ 
    | |  $\alpha_r(k') \leftarrow \min(\alpha_r(k'), \omega/2)$ 
    | else
    | return Kein Layout möglich
  end

```

Prozedur **setAvailableAngle**(Knoten *b*)

<p>Input : Ein Knoten v aus $H(b)$</p> <p>Output : Liegt v auf der konvexen Hülle von $H(b)$</p> <p>Output : Falls ja, welches sind die Richtungen der zu v inzidenten Kanten dieser Hülle</p> <pre> 1 begin 2 $\delta \leftarrow$ Innenwinkel von ∂b an v 3 $v' \leftarrow$ Nachfolger von v in $H(b)$ 4 if $\delta \geq \pi$ then return FALSE $\phi \leftarrow \delta/2$ 5 $minLeft \leftarrow 0$ 6 $maxRight \leftarrow 0$ 7 forall Knoten $c \in H(b) \setminus \{v\}$ do 8 $\alpha \leftarrow \angle v'vc$ 9 if $\alpha = \phi + \pi$ then return FALSE if $\alpha \in]\phi, \phi + \pi[$ then 10 $maxRight \leftarrow \max(maxRight, \alpha)$ 11 else 12 $minLeft \leftarrow \min(minLeft, \alpha)$ 13 14 if $maxRight - minLeft < \pi$ then 15 return TRUE, $maxRight, minLeft$ 16 else 17 return FALSE 18 19 end </pre>

Algorithmus 28 : Entscheidung, ob ein Knoten auf der konvexen Hülle liegt

Wenn $H(b)$ viele Schnittknoten enthält, ist im allgemeinen die explizite Bestimmung von $CH(b)$ effizienter, da für das alternative Verfahren die Laufzeit zur kompletten Bearbeitung von b im worst-case quadratisch in $|H(b)|$ ist.

Damit sind die verfügbaren Sektoren für alle Austrittsknoten bestimmt, und die daran hängenden Teilbäume können nun komplett gezeichnet werden.

Sektorbestimmung für Eintrittsknoten

Sei nun v ein Eintrittsknoten bzgl. b . Dann müssen noch die Größen des von b belegten Sektors sowie die Größen aller an den Austrittsknoten von b hängenden Teilbäume, allerdings bzgl. v , bestimmt werden.

Im modifizierten Sektorlayout werden dann diese Werte für das Layout des gesamten Teilbaums $T(v)$ verwendet.

Die Größe des von b selbst belegten Sektors $S(b)$ lässt sich unter den obigen Voraussetzungen leicht bestimmen: Die nach links bzw. nach rechts belegten Winkel sind wieder gerade die Winkel zwischen den entsprechenden Kanten von $CH(b)$ an v und der der Eintrittskante, und für die Gesamtlänge $l(v, b)$ des Sektors für b gilt einfach $l(v, b) = \max_{w \in b} d(v, w)$ (s. Abb. 7.5(a)).

Etwas komplizierter ist die Bestimmung der Winkel und Längen der Teilbäume an Austrittsknoten. Sei dazu v' ein Austrittsknoten von b und $S(c)$ der schon bestimmte Sektor für einen den Teilbaum an v' . Die Abmessungen des Sektors $S(v')$ bzgl. v erhält man dann einfach, indem man v' als Kind von v betrachtet und alle Berechnungen bzgl. v und einer virtuellen Kante (s. Abb. 7.5(b)). Damit kann man den Platzbedarf aller Teilbäume $T(v')$ bzgl. v , wobei v' die Austrittsknoten von b sind, bestimmen. Für die Bestimmung der Distanzen und Winkel für $T(v)$ werden dann später diese Sektoren $S_v(c)$ und $S(b)$ verwendet.

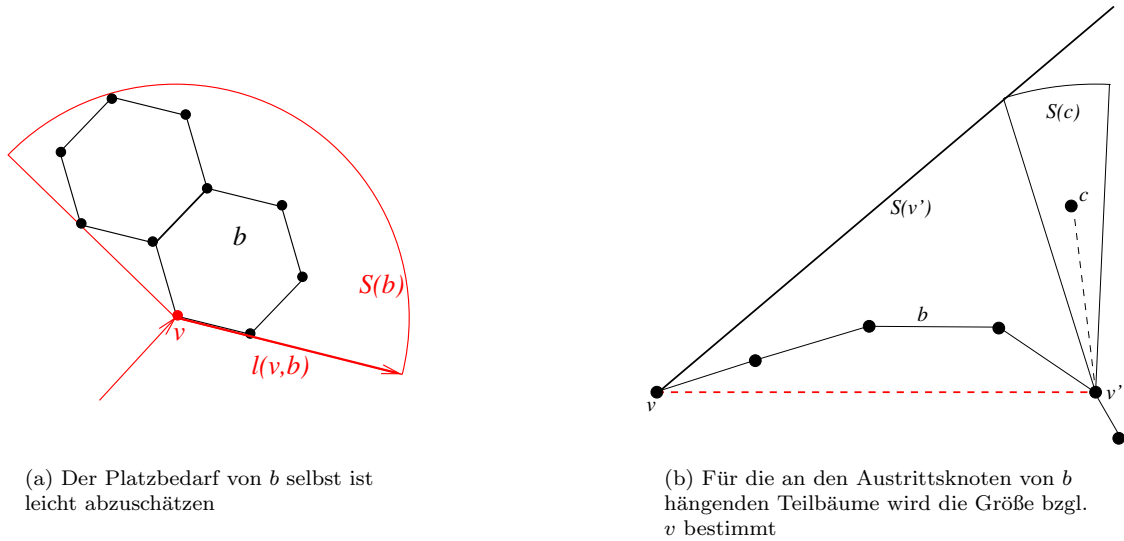


Abbildung 7.5: Bestimmung der Teilbaumgrößen für Eintrittsknoten

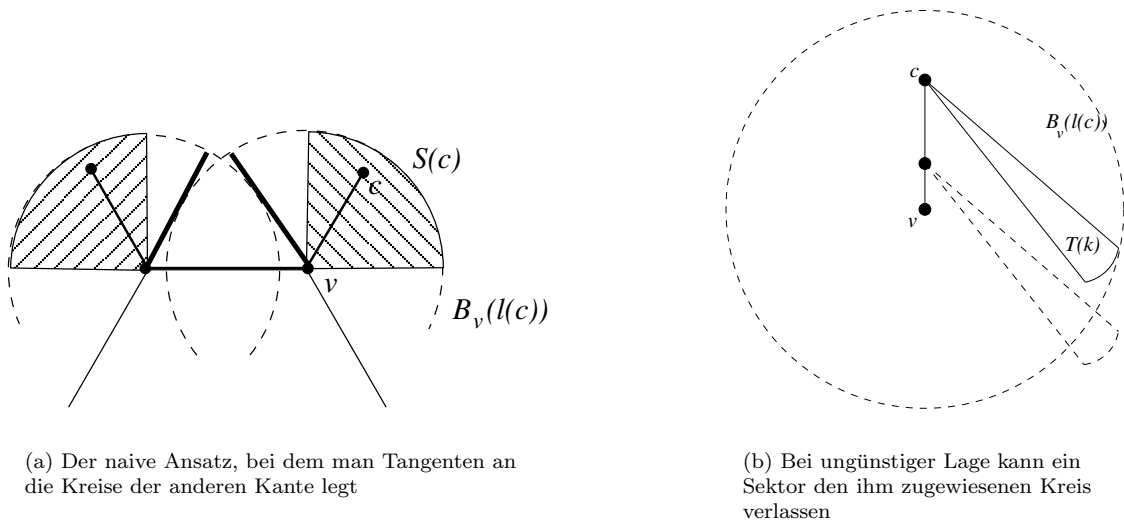


Abbildung 7.6: Vergrößerung des Sektors im einfachsten Fall

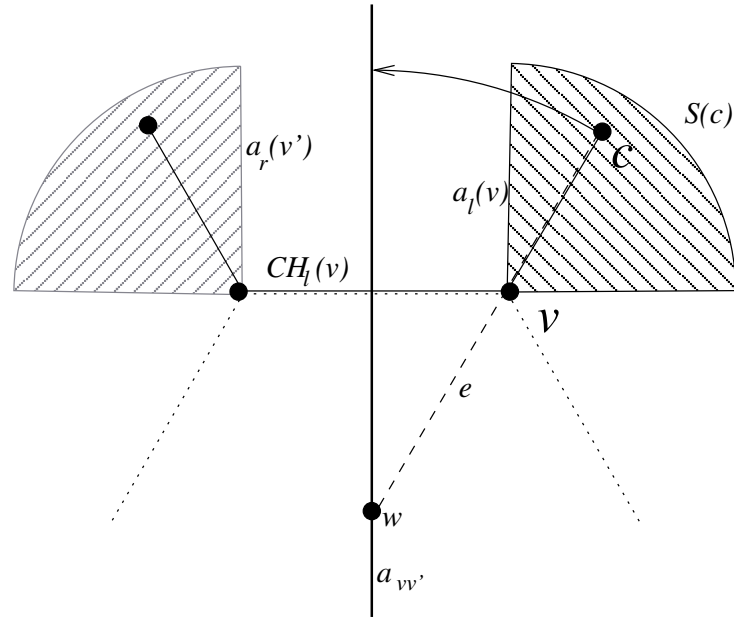


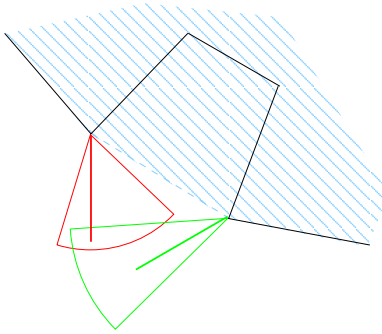
Abbildung 7.7: Zur Vergrößerung des Sektors kann man als linke/rechte Begrenzung die Mittelparallele zwischen den ursprünglichen Sektorgrenzen verwenden

Optimierungsmöglichkeiten

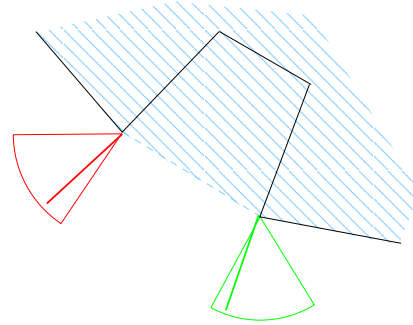
Bei diesem einfachen Ansatz sind einige Optimierungen denkbar. Neben den im vorigen Kapitel beschriebenen Verfahren, um Distanzen weiter zu verringern, ist insbesondere eine bessere Bestimmung der initialen Sektorgrößen interessant. Wie man in Abb. 7.3(b) sieht, wird durch die Wahl der Sektoren, wie sie oben vorgenommen wurde, der Platz im Allgemeinen nicht optimal genutzt, so könnte man hier für benachbarte Schnittknoten v und v' noch zusätzlichen Platz zwischen den beiden Sektoren verwenden. Da aber zum Zeitpunkt der initialen Sektorbestimmung die Längen der Teilbäume an v und v' noch nicht bekannt sind, muss dies nachträglich geschehen, so dass man dann durch eine Vergrößerung der Sektoren $S(c)$ und $S(c')$ die Distanzen der Kinder c und c' von v und v' noch verringern könnte. Der naive Ansatz, einfach die Tangenten an den jeweils benachbarten Kreis $B_v(l(c))$ zu verwenden (s. Abb. 7.6(a)), ist allerdings problematisch: Durch die Verringerung der Distanz $d(c)$ kann nämlich bei ungünstiger Lage eines Teilbaumes $T(k)$ dieser den Kreis $B_v(l(c))$ verlassen (s. Abb. 7.6(b)). Eine bessere Möglichkeit beruht auf der Beobachtung, dass die Strahlen $a_r(v')$ und $a_l(v)$ parallel sind, wenn man die Beschränkungen durch die konvexe Hülle vorerst nicht berücksichtigt, da sich diese Sektoren lediglich durch Verschiebung zweier an einem Punkt direkt benachbarter Sektoren ergeben haben. Für die linke Begrenzung von $S(c)$ kann man daher auch zusätzlich die Mittelparallele $a_{vv'}$ zwischen $a_r(v')$ und $a_l(v)$ heranziehen. Damit bestimmt man für einen links liegenden Sektor $S(k)$ an c die minimal mögliche Distanz

$$d_k(c) = \max\{l_{min}, d_{k,w,a_{vv'}}(c) - d(c,v), d_{k,v,CH_l(v)}(c)\}$$

wobei w der Schnittpunkt zwischen der Verlängerung der Kante $\{v, c\}$ und $a_{vv'}$ ist, und $d_{k,w,a_{vv'}}(c)$ die Minimaldistanz von c für den Sektor $S(k)$ innerhalb des Sektors mit Spitze w und linker Begrenzung $a_{vv'}$ bezeichne, sowie $d_{k,v,CH_l(v)}(c)$ entsprechend mit Spitze v und linker Begrenzung $CH_l(v)$ als linke Kante an v der konvexen Hülle des Blocks (s. Abb. 7.7). Analog geht man für rechts liegende Sektoren an c' vor. Hierfür muss insbesondere die Länge der Teilbäume an v und v' nicht im voraus bekannt sein, was man sich allerdings mit zusätzlichen Berechnungen bei der Distanzbestimmung erkauft.



(a) Hier schneiden sich die Sektoren zu den beiden Kanten auf jeden Fall, da der Winkel zwischen den Kanten negativ ist



(b) Wählt man die Rotationswinkel winkeluniform zur konvexen Hülle, so zeigen beide Kanten wieder voneinander weg

Abbildung 7.8: Wahl der Rotationswinkel bei sich schneidenden Sektoren

7.2.2 Andere Fälle

Wenn eine der Bedingungen von oben nicht erfüllt ist, kann man bei uniformer Winkelzuweisung im Allgemeinen die Sektoren für die Ein- und Austrittsknoten von b nicht so wählen, dass sich diese nicht gegenseitig oder mit b überlappen. Im Folgenden wird jeweils kurz beschrieben, welche Probleme auftreten können, und welche Strategien man zur Problembehebung anwenden könnte, wobei es oft verschiedene Möglichkeiten gibt.

Nicht positive Winkel zwischen benachbarten Aus-/Eintrittskanten, alle anderen Bedingungen erfüllt

In diesem Fall können sich die Sektoren benachbarter Aus-/Eintrittskanten überlappen (s. Abb. 7.8(a)). Die einfachste Abhilfe besteht darin, die Winkel der Aus-/Eintrittskanten nicht winkeluniform bezüglich ∂b , sondern bezüglich $CH(b)$ zu wählen (s. Abb. 7.8(b)). Damit ist wegen der Konvexität von $CH(b)$ der Winkel zwischen benachbarten Aus-/Eintrittskanten wieder positiv.

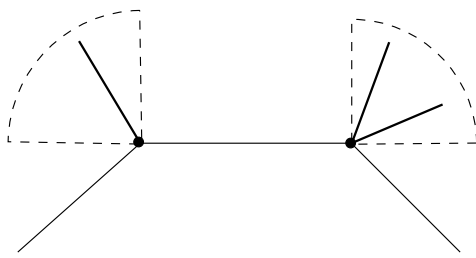
Stellt man nach dem Auslegen der Teilbäume an benachbarten Schnittpunkten v und v' fest, dass $B_v(l(v))$ und $B_{v'}(l(v'))$ sich nicht schneiden, so kann man den beiden Austrittskanten dann wieder uniforme Winkel bzgl. ∂b zuweisen. In diesem Fall handelt es sich jeweils um eine bloße Rotation von $S(v)$ bzw. $S(v')$, die die Längen der Sektoren nicht verändert, so dass das im vorigen Abschnitt beschriebene Problem nicht auftreten kann.

Mehrere Austrittskanten, Schnittpunkte sind Ecken von $CH(b)$

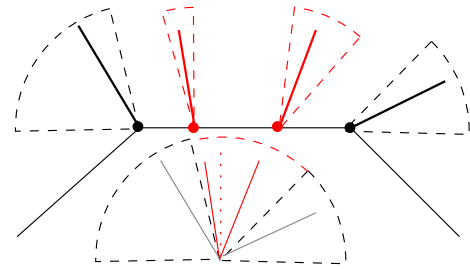
Dieser Fall unterscheidet sich nicht wesentlich von den vorhergehenden, auch hier kann man dem gesamten Teilbaum an einem Schnittpunkt v wie im vorangegangenen Abschnitt einen Sektor so zuweisen, dass sich dieser mit anderen nicht überschneidet, und diesen Winkel zwischen Austrittskanten an v gleichmäßig zur Verfügung stellen, wobei natürlich die relativen Rotationswinkel der einzelnen Kanten entsprechend zu wählen sind (s. Abb. 7.9(a)). Auch in diesem Fall ist eine Optimierung wie im vorigen Abschnitt problemlos möglich.

Nicht alle Schnittpunkte sind Ecken von $CH(b)$

Falls lediglich einige Schnittpunkte auf dem Rand von $CH(b)$ liegen, aber keine Ecken sind, kann es eintreten, dass die zugehörigen Austrittskanten bei uniformer Winkelzuweisung sowohl bzgl. ∂b



(a) Mehrere Austrittskanten an demselben Schnittknoten teilen sich den Sektor



(b) Parallelen Austrittskanten kann man ebenfalls einen Sektor zuweisen, den sie sich dann teilen

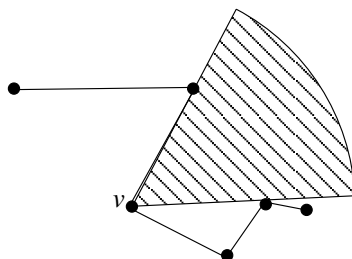
Abbildung 7.9: Sektoren und Rotationswinkel bei mehreren oder parallelen Austrittskanten

als auch $CH(b)$ parallel sind, wodurch kein Sektor bestimmt werden kann. In diesem Fall kann man aber wieder eine Sektoraufteilung wie im vorletzten Abschnitt finden, indem man für alle parallelen Austrittskanten gemeinsam einen Sektor bestimmt, und diesen dann wieder gleichmäßig so aufteilt, dass die relativen Rotationswinkel entlang der Kante streng monoton wachsen (s. Abb 7.9(b)).

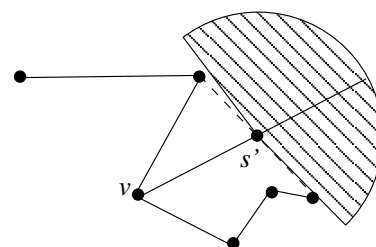
Schwieriger ist der Fall zu behandeln, bei dem Schnittknoten im Inneren von $CH(b)$ liegen. Betrachtet man zuerst die Situation für einen Schnittknoten v isoliert von den anderen, so hat man im allgemeinen zwei Möglichkeiten, einen Sektor für $T(v)$ zu bestimmen:

1. Bestimme einen Sektor mit Spitze v so, dass dieser sich nicht mit b überschneidet (s. Abb. 7.10(a)), und weise Winkel uniform bzgl. dieses Sektors zu: Hierzu muss man lediglich Algorithmus 28 so modifizieren, dass in jedem Fall $minLeft$ und $maxRight$ bestimmt werden.
2. Bestimme den Schnittpunkt s' zwischen dem Strahl durch v in Richtung der Austrittskante bei uniformer Winkelzuweisung bzgl. obigem Sektor und $CH(b)$, weise dem an dieser Kante hängenden Teilbaum einen Winkel zu, der maximal durch die Winkel zwischen s' und der durchstoßenen Kante von $CH(b)$ begrenzt ist sowie eine Minimaldistanz von $d(v, s')$ (s. Abb. 7.10(b)).

Die Entscheidung, welches der beiden Verfahren für eine Austrittskante $e = (v, w)$ günstiger ist, hängt von der Form von $T(w)$ ab, im Zweifel wird man daher für beide Sektoren die minimal mögliche Distanz von w bestimmen und dann den besseren Wert wählen.



(a) 1. Fall, hier wird ein Sektor direkt an v bestimmt



(b) 2. Fall, hier wird das Kind s' ausreichend weit herausbewegt

Abbildung 7.10: Sektoren an Schnittknoten, die nicht auf der konvexen Hülle liegen

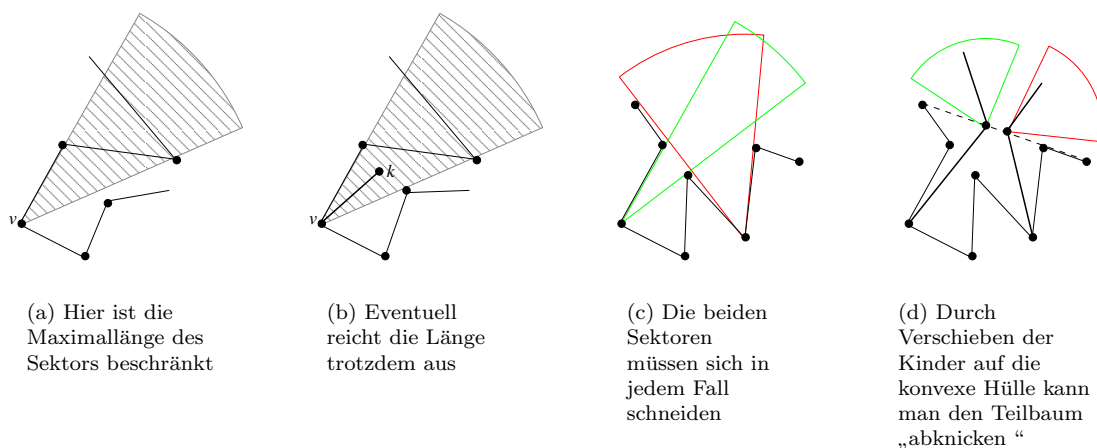


Abbildung 7.11: Probleme bei Schnittknoten, die im Innern der konvexen Hülle liegen

Fatalerweise kann es eintreten, dass sich zwar Sektoren mit einem der beiden Verfahren bestimmen lassen, aber die Austrittskante sich nicht verlängern lässt, ohne eine Kante von b zu treffen, oder der Sektor auf jeden Fall b trifft. Dies liegt daran, dass beide Verfahren zur Sektorberechnung b als Punktmenge betrachten, ohne auf die Kanten von b Rücksicht zu nehmen, wie in Abb. 7.11(a) zu sehen ist. Diese Situation ist insbesondere auch möglich, wenn b außenplanar ist, und kann einfach durch maximal $O(b)$ Schnitttests mit allen Kanten in $H(b)$ erkannt werden.

Eine ähnliche Situation liegt in Abb. 7.11(c) vor, hier kann man die Richtungen der Austrittskanten nicht so bestimmen, dass die zugehörigen Sektoren sich gegenseitig nicht überschneiden und gleichzeitig die Kanten den für sie möglichen Bereich nicht verlassen, so dass die im vorigen Abschnitt beschriebene Methode für nicht positive Winkel zwischen Austrittskanten hier nicht anwendbar ist. Auch diese Situation lässt sich leicht erkennen.

Zur Abhilfe sind mehrere Ansätze denkbar:

1. Man zeichnet trotzdem in die vorgegebenen Sektoren und hofft, dass die Teilbäume klein genug sind, um in den verfügbaren Platz zu passen. Wenn dies möglich ist, stellt dies sicher die beste Lösung dar. Evtl. kann man durch Zusammenfassen von Ketten und Teilbäumen zu Superatomen erreichen (s. Abb. 7.11(b)), dass sich der Platzbedarf ausreichend verringert. Alternativ kann man die betreffenden Teilbäume so weit herunterskalieren, dass sie in den verfügbaren Platz passen, was allerdings nur bis zu einer gewissen Teilbaumgröße sinnvoll machbar ist.
2. Bei der Situation in Abb. 7.11(c) kann man alternativ versuchen, den ersten Knoten jedes Teilbaums so zu platzieren, dass er noch nicht im anderen Sektor liegt, aber für die Kinder dieses Knotens möglichst viel Bewegungsfreiheit zur Verfügung steht. Dies kann man beispielsweise erreichen, indem man diese Knoten auf den Durchstoßpunkten der Kante durch die konvexe Hülle platziert, sofern sich die Austrittskanten nicht schon im Innern der Hülle schneiden (s. Abb. 7.11(d))
3. Man sorgt dafür, dass b nur Schnittknoten auf der konvexen Hülle hat. Da b als außenplanar vorausgesetzt wurde, ist dies stets möglich, da man hierfür ein Kreislayout von b wählen kann. Um das Layout etwas auszugleichen, genügt es, nur die Positionen der Schnittknoten im Kreislayout zu fixieren und dann beispielsweise mit dem Spring Embedder aus Kapitel 5 das Layout zu verbessern. Hierzu muss man verhindern, dass die anderen Knoten die konvexe Hülle zu weit nach außen überschreiten, was beispielsweise dadurch geschehen kann, dass

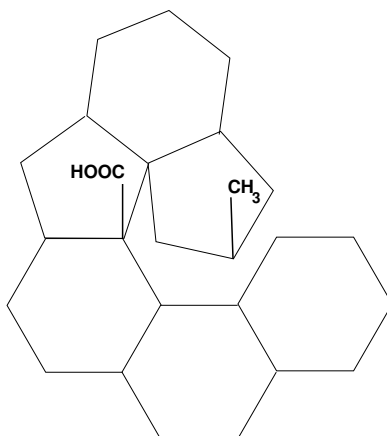


Abbildung 7.12: Hier ist es am günstigsten, die Teilbäume in das Innere der Ringe zu zeichnen

man zusätzlich dafür sorgt, dass alle Knoten im Inneren des Umkreises für das Kreislayout bleiben. Problematisch ist hierbei jedoch die starke Verzerrung der Ringsysteme.

4. Als letzte Möglichkeit bleibt schließlich, auf Ringtreue und/oder Planarität zu verzichten. So kann man kurze Teilbäume, die beispielsweise nur aus einem Atom bestehen, auch in das Innere der anliegenden Ringe zeichnen (s. Abb. 7.12), oder man weicht auf eine Pseudo-3D-Darstellung aus.

Entsprechend kann man auch im Fall von Eintrittsknoten verfahren, die nicht auf $CH(b)$ liegen. Problematisch ist allerdings, dass man keine Informationen über die weiter oben in $kbc(G)$ liegenden Knoten hat, so dass hier die Auswahl einer optimalen Strategie nicht so einfach möglich ist. Insbesondere kann man bei Konflikten nicht einfach Lösung 1. und 2. verwenden. Falls es in dem Molekül nur einen problematischen Block gibt, bietet es sich daher an, diesen und nicht das Zentrum als Startknoten für das Layout zu wählen, so dass höchstens bei den Austrittsknoten Probleme auftreten.

7.2.3 Blöcke in Spiro-Anordnung

Für Blöcke, die sich in Spiro-Anordnung befinden, ist es nicht sinnvoll, die einzelnen Blöcke getrennt zu zeichnen und dann mit einem Baumlayouter zusammenzufügen, da man die Distanz zwischen zwei Blöcken nicht verändern kann. Besser ist es, diese so zusammenhängenden Blöcke gemeinsam auszulegen, beispielsweise, in dem man das uniforme Zeichenverfahren bzw. den Algorithmus von Bertault für Blöcke aus Kapitel 4 passend adaptiert.

7.3 Fazit

Prinzipiell ist die Modifikation bestehender Algorithmen für das Layout von Bäumen geeignet, um daraus ein Gesamtlayout für Strukturformeln zu erhalten. Der Hauptaufwand besteht darin, den Platzbedarf für die Blöcke korrekt abzuschätzen. Hierbei treten Probleme insbesondere dann auf, wenn die Schnittpunkte nicht mehr auf der konvexen Hülle liegen, so dass oft nur unter Verzicht auf mehrere Layoutkriterien aus 2.2 ein planares Layout möglich ist. Durch den Einsatz eines anderen Layoutalgorithmus für Bäume lassen sich auch nicht alle der angesprochenen Probleme lösen, da oft der zur Verfügung stehende Platz insgesamt zu gering ist. Für eine konkrete Implementation ist es daher unter Umständen sinnvoll, den Benutzer aus mehreren Möglichkeiten auswählen zu lassen.

Kapitel 8

Zusammenfassung und Ausblicke

In den vorangegangenen Kapiteln wurde versucht, für zwei spezielle Aspekte des Layouts chemischer Strukturformeln, nämlich das Zeichnen von Ringsystemen und von Baumstrukturen, Layoutalgorithmen zu entwerfen, die längen- und winkeluniforme Layouts liefern. Dabei wurde die Klasse zeichenbarer Ringsysteme auf außenplanare Blöcke eingeschränkt, da für diese die Analyse und Darstellung der Ringsysteme deutlich leichter zu handhaben ist, gleichzeitig aber die allermeisten Ringsysteme diese Voraussetzung auch erfüllen. Allerdings ist in wenigen Fällen kein uniformes Layout möglich, so dass ein weiterer kräftebasierter Layouter, nämlich der *einbettungserhaltende Spring Embedder nach Bertault* als Fallback vorgestellt wurde.

Für das Layout von Baumstrukturen wurde mit dem *sektoroptimierenden* Layouter gezielt auf Winkeluniformität hin optimiert, was bei vielen Substanzen durchaus befriedigende Ergebnisse liefert. Dies war man bei einer derart einseitigen Orientierung auf ein einziges Designziel hin nicht unbedingt zu erwarten. Insgesamt erwies sich dieses Verfahren für das spezielle Problem als durchaus konkurrenzfähig gegenüber etablierten und kommerziell eingesetzten Algorithmen. Zudem können hier tatsächlich für die Optimalität eines nichttrivialen Layoutkriteriums (nämlich für die Winkelauflösung) Garantien angegeben werden. Allerdings zeigt das verwendete Verfahren im Ergebnis noch deutliche Schwächen, da hierbei oft starke Abweichungen von einheitlichen Kantenlängen in Kauf zu nehmen sind, und die Ergebnisse teilweise stark von der Wahl des Startknotens abhängig sind. Insofern eignet sich das beschriebene Verfahren ohne weitere Modifikationen nur bedingt für die praktische Anwendung.

Als weiterführender Ausblick wurde zum Abschluss vorgestellt, inwiefern man die beschriebenen Algorithmen verbinden kann, um ein Layout für ein komplettes Molekül zu erhalten. Hierzu wurde statt des Moleküls selber der zugehörige *Block-Artikulations-Baum* betrachtet, und dieser mit dem Sektorlayout ausgelegt. Prinzipiell lässt sich dieses Verfahren für beliebige Layoutalgorithmen für Bäume verallgemeinern, wenn es gelingt, für die Blockknoten entsprechende Beschränkungen für den belegten Platz anzugeben, die sich in das Baumlayout einbinden lassen. Hier war das einfach möglich, da sich die verwendeten geometrischen Containerobjekte relativ einfach bestimmen ließen.

Die beiliegende CD

Auf der CD, die der gedruckten Fassung beiliegt befinden sich:

- Die Moleküldaten des NCI in gepackter Form, konvertiert nach GraphML (im Verzeichnis `/data/nciopen`) zusammen mit einigen Listen für Klassen von Graphen; die synthetischen Bäume (im Verzeichnis `/data/trees`); die nicht uniform einbettbaren außenplanaren Graphen (im Verzeichnis `/data/misc`). Einige Proteindaten liegen in `/data/proteins`
- Eine Version dieser Arbeit als PDF im Wurzelverzeichnis.
- Implementationen der Algorithmen aus Kapitel 5 und 6: Uniformer Blocklayouter, Einbettungserhaltender Spring Embedder, Ballonlayout, Sektorlayout, diverse Utilities und kleinere Algorithmen (im Verzeichnis `/Diplomarbeit`). Die Algorithmen wurden in Java mithilfe der Bibliothek `yFiles` (nicht beigelegt) implementiert. Der Quellcode befindet sich im Unterverzeichnis `src`, die Klassen in `classes`. Dokumentation im JavaDoc-Format findet sich in `doc`. Für die Layoutalgorithmen wurden `yModule`-Klassen erzeugt, so dass diese direkt eingebunden werden können (z.B. in `yEd`). Außerdem liegt in dem Verzeichnis `xslt` ein XSLT-Stylesheet zur Konvertierung von CML nach GraphML.

Literaturverzeichnis

- [BCF00] J.D. Boissonat, F. Cazals, and J. Flötotto. 2d-structure drawings of similar molecules. Technical report, INRIA Sophia Antipolis, 2000. 2.3
- [BCF01] Boissonat, Cazals, and Flötotto. *2D-Structure Drawings of Similar Molecules*, volume 1984 of *Lecture Notes in Computer Science*, pages 115–126. Springer, 2001. 2.3, 3.2.1, 4
- [Ber99] Francois Bertault. *A Force-Directed Algorithm that Preserves Edge Crossing Properties*, volume 1731 of *Lecture Notes in Computer Science*, pages 351–358. Springer, 1999. 5.3.4, 5.3.4, 2
- [BETT99] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing. Algorithms for the Visualization of Graphs*. Prentice Hall, 1999. 2.3, 1, 6.2.1, 6.2.1, 6.2.2
- [EW96] P. Eades and S. Whitesides. The logic engine and the realization problem for nearest neighbor graphs. *Theor. Comput. Sci.*, 169:23–37, 1996. 2.3
- [FR91] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Software-Practice and Experience*, 21(11):1129–1164, 1991. 5.3.4
- [gra] GraphML, <http://graphml.graphdrawing.org/>. 4.1
- [Har74] Frank Harary. *Graphentheorie*. R. Oldenbourg Verlag, 1974. 2.1, 2.16, 7.1, 7.2
- [HT74] J. Hopcroft and R. Tarjan. Efficient planarity testing. *Journal ACM*, 21(4):229–232, 1974. 5.1.1
- [IE98] Wolf-D. Ihlenfeldt and K. Engel. Visualizing chemical data in the internet - data-driven and interactive graphics, 1998. 3
- [KW01] Michael Kaufmann and Dorothea Wagner, editors. *Drawing Graphs - Methods and Models*. Number 2025 in *Lecture Notes in Computer Science*. Springer, 2001. 1
- [LS97] Josef Leydold and Peter F. Stadler. Minimal cycle bases of outerplanar graphs, 1997. 2.1, 2.1.2, 5.1, 5.9, 5.1.2, 5.10
- [MH98] Guy Melancon and Ivan Herman. Circular drawings of rooted trees. Technical Report INS-R9817, Centrum voor Wiskunde en Informatica CWI, 1998. 6.3, preorder
- [Mit79] Sandra L. Mitchell. Linear algorithms to recognize outerplanar and maximal outerplanar graphs. *Information Processing Letters*, 9(5):229–232, 1979. 5.3, 5.5, 5.6, 5.7, 5.1.1
- [Mor65] H.L. Morgan. The generation of a unique machine description for chemical structures - a technique developed at chemical abstracts service. *J. Chem. Doc.*, 5:107–113, 1965. 3.1.1

- [ope] OpenBabel, <http://openbabel.sourceforge.net>. 4.1
- [PPC95] *PPCHTeX*, Proceedings of the 9th European TeX Conference EuroTeX'95, 1995. 3
- [RT81] E. Reingold and J. Tilford. Tidier drawings of trees. *IEEE Trans. Softw. Eng.*, SE-7(2):223–228, 1981. 6.2.1
- [She83] Craig A. Shelley. Heuristic approach for displaying chemical structures. *J. Chem. inf. Comput. Sci.*, 23:61–65, 1983. 3.1
- [Vis97] P. Vismara. Union of all the minimum cycle bases of a graph. *Electronic J. Comb.*, 4(R9), 1997. 5.5
- [Vol96] Lutz Volkmann. *Fundamente der Graphentheorie*. Springer, 1996. 2.1
- [yfi] yFiles, http://www.yworks.com/en/products_yfiles_about.htm. 4.1.3

Index

- 2-Knoten, **28**
- Abstand
 - zweier Knoten, 50
- Abstoßungskräfte, 41
- Albocyclin, **2**
- Algorithmus
 - von Tilford u. Reingold, **51**
- Anziehungskräfte, 41
- Aromaten, 12
- Atom, **3**
- Außenplanarität, **6**
 - Test auf, **29**
- Austrittskante, **84**
- Austrittsknoten, **84**

- Backtracking, 19
- Ballonlayout, **53**
- Baum, **22, 49**
 - freier, **49**
 - Höhe, **49**
 - n-ärer, **49**
 - Ordnung, 49
 - partielle Ordnung, 49
 - rekursive Definition, **50**
 - Tiefe, **49**
 - vollständig n-ärer, **49**
- Bertault, F., 40
- Bindung, **3**
- Blatt, **49**
- Block-Artikulations-Baum, **83**
 - komprimierter, **84**
- Blockgraph, **83**
- Boissonat, **18**
- Brücke, **16**
- Breitenminimierung, 58
- Bucketsort, 68

- Ceva-Bedingung, **37**
- Containerobjekte, 58
- Cuban, **2**

- Divide-and-Conquer-Ansatz, 49
- Dreiecksgraph, **37**
- Dualgraph, 7, 28, 31

- Einbettung
 - geradlinige planare, 5
 - kantenuniforme, **9**
 - kombinatorische, **5, 35**
 - konvex, **8, 27, 45**
 - nicht uniforme, 35
 - planare, **5**
 - uniforme, **9, 34**
 - winkeluniforme, **9**
- Eintrittskante, **84**
- Eintrittsknoten, **33, 84**
- Elternknoten, **49**
- Enzyme, 21
- Eulersche Polyederformel, 6
- Exzentrizität, **50**

- Facette, **5**
 - äußere, **5**
 - innere, **5**
 - Rand, **5**
- Facettenbedingung, 36
- funktionale Gruppe, 12

- Gesamtlayout, 83
 - Optimierungen, 93
- Gitterlayout, 50
- Graph
 - außenplanar, **6, 22, 27**
 - Größe, 21
 - hamiltonsch, 27
 - maximal außenplanar, **6**
 - maximal planar, **5**
 - planar, **5, 22**
- GraphML, 21

- Hamiltonkreis, 27
- Heuristik, 12, 18, 39

- Karzinogenität, 21
- Kindknoten, **49**
- Knoten
 - kritisch, 39
- Knotenbedingung, 36
- Knotengrad, 21
- Kontur, **51**
- Kreislayout, **27, 45**

- Kreuzungserkennung, 38
- Lagenlayout, **50**
- Layout, **5**
- eines Moleküls, **10**
 - hierarchisches, **50**
 - ringtreues, **7, 10**
 - stark ringtreues, **7**
 - von Bäumen, **49**
 - von Blöcken, **27**
- logic engine, 13
- lokale Konsistenz, **36**
- Mehrfachbindung, 3, 12
- Molekül, **3**
- Morgan-Codierung, 15
- Nachfolger, **49**
- Norbornan, **16**
- OpenBabel, 21
- Ordnung, 31
- Orientierung, **6**
- Orientierungserhaltung, 33
- Oszillationen, 41
- Planarität, **5**
- Proteine, 21
- Rad, **37**
- Radiallayout, **51**
- Radius, **50**
- Ring, **4**
- Ringsystem
- bizyklisch, **16**
 - einfaches, **16**
 - Komplexität, **16, 22**
- Ringzusammenhangsgraph, **4, 7, 15**
- Schnittknoten, 83
- Sehne, **3, 27, 29**
- Sektorlayout, **56**
- Distanzberechnung, 64
 - Effizienzbedingung, 58
 - Initialisierung, 60
 - Invarianzbedingung, 58
 - Konturierungsbedingung, 58
 - Koordinatengenerierung, 66
 - Längenberechnung, 63
 - Laufzeit, 66
 - Optimierungen, 67
 - Platzbedarf, 66
 - Verwendung von Nachbarsektoren, 69
 - Winkelberechnung, 62
 - Winkelneuverteilung, 66, 68
- Sektoroptimierung, **61**
- Shelley, Craig A., **15**
- Sinussatz, 64
- Skalarmultiplikation, 4
- Spiro-Anordnung, 97
- Spring Embedder, 17, 40
- einbettungserhaltender, 40
 - Performance, 41
- Strukturanalyse von Ringsystemen, 3, 15
- Superatome, 12
- Sweep-Line-Algorithmus, 39
- Teilbaum, **49**
- Triangulierungskanten, 29
- ULPGD, **13**
- uniform, 10
- Uniformität, **9**
- UPD, **13**
- Vektorraum, 4
- Basis, 4
- virtuelle Kanten, 45
- Winkelauflösung, **7, 10, 35, 52**
- bei planaren Graphen, 35
- Winkelausgleichskräfte, 44
- Winkelmaximierung, 35
- Winkelzuweisung, Realisierung einer, 37
- Wurzel, **49**
- Wurzelbaum, **49**
- Zentrum, **50**
- Zonenbestimmung, 42
- Zusammenhangskomponente, 22
- 2-fache, 22
- Zykel, **3**
- adjazente, 4
 - einfacher, **3**
 - elementarer, **3**
 - fundamentaler, 29
 - Menge aller, 4
 - relevanter, *siehe* Ring
 - strategische, 16
- Zykelbasis, **4, 22**
- außenplanarer Graphen, 29
 - Bestimmung einer minimalen, 31
 - fundamentale, 29
 - Länge, **4**
 - minimale, **4, 7, 29**
 - planare, **7, 31**
- Zykelraum, **4**
- zyklomatische Zahl, **4, 22**