Neue Anwendungen von SPQR-Bäumen im Graphenzeichnen

René Weiskircher

Abstract: Wir untersuchen zwei Probleme auf dem Gebiet des Zeichnens von Graphen. Bei beiden Problemen geht es darum, eine Funktion über der Menge aller Einbettungen eines planaren Graphen zu optimieren und wir verwenden jeweils SPQR-Bäume um die Probleme zu lösen. Das erste von uns betrachtete Problem ist das Einfügen einer zusätzlichen Kante in einen planaren Graphen mit möglichst wenigen Kreuzungen. Dies is der erste Algorithmus, der das Problem löst und er hat lineare Laufzeit. Das zweite Problem ist das Berechnen einer orthogonalen Zeichnung mit der minimalen Anzahl von Knicken. Es ist bekannt, dass dieses Problem NP-schwer ist. Hier benutzen wir den SPQR-Baum, um ein ganzzahliges lineares Programm zu entwickeln, dessen Lösungen den Einbettungen des Graphen entsprechen. Dies ist die Grundlage für unseren Algorithmus für die Berechnung einer knick-minimalen Zeichnung, der sich in unseren Experimenten im Vergleich mit der bisher verwendeten Methode überlegen gezeigt hat.

1 Einführung

Es gibt viele wissenschaftliche Gebiete, in denen Graphen benutzt werden, um Beziehungen zwischen Objekten zu modellieren. Der Grund ist, dass viele Wissenschaften sich mit komplexen Systemen interagierender Objekte befassen. Die Objekte des Systems entsprechen den Knoten des Graphen und die Beziehungen den Kanten.

Eine Zeichnung eines Graphen kann das Verstehen eines Systems für einen menschlichen Betrachter sehr vereinfachen. Ein Beispiel für eine gelungene Visualisierung aus dem Gebiet der Wirtschaftswissenschaften zeigt Abbildung 1 [HJ00]. Dargestellt sind die Anteils-Beziehungen einiger großer spanischer Unternehmen. Diese Zeichnung ist eine orthogonale Zeichnung d.h. die Kanten sind als Folgen vertikaler und horizontaler Geradensegmente gezeichnet.

Eines der wichtigsten Kriterien für die Qualität einer orthogonalen Zeichnung ist die Anzahl der Kreuzungen. Enthält die Zeichnung eines Graphen viele Kreuzungen, so ist es für den Betrachter meist schwierig zu erkennen, welche Knoten durch eine Kante miteinander verbunden sind. Auch ist es in orthogonalen Zeichnungen wünschenswert, möglichst wenige Segmente pro Kante zu verwenden. Einer Kante, die aus vielen Segmenten besteht und somit viele Knicke hat, kann das Auge nicht so gut folgen wie einer Kante mit wenigen Knicken. Abbildung 2 zeigt zwei Zeichnungen des selben Graphen, die linke hat 11 Kreuzungen und 15 Knicke während die rechte nur eine Kreuzung und neun Knicke hat und deshalb viel übersichtlicher ist.

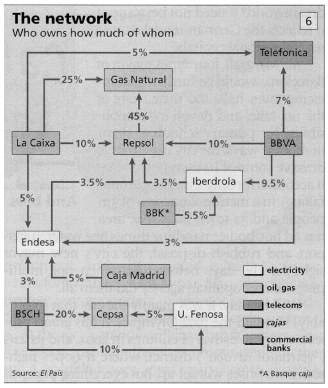


Abbildung 1: Ein Graph, der die Firmenbeteiligungen zwischen verschiedenen spanischen Firmen zeigt.

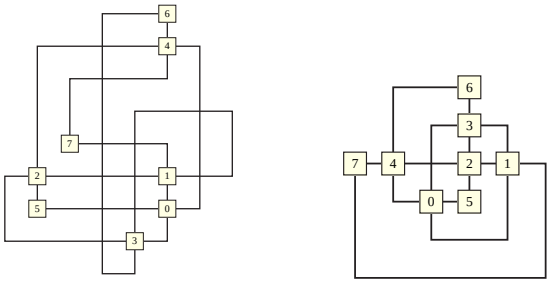


Abbildung 2: Zwei verschiedene Zeichnungen des selben Graphen mit unterschiedlicher Anzahl von Kreuzungen und Knicken

Hier beschäftigen wir uns sowohl mit der Minimierung der Anzahl der Kreuzungen in einer Zeichnung als auch mit der Minimierung der Anzahl der Knicke. Bei beiden Problemen geht es darum, eine Eigenschaft einer Zeichnung über alle möglichen Einbettungen zu optimieren und beides mal benutzen wir SPQR-Bäume, um das Problem zu lösen. Abschnitt 2 gibt eine kurze Einführung in SPQR-Bäume. In Abschnitt 3 untersuchen wir das Problem, wie man eine neue Kanten in einen planaren Graphen einfügen kann, so dass möglichst wenige Kreuzungen entstehen. Der resultierende Algorithmus kann in einer Heuristik verwendet werden, die nicht-planare Graphen mit wenigen Kreuzungen zeichnet. In Abschnitt 4 entwickeln wir einen Algorithmus, der für einen Graphen eine orthogonale Zeichnung mit der kleinsten möglichen Anzahl von Knicken berechnet. Das dort vorgestellte ganzzahlige lineare Programm ermöglicht es erstmals, Algorithmen aus der linearen Programmierung auf Probleme anzuwenden, bei denen über die Menge der Einbettungen eines Graphen optimiert werden muss.

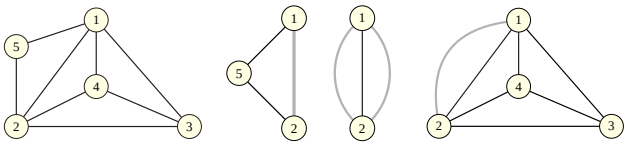
2 SPQR-Bäume

SPQR-Bäume wurden von Di Battista und Tamassia eingeführt [BT89] und man kann sie benutzen, um alle Einbettungen eines Graphen aufzuzählen. Einbettungen beschreiben die Topologie einer planaren Zeichnung eines Graphen. Man kann eine Einbettung definieren, indem man für jeden Knoten die Reihenfolge der inzidenten Kanten im Uhrzeigersinn angibt.

Die Eigenschaft von SPQR-Bäume, alle Einbettungen eines Graphen aufzählen zu können nutzen Bertolazzi, Didimo und Di Battista aus, um mit Hilfe eines Branch & Bound Algorithmus die Knicke einer Zeichnung zu minimieren [BBD00] und um Zeichnungen zu erzeugen, in denen möglichst viele Kanten nach oben gerichtet sind [BBD02]. Das hat uns motiviert, sie bei der Lösung der beiden Probleme, die hier betrachtet werden, zu verwenden. Die Größe der Datenstruktur SPQR-Baum eines Graphen G wächst nur linear mit der Anzahl der Kanten in G und das gilt auch für die benötigte Zeit um den Baum zu berechnen [GM01].

Jeder Knoten des Baums ist mit einem speziellen Graphen assoziiert, dem *Skelett* des Knotens, den man als vereinfachte Sicht von G interpretieren kann. Die Knoten eines Skelettes sind Knoten von G während die Kanten entweder Kanten oder Teilgraphen von G entsprechen. Es gibt vier Typen von Knoten im Baum (S-,P-, Q- und R-Knoten), die sich durch die Struktur ihrer Skelette unterscheiden. Die Q-Knoten sind die Blätter des Baums.

Abbildung 3 zeigt einen Graphen 3(a) und die Skelette seiner inneren Knoten. Diese sind ein S-Knoten 3(b), ein P-Knoten 3(c) und ein R-Knoten 3(d). Die grauen Kanten in den Skeletten stehen für Teilgraphen während die schwarzen Kanten auch im Originalgraphen enthalten sind.



(a) (b) (c) (d)

Abbildung 3: Ein Graph 3(a) und die Skelette der inneren Knoten seines SPQR-Baums

Die Eigenschaft, die wir uns bei unseren Algorithmen hauptsächlich zu nutzen machen ist, dass eine Einbettung für den Graphen eine Einbettung für alle Skelette definiert und umgekehrt.

3 Das Einfügen einer Kante in einen planaren Graphen

Enthält die Zeichnung eines Graphen viele Kreuzungen ist sie meist nicht gut lesbar. Unglücklicherweise ist es NP-schwer, für einen Graphen eine Zeichnung mit der minimalen Anzahl von Kreuzungen zu finden [GJ83]. Selbst für relativ kleine Graphen (z.B. für Graphen mit 20 Knoten), ist kein Algorithmus mit vertretbarer Laufzeit bekannt, der das Problem löst.

Aus diesem Grund benutzt man Heuristiken um Zeichnungen von nicht-planaren Graphen mit wenigen Kreuzungen zu berechnen. Eine der besten Methoden, die man hierfür verwendet, ist die *Planarisierungsmethode*. Dabei werden erst Kanten aus dem Eingabegraphen gelöscht, um einen *planaren*, d.h. kreuzungsfrei zeichenbaren Graphen zu erzeugen. In diesen werden dann die gelöschten Kanten nacheinander wieder eingefügt, wobei man die entstehenden Kreuzungen durch neue Knoten mit vier inzidenten Kanten ersetzt (Abbildung 4 zeigt diesen Vorgang). Dadurch liegt zu jedem Zeitpunkt ein planarer Graph vor. Eine Zeichnung des entstandenen Graphen kann man leicht in eine Zeichnung des ursprünglichen Graphen umwandeln, indem man die neuen Knoten wieder durch Kreuzungen ersetzt.

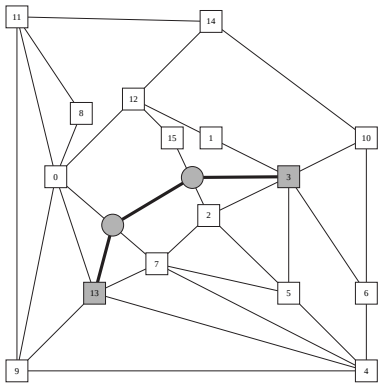


Abbildung 4: Künstliche Knoten (hier als graue Kreise gezeichnet) wurden in den Graphen eingefügt, um die Kreuzungen zu beseitigen, die beim Einfügen der neuen Kante entstanden sind.

Das Problem, dass sich beim Wiedereinfügen der Kanten stellt haben wir mit Hilfe von SPQR-Bäume gelöst. Es geht darum, eine Kante in einen planaren Graphen G so einzufügen, dass dabei möglichst wenige Kreuzungen entstehen und dass sich keine Kanten von G kreuzen.

Bevor wir uns mit dem Problem beschäftigt haben, wurde das Einfügen der Kanten wie folgt durchgeführt: Man wählt eine beliebige Einbettung des planaren Graphen und berechnet einen kürzesten Pfad in dem dualen Graphen, der durch die Einbettung definiert ist. Die erhaltene Lösung garantiert, dass die Anzahl der entstandenen Kreuzungen für die gewählte Einbettung minimal ist. Allerdings kann es für andere Einbettungen bessere Lösungen geben.

Unser neuer Algorithmus minimiert die Anzahl der Kreuzungen über alle Einbettungen. Er zerlegt zuerst den Graphen in seine Zusammenhangskomponenten. Sind die beiden Knoten u und v, die es zu verbinden gilt, in verschiedenen Zusammenhangskomnponenten enthalten (es gibt also keinen Pfad im Graphen der sie verbindet), so können wir stets die Kante (u, v) in den Graphen einfügen, ohne eine Kreuzung zu erzeugen.

Liegen u und v in der gleichen Zusammenhangskomponente, so wird für diese der *Block-Baum* B berechnet. Dieser enthält außer den Knoten des ursprünglichen Graphen auch Knoten, die die zweizusammenhängenden Komponenten des Graphen darstellen (maximale Teilgraphen, die nicht durch Löschung eines Knotens zerfallen). Der Algorithmus berechnet den Pfad P in B von u nach v und für jede Komponente auf dem Pfad ein paar von Knoten, das darin u und v repräsentiert. Dann ruft der Algorithmus für jede der Komponenten die Funktion für zweizusammenhängende Graphen auf. Dieser berechnet dann jeweils eine Folge von Kanten, den *Einfügepfad*, die durch die neue Kante gekreuzt werden. Der Einfügepfad für den gesamten Graphen ergibt sich durch Aneinanderhängen der Einfügepfade für die Komponenten.

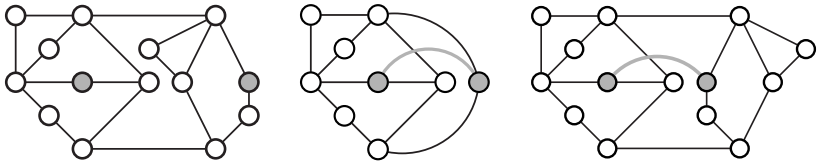
Der Algorithmus für zweizusammenhängende Graphen ist das Kernstück des Verfahrens. Er nützt aus, dass nur die Skelette der R-Knoten im SPQR-Baum des Graphen für die Anzahl der notwendigen Kreuzungen entscheidend sind. Der Algorithmus betrachtet alle R-Knoten Skelette auf dem Pfad im SPQR-Baum zwischen den zu verbindenden Knoten u und v. In jedem dieser Skelette erzeugen wir zwei neue Knoten u' und v', die u und v repräsentieren.

Danach ersetzen wir alle Kanten des Skelettes, die nicht inzident zu u' oder v' sind durch die Teilgraphen von G, die sie repräsentieren. Für eine beliebige Einbettung des erhaltenen Graphen, den wir den *Erweiterungsgraphen* des Skelettes nennen, können wir nun den kürzesten Pfad im dualen Graphen berechnen, der u' mit v' verbindet. Dieser Pfad entspricht einem Einfügepfad für u' und v'. Hängen wir die Einfügepfade aller R-Knoten Skelette auf dem Pfad aneinander, so erhalten wir einen Einfügepfad für den gesamten Graphen.

Abbildung 5(a) zeigt einen Graphen dessen SPQR-Baum nur einen R-Knoten enthält mit den beiden zu verbindenden Knoten. Abbildung 5(b) zeigt den Erweiterungsgraphen des Skelettes des R-Knotens mit der neuen Kante. In Abbildung 5(c) ist die entsprechende Kante im Originalgraphen so eingefügt, dass die gleiche Kante wie im Erweiterungsgraphen gekreuzt wird. Dazu wurde die Einbettung des Originalgraphen entsprechend geändert.

Wir können zeigen, dass der von unserem Algorithmus berechnete Einfügepfad für die neue Kante minimale Länge hat. Dies ist auch unter Berücksichtigung aller möglichen Einbettungen des Graphen wahr und somit löst er das Problem des Einfügens einer Kante optimal. Die Laufzeit unsere Algorithmus ist linear in der Größe des Eingabegraphen. Auch die bisher verwendete nicht optimale Methode zum Einfügen einer Kante in einen planaren Graphen hat lineare Laufzeit.

Wir wollten nun wissen, welche Vorteile es bringt, unseren neuen Algorithmus im Rahmen der Planarisierungsmethode zu benutzen. Wir verwendeten dazu die 8249 nicht-planaren Graphen aus der bekannte Benchmark-Suite aus [BGL+97] (Graphen aus industriellen



(a) (b) (c)

Abbildung 5: Ein Graph mit zu verbindenden Knoten (a), dem Erweiterungsgraphen des R-Knoten Skelettes mit der eingfügten Kante (b) und das entsprechende Resultat im ursprünglichen Graphen (c)

Anwendungen). Abbildung 6 zeigt die durchschnittliche prozentuale Verringerung der Anzahl der Kreuzungen bei Verwendung unseres neuen Verfahrens statt dem Standardverfahren. Es gab nur wenige kleine Graphen in der Benchmark-Suite, weshalb die Durchschnitte erst ab etwa 40 Knoten aussagekräftig werden. Dann zeigt sich aber eine durchschnittliche Verbesserung von etwa 15%.

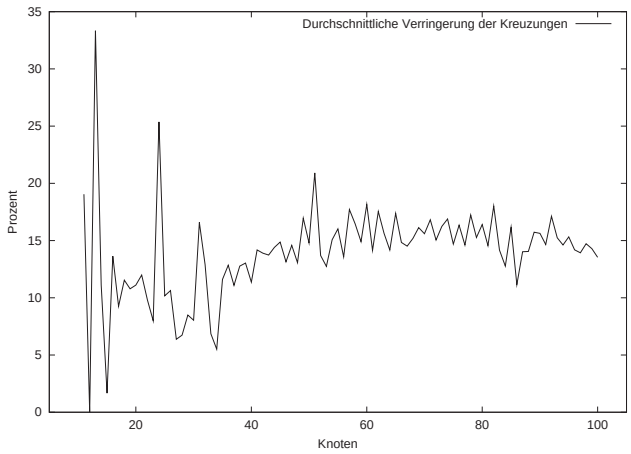


Abbildung 6: Durchschnittliche Verringerung der Kreuzungszahl in Prozent bei Verwendung unserer neuen Methode

4 Knickminimierung in Orthogonalen Zeichnungen

In einer orthogonalen Zeichnung eines Graphen werden die Kanten als Folgen von horizontalen und vertikalen Geradensegmenten gezeichnet. Ein Beispiel ist die Zeichnung in

Abbildung 1. Der Punkt, wo ein vertikales Segment einer Kante sich mit einem horizontalen Segment der gleichen Kante schneidet, wird *Knick* genannt.

Es ist schon länger bekannt, dass man für eine fixe Einbettung eines Graphen in polynomieller Zeit eine Zeichnung mit der minimalen Anzahl von Knicken berechnen kann. Dies geschieht, indem man das Problem in ein Fluss-Netzwerk umwandelt, indem man dann einen Fluss mit minimalen Kosten bestimmt. Der erste Algorithmus dieser Art wurde von Tamassia entwickelt [Ta87]. Ist die Einbettung eines Graphen nicht festgelegt, so ist es ein NP-schwer, eine Zeichnung mit der kleinstmöglichen Anzahl von Knicken zu berechnen [GT02].

Abbildung 7 zeigt zwei orthogonale Zeichnungen des gleichen Graphen, die zwei unterschiedliche Einbettungen darstellen. Beide Zeichnungen sind knickminimal für die dargestellte Einbettung. Während die linke Zeichnung 13 Knicke aufweist, hat die rechte Zeichnung nur 7 Knicke. Dies zeigt, dass die Wahl der Einbettung großen Einfluß auf die Anzahl der Knicke in der fertigen Zeichnung haben kann.

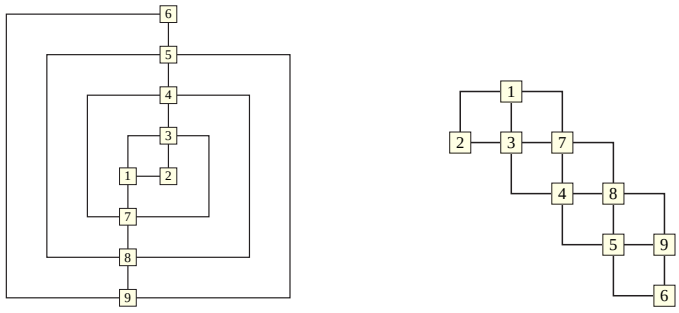


Abbildung 7: Zwei orthogonale Zeichnungen des gleichen Graphen, die jeweils knickminimal sind für die dargestellte Einbettung

Es gibt bereits einen Branch & Bound Algorithmus, um einen planaren zweizusammenhängenden Graphen mit der minimalen Anzahl von Knicken zu zeichnen [BBD00]. Wir sind dieses Problem mit Hilfe von Methoden der ganzzahligen linearen Programmierung angegangen und haben dadurch einen beachtlichen Geschwindigkeitsvorteil für große Probleminstanzen erzielt.

Zur Beschreibung von Einbettungen als Vektoren mit binären Einträgen verwenden wir die Tatsache, dass man eine Einbettung auch durch die Menge der gerichteten Kreise charakterisieren kann, die Regionen begrenzen. Ein Kreis begrenzt eine Region, wenn in einer Zeichnung, die die Einbettung realisiert, die linke Seite des Kreises leer ist. Man nennt diese Kreise auch *Regionen-Kreise*. Abbildung 8 zeigt ein Beispiel für einen Regionen-Kreis.

Wieder nutzen wir die Tatsache aus, dass ein SPQR-Baum alle Einbettungen eines Graphen repräsentieren kann. Wie schon vorher erwähnt, ist jeder Knoten in dem Baum mit einem speziellen Graphen, seinem Skelett, assoziiert. Weil die Skelette eine einfache Struktur haben ist es nicht schwierig, ein ganzzahliges lineares Programm (GLP) zu konstruieren, welches die Einbettungen eines Skelettes beschreibt. Wir führen für jeden gerichteten

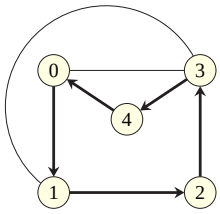


Abbildung 8: Der fett gezeichnete gerichtete Kreis ist ein Regionen-Kreis der dargestellten Einbettung.

Kreis, der in zumindest einer Einbettung des Skelettes ein Regionen-Kreis ist, eine binäre Variable ein. Hat diese Variable Wert 1, so ist der Kreis in der dargestellten Einbettung ein Regionen-Kreis.

Wir berechnen das GLP für komplexere Graphen rekursiv, indem wir den SPQR-Baum an einem inneren Knoten in kleinere SPQR-Bäume (die *Spalt-Bäume*) aufspalten. Dieser Vorgang ist in Abbildung 9 dargestellt. Dabei führen wir neue Q-Knoten ein, denn auch die Spalt-Bäume müssen als SPQR-Bäume die Eigenschaft haben, dass alle Blätter Q-Knoten sind.

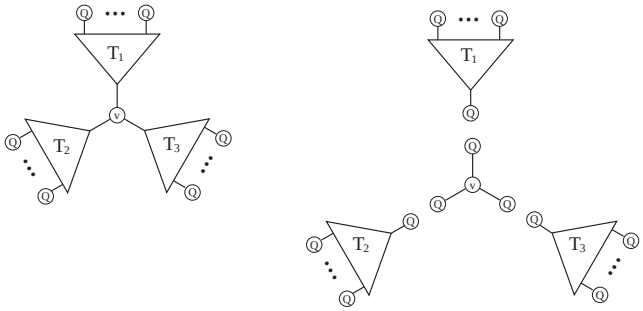


Abbildung 9: Aufspalten eines SPQR-Baums am inneren Knoten v in kleinere Bäume

Um aus den GLPs der Spalt-Bäume ein GLP für den ursprünglichen Baum zu konstruieren, bestimmen wir erst die Variablenmenge für das gesamte Problem. Dazu kombinieren wir Kreise, die durch Variablen in den GLPs der Spalt-Bäume repräsentiert sind um Kreise im Originalgraphen zu erhalten. Unser Verfahren garantiert, dass wir genau für die Kreise Variablen einführen, die Regionen-Kreise in mindestens einer Einbettung des Graphen sind. Also führen wir die kleinste mögliche Menge von Variablen ein die nötig ist, um alle Einbettungen darstellen zu können.

Einen Teil der Ungleichungen für das GLP des ursprünglichen Graphen erhalten wir, indem wir die Ungleichungen in den GLPs der Spalt-Graphen auf die Variablenmenge des ursprünglichen Graphen anpassen. Wir führen Außerdem noch zusätzliche Ungleichungen ein, die den Zusammenhang zwischen den Spalt Graphen modellieren. Wir konnten zeigen, dass die Lösungen des so gewonnenen GLPs exakt der Menge aller Einbettungen des

Graphen entsprechen.

Um nun unser ursprüngliches Knickminimierungsproblem zu lösen, kombinieren wir das GLP, das die Einbettungen des Graphen beschreibt mit einem linearen Programm (LP), das die orthogonalen Repräsentationen des Graphen für eine feste Einbettung beschreibt. Das LP ist abgeleitet von der Formulierung des Knickminimierungsproblems für feste Einbettungen als Netzwerk-Fluss Problem [BBD00].

Die Kombination des GLP mit dem LP erzeugt ein gemischt ganzzahliges lineare Programm, welches die Menge der orthogonalen Repräsentationen eines Graphen beschreibt. Eine Optimallösung des Programms entspricht einer orthogonalen Repräsentation mit der minimalen Anzahl von Knicken. Wir lösen das Programm mit Hilfe von CPLEX, einem kommerziellen Löser für gemischt ganzzahlige lineare Programme. Unser Einbettungs-GLP kann man auf diese Weise immer dort benutzen, wo man eine Funktion über alle Einbettungen optimieren will und das Problem sich für eine feste Einbettung als (ganzzahliges) lineares Programm beschreiben lässt. Durch Verbindung der linearen Prgramme kann man dann die Funktion über alle Einbettungen optimieren.

Unsere Experimente zeigen, dass unser neuer Ansatz dem schon bekannten Branch & Bound Ansatz von [BBD00] überlegen ist. Wir haben mit dem Graphen-Generator, den auch Bertolazzi, Di Battista und Didimo verwendet haben 500 Graphen generiert, die sich durch eine große Anzahl von Einbettungen auszeichnen. Auf diese Graphen haben wir die Implementierung des Branch & Bound Algorithmus aus [BBD00] und die Implementierung unseres neuen Algorithmus angewendet. Beide Programme liefen auf dem gleichen Rechner unter den gleichen Bedingungen. Die x-Achse in Abbildung 10 zeigt die Anzahl der Knoten der Graphen und die y-Achse die durchschnittliche Laufzeit der Algorithmen für die Graphen mit der entsprechenden Anzahl von Knoten. Das Diagramm zeigt, dass unser Algorithmus (GLP) fast doppelt so schnell ist wie der Branch & Bound Algorithmus (B & B).

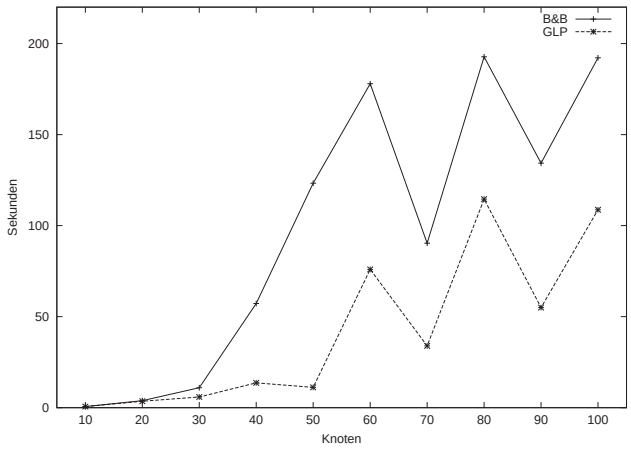


Abbildung 10: Laufzeitvergleich mit dem Branch & Bound Algorithmus

5 Ausbildung und Akademische Laufbahn

10/92-07/97	Informatik Studium an der Universität des Saarlandes in Saarbrücken
08/97-10/97	Wissenschaftlicher Mitarbeiter am Max–Planck–Institut für Informatik in Saarbrücken
11/97-09/99	Mitglied des Graduiertenkollegs “Effizienz und Komplexität von Algorithmen und Rechenanlagen” an der Universität des Saarlandes
10/99-06/00	Universitätsassistent an der Technischen Universität Wien, Institut für Computergraphik und Algorithmen
07/00-12/00	Wissenschaftlicher Mitarbeiter an der University of Newcastle, Australien
Seit 01/01	Universitätsassistent an der Technischen Universität Wien, Institut für Computergraphik und Algorithmen
23.05.02	Promotion an der Universität des Saarlandes

Literatur

[BBD00] Bertolazzi, P., Battista, G. D., und Didimo, W.: Computing orthogonal drawings with the minimum number of bends. *IEEE Transactions on Computers*. 49(8):826–840. 2000.

[BBD02] Bertolazzi, P., Battista, G. D., und Didimo, W.: Quasi-upward planarity. *Algorithmica*. 32(3):474–506. 2002.

[BGL+97] Battista, G. D., Garg, A., Liotta, G., Tamassia, R., Tassinari, E., und Vargiu, F.: An experimental comparison of four graph drawing algorithms. *Comput. Geom. Theory Appl.* 7:303–326. 1997.

[BT89] Battista, G. D. und Tamassia, R.: Incremental planarity testing. In: *Proc. 30th Annual IEEE Symp. on Foundations of Computer Science (FOCS)*. S. 436–441. 1989.

[GJ83] Garey, M. R. und Johnson, D. S.: Crossing number is NP-complete. *SIAM Journal Alg. Disc. Methods*. 4:312–316. 1983.

[GM01] Gutwenger, C. und Mutzel, P.: A linear time implementation of SPQR-trees. In: Marks, J. (Hrsg.), *Graph Drawing (Proc. 2000)*. volume 1984 of *LNCS*. S. 77–90. Springer-Verlag. 2001.

[GT02] Garg, A. und Tamassia, R.: On the computational complexity of upward and rectilinear planarity testing. *SIAM Journal on Computing*. 31(2):601–625. April 2002.

[HJ00] Hugh-Jones, S.: Survey: Spain. *The Economist*. 2000.

[Ta87] Tamassia, R.: On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing*. 16(3):421–444. 1987.