Algorithmen zum automatischen Zeichnen von Graphen

Franz J. Brandenburg [*] Michael Jünger [†] Petra Mutzel [‡]

Zusammenfassung

Das Zeichnen von Graphen ist ein junges aufblühendes Gebiet der Informatik. Es befaßt sich mit Entwurf, Analyse, Implementierung und Evaluierung von neuen Algorithmen für ästhetisch schöne Zeichnungen von Graphen. Anhand von selektierten Anwendungsbeispielen, Problemstellungen und Lösungsansätzen wollen wir in dieses noch relativ unbekannte Gebiet einführen und gleichzeitig einen Überblick über die Aktivitäten und Ziele einer von der DFG im Rahmen des Schwerpunktprogramms „Effiziente Algorithmen für Diskrete Probleme und ihre Anwendungen“ geförderten Arbeitsgruppe aus Mitgliedern der Universitäten Halle, Köln und Passau und des Max-Planck-Instituts für Informatik in Saarbrücken geben.

Schlüsselwörter: Automatisches Graphenzeichnen, Algorithmen, Planarisierung, Kreuzungsminimierung, Grapheneditoren

Graph drawing is a new and growing area in Computer Science. It is concerned with the design, analysis, implementation and evaluation of new algorithms for aesthetically nice drawings of graphs. Through the use of some selected examples of applications, typical problems, and solutions, we would like to provide an introduction into this still relatively unknown field. And we survey activities and goals of a working group consisting of members of the universities of Halle, Köln and Passau and the Max-Planck-Institut für Informatik in Saarbrücken, that is funded by the German Science Foundation DFG under the program „Efficient Algorithms for Discrete Problems and their Applications“.

Key words: Automatic Graph Drawing, Planarization, Algorithms, Crossing Minimization, Graph Editors

Computing Reviews Classification: D.2.2, G.2.2, I.3.3

[*] *Fakultät für Mathematik und Informatik, Universität Passau*
[†] *Institut für Informatik, Universität zu Köln*
[‡] *Max-Planck-Institut für Informatik, Saarbrücken*

Erscheint im Informatik Spektrum, 1997

1 Was ist automatisches Zeichnen von Graphen?

Vor drei Jahren wandte sich der Berliner Astrophysiker Holger Beck an die Autorin mit der Frage, ob Informatiker helfen könnten, Diagramme wie das in Abb. 1 automatisch zu erstellen. Es handelt sich dabei um ein in [1] publiziertes chemisches Reaktionsflußdiagramm, wie sie von Chemikern in großer Anzahl produziert werden. In einem zeitraubenden Verfahren werden die chemischen Elemente (Knoten) mit Hilfe einer Standard-Graphiksoftware von Hand mit der Maus plaziert und die Reaktionen (Kanten) eingezeichnet. Ist das Resultat nicht übersichtlich genug, so werden Knoten verschoben, Kanten verlegt, oder man fängt gleich noch einmal von vorne an.

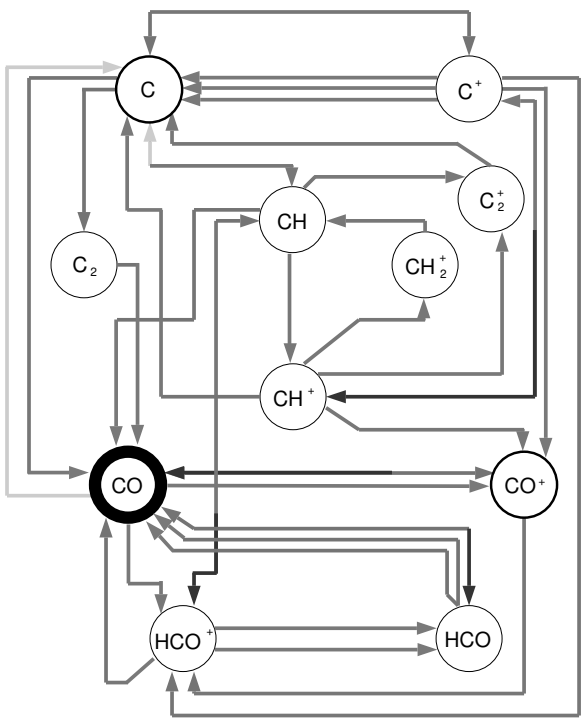

Abb. 1: Computerunterstützt gezeichnetes chemisches Reaktionsflußdiagramm

Ähnliche Probleme treten in vielen anderen Bereichen auf, z.B. bei Datenstrukturen, ER-Diagrammen, Flußgraphen, Petrinetzen, Schaltplänen oder Skizzen. Es verbleibt

beim Anwender, die Objekte zu plazieren und die Verbindungen zu ziehen. Hier setzt das automatische Zeichnen von Graphen an. Gesucht werden Algorithmen für ästhetisch schöne Layouts von Graphen. „Ästhetisch schön“ steht hier für „übersichtlich“ und „verständlich“ und wird formal durch Ästhetikkriterien wie z.B. die Anzahl der Kreuzungen oder Knicke von Kanten beschrieben. Erste wahrnehmungspsychologische Untersuchungen zur Ästhetik und Verständlichkeit von Zeichnungen bestätigen die Bedeutung dieser Kriterien [38]. Darüber hinaus mißt man die Auflösung von Zeichnungen. Bei vorgegebenem Einheitsabstand ist dies z.B. die benötigte Fläche, die Längen der Kanten oder die Größe der Winkel zwischen benachbarten Kanten. Ästhetisch schöne Layouts zeichnen sich durch Kompaktheit, eine hohe Uniformität der Kantenlängen und gleichmäßige Winkel aus. Die Hervorhebung von Symmetrien ist bei einigen Anwendungen wünschenswert. Diese Optimalitätskriterien sollen unter Einhaltung von vorgegebenen Restriktionen, z.B. rechtwinkliges oder kreuzungsfreies Zeichnen, erreicht werden. Welches Ästhetikkriterium passend ist, hängt von der Anwendung und dem gewohnten Kontext ab. Ein Beispiel ist der Würfel mit einer planaren, d.h. kreuzungsfreien und einer perspektivischen Darstellung (Abb. 2).

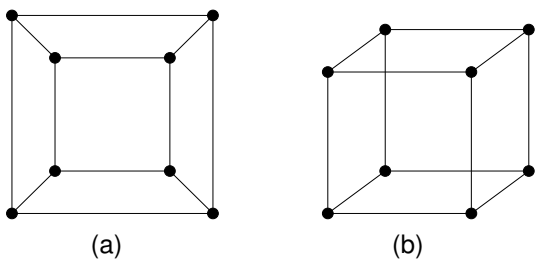

Abb. 2: Eine kreuzungsfreie (a) und eine perspektivische (b) Darstellung des Würfels

Das Problem des Zeichnens von Graphen läßt sich als kombinatorisches Optimierungsproblem formulieren, in dem die Zeichnungen vorgegebenen Restriktionen genügen und eine Kostenfunktion minimiert wird. Es hat sich herausgestellt, daß das Zeichnen von Graphen fast immer NP-schwierig ist [13], [3]. Überraschenderweise gilt dies sogar für Binärbäume, z.B. wenn diese frei mit minimaler Fläche oder der Kantenlänge eins gezeichnet werden sollen, oder ebenenweise unter bestimmten Isomorphiebedingungen mit minimaler Breite. Andererseits kann man Bäume ohne diese Bedingungen in linearer Zeit ästhetisch ansprechend zeichnen [39], [48].

Die Anfrage von Herrn Beck kam zu einem Zeitpunkt, als sich bereits eine starke Interessengruppe für das Thema Graphenzeichnen gebildet hatte. Man hat die von Tutte in seinem 1963 erschienenen Aufsatz initiierte Frage „How to draw a graph“ [46] wieder aufgegriffen. Im Jahre 1992 fand in Rom die erste „Graph Drawing“ Konferenz statt, gefolgt von „Graph Drawing '93“ in Paris, „Graph Drawing '94“ in Princeton [44], „Graph Drawing '95“ in Passau [6] und „Graph Drawing '96“ in Berkeley [37].

In Kapitel 2 geben wir einen Einblick in Techniken, die Gegenstand aktueller Forschung sind und die in verschiedenen Varianten in mehreren Softwaresystemen zum automatischen Graphenzeichnen eingesetzt werden. In den folgenden Kapiteln widmen wir uns Fragestellungen, auf die wir uns im Rahmen eines von der DFG geförderten gemeinsamen Projekts an den Universitäten Halle, Köln und Passau sowie dem MPI Saarbrücken spezialisiert haben. In Kapitel 3 behandeln wir die Planarisierung von Graphen, in Kapitel 4 die Minimierung von Kantenkreuzungen, in Kapitel 5 die Vermeidung kleiner Winkel in planaren Graphen und in Kapitel 6 die Entwicklung des in unserem Projekt entstehenden Systems Graphlet sowie einer Software-Bibliothek, die in Verbindung mit Graphlet den Anwendern benutzerfreundliche Implementierungen der grundlegenden (Kapitel 2) und aller von uns entwickelten (Kapitel 3-5) und zu entwickelnden Algorithmen bereitstellen soll.

In Kapitel 3 stellen wir auch unseren ersten „Lösungsversuch“ für das chemische Reaktionsflußproblem vor, der noch „halbautomatisch“ erzeugt wurde, aber schließlich mit der in Kapitel 6 beschriebenen Software vollautomatisch erzeugbar sein soll.

2 Einige Zeichenalgorithmen

Konzentrieren wir uns zunächst auf den Spezialfall (ungerichteter) planarer Graphen, das sind Graphen, die ohne Kantenüberkreuzungen in der Ebene gezeichnet werden können („planare Zeichnung“).

Die Tatsache, daß jeder planare Graph auch geradlinig planar gezeichnet werden kann, geht auf den Graphentheoretiker Wagner [47] zurück und wurde mehrfach wiederentdeckt. Ein entsprechender Zeichenalgorithmus wurde von Tutte [46] angegeben. Dieser liefert für 3-zusammenhängende Graphen sogar eine konvexe Zeichnung, d.h. alle Innenflächen und der Gesamtgraph sind konvex. (In einem k-zusammenhängenden Graphen müssen wenigstens k Knoten entfernt werden, um den Graphen unzusammenhängend zu machen.) Chiba, Onoguchi und Nishizeki [9] zeichnen 2-zusammenhängende, planare Graphen konvex, soweit es möglich ist (Abb. 3a). De Fraysseix, Pach, und Pollack [11] und Schnyder [40] zeigten unabhängig voneinander, daß jeder planare Graph mit n Knoten geradlinig auf einem Gitter der Größe $O(n^2)$ gezeichnet werden kann. Für 3-zusammenhängende planare Graphen ist sogar eine konvexe Zeichnung möglich [27], [10] (Abb. 3b). Von Bedeutung ist dabei die Ganzzahligkeit durch das Gitter und die asymptotisch minimale Fläche der Größe $(n-2) \times (n-2)$. Die Zeichnungen lassen sich zudem in Zeit $O(n)$ konstruieren.

Wenn Knicke erlaubt sind, d.h. die Kanten stückweise aus geraden Linien zusammengesetzt sind, so können wir z.B. einen einfachen Zeichenalgorithmus von Woods [49] verwenden (Abb. 3c). Ist der Graph 4-gradbeschränkt, d.h. jeder Knoten inzidiert mit höchstens vier Kanten, so bieten sich orthogonale Zeichenverfahren an, die planare Zeichnungen liefern, bei denen alle Kanten aus horizontalen und vertikalen Geradenstücken zusammengesetzt sind. Das Verfahren von Tamassia [43] minimiert dabei die Anzahl der Kantenknicke bei fixer topologischer Einbettung (Abb. 3d). Alle vier Algorithmen profitieren von der Tatsache, daß man topologische Einbettungen planarer Graphen in linearer Zeit bestimmen kann. Entsprechende Algorithmen wurden von Chiba, Nishizeki, Abe und Ozawa [8] (basierend auf dem Planaritätstest von Booth und Lueker [2]) und von Mehlhorn und Mutzel [31] (basierend auf dem Planaritätstest von Hopcroft und Tarjan [20]) entwickelt.

Schon die einfachen Beispiele in Abb. 3 zeigen, daß insbesondere die drei erstgenannten Algorithmen nicht notwendigerweise gute Zeichnungen liefern. Die Zeichenfläche wird oft nicht gleichmäßig ausgenutzt, und es werden viele kleine Winkel erzeugt. Dies macht die Zeichnungen unübersichtlich. Wir kommen darauf in Kapitel 5 zurück.

Eine weitere Schwierigkeit besteht darin, daß diese (und andere) Algorithmen neben der Planarität weitere Eigen-

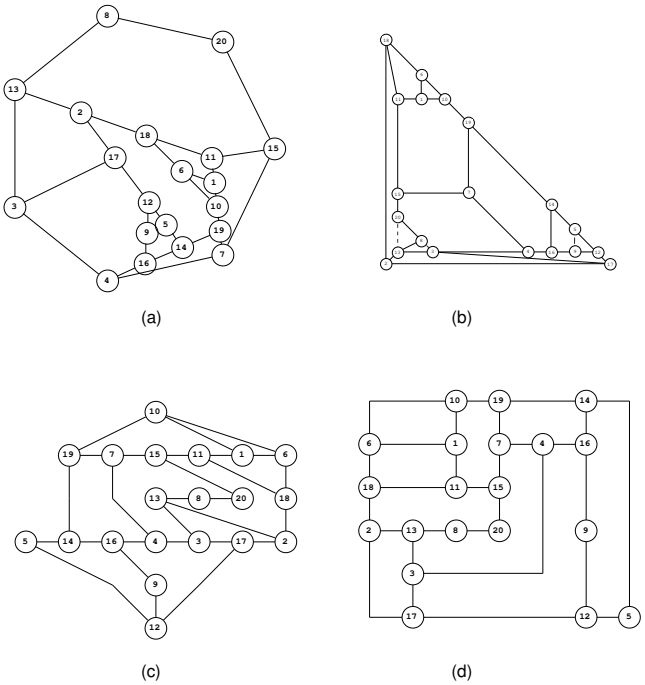

Abb. 3: Verschiedene planare Zeichnungen eines planaren Graphen

schaften des Graphen wie 2- oder 3-Zusammenhang oder eine Knotengradbeschränkung voraussetzen. Will man solche Algorithmen auf nicht planare, nicht 2-zusammenhängende, nicht gradbeschränkte Graphen anwenden, so muß man diese Bedingungen künstlich durch Hinzufügen bzw. Entfernen von Kanten erzeugen und nach der Layoutphase wieder reparieren, siehe Kapitel 3.

Drei weitere Verfahren wollen wir an einem Beispiel vorstellen. Eades und Marks haben zu den Graph Drawing Symposien GD'94, GD'95 und GD'96 einen „Graph Drawing Contest" organisiert. Es galt, ihre Wettbewerbsgraphen so schön wie möglich zu zeichnen. Die Einsendungen wurden von einer Jury bewertet und mit Preisen prämiert [44], [6], [37]. Solche Wettbewerbe, so spielerisch sie zunächst erscheinen mögen, helfen, den Begriff „Schönheit" auf verschiedene Weisen formal zu präzisieren.

Wir wenden uns einem Wettbewerbsgraphen der GD'94 in Princeton zu. Besonders leicht verständlich ist die von Eades [14] eingeführte „spring embedder" Methode, die auf einem physikalischen Kräftemodell basiert. Die Knoten stoßen sich gegenseitig ab, die Kanten sind Federn, die den Abstoßkräften entgegenwirken. Die Idee besteht darin, in diesem physikalischen System einen Zustand minimaler Energie zu bestimmen bzw. zu approximieren. In der Literatur sind zahlreiche Ansätze vorgestellt worden, diese Grundidee effizient zu realisieren. In [5] wird ein Überblick über solche Ansätze mit einer experimentellen Evaluation gegeben. Die so konstruierten Zeichnungen haben eine ausgewogene Verteilung der Knoten und der Kantenlängen und machen Symmetrien in Graphen transparent. Besonders gut ist ihr Verhalten bei der Darstellung von dreidimensionalen Körpern wie Würfel, Ikosaeder, Dodekaeder, etc. Die Zahl der Kreuzungen oder gar Planarität werden jedoch nicht beachtet. Abb. 4 zeigt eine „spring embedder" Zeichnung des Wettbewerbsgraphen.

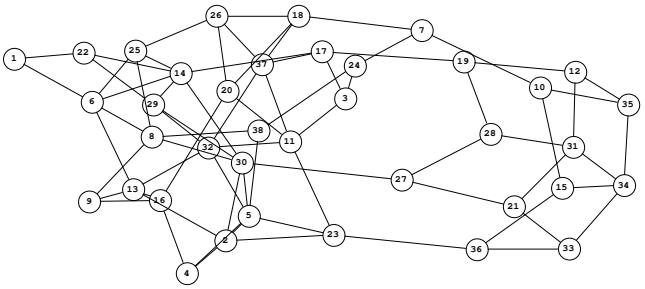

Abb. 4: Zeichnung des Wettbewerbsgraphen mit Eades' Algorithmus

Als nächstes wenden wir eine Adaption einer von Sugiyama, Tagawa und Toda [42] für die Zeichnung gerichteter Graphen vorgeschlagenen Methode auf unseren (ungerichteten) Wettbewerbsgraphen an. Die Knoten werden so auf Schichten verteilt, daß innerhalb der Schichten keine Kanten verlaufen und möglichst wenige Kantenüberkreuzungen entstehen. Das Verfahren liefert die in Abb. 5 dargestellte Zeichnung. Auf die Problematik, daß hier NP-schwierige Probleme mit geeigneten Heuristiken approximativ gelöst werden müssen, kommen wir in Kapitel 4 zurück.

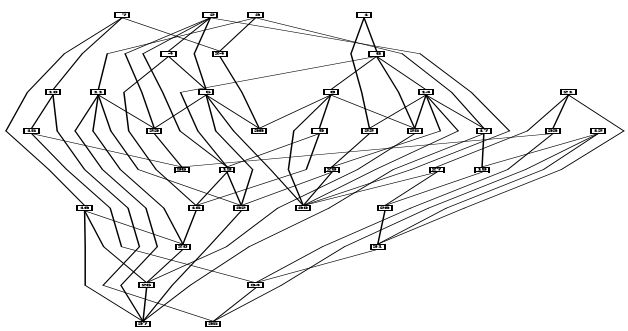

Abb. 5: Zeichnung des Wettbewerbsgraphen mit Sugiyama's Algorithmus

Eine weitere Methode besteht darin, durch temporäres Löschen von Kanten Planarität herzustellen, dann etwa eine der oben erwähnten Methoden zum Zeichnen planarer Graphen anzuwenden und die gelöschten Kanten anschließend wieder geeignet einzuzeichnen. Wir wollen Tamassia's Algorithmus [43] verwenden und mit minimalen Veränderungen (Anzahl der entfernten Kanten) die Voraussetzungen für seine Anwendung herstellen. Unser Wettbewerbsgraph ist nicht planar, und manche Knoten haben den Grad sechs. Man kann zeigen (darauf kommen wir in Kapitel 3 zurück), daß wenigstens zehn Kanten entfernt werden müssen, um einen planaren Graphen mit Gradbeschränkung vier zu erhalten. Wir zeichnen den resultierenden Graphen mit Tamassia's Algorithmus und fügen die entfernten Kanten unter gleichzeitigen geeigneten Veränderungen der Knotenpositionen nachträglich ein. Das Ergebnis ist in Abb. 6 dargestellt. Diese Zeichnung (von Mutzel und Odenthal) gewann in Princeton den ersten Preis [44].

Dies war nur ein kleiner Ausschnitt der bekannten Verfahren zum Zeichnen von Graphen. Eine kommentierte Bibliographie findet man in [13]. Neuere Arbeiten wurden auf den Graph Drawing Symposien vorgestellt [44], [6], [37].

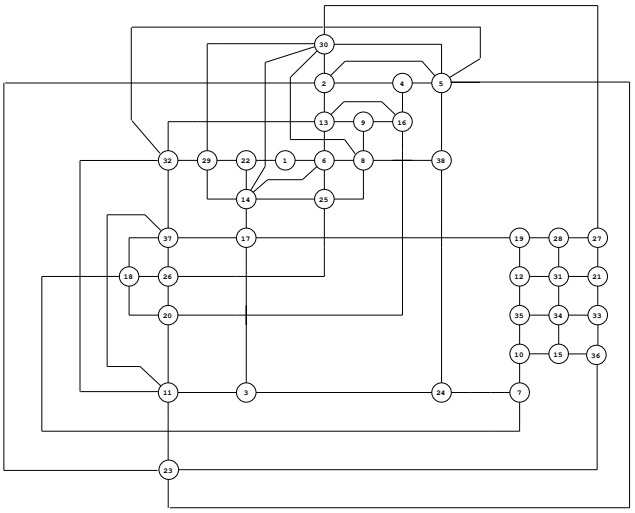

Abb. 6: Gewinner-Zeichnung des Wettbewerbsgraphen

3 Planarisierung, Augmentierung und Gradbeschränkung

Für planare Graphen gibt es eine reiche Auswahl von Zeichenverfahren, die jedoch neben der Planarität in der Regel weitere Voraussetzungen wie k-Zusammenhang mit $k = 2$ oder $k = 3$ und Beschränkungen des Knotengrades meist zwischen 4 und 8 erfordern. Das Herstellen dieser Voraussetzungen mit minimaler Veränderung des zu zeichnenden Graphen (Anzahl hinzugefügter plus Anzahl entfernter Kanten) durch Augmentierung und Planarisierung läßt sich im allgemeinen nicht schrittweise erreichen, ohne die Optimalität zu verlieren.

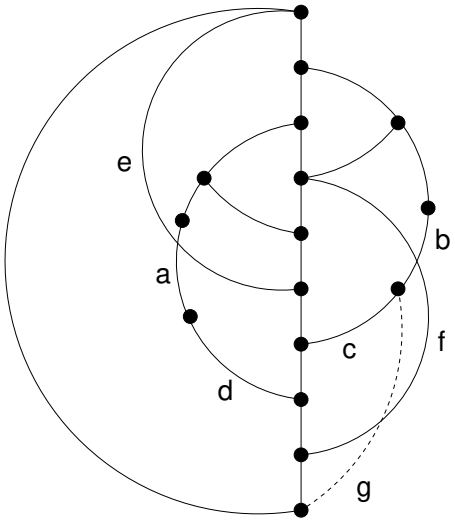

Abb. 7: Beispiel für die Nützlichkeit der Kombination von Planarisierung und Augmentierung

Wir demonstrieren dies in Abb. 7. Wir wollen einen Graphen G planar und 2-zusammenhängend machen. Ist G der mittels der durchgezogenen Linien definierte Graph, so müssen wenigstens zwei Kanten entfernt werden, um G zu planarisieren, z.B. ist die Entfernung von a und b optimal. Zur Herstellung des 2-Zusammenhangs müssen nun wenigstens drei Kanten hinzugefügt werden, d.h. insgesamt fünf Veränderungen. Die Entfernung von e und f (zwei Veränderungen) ist eine Optimallösung bei gleichzeitiger Betrachtung beider Ziele. Der Graph G ohne die Kante c demonstriert, daß auch die andere Reihenfolge fehlschlägt. Hinzufügen von c stellt 2-Zusammenhang mit minimalem Aufwand her, dann müssen allerdings zwei Kanten, etwa e und f, entfernt werden, um Planarität zu erreichen, d.h. insgesamt drei Veränderungen. Bei gleichzeitiger Betrachtung beider Ziele reichen aber zwei Veränderungen: Entferne e und füge g hinzu.

Während es leicht ist, einen beliebigen Graphen $G = (V, E)$ mit Knotenmenge V und Kantenmenge E unter Hinzufügung einer minimalen Anzahl von Kanten 2- bzw. 3-zusammenhängend zu machen, ist die Bestimmung einer Kantenmenge minimaler Kardinalität, deren Entfernung aus einem gegebenen Graphen zu einem planaren Graphen führt, NP-schwierig. Deshalb wurden mehrere schnelle Heuristiken entwickelt, die eine Kantenmenge entfernen, so daß ein maximal planarer Subgraph entsteht, dessen Planarität beim Einfügen jeder einzelnen entfernten Kante verloren gehen würde. Man kann zeigen, daß die Kardinalität der entfernten Kantenmenge erheblich größer als die optimale sein kann. Solche Heuristiken wurden etwa von Cai, Han und Tarjan [7] (mit Laufzeit $O(|E| \log |V|)$) oder Jayakumar, Thulasiraman und Swamy [21] (mit Laufzeit $O(|V|^2)$) entwickelt.

Selbst für Graphen praxisrelevanter Größe ist es aber oft möglich, Optimallösungen mit Hilfe eines in [22] vorgestellten Branch&Cut Verfahrens zu finden. Die Beschreibung dieses Verfahrens sprengt den Rahmen dieser Einführung, die Idee wollen wir aber hier skizzieren: Nach einem klassischen Resultat von Kuratowski [28] ist ein Graph genau dann planar, wenn er weder eine Unterteilung des vollständigen Graphen auf 5 Knoten K_5 noch eine Unterteilung des vollständigen bipartiten Graphen $K_{3,3}$ enthält (Abb. 8). Bei einer Unterteilung eines Graphen ist es erlaubt, beliebige zusätzliche Knoten auf den Kanten des ursprünglichen Graphen anzubringen. Die Unterteilungen von K_5 und $K_{3,3}$ heißen auch Kuratowski-Graphen.

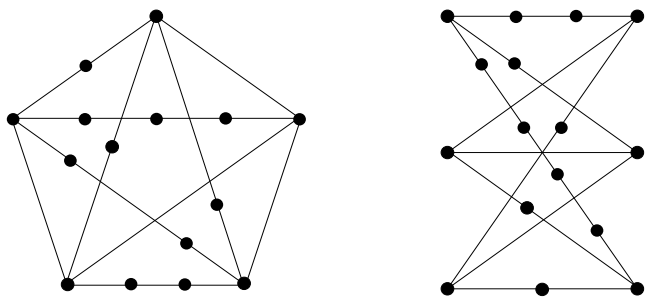

Abb. 8: Unterteilungen der verbotenen Untergraphen

Wir ordnen nun jeder Teilmenge F der Kantenmenge E eines gegebenen Graphen $G = (V, E)$ einen $|E|$-dimensionalen charakteristischen Vektor $\chi^F \in \{0, 1\}^E$ zu, dessen Komponenten mit den Kanten $e \in E$ indiziert sind und setzen $\chi^F_e = 1$ falls $e \in F$ und $\chi^F_e = 0$ falls $e \neq F$. Dann sind die charakteristischen Vektoren aller planarer Subgraphen von G genau die ganzzahligen Lösungen $x \in R^E$ des folgenden Ungleichungssystems:

$$\begin{array}{ll} 0 \leq x_e \leq 1 & \text{für alle } e \in E, \\ \sum_{e \in K} x_e \leq |K| - 1 & \text{für alle } K \subseteq E \text{ die die Kantenmenge eines Kuratowski Subgraphen von } G \text{ definieren} \end{array} .$$

Fügen wir die Zielfunktion

$$\text{maximiere} \sum_{e \in E} x_e$$

und die Ganzzahligkeitsbedingungen

$$x_e \text{ ganzzahlig für alle } e \in E$$

hinzu, so haben wir unser Problem als ganzzahliges lineares Optimierungsproblem formuliert, denn die Ungleichungen schließen die verbotenen Kuratowski-Subgraphen aus. Kantenprioritäten können durch Gewichte c_e auf den Kanten $e \in E$ und der Zielfunktion

$$\text{maximiere} \sum_{e \in E} c_e x_e$$

berücksichtigt werden. Läßt man die Ganzzahligkeitsbedingungen weg, so entsteht ein lineares Optimierungsproblem, dessen optimaler Zielfunktionswert eine obere Schranke für die Anzahl der Kanten in einem planaren Subgraphen von G ist. Durch Hinzufügen weiterer Ungleichungen kann man diese Schranke verschärfen. In [25] wird dargestellt, wie man solche Relaxierungen mit Hilfe eines Schnittebenenverfahrens („Cut“) lösen kann und wie die Lösungen solcher Relaxierungen als Schranken in einem Enumerationsverfahren („Branch“) zur Optimallösung des Planarisierungsproblems ausgenutzt werden können. Dieses „Branch&Cut“-Verfahren ist so angelegt, daß auch gute zulässige Lösungen (charakteristische Vektoren planarer Subgraphen großer Kardinalität) gefunden werden, so daß zu jeder Zeit eine Lösung mit einer aus der oberen Schranke resultierenden Gütegarantie angegeben werden kann. So entsteht als Nebenprodukt eine Heuristik, die Lösungen mit sehr hoher Qualität im Vergleich zu einfacheren Heuristiken produziert [25], [34].

Als Herr Beck uns das chemische Reaktionsflußdiagramm schickte, hatten wir gerade eine erste Version unserer Planarisierungssoftware fertiggestellt. Wendet man sie auf den zugrundeliegenden Graphen (Vernachlässigung der Richtungen und Ersetzen von Mehrfachkanten durch Einfachkanten) an, so stellt sich heraus, daß dieser Graph nicht planar ist, aber die Entfernung der Kante zwischen CH und HCO zu einem planaren Graphen führt. Die Implementierung des Hopcroft-Tarjan Algorithmus von Mehlhorn, Mutzel und Näher [32] lieferte uns eine topologische Einbettung, aus der wir (noch halbautomatisch) das neue in Abb. 9 gezeigte Diagramm konstruierten.

Kommen wir nun auf das Problem der optimalen Planarisierung unter Zusammenhangs- und Gradbeschränkungen zurück. Es ist prinzipiell kein Problem, unseren oben skizzierten Branch&Cut Algorithmus entsprechend zu modifizieren. Dazu vervollständigen wir unseren Eingabegraphen $G = (V, E)$ (es sei $n := |V|$) durch Hinzunahme aller fehlenden Kanten zum vollständigen Graphen $K_n = (V, E_n)$. Stoer [41] hat gezeigt, wie man die k-Zusammenhangsbedingungen (analog zu den „Kuratowski-Bedingungen“) als Ungleichungen formulieren kann. Für k-Zusammenhang ist es hinreichend, zu fordern, daß der Graph nach Entfernen von $k-1$ beliebigen Knoten immer noch zusammenhängend ist, und das ist der Fall, wenn zwischen jeder Knotenteilmenge im Restgraphen und deren Komplement wenigstens eine Kante verläuft. Für $Y \subseteq V$ bezeichnen wir mit $G \setminus Y$ den Graphen, der aus G durch Entfernung aller Knoten in Y und der zu diesen inzidenten Kanten resultiert. Für $W \subseteq V \setminus Y$ sei dann $\delta_{G \setminus Y}(W)$ die Menge der Kanten, die in $G \setminus Y$ die

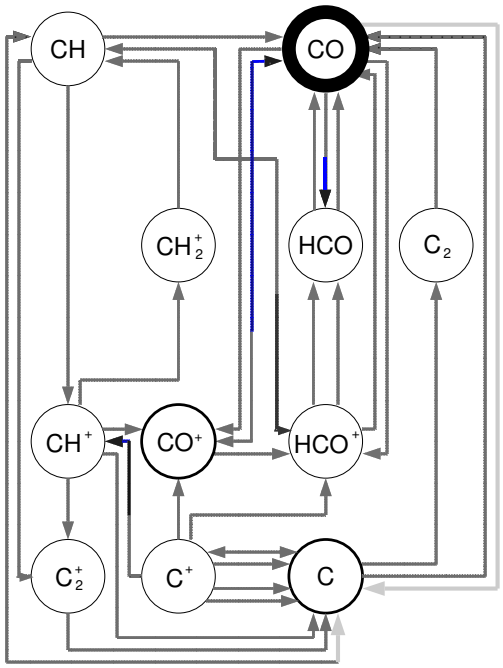

Abb. 9: Das mit der beschriebenen Methode (noch halbautomatisch) gezeichnete chemische Reaktionsflußdiagramm aus Abb. 1

Knotenmenge W mit ihrem Komplement verbindet. Die Ungleichungen lauten dann

$$\sum_{e \in \delta_{K_n \setminus Y}(W)} x_e \geq 1 \text{ für alle } Y \subseteq V, |Y| = k-1, W \subseteq V \setminus Y.$$

Gradbeschränkungen sind ganz einfach als Ungleichungen formulierbar:

$$\sum_{e \in \delta_{K_n}(\{v\})} x_e \leq l \text{ für alle } v \in V.$$

Die Zielfunktion

$$\text{maximiere} \sum_{e \in E} x_e - \sum_{e \in E_n \setminus E} x_e$$

sorgt nun dafür, daß viele Kanten aus unserem ursprünglichen Graphen $G = (V, E)$ genommen werden, aber nur wenige der „neuen“ Kanten.

Es ist zur Zeit noch unklar, welche zusätzlichen Ungleichungen nach Weglassen der Ganzzahligkeitsbedingungen für ein Branch&Cut Verfahren geeignet sind. Dies erfordert umfangreiche theoretische und experimentelle Studien, deren Beschreibung (wie schon bei „reiner“ Planarisierung) den Rahmen dieser Übersicht sprengen. Vorläufige Untersuchungen stimmen uns aber recht zuversichtlich [23], [35], [12].

Wir haben viele Details und ganze Teilprobleme nicht diskutiert. Z.B. ist es natürlich leicht, in der abschließenden „Reparaturphase“ künstlich hinzugefügte Kanten aus der Zeichnung zu entfernen, aber es ist NP-schwierig, künstlich weggelassene Kanten wieder hinzuzufügen, so daß möglichst wenige zusätzliche Kantenüberkreuzungen entstehen. Man verwendet zur Zeit üblicherweise Heuristiken, die auf kürzeste-Wege-Berechnungen im dualen Graphen eines planaren Graphen beruhen. Wir wollen hier nicht auf diese Problematik eingehen.

4 Kreuzungsminimierung

Das am weitesten verbreitete Verfahren zum Zeichnen gerichteter Graphen ist das eingangs skizzierte Sugiyama-Verfahren [42]. Hier wird in einem ersten Schritt eine Partitionierung der Knotenmenge in Schichten vorgenommen. In einem zweiten Schritt wird versucht, Permutationen der Knoten innerhalb der einzelnen Schichten zu finden, die eine Zeichnung mit wenigen Kreuzungen ergibt.

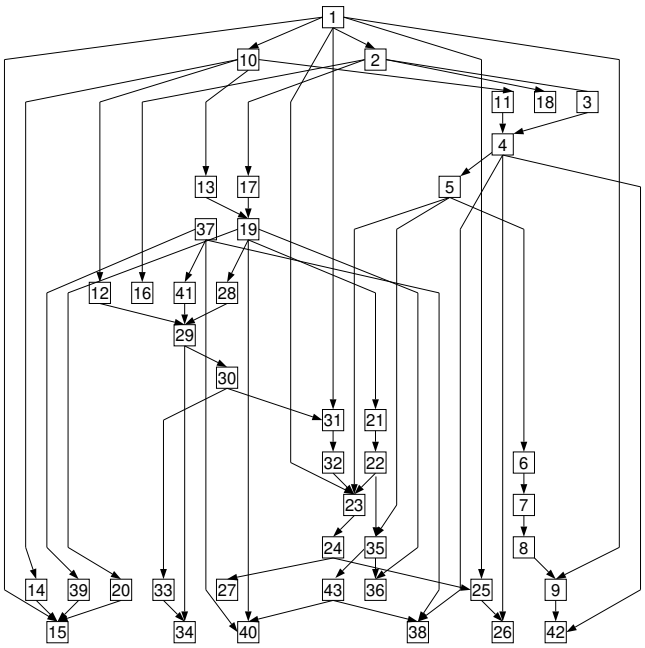

Abb. 10: Das Sugiyama-Verfahren für gerichtete Graphen

Während es für die Schichtenfestlegung verschiedene bewährte Verfahren gibt, z.B. die topologische Sortierung bei azyklischen Graphen, sind die Ansätze für das Kreuzungsminimierungsproblem bisher unbefriedigend (Abb. 10). Das Kreuzungsminimierungsproblem über k Schichten wird in der Praxis oft wie folgt behandelt. Man betrachtet jeweils zwei benachbarte Schichten und versucht, die Anzahl der Kreuzungen, die innerhalb dieser beiden Schichten auftreten, zu minimieren. Hierbei werden für alle Kanten, die die beiden Schichten traversieren, künstliche Knoten eingeführt.

Die Knoten der einen Schicht werden fixiert, und man versucht, die Knoten der anderen Schicht so zu permutieren, daß die Anzahl der Überkreuzungen minimal ist. Eades und Wormald [15] haben gezeigt, daß dies NP-schwierig ist. Deshalb wurden in der Literatur diverse Heuristiken für dieses Problem publiziert.

Betrachten wir das in Abb. 11a gezeigte Beispiel. Die obere Schicht sei fixiert und es ist unsere Aufgabe, eine Knotenpermutation für die untere Schicht zu bestimmen, so daß die Anzahl der Kantenüberkreuzungen minimal wird. Für ein Knotenpaar i, j der unteren Schicht bezeichne c_{ij} die Anzahl der Kreuzungen zwischen den mit i und j inzidenten Kanten, falls i links von j plaziert wird. In unserem Beispiel ist $c_{21} = 2$ und $c_{12} = 4$. Unsere Aufgabe kann also folgendermaßen formuliert werden: Finde eine Permutation π von $\{1, 2, \ldots, n\}$ ($n = 8$ im Beispiel), die

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{\pi(i),\pi(j)}$$

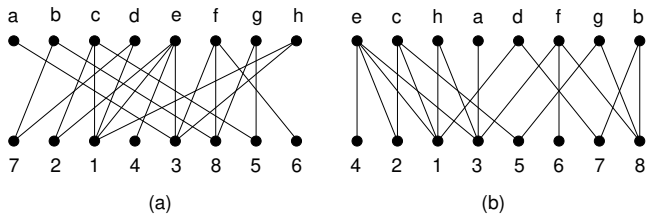

Abb. 11: Zwei-Schichten Zeichnungen mit minimaler Kreuzungszahl wobei (a) die obere Schicht fixiert ist, (b) beide Schichten frei permutiert werden können

minimiert. Dieses Problem ist bekannt als das Lineare Ordnungsproblem, für das in [16] der erste publizierte Branch&-Cut Algorithmus angegeben wurde. Eine Neuimplementierung dieses Algorithmus versetzte uns erstaunlicherweise in die Lage, Probleminstanzen von praxisrelevanter Größenordnung schneller optimal zu lösen als die meisten der gängigen Heuristiken approximativ, d.h. in weniger als einer Sekunde auf einer Workstation [24].

Wie weit hilft uns jedoch die Lösung des vorgegebenen Problems, um die Anzahl der Kreuzungen in k-Schichten Graphen zu minimieren? Wir haben experimentell für $k = 2$ das obige Verfahren mit einem zeitaufwendigen Enumerationsschema für die Permutationen in der oberen Schicht verbunden und konnten so Optimallösungen berechnen, wenn diese nicht mehr als 15 Knoten enthält (siehe z.B. Abb. 11b). Die Rechenzeiten betragen allerdings hier bis zu einer Stunde auf einer Workstation. Wenn das letztere Verfahren für große Graphen auch nicht praktikabel ist, so erlaubt es uns doch wenigstens eine experimentelle Evaluation populärer Heuristiken. Unsere Experimente auf 2-Schichten-Graphen haben ergeben, daß die durch traditionelle Verfahren erhaltene Kantenüberkreuzungszahl sehr weit (ca. 500 Prozent) von der minimalen Kreuzungszahl entfernt ist [24]. Hier sind neue Ideen erforderlich.

5 Winkel in planaren Graphen

Wie schon in Kapitel 2 erwähnt, liefern einige Verfahren zum Zeichnen von planaren Graphen auf dem Gitter ästhetisch unschöne Zeichnungen mit zu vielen kleinen Winkeln. Derartige Zeichnungen wirken unübersichtlich. Durch die Maximierung des kleinsten Winkels soll dem entgegengewirkt werden. Auch dieses Problem ist NP-schwierig.

Wir formulieren diese Aufgabe wieder als ein Optimierungsproblem. Gleichungen ergeben sich aus der elementaren Beobachtung, daß die Winkelsumme an jedem Knoten 360°, die (Innen-)Winkelsumme in jeder durch k Kanten begrenzten Innenfläche $(k-2)180°$ und die (Außen-) Winkelsumme der durch k Kanten begrenzten Außenfläche $(k+2)180°$ ist. Diese Gleichungen reichen jedoch nicht aus, denn sie sind z.B. auch dann erfüllt, wenn die berechneten Winkel an den in Abb. 2a diagonal gezeichneten Kanten statt $(45°, 45°, 135°, 135°)$ etwa $(30°, 60°, 120°, 150°)$ sind, aber der Graph ist dann mit den berechneten Vorgabewinkeln nicht zeichenbar. Falls jedoch die Winkel passen, so reduziert sich das Problem auf die Berechnung der Längen der Kanten. Bei triangulierten Graphen genügt es dann, die Länge einer Kante festzulegen, d.h. die Skalierung. Die Zeichnung selbst ist eindeutig bestimmt. Im allgemeinen Fall kann man die kürzesten Kantenlängen mit einem linearen Programm maximieren und dadurch gute Zeichnungen konstruieren.

Vorgabewinkel zwischen benachbarten Kanten lassen sich gut dazu verwenden, eine gegebene planare Zeichnung zu verbessern und das Layout zu verschönern. Dies gelingt mit einer Erweiterung der in Kapitel 2 erwähnten „spring embedder“ Methode. Wir betrachten die Kanten eines gezeichneten Graphen einerseits als elastische Federn, die weit entfernte Knoten anziehen und nahe beieinanderliegende Knoten abstoßen. Andererseits sind Kanten starr und sollen gemäß der vorgegebenen oder vorberechneten Winkel in eine bestimmte Richtung gedreht werden. Dann wirken sie als Hebel an ihren Endknoten. Die so entstehenden Kräfte überlagern sich. Das Gesamtsystem versucht, ein Kräftegleichgewicht herbeizuführen. Durch zusätzliche Restriktionen wird die Planarität der Zeichnungen garantiert. Die bisherigen Experimente mit diesem Verfahren liefern gute Ergebnisse. Die unschönen Zeichnungen mit den Verfahren von de Fraysseix et al. [11] oder Chrobak und Kant [10] werden auseinandergefaltet und bringen dort nicht erkennbare Symmetrien zum Vorschein [4], [17].

6 Systeme zum Zeichnen von Graphen

Neben chemischen Reaktionsflußdiagrammen gibt es zahlreiche andere Herausforderungen an die Algorithmenentwicklung zum Zeichnen von Graphen, denen wir uns stellen müssen. Es gibt schon jetzt mehrere bereits etablierte Kooperationen mit der Industrie (übersichtliche automatische Erstellung von Organisations- und Produktions-Ablaufdiagrammen) oder im universitären Bereich (Visualisierung von Regeln im „Data Mining“, Erstellung von Harris-Matrix Diagrammen in der Archäologie). Neben den in den Kapiteln 3-5 beschriebenen algorithmischen Fragestellungen widmen wir uns in unserem DFG-Projekt auch den hiermit verbundenen neuen Aufgaben. Den interessierten Anwendern ist natürlich nicht mit der Veröffentlichung wissenschaftlicher Aufsätze wie den hier zitierten gedient. Sie brauchen konkrete Unterstützung in Form von benutzerfreundlicher Software.

Systeme zum Zeichnen von Graphen erfordern Grapheneditoren und Layoutalgorithmen in einer integrierten Umgebung. Es ist das Hauptziel unseres DFG-Projekts, einen Grapheneditor und eine Software-Bibliothek von Layoutalgorithmen bereitzustellen. Durch die Integration beider Komponenten in ein Gesamtsystem sollen die Anwender gute neue Werkzeuge zur automatischen Erstellung ihrer Layouts erhalten, ohne sich um algorithmische Details kümmern zu müssen. Der modulare Aufbau der Software-Bibliothek wird es erlauben, neue Layoutsoftware in das System zu integrieren, wobei auf geeignete bereits entwickelte Datenstrukturen und Algorithmen zurückgegriffen werden kann.

Die Realisierung eines Grapheneditors stellt hohe Ansprüche an die Entwickler. Editoren für Graphen sind deutlich komplexer als normale Editoren für Texte oder Graphiken. Dies liegt an der relationalen Abhängigkeit, die durch die Kanten zum Ausdruck kommt. Diese Abhängigkeit müssen die Editoren berücksichtigen. Kanten sind an Knoten gebunden und Beschriftungen an Knoten, Kanten oder sogar Flächen, die von Knoten und Kanten eingerahmt werden. Bei Bewegungen oder Löschungen von Knoten müssen die damit zusammenhängenden Kanten bzw. Beschriftungen entsprechend mitbewegt bzw. gelöscht werden. Knoten müssen sich durch verschiedene Graphikprimitive darstellen lassen, z.B. als Rechtecke, Kreise oder benutzerdefinierte Icons. Die Darstellung der Kanten erfolgt durch Linien in gewissen Stilen, Strichstärken und Farben. Kanten müssen an ihre Endknoten angepaßt werden. Man möchte Graphen zoomen oder animieren. Weitere Basisanforderungen sind in [18], [29] zusammengestellt.

Es gibt sehr viele Grapheneditoren. Allein auf den Symposien über Graph Drawing 1994, 1995 und 1996 wurden über 20 Systeme präsentiert. Die meisten dieser Systeme sind für eine spezielle Anforderung erstellt worden. Allgemeine Anforderungen an Portabilität, Erweiterbarkeit und Flexibilität werden oft nicht angemessen berücksichtigt. (Es ist nicht das Ziel dieses Aufsatzes und auch außerhalb unserer Möglichkeiten, hier eine Auflistung oder gar Bewertung der existierenden Grapheneditoren und darüberhinaus der Systeme zum Zeichnen von Graphen zu geben.) Die einzelnen Systeme unterscheiden sich stark in der Mächtigkeit und den Möglichkeiten des Editierens und automatischen Zeichnens von Graphen. Keines der bekannten Systeme erfüllt alle Basisanforderungen. Der Standard-Grapheneditor ist (noch) nicht auf dem Markt! In eine neue Richtung soll das im Rahmen des DFG-Projekts zu entwickelnde Graphlet-System zusammen mit einer „Layout-Bibliothek“ gehen. Graphlet beinhaltet einen Grapheneditor und ein Interface zu Layoutalgorithmen.

Um die Implementierung von Graphenalgorithmen und Oberflächen zu vereinfachen, stellt Graphlet die Programmiersprache GraphScript zur Verfügung. Algorithmen können direkt in GraphScript geschrieben werden oder über eine Schnittstelle in GraphScript eingebunden werden. GraphScript integriert Algorithmen und Benutzerinteraktionen, so daß z. B. die Animation von Algorithmen nur einen geringen Zusatzaufwand erfordert. Die Verwendung von Tcl/Tk zur Implementierung der Benutzerschnittstelle garantiert weitreichende Portabilität. Graphlet ist für UNIX und Microsoft Windows verfügbar.

Die Software-Bibliothek der Layout-Algorithmen ist auf LEDA [33] als algorithmischer Basis aufgebaut. LEDA („a Library for Efficient Data types and Algorithms“) stellt in C++ objektorientierte Realisierungen von Datentypen und Algorithmen bereit, die es erlauben, gleichzeitig elegante und effiziente Software für kombinatorische und geometrische Aufgaben zu erstellen. Zu der bestehenden reichhaltigen Sammlung grundlegender Datenstrukturen und Algorithmen (wie Prioritätsschlangen, Wörterbücher, etc.) werden für unser Projekt wichtige Komponenten hinzugefügt, wie z.B. st-Numerierungen, PQ-Bäume oder duale planare Graphen.

Die Branch&Cut Algorithmen unserer Layout-Bibliothek werden in ABACUS [45], [26] implementiert. ABACUS („A Branch And CUt System“) stellt in C++ einen objektorientierten Rahmen zur eleganten und effizienten Realisierung von Branch&Cut Algorithmen bereit. Neben der in Kapitel 3 beschriebenen Planarisierung und der in Kapitel 4 beschriebenen Kreuzungsminimierung gibt es in unserem DFG-Projekt weitere Anwendungen von ABACUS im Zusammenhang mit alternativen Verfahren zur Kreuzungsreduktion [36] oder auch der Bestimmung maximaler azyklischer Subgraphen, die in gewissen Zeichenalgorithmen für gerichtete Graphen eine zentrale Rolle spielen.

Aufbauend auf LEDA und ABACUS werden die in der Software-Bibliothek zusammengefaßten Layout Algorithmen zum automatischen Graphenzeichnen implementiert. Umgekehrt erhalten LEDA and ABACUS via Graphlet und der Software-Bibliothek eine komfortable Benutzeroberfläche für die Manipulation von Graphen und die Animation von Algorithmen. Ein kompaktes Spezialwerkzeug zum Zeichen von Bäumen, das wir insbesondere zur Animation von Branch&Cut Algorithmen verwenden, wurde im Rahmen des DFG-Projekts bereits fertiggestellt [30].

Da es kein standardisiertes Dateiformat für Graphen mit Layoutattributen gibt, arbeiten wir in unserem DFG-Projekt in Abstimmung mit Vertretern der internationalen „Graph Drawing Community“ an einem geeigneten Vorschlag [19].

Die Deutsche Forschungsgemeinschaft unterstützt unsere Arbeiten im Rahmen des Schwerpunktprogramms „Effiziente Algorithmen für Diskrete Probleme und ihre Anwendungen“ sowohl bei unseren theoretischen Arbeiten, als auch bei der konkreten Software-Erstellung. Am Projekt beteiligt sind Arbeitsgruppen an den Universitäten Halle (Näher, Alberts), Köln (Jünger, Lange, Leipert) und Passau (Brandenburg, Bachl, Himsolt, Stübinger, Wetzel) und am Max-Planck-Institut für Informatik in Saarbrücken (Mehlhorn, Mutzel, Hundack, Ziegler). Weitere Informationen sind in

`http://www.gmd.de/SCAI/dfg/e-version/projects.html`

zu finden. Dort ist auch beschrieben, wie man sich die im Rahmen des Projekts entstandene Software beschaffen kann.

Literatur

[1] H.K.B. Beck, H.-P. Galil, R. Henkel, und E. Sedlmayr. Chemistry in circumstellar shells, I. chromospheric radiation fields and dust formation in optically thin shells of M-giants. *Astron. Astrophys.*, 265:626–642, 1992.

[2] K. Booth und G. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. System Sci.*, 13:335–379, 1976.

[3] F.J. Brandenburg. Nice drawings of graphs are computationally hard. *LNCS*, 439:1–15, 1990.

[4] F.J. Brandenburg. On drawing planar angle graphs. *MIP Bericht, Universität Passau*, 1996.

[5] F.J. Brandenburg, M. Himsolt, und C. Rohrer. An experimental comparison of force-directed and randomized graph drawing algorithms. *Proc. Graph Drawing '95, LNCS*, 1027:76–87, 1996.

[6] F.J. Brandenburg (ed.). Proceedings Graph Drawing '95. *LNCS*, 1027, 1996.

[7] J. Cai, X. Han, und R.E. Tarjan. An $O(m \log n)$-time algorithm for the maximal planar subgraph problem. *SIAM J. Comput.*, 22:1142–1162, 1993.

[8] N. Chiba, T. Nishizeki, S. Abe, und T. Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. *J. Comput. System Sci.*, 30:54–76, 1985.

[9] N. Chiba, K. Onoguchi, und T. Nishizeki. Drawing planar graphs nicely. *Acta Informatica*, 22:187–201, 1985.

[10] M. Chrobak und G. Kant. Convex grid drawings of 3-connected planar graphs. *Technical Report RUU-CS-93-45, Dept. of Comp. Sci., Utrecht University*, 1993.

[11] H. De Fraysseix, J. Pach, und R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10:41–51, 1990.

[12] C. De Simone und M. Jünger. On the two-connected planar spanning subgraph polytope. *Technical Report No. 96.229, Institut für Informatik, Universität zu Köln*, 1996.

[13] G. Di Battista, P. Eades, R. Tamassia, und I.G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geometry: Theory Appl.*, 4:235–282, 1994.

[14] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[15] P. Eades und N.C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 10:379–403, 1994.

[16] M. Grötschel, M. Jünger, und G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32:1195–1220, 1984.

[17] H. Heine. Das Zeichnen planarer Graphen mit hoher Winkelauflösung. *Diplomarbeit, Universität Passau*, 1995.

[18] M. Himsolt. *Konzeption und Implementierung von Grapheneditoren.* Shaker Verlag, Aachen, 1993.

[19] M. Himsolt. A graph file format. *Manuskript, Universität Passau*, 1996.

[20] J. Hopcroft und R.E. Tarjan. Efficient planarity testing. *J. Assoc. Comput. Mach.*, 21:549–568, 1974.

[21] R. Jayakumar, K. Thulasiraman, und M.N.S. Swamy. $O(n^2)$ algorithms for graph planarization. *IEEE Trans. on Computer-aided Design*, 8:257–267, 1989.

[22] M. Jünger und P. Mutzel. Solving the maximum planar subgraph problem by branch and cut. In L.A. Wolsey und G. Rinaldi (Ed.), *Proc. 3rd IPCO Conference, Erice*, 479–492, 1993.

[23] M. Jünger und P. Mutzel. The polyhedral approach to the maximum planar subgraph problem: New chances for related problems. *Proc. Graph Drawing '94, LNCS*, 894:119–130, 1995.

[24] M. Jünger und P. Mutzel. Exact and heuristic algorithms for 2-layer straightline crossing minimization. *Proc. Graph Drawing '95, LNCS*, 1027:33–59, revised version to appear in J. Graph Algorithms and Applications, 1996.

[25] M. Jünger und P. Mutzel. Maximum planar subgraphs and nice embeddings: Practical layout tools. *Algorithmica*, 16:33–59, 1996.

[26] M. Jünger und S. Thienel. The design of the branch and cut system ABACUS. *Technical Report No. 97.260, Institut für Informatik, Universität zu Köln*, 1997.

[27] G. Kant. Drawing planar graphs using the lmc-ordering. *Proc. 33rd Symp. on Found. of Comp. Sci.*, 101–110, 1992.

[28] K. Kuratowski. Sur le problème des courbes gauches en topologie. *Fund. Math.*, 15:271–283, 1930.

[29] H. Lauer. Grapheneditoren: eine Wunschliste. *Bericht WSI-95-4, Univ. Tübingen*, 1995.

[30] S. Leipert. The Tree Interface - Version 1.0 User Manual. *Technical Report No. 96.242, Institut für Informatik, Universität zu Köln*, 1996.

[31] K. Mehlhorn und P. Mutzel. On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm. *Algorithmica*, 16:233–242, 1996.

[32] K. Mehlhorn, P. Mutzel, und S. Näher. An implementation of the Hopcroft and Tarjan planarity test and embedding algorithm. *Technical Report MPI-I-93-151, Max-Planck-Institut für Informatik, Saarbrücken*, 1993.

[33] K. Mehlhorn und S. Näher. LEDA: A platform for combinatorial and geometric computing. *Comm. Assoc. Comput. Mach.*, 38:96–102, 1995.

[34] P. Mutzel. The maximum planar subgraph problem. *Dissertation, Universität zu Köln*, 1994.

[35] P. Mutzel. A polyhedral approach to planar augmentation and related problems. *Proc. ESA '95, LNCS*, 979:494–507, 1995.

[36] P. Mutzel. An alternative approach for drawing hierarchical graphs. *Proc. Graph Drawing '96, LNCS*, erscheint 1997.

[37] S. North (ed.). Proceedings Graph Drawing '96. *LNCS*, erscheint 1997.

[38] H.C. Purchase, R.F. Cohen, und M. James. Validating graph drawing aesthetics. *Proc. Graph Drawing '95, LNCS*, 1027:435–446, 1996.

[39] E.M. Reingold und J.S. Tilford. Tidier drawings of trees. *IEEE Trans. Software Engineering*, SE-7:223–228, 1981.

[40] W. Schnyder. Embedding planar graphs on the grid. *Proc. 1st ACM–SIAM Symp. on Discrete Algorithms*, 138–147, 1990.

[41] M. Stoer. *Design of Survivable Networks*. Lecture Notes in Mathematics, Springer-Verlag, Berlin, 1992.

[42] K. Sugiyama, S. Tagawa, und M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Syst. Man, Cybern.*, SMC-11:109–125, 1981.

[43] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Computing*, 16:421–444, 1987.

[44] R. Tamassia und I.G. Tollis (eds.). Proceedings Graph Drawing '94. *LNCS*, 894, 1995.

[45] S Thienel. ABACUS - A Branch And CUt System. *Dissertation, Universität zu Köln*, 1995.

[46] W.T. Tutte. How to draw a graph. *Proc. London Math. Soc.*, 13:743–768, 1963.

[47] K. Wagner. Bemerkungen zum Vierfarbenproblem. *Jahresbericht Deutsch. Math.*, 46:26–32, 1936.

[48] J.Q. Walker II. A node-positioning algorithm for general trees. *Software – Practice and Experience*, 20:685–705, 1990.

[49] D.R. Woods. Drawing planar graphs. *Technical Report STAN-CS-82-943, Computer Science Dept., Stanford University*, 1981.