Julius-Maximilians-Universität Würzburg
Institut für Informatik
Lehrstuhl für Informatik I
Effiziente Algorithmen und wissensbasierte Systeme

**Bachelorarbeit**

# Automatisches Zeichnen von U-Bahn-Linienplänen unter Verwendung von Bézierkurven

Julian Schuhmann

Eingereicht am 29. Juli 2011

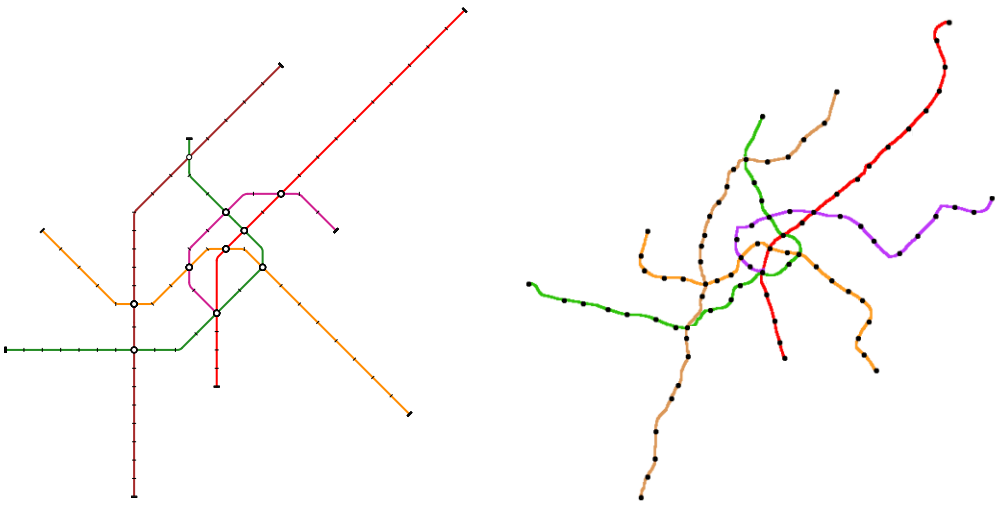
Betreuer:
Prof. Dr. Alexander Wolff
Dipl.-Inf. Martin Fink

# Inhaltsverzeichnis

# 1 Einleitung

U-Bahn-Linienpläne sind für Menschen in Großstädten ein wichtiges Mittel, um sich in den komplexen U-Bahn-Netzwerken zurechtzufinden. Sie müssen gut lesbar sein, damit sie auch für Auswärtige, die den Plan zum ersten Mal sehen, gut verständlich sind. Mit ihnen müssen die Passagiere Fragen beantworten können wie: „Welche Linie(n) muss ich nehmen, um möglichst schnell von Station A zu Station B zu kommen?“ oder „Wo muss ich auf welche Linie umsteigen?“. Dazu müssen die Pläne nicht wie bei einer Straßenkarte genau der Realität entsprechen. Es ist vielmehr nötig, das Netzwerk übersichtlich darzustellen, damit aufkommende Fragen schnell und problemlos beantwortet werden können.

Zu diesem Zweck erstellte Henry Beck bereits 1933 eine schematische Darstellung des U-Bahn-Systems von London. Er führte ein Layout ein, das als *oktilinear* bezeichnet wird. Hier dürfen Linienabschnitte nur senkrecht, waagrecht oder diagonal gezeichnet werden. Dieses Layout erwies sich als gut geeignet, weshalb bis heute die meisten U-Bahn-Linienpläne so dargestellt werden. Abb. 1.1a zeigt ein Beispiel für eine oktilineare Zeichnung des Wiener U-Bahn-Netzes. Diese Zeichnung wurde automatisch von einem Programm von Nöllenburg und Wolff [NW11] erstellt. Zum Vergleich zeigt Abb. 1.1b die geographisch korrekte Zeichnung des U-Bahn-Netzes.



**(a)** Das U-Bahn-Netz von Wien in oktilinearem Layout

**(b)** Die geographisch korrekte Zeichnung des Wiener U-Bahn-Netzes

**Abb. 1.1:** Verschiedene Zeichnungen des Wiener U-Bahn-Netzes

In den letzten Jahren wurde auch das Problem untersucht, Zeichnungen von oktilinearen U-Bahn-Linienplänen automatisch zu erstellen. Dieses Problem ist NP-schwer. Trotzdem wurden praxistaugliche Methoden entwickelt [NW11, SRMW11], die in annehmbarer Zeit gute Lösungen liefern.

Benutzerstudien von Roberts et al. [RND+11] zeigen jedoch, dass oktilineare Pläne nicht optimal sind, was die Effektivität bei der Planung einer Fahrt angeht. Die Probanden hatten die Aufgabe, eine möglichst schnelle Route mit möglichst wenig Umsteigevorgängen von einer Station zu einer anderen zu finden. Die besten Ergebnisse wurden hier mit Plänen erzielt, in denen die Linien nicht aus geraden Strecken, sondern aus Kurven bestehen. Diese Untersuchungen lieferten die Motivation zu dieser Arbeit.

Einen Plan mit kurvigen Linien per Hand zu erstellen ist allerdings sehr aufwändig. Deshalb ist das Ziel dieser Arbeit einen Algorithmus zu entwickeln, der automatisch einen Plan erstellt, in dem alle Linien durch Kurven (sogenannte *Bézierkurven*) dargestellt werden. Dabei soll das Ergebnis so übersichtlich wie möglich sein, das heißt es sollen zum Beispiel keine scharfen Kurven vorkommen und Überschneidungen von Linien außerhalb von Stationen sollen verhindert werden. Abb. 1.2 zeigt am Beispiel von Wien, wie eine Ausgabe in etwa aussehen soll.

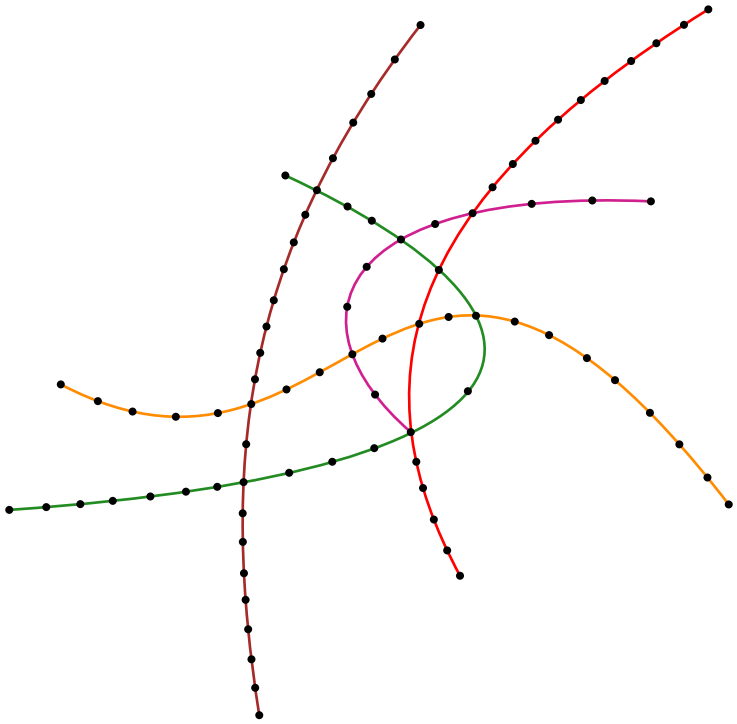


**Abb. 1.2:** Das U-Bahn-Netz von Wien mit kurvigen Linien

Der entwickelte Algorithmus baut dabei auf sogenannten *kräftebasierten* Verfahren auf. Das heißt er erhält als Eingabe bereits eine Zeichnung des U-Bahn-Linienplans. Dann werden verschiedene Kräfte berechnet, die auf die Zeichnung einwirken. Diese Kräfte werden nach dem Ändern der Zeichnung jeweils neu berechnet und sie modi-

fizieren die Zeichnung so lange, bis eine schöne Ausgabe erzeugt wurde. Mit solchen kräftebasierten Verfahren haben sich bisher unter anderem Bertault [Ber99] und Finkel und Tamassia [FT05] beschäftigt. Anhand deren Arbeiten wird in Kapitel 2 die generelle Vorgehensweise von kräftebasierten Verfahren näher erläutert.

Das Neue in dieser Arbeit ist die Anwendung eines kräftebasierten Verfahrens auf eine Zeichnung, die aus Bézierkurven besteht. Finkel und Tamassia haben sich bereits mit diesem Thema beschäftigt, allerdings simuliert ihr Algorithmus die Kurven nur und fügt sie erst am Ende in die Zeichnung ein. Damit der Algorithmus tatsächlich mit Bézierkurven arbeiten kann, werden in dieser Arbeit neue Kräfte eingeführt, die speziell darauf ausgelegt sind, für eine übersichtliche Zeichnung mit Bézierkurven zu sorgen. Kapitel 3 beschreibt den gesamten Algorithmus und die verwendeten Kräfte. Im Verlauf des Algorithmus könnte es außerdem dazu kommen, dass sich in der Zeichnung Linien überschneiden, die das in der Realität nicht tun. Weil das zu missverständlichen Plänen führen kann, wird ein Verfahren vorgestellt, welches Überschneidungen verhindert. In Kapitel 4 wird dann näher auf die Implementierung des Algorithmus und die Ausgaben, die dieser produziert, eingegangen. Abschließend werden in Kapitel 5 einige Ideen vorgestellt, wie man den Algorithmus noch verbessern könnte.

# 2 Grundlagen und bisherige Arbeiten

Da im Folgenden Algorithmen vorgestellt werden, die Graphen zeichnen, muss zunächst einmal definiert werden, was ein Graph ist. Ein *Graph* ist ein Paar $G = (V, E)$. Dabei ist $V$ eine Menge von Objekten, die man als *Knoten* bezeichnet, und $E$ ist eine Menge, die aus Paaren von Knoten besteht. Diese Paare bezeichnet man als *Kanten*. Jede Kante verbindet immer genau zwei Knoten. Eine Kante, die die Knoten $u$ und $v$ verbindet, wird mit $(u, v)$ bezeichnet. Zwei Knoten $u$ und $v$ heißen *benachbart*, wenn es eine Kante gibt, die $u$ und $v$ verbindet. Ein Knoten $v$ und eine Kante $e$ heißen *inzident*, wenn $v$ einer der beiden Knoten ist, die $e$ miteinander verbindet. Die Inzidenz-Relation lässt sich auch für zwei Kanten definieren: Im Folgenden heißen zwei Kanten $e$ und $f$ inzident, wenn ein Knoten existiert, der sowohl zu $e$ als auch zu $f$ inzident ist. Eine gute Einführung in die Graphentheorie geben Krumke und Noltemeier [KN09]. In dieser Arbeit sind die Knoten des Graphen die Stationen der U-Bahn-Linienpläne und die Kanten geben an, dass zwischen zwei Stationen Schienen verlaufen, die die Stationen direkt miteinander verbinden.

Graphen sind zunächst eine sehr abstrakte Datenstruktur. Um die darin enthaltene Information für Menschen verständlich darzustellen, werden Graphen gezeichnet. Dabei werden die Knoten meist als Punkte oder kleine Kreise dargestellt und die Kanten werden als Linien abgebildet, die die entsprechenden Knoten miteinander verbinden. Die einfachste Möglichkeit, um Graphen darzustellen, ist mit einer *geradlinigen* Zeichnung. Hier erhält jeder Knoten eine Position auf der Ebene und die Kanten werden als Strecken dargestellt. Für komplexe Graphen ist eine solche Zeichnung aber nicht immer sinnvoll, weil sie bei vielen Knoten und Kanten schnell unübersichtlich wird. Aus diesem Grund werden U-Bahn-Linienpläne meist *oktilinear* gezeichnet. Das heißt, dass jede Kante nur aus waagrechten, senkrechten oder diagonalen Abschnitten bestehen darf. Das führt dazu, dass sich die einzelnen Kanten deutlich voneinander abheben, was die Übersichtlichkeit erhöht. Die Bezeichnung oktilinear kommt daher, dass eine Kante einen Knoten in genau acht Richtungen verlassen kann.

Damit man nicht alle Graphen per Hand zeichnen muss, beschäftigt man sich schon länger damit, wie man automatisch eine übersichtliche Zeichnung generieren kann. Eine Möglichkeit, die auch in dieser Arbeit verwendet wird, ist die Anwendung von kräftebasierten Verfahren. In diesen Algorithmen werden Kräfte berechnet, die auf Knoten und Kanten des Graphen einwirken. Üblicherweise stoßen sich alle Knoten gegenseitig ab, um eine möglichst gleichmäßige Verteilung der Knoten zu erreichen. Außerdem ziehen sich benachbarte Knoten an, da diese nah beieinander gezeichnet werden sollen. So ist es für den Betrachter leichter, alle Nachbarn eines Knotens zu finden.

Der Ablauf solcher Algorithmen sieht folgendermaßen aus: Man geht von einer beliebigen Zeichnung des Eingabegraphen aus. Anschließend werden die anziehenden und

abstoßenden Kräfte für alle Knoten und Kanten berechnet, und die Zeichnung wird entsprechend angepasst. Das passiert über viele Iterationen, bis sich ein Kräftegleichgewicht einstellt und sich die Knoten nicht mehr nennenswert verschieben. Abhängig von der Definition der Kräfte ergibt sich dann eine mehr oder weniger schöne Zeichnung des Graphen.

Solche Algorithmen, die Kräfte benutzen um Graphen übersichtlich darzustellen, existieren schon lange. Bereits im Jahr 1984 entwickelte Eades [Ead84] den ersten kräftebasierten Algorithmus. Er verwendete zur Anschauung Metallringe, die den Knoten entsprechen. Die Kanten modellierte er als Metallfedern, die sich so lange zusammenziehen, bis die gewünschten Kantenlängen erreicht sind.

Ein anderer wichtiger Algorithmus stammt von Fruchterman und Reingold [FR91]. Sie erweiterten die Idee von Eades um abstoßende Kräfte zwischen zwei Knoten, um eine gleichmäßigere Verteilung der Knoten zu erreichen. Außerdem führten sie eine Grenze ein, die angibt, wie weit sich die Knoten in einer Iteration verschieben dürfen. Diese Grenze nimmt nach jeder Iteration ab, um besser ein Gleichgewicht herstellen zu können. Denn ohne die Begrenzung kann es vorkommen, dass sich die Zeichnung auch nach sehr vielen Iterationen noch stark verändert, wenn das Kräftegleichgewicht sehr instabil ist.

Es wird nun auf einige bekannte Kräfte näher eingegangen. Diese werden anhand eines Algorithmus von Bertault [Ber99] vorgestellt. Das Neue an seinem Algorithmus ist, dass er Kantenkreuzungen bewahrt. Das heißt zwei Kanten schneiden sich in der Ausgabezeichnung genau dann, wenn sie sich in der Eingabezeichnung geschnitten haben. Planare Graphen zeichnet dieser Algorithmus kreuzungsfrei, indem er im ersten Schritt mit bereits bekannten Algorithmen [Kan92, MM96] eine planare Zeichnung des Eingabegraphen sucht. Der Nachteil einer planaren Zeichnung ist jedoch, dass diese häufig sehr unübersichtlich ist, da es sehr kurze Kanten geben kann und andere Kanten im Vergleich dazu extrem lang sind. Deshalb wird im zweiten Schritt eine kräftebasierte Methode verwendet, um die Zeichnung übersichtlicher zu machen. Um neue Kantenkreuzungen zu verhindern, dürfen die Knoten dabei nicht beliebig weit verschoben werden, sondern nur so weit, dass unmöglich neue Kantenkreuzungen entstehen können.

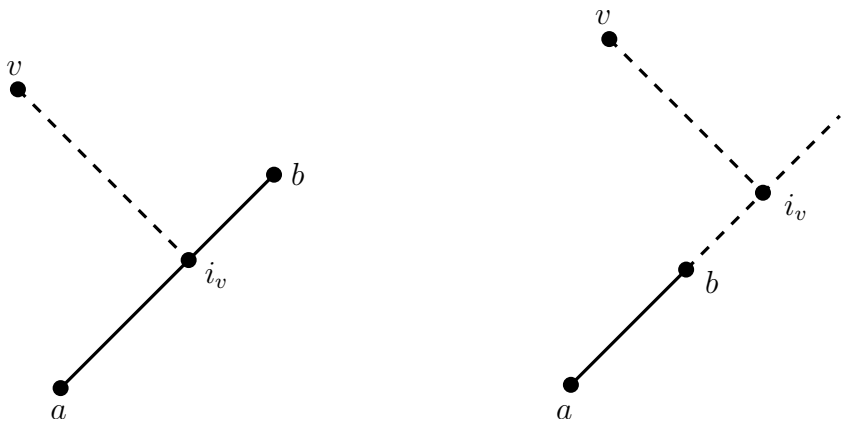
Die von Bertault benutzten Kräfte sind denen in anderen kräftebasierten Algorithmen sehr ähnlich. Es werden drei Kräfte zwischen Knoten und Kanten betrachtet: die abstoßende Kraft zwischen allen Paaren von Knoten, die Anziehungskraft zwischen benachbarten Knoten und die abstoßende Kraft zwischen Knoten und Kanten. Der Vektor von einem Knoten $u$ zu einem Knoten $v$ sei im Folgenden mit $\overrightarrow{uv}$ bezeichnet. Der euklidische Abstand zwischen zwei Knoten $u$ und $v$ wird mit $d(u, v)$ bezeichnet und $\delta$ ist die gewünschte Kantenlänge für alle Kanten. Die abstoßenden und anziehenden Kräfte zwischen zwei Knoten $u$ und $v$ sind im Artikel von Bertault nicht korrekt definiert, da die Richtungen falsch sind. Die richtigen Definitionen für die Kräfte, die von einem Knoten $u$ auf den Knoten $v$ wirken, lauten:

$$\text{Abstoßende Kraft (repelling):} \qquad F^{\mathrm{r}}(u, v) = \frac{\delta^2}{d(u, v)^2} \cdot \overrightarrow{uv}$$

$$\text{Anziehende Kraft (attracting):} \qquad F^{\mathrm{a}}(u, v) = \frac{d(u, v)}{\delta} \cdot \overrightarrow{vu}$$

Die abstoßende Kraft $F^{\mathrm{e}}(v,(a,b))$ zwischen einem Knoten $v$ und einer Kante $(a,b)$ wirkt sowohl auf den Knoten $v$ als auch auf die beiden Endknoten der Kante $(a,b)$. Zur Berechnung dieser Kraft wird ein virtueller Knoten $i_v$ betrachtet. Dieser ist definiert durch die orthogonale Projektion von $v$ auf die Stecke $\overline{ab}$ (Abb. 2.1). Es wirkt nur dann eine Kraft auf $v, a$ und $b$, wenn $i_v$ auf der Strecke $\overline{ab}$ liegt und wenn der Abstand zwischen $v$ und $i_v$ kleiner ist als ein Wert $\gamma$. Bertault verwendet $\gamma = 4\delta$. Das führt dazu, dass weit voneinander entfernte Knoten und Kanten keine Kraft aufeinander ausüben. Außerdem wirkt keine Kraft, wenn $v = a$ oder $v = b$ gilt. Falls diese Bedingungen erfüllt sind, gilt für die Kraft, die die Kante $(a,b)$ auf den Knoten $v$ ausübt:

$$F^{\mathrm{e}}(v,(a,b)) = \frac{(\gamma - d(v,i_v))^2}{d(v,i_v)} \cdot \overrightarrow{i_v v}$$




**(a)** $i_v$ liegt auf der Kante $(a,b)$ **(b)** $i_v$ liegt nicht auf der Kante $(a,b)$

**Abb. 2.1:** Bestimmung des Fußpunkts von $v$ auf der Geraden durch $a$ und $b$

Die resultierende Kraft für einen Knoten $v$ ergibt sich somit aus der Summe der anziehenden und abstoßenden Kräfte:

$$F(v) = \sum_{(u,v)\in E} F^{\mathrm{a}}(u,v) + \sum_{u\in V} F^{\mathrm{r}}(u,v) + \sum_{\substack{(a,b)\in E \\ a,b\neq v}} F^{\mathrm{e}}(v,(a,b)) - \sum_{\substack{u\in V \\ (v,w)\in E}} F^{\mathrm{e}}(u,(v,w))$$

Durch diese Kräfte entsteht eine gleichmäßige Verteilung der Knoten und die Kantenlängen sind ungefähr gleich den gewünschten Kantenlängen [Ber99]. Das führt dazu, dass die Zeichnung übersichtlich und gut lesbar wird. Beim Anwenden der Kräfte kann es allerdings passieren, dass neue Kantenüberschneidungen entstehen. Um das zu verhindern, müssen die Kräfte in manchen Fällen begrenzt werden, wenn sie zu Überschneidungen führen würden. Ob das nötig ist, hängt auch von der Richtung der Kräfte ab. Da es unendlich viele Richtungen für die Kräfte gibt, kann man nicht für jede Kraftrichtung eine Grenze angeben. Deshalb wird jedem Knoten $v$ eine sogenannte *Zone* $Z(v)$ zugeordnet. Diese Zone $Z(v)$ ist definiert durch acht Kreisbögen $Z_1(v), \ldots, Z_8(v)$. Die Radien

dieser Kreisbögen werden mit $R_1(v), \ldots, R_8(v)$ bezeichnet und bestimmen, wie weit ein Knoten in die entsprechende Richtung verschoben werden darf, damit sich dadurch keine Kantenüberschneidungen ergeben. Abb. 2.2 zeigt die Zone $Z(v^{(i)})$ um einen Knoten $v$ nach der $i$-ten Iteration des Algorithmus. Der Knoten wird während der $(i+1)$-ten Iteration in Richtung der Kraft $F(v^{(i)})$ bewegt, allerdings nur so weit, dass $v^{(i+1)}$ noch in der Zone liegt. Ein Kreisbogen mit unendlichem Radius ist gestrichelt gezeichnet.

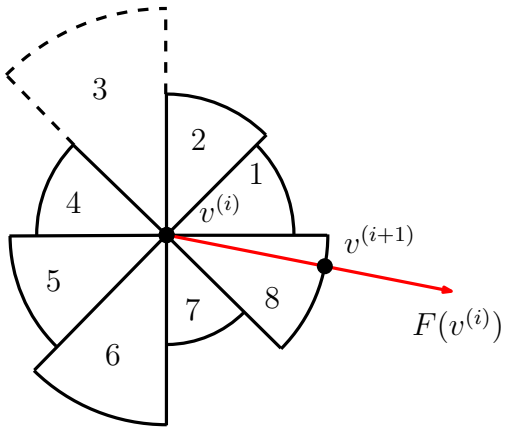



**Abb. 2.2:** Beispiel für die Zone $Z(v)$ eines Knotens $v$

Der Radius der Kreisbögen muss dabei in jeder Iteration neu berechnet werden. Das wirkt sich aber nicht negativ auf die asymptotische Laufzeit des Algorithmus aus, denn die Zeit für die Radius-Berechnungen liegt in $O(|V| \cdot |E|)$. Zum Vergleich: Die Zeit für die Berechnung der Kräfte liegt in $O(|V|^2 + |V| \cdot |E|)$, was auch die Gesamtlaufzeit für eine Iteration des Algorithmus ist.

Um Graphen noch übersichtlicher darzustellen, benutzen Finkel und Tamassia [FT05] für Kanten keine geraden Linien, sondern stellen Kanten durch Bézierkurven dar. Dadurch werden Kanten, die in den gleichen Knoten eingehen, besser unterscheidbar.

Eine *Bézierkurve* ist eine parametrisch modellierte Kurve, definiert durch einen Anfangspunkt und Endpunkt sowie beliebig viele Kontrollpunkte. Die allgemeine Definition für Punkte, die auf einer Bézierkurve mit Anfangspunkt $P_0$, Endpunkt $P_n$ und Kontrollpunkten $P_1, \ldots, P_{n-1}$ liegen, lautet:

$$B_n(t) = \sum_{k=0}^{n} \binom{n}{k} (1-t)^{n-k} t^k \cdot P_k \text{ mit } t \in [0;1]$$

Je mehr Kontrollpunkte benutzt werden, desto komplexer wird die Kurve und damit das Polynom, das die Kurve definiert. In dieser Arbeit werden kubische Bézierkurven verwendet, welche zwei Kontrollpunkte benutzen. Sie heißen kubisch, weil das die Kurve definierende Polynom ein Polynom dritten Grades ist. Die Punkte, die auf einer kubischen Bézierkurve liegen, sind folgendermaßen festgelegt:

$$B_3(t) = (1-t)^3 \cdot P_0 + 3(1-t)^2 t \cdot P_1 + 3(1-t) t^2 \cdot P_2 + t^3 \cdot P_3 \text{ mit } t \in [0;1]$$

Dabei ist $P_0$ der Anfangspunkt der Kurve und $P_3$ der Endpunkt. $P_1$ und $P_2$ sind die Kontrollpunkte. Bézierkurven haben die Eigenschaft, dass alle Punkte der Kurve innerhalb der konvexen Hülle der Anfangs-, End- und Kontrollpunkte liegen [BFK84]. Das wird später ausgenutzt, um Kantenüberschneidungen zu verhindern. Abb. 2.3 zeigt eine kubische Bézierkurve sowie deren konvexe Hülle.

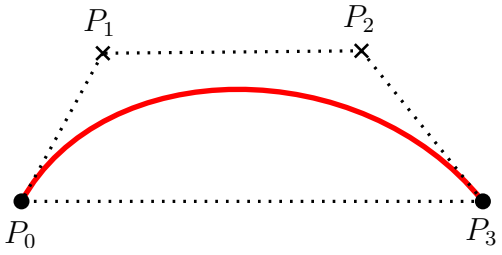



**Abb. 2.3:** Eine kubische Bézierkurve und deren konvexe Hülle (gepunktet)

Die *Winkelauflösung* in einem Knoten $v$ bezieht sich auf die Winkel zwischen zwei beliebigen Kanten, die inzident zu diesem Knoten sind. Dazu wird der kleinste Winkel $\alpha$ zwischen zwei Kanten betrachtet und mit dem optimalen kleinsten Winkel $\beta$ verglichen. Der optimale kleinste Winkel ist $\beta = 360/\deg(v)$. In diesem Fall sind alle Winkel zwischen zwei nebeneinander liegenden Kanten gleich groß und die Winkelauflösung optimal. Je kleiner die Differenz zwischen $\alpha$ und $\beta$, desto besser ist die Winkelauflösung in $v$ und desto übersichtlicher sind die Kanten an $v$ gezeichnet.

Der Algorithmus von Finkel und Tamassia funktioniert folgendermaßen: Zuerst wird in jede Kante eines Graphen $G$ ein Dummy-Knoten eingebaut, das heißt jede Kante $(a, c)$ wird durch zwei Kanten $(a, b)$ und $(b, c)$ ersetzt. Dadurch entsteht ein Graph $G'$. Nun wird mit Hilfe eines kräftebasierten Algorithmus eine geradlinige Zeichnung von $G'$ erstellt. Hier werden die Dummy-Knoten wieder entfernt, indem ein Pfad $(a, b, c)$ mit Dummy-Knoten $b$ durch eine Bézierkurve mit Endpunkten $a$ und $c$ und Kontrollpunkt $b$ ersetzt wird. Die Anzahl von Dummy-Knoten und damit die Anzahl von Kontrollpunkten pro Kante kann variiert werden, um den Kurven-Effekt zu vergrößern. In der Praxis zeigen ein oder zwei Kontrollpunkte pro Kante gute Ergebnisse.

Bei dieser Methode können allerdings zusätzliche Kantenüberschneidungen entstehen. Abb. 2.4 zeigt einen solchen Fall. Um das zu verhindern, wird eine Technik namens *Bindung* von Brandes und Wagner [BW00] verwendet. Dabei werden verborgene Kanten eingeführt. Diese verbinden die Kontrollpunkte aller Kanten, die in einen Knoten eingehen, in der Reihenfolge miteinander, wie die Endpunkte der Kanten um den Knoten herum liegen (Abb. 2.5). Diese verborgenen Kanten werden nicht gezeichnet, aber sie sorgen dafür, dass die Kontrollpunkte verschoben werden und verhindern so viele Kantenüberschneidungen. Um diese Technik anzuwenden, muss der Algorithmus erst einmal angewandt werden und anschließend mit den verborgenen Kanten wiederholt werden. Es zeigte sich, dass bei der Benutzung von kubischen Bézierkurven die Bindung von Kontrollpunkten eine sehr effektive Technik ist. Bei quadratischen Bézierkurven hinge-

gen sind die Bindungen nicht sehr effektiv. Zum Testen wurden die „Rome-Graphen“[1] benutzt.

Insgesamt konnten Finkel und Tamassia durch die Einführung von gekrümmten Kanten und den Bindungen die Winkelauflösung um 46% gegenüber geradlinigen Kanten verbessern und die Anzahl von Kantenüberschneidungen um 6% reduzieren.

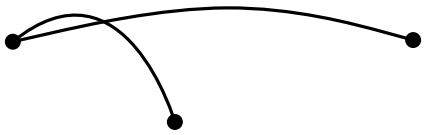


**Abb. 2.4:** Kantenüberschneidung, die durch Bindungen verhindert werden kann

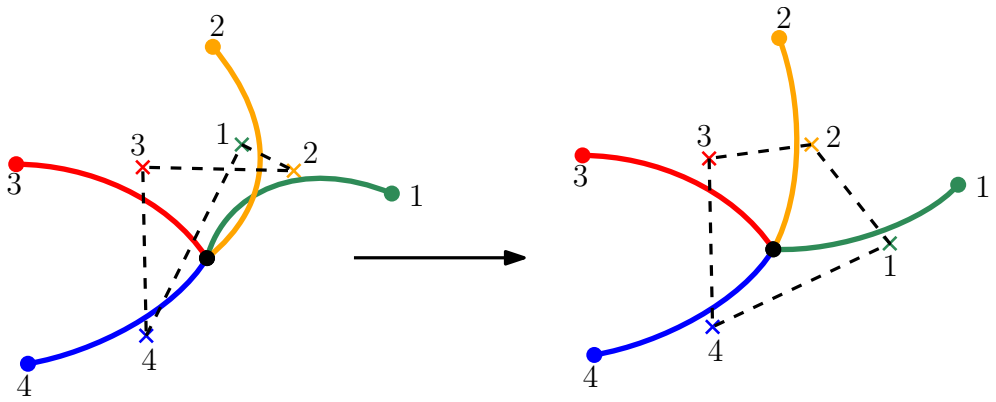



**Abb. 2.5:** Der Effekt von Bindungen (Kreuze entsprechen den Kontrollpunkten)

Auch das spezielle Problem des automatischen Zeichnens von U-Bahn-Linienplänen ist schon untersucht worden. Hong et al. [HMdN06] verwenden dazu etwa eine Variation des Algorithmus von Eades. Stott und Rodgers [SRMW11] benutzen einen Hillclimbing-Algorithmus, der nach festgelegten ästhetischen Kriterien einen möglichst schönen U-Bahn-Linienplan erstellt. Nöllenburg und Wolff [NW11] verwenden zum automatischen Zeichnen ein gemischt ganzzahliges Programm. Hier wird eine Menge von harten Bedingungen festgelegt, die in der Ausgabe alle zwingend erfüllt sein müssen. Außerdem werden weiche Kriterien aufgestellt, nach denen die Ausgabe optimiert werden soll. So entsteht eine Zeichnung, die einerseits alle harten Bedingungen erfüllt und andererseits optimal im Sinne der weichen Kriterien ist. Das Programm produziert oktilineare Zeichnungen, wobei die Knoten auf dem ganzzahligen Gitter liegen. Die Laufzeit ist aufgrund der Komplexität des Problems recht hoch. Für London benötigt der Algorithmus ungefähr eine Nacht bis die Zeichnung erstellt ist. Die Ausgaben dieses Programms werden in der vorliegenden Arbeit in modifizierter Form als Eingaben verwendet.

[1] verfügbar auf http://graphdrawing.org

# 3 Zeichen-Algorithmus

Zum automatischen Zeichnen der U-Bahn-Linienpläne wird ein Algorithmus entwickelt, der auf den bekannten kräftebasierten Verfahren aufbaut. Dadurch, dass aber auf jeder Kante des Graphen mehrere U-Bahn-Linien verlaufen können und die Kanten durch Bézierkurven dargestellt werden sollen, muss das Verfahren noch an einigen Stellen erweitert werden.

So reicht es nicht aus, den Graph als einfache Mengen von Knoten und Kanten zu speichern, da noch viel zusätzliche Information benötigt wird. Außerdem ist das Einlesen des Graphen wesentlich komplexer, da hierbei schon eine rudimentäre Zeichnung mit Bézierkurven erstellt wird. Dabei muss vor allem darauf geachtet werden, dass bei der Umwandlung einer oktilinearen Zeichnung keine Kantenüberschneidungen entstehen.

Da anders als bei Finkel und Tamassia die Kanten tatsächlich aus Bézierkurven bestehen und die Kurven nicht nur durch Dummy-Knoten simuliert werden, sind zusätzliche Kräfte nötig. Diese wirken vor allem auf die Kontrollpunkte der Kurven, damit diese eine möglichst schöne Form erhalten.

Darüber hinaus soll im Verlauf des Algorithmus darauf geachtet werden, dass, ähnlich wie bei Bertault, keine Kantenüberschneidungen entstehen. Das von Bertault entwickelte Verfahren zum Vermeiden von Kantenüberschneidungen kann hier jedoch nicht angewandt werden, da es auf geradlinige Zeichnungen ausgelegt ist. Um Überschneidungsfreiheit zu garantieren, wird, nachdem in einer Iteration die Kräfte angewandt wurden, die Zeichnung auf Kantenüberschneidungen getestet. Sollten hierbei Überschneidungen aufgetreten sein, dann werden die Beträge der dafür verantwortlichen Kräfte so lange reduziert, bis sich die Überschneidungen auflösen.

Am Ende jeder Iteration soll dann noch getestet werden, ob sich mehrere Bézierkurven zu einer kombinieren lassen, um längere und damit schönere Kurven zu erhalten. Auch hierbei sollen natürlich keine Überschneidungen entstehen und die Einbettung des Graphen soll nicht verändert werden.

Algorithmus 1 zeigt das grobe Vorgehen in Pseudocode. Die angesprochenen Erweiterungen werden in den nächsten Kapiteln näher erläutert, während der Algorithmus genauer vorgestellt wird.

## 3.1 Repräsentation des Graphen

Wenn eine kräftebasierte Methode auf einen Graphen mit geradliniger Zeichnung angewandt wird, müssen nur wenige Informationen gespeichert werden. Es ist lediglich nötig, die Struktur des Graphen zu speichern und für jeden Knoten die aktuelle Position zu sichern. Diese Informationen reichen aus, um in jeder Iteration die Kräfte zu berechnen

**Algorithmus 1:** Automatisches Zeichnen von U-Bahn-Linienplänen unter Verwendung von Bézierkurven

**Eingabe** : oktilineare Zeichnung $Z$, maximale Anzahl $k$ von Iterationen
**Ausgabe** : Zeichnung $Z'$ mit Bézierkurven

```
1  lies die oktilineare Zeichnung Z ein und wandle sie in eine Zeichnung Z' mit
   Bézierkurven um
2  i = 0
3  while i < k do
4      berechne die Kräfte für Z'
5      wende die Kräfte auf Z' an
6      if es existieren Kantenüberschneidungen then
7          reduziere die Kräfte, bis sich die Überschneidungen auflösen
8      if es ist möglich Kanten zusammenzulegen ohne dass Überschneidungen
       entstehen then
9          entferne die entsprechenden Knoten und lege die Kanten zusammen
10     i = i + 1
11 return Z'
```

und gegebenenfalls den Graphen zu zeichnen. In dieser Arbeit muss jedoch wesentlich mehr Information über den Graphen gespeichert werden.

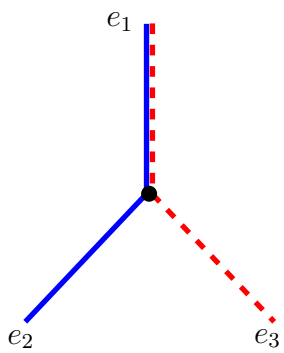
Das liegt zum einen daran, dass die Grundstruktur des Graphen lediglich das Schienennetz der U-Bahn angibt. Da aber in vielen Fällen mehrere unterschiedliche U-Bahn-Linien die gleichen Schienen benutzen, muss für jede Kante gespeichert werden, welche Linien darauf verlaufen.

Zum anderen werden Kanten durch Bézierkurven dargestellt. Das führt dazu, dass es nicht ausreicht, die beiden Endpunkte jeder Kante zu speichern. Es müssen zusätzlich die Kontrollpunkte der Kurven gesichert werden. Da in dieser Arbeit kubische Bézierkurven verwendet werden, hat jede Kante zwei Kontrollpunkte, wobei jeder Kontrollpunkt zu einem Endpunkt gehört.

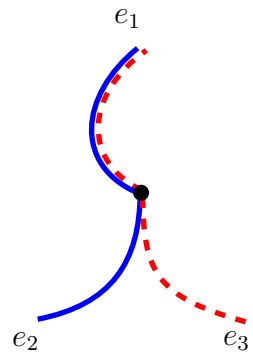
In vielen Fällen können die Kontrollpunkte der Kanten nicht beliebig gesetzt werden. Denn dann können im Wesentlichen zwei Probleme auftreten.

**Problem 1:** Linien machen in einer Station einen Knick, wodurch es für den Betrachter schwerer wird, dem Verlauf einer Linie zu folgen. Man betrachte dazu einen Fall wie in Abb. 3.1, wobei hier nur die Kontrollpunkte interessieren, die zu dem gezeichneten Knoten gehören. Abb. 3.1a zeigt die oktilineare Eingabe. Wenn die Kontrollpunkte beliebig gesetzt werden dürften, könnte das dazu führen, dass die Linien in der Station einen Knick machen (Abb. 3.1b). Deshalb müssen in diesem Fall die Kontrollpunkte von $e_2$ und $e_3$ beide auf der Gerade durch den Knoten und den Kontrollpunkt von $e_1$ liegen (Abb. 3.1c). Der Abstand der Kontrollpunkte vom Knoten ist jedoch nicht eingeschränkt (Abb. 3.1d).

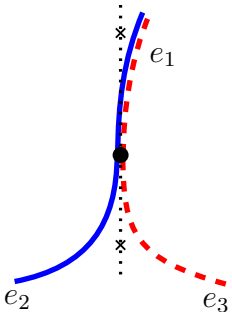
**Problem 2:** Es entstehen unnötige Kantenüberschneidungen. Abb. 3.2 zeigt einen



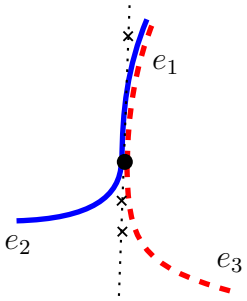

**(a)** Die oktilineare Eingabe




**(b)** Falsch gesetzte Kontrollpunkte




**(c)** Korrekt gesetzte Kontrollpunkte mit gleichem Abstand




**(d)** Korrekt gesetzte Kontrollpunkte mit unterschiedlichem Abstand

**Abb. 3.1:** Beispielfall für abknickende Linien

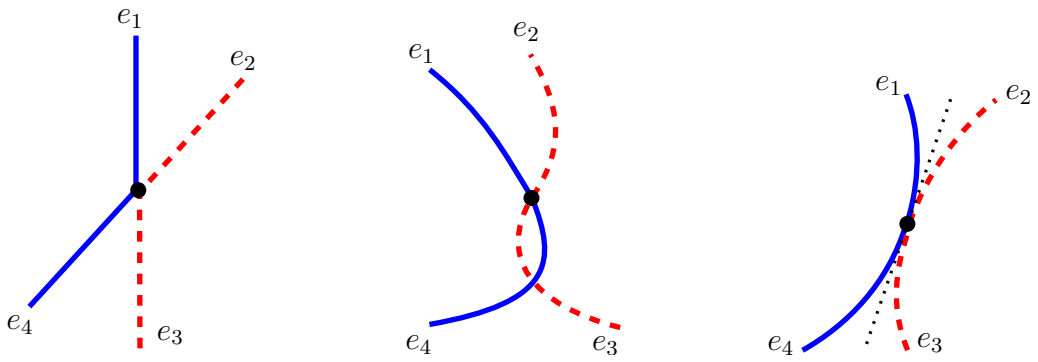
solchen Fall. Auch dieses Problem kann man verhindern, wenn man die Kontrollpunkte korrekt setzt. In Abb. 3.2b gibt es keine Einschränkungen für das Setzen der Kontrollpunkte, wodurch es zu einer Überschneidung der Kanten $e_3$ und $e_4$ kommt. Um das zu verhindern, müssen die Kontrollpunkte von $e_3$ und $e_4$ auf einer Gerade mit dem Knoten liegen. Das gleiche gilt natürlich auch für $e_1$ und $e_2$. In Abb. 3.2c sind die Kontrollpunkte richtig gesetzt, um Überschneidungen zu verhindern. Die gepunktete Linie zeigt die Gerade, auf der alle Kontrollpunkte liegen.

Beide Probleme lassen sich lösen, indem man allgemeine Bedingungen für das Setzen der Kontrollpunkte festlegt:

1. Die Kontrollpunkte zweier Kanten, die mindestens eine gemeinsame Linie haben, müssen mit dem Knoten auf einer Geraden liegen.

2. Wenn sich zwei Linien in einem Knoten nicht kreuzen, müssen die Kontrollpunkte von allen beteiligten Kanten mit dem Knoten auf einer Gerade liegen.

Um diese Bedingungen einzuhalten und im Graphen abzuspeichern, erhält jede Linie in jedem Knoten eine Tangente, die angibt, auf welcher Gerade die Kontrollpunkte liegen müssen. Dadurch ist die erste Bedingung erfüllt und das Abknicken einer Linie in einem




**(a)** Die oktilineare Eingabe

**(b)** Falsch gesetzte Kontrollpunkte

**(c)** Korrekt gesetzte Kontrollpunkte

**Abb. 3.2:** Beispielfall für unnötige Kantenüberschneidungen

Knoten wird verhindert. Um Kantenüberschneidungen zu verhindern und die zweite Bedingung umzusetzen, müssen sich manche Linien wie in Abb. 3.2c die gleiche Tangente teilen. Um die tatsächliche Position der Kontrollpunkte zu bestimmen, wird in den Kanten gespeichert, auf welcher Seite bezüglich der Tangentenrichtung der Kontrollpunkt liegt. Zusammen mit dem Abstand vom Knoten ist die Position des Kontrollpunktes dann eindeutig festgelegt.

Um gleichmäßig gebogene Bézierkurven zu erhalten, werden im Verlauf des Algorithmus außerdem mehrere aufeinander folgende Kanten zu einer Kante zusammengefasst. Deshalb ist es nötig, alle Stationen zu speichern, die dann auf einer Kante liegen. Diese Stationen sind dann keine Knoten im Graph mehr, weshalb für sie auch keine Kräfte mehr berechnet werden. Sie werden aber gebraucht, um die optimale Kantenlänge zu berechnen, denn je mehr Stationen auf einer Kante liegen, desto länger soll diese Kante gezeichnet werden.

Abschließend geben wir noch einmal eine Übersicht darüber, welche zusätzliche Information im Vergleich zu kräftebasierten Verfahren für geradlinige Zeichnungen gespeichert werden muss:

- In jedem Knoten wird für jede Linie eine Tangente gespeichert, wobei sich mehrere Linien dieselbe Tangente teilen können.
- In jeder Kante wird für jeden Endpunkt gespeichert, in welcher Richtung bezüglich der Tangentenrichtung die Kante den Knoten verlässt und welchen Abstand die Kontrollpunkte vom Knoten haben.
- Es wird auf jeder Kante gespeichert, welche Linien darauf verlaufen und welche Stationen auf der Kante liegen.

## 3.2 Einlesen der oktilinearen Zeichnung

Die Eingabe für den Algorithmus ist eine oktilineare Zeichnung des U-Bahn-Linienplans, welcher mit Hilfe eines gemischt ganzzahligen Programms von Nöllenburg und Wolff [NW11] automatisch erstellt wurde. In dieser Zeichnung besteht jede Kante aus einer Strecke, d.h. Linien können nur an einem Knoten abknicken, und die Knoten liegen auf dem ganzzahligen Gitter. Diese Zeichnung wird bereits beim Einlesen in eine Zeichnung mit Bézierkurven transformiert. Dabei soll verhindert werden, dass sich Kanten überschneiden. Für zwei nicht inzidente Kanten ist diese Bedingung immer erfüllt, wenn die Kontrollpunkte korrekt gesetzt werden.

**Satz 1.** *Sei $Z$ eine oktilineare Zeichnung eines Graphen $G$ auf dem Gitter. Dann existiert eine Zeichnung $Z'$ von $G$ mit folgenden Eigenschaften:*

- *Alle Kanten werden durch kubische Bézierkurven dargestellt.*
- *Es existiert kein Paar von nicht inzidenten Kanten $e$ und $f$, so dass sich $e$ und $f$ überschneiden.*

*Beweis.* Seien $e$ und $f$ zwei beliebige, nicht inzidente Kanten in $G$. Die Kontrollpunkte für die Kanten in $Z'$ seien alle so gesetzt, dass sie einen Abstand von 0,3 vom entsprechenden Knoten haben. Dann ist jeder Punkt der konvexen Hülle von Kontrollpunkten und Endpunkten von $e$ in $Z'$ höchstens 0,3 von der Zeichnung von $e$ in $Z$ entfernt. Das gleiche gilt für Punkte der konvexen Hülle von Kontrollpunkten und Endpunkten von $f$. Da $Z$ eine Zeichnung auf dem Gitter ist, haben $e$ und $f$ in $Z$ mindestens einen Abstand von $\frac{\sqrt{2}}{2} \approx 0,707$. Die konvexen Hüllen von Kontrollpunkten und Endpunkten von $e$ und $f$ überschneiden sich folglich nicht. Weil Bézierkurven immer komplett innerhalb oder auf dem Rand der konvexen Hülle von Kontrollpunkten und Endpunkten liegen, existiert auch in $Z'$ keine Überschneidung von $e$ und $f$. □

Für zwei inzidente Kanten lässt sich keine solche Aussage treffen. In den allermeisten Fällen ist es möglich, eine Anordnung der Kanten zu finden, so dass sich keine Kanten überschneiden, auch für sehr komplexe Fälle wie in Abb. 3.3. Theoretisch kann es aber vorkommen, dass es unmöglich ist, Kantenüberschneidungen an einem Knoten zu verhindern. Abb. 3.4 zeigt einen solchen Fall. Ohne die blaue Linie ist es noch möglich, mit ihr ist es aber unmöglich, weil entweder eine Überschneidung mit der orangen Linie entsteht (gepunktet) oder die blaue und grüne Linie unterschiedlich verlaufen, obwohl sie auf einer Kante liegen sollten (gestrichelt).

### Kantenannotationen

Damit das Setzen der Tangenten auch in komplexen Knoten funktioniert, sind die Tangenten und die Richtungen, in der die entsprechenden Kanten den Knoten verlassen, Teil der Eingabe. Dazu kann jede Kante an beiden Endpunkten annotiert werden. Das heißt für jeden Endpunkt der Kante wird angegeben, mit welcher anderen Kante (die natürlich inzident zum gleichen Knoten sein muss) sie sich eine Tangente in diesem gemeinsamen

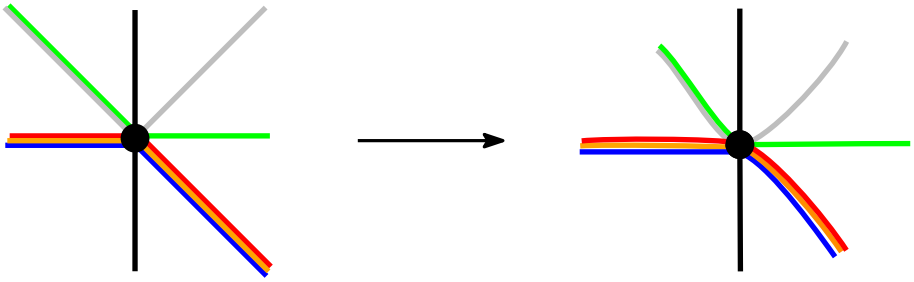


**Abb. 3.3:** Trotz vieler unterschiedlicher Linien ist es möglich, die Kanten ohne Überschneidungen zu zeichnen

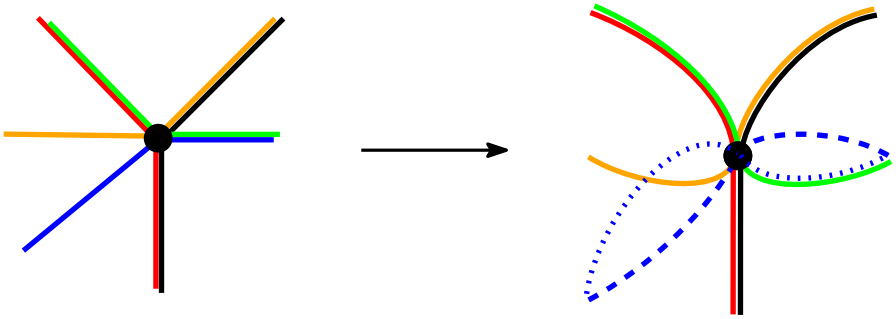


**Abb. 3.4:** Die Kanten können nicht überschneidungsfrei gezeichnet werden

Knoten teilen soll. Außerdem muss dann angegeben werden, ob die Kante den Knoten in derselben Richtung verlässt wie die referenzierte Kante oder in der entgegengesetzten Richtung. Kanten, die keine Annotation haben, erhalten eine eigene Tangente.

Abb. 3.5 zeigt ein Beispiel für die Annotationen. Die Kanten $e_2$ und $e_7$ haben keine Annotation, das heißt beide legen eine eigene Tangente fest. Die Kante $e_4$ referenziert $e_7$, sie hat also die gleiche Tangente wie $e_7$. Das „diff.“ (different) gibt an, dass $e_4$ den Knoten in der anderen Richtung als $e_7$ verlässt. Die Annotationen der übrigen Kanten funktionieren genauso. Sie referenzieren jeweils $e_2$, wobei $e_1$ und $e_3$ den Knoten in derselben Richtung („same“) verlassen wie $e_2$, der Rest verlässt den Knoten in der anderen Richtung.

Grundsätzlich müssen alle Kanten annotiert werden. In eindeutigen Fällen können die Annotationen jedoch weggelassen werden. Abb. 3.6 zeigt, wie die Kontrollpunkte in diesen Fällen automatisch gesetzt werden. Kanten an Grad-2-Knoten müssen beispielsweise nicht annotiert werden (3.6a). Auch wenn sich zwei Linien in einem Knoten schneiden (Abb. 3.6b) oder sich eine Linie aufteilt (Abb. 3.6c) ist es nicht nötig, die Kanten zu annotieren.

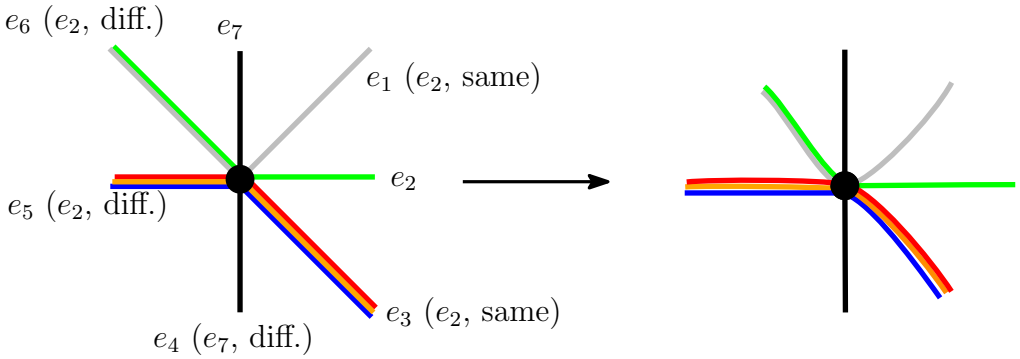



**Abb. 3.5:** Beispiel für die Annotation von Kanten

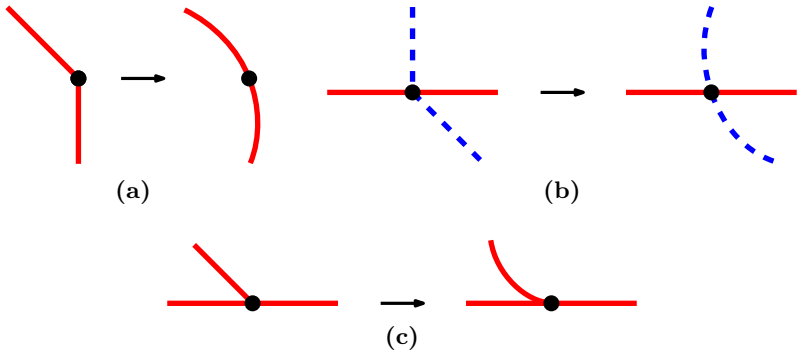



**Abb. 3.6:** Beispiele, in denen keine Annotationen gesetzt wurden

## 3.3 Die Kräfte

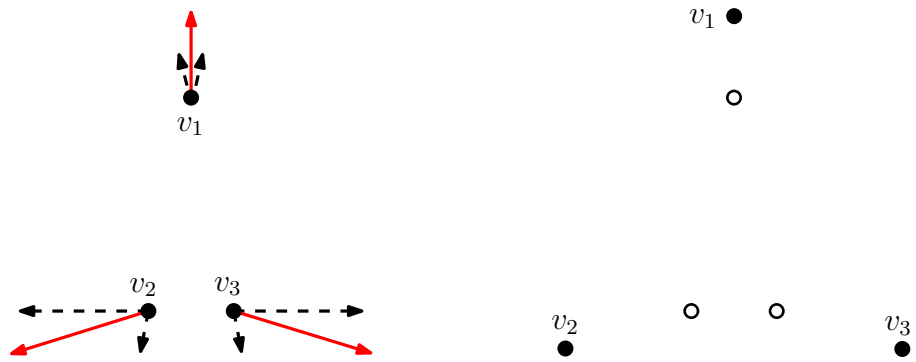
Der zentrale Punkt in einem kräftebasierten Algorithmus ist die Definition der Kräfte. Dieses Kapitel beschreibt nun die unterschiedlichen Arten der Kräfte in dieser Arbeit, die auf die Knoten und Kanten des Graphen wirken, sowie die Auswirkungen dieser Kräfte auf die endgültige Zeichnung.

### 3.3.1 Abstoßende Kraft für Knoten

Eine sehr wichtige Kraft für den Algorithmus ist die abstoßende Kraft für jedes Paar von Knoten. Diese Kraft wird in nahezu jedem kräftebasierten Modell verwendet. Sie sorgt dafür, dass sich die Knoten in der ausgegebenen Zeichnung möglichst gleichmäßig auf der Zeichenfläche verteilen und ist damit besonders wichtig für die Übersichtlichkeit der Zeichnung. Die Kraft ist dabei so definiert, dass die Abstoßung von zwei Knoten schwächer wird, je weiter die Knoten voneinander entfernt sind. Das führt einerseits dazu, dass sich zwei eng beieinander liegende Knoten stark abstoßen, wodurch vermieden wird, dass zu viele Knoten zu nahe beisammen liegen. Andererseits hat es zur Folge, dass sich weit entfernte Knoten nur sehr schwach abstoßen, wodurch diese von anderen Kräften

stärker beeinflusst werden können. Außerdem existiert für den Abstand eine Schranke, ab der keine Kraft mehr wirkt, um die Beeinflussung von sehr weit entfernten Knoten noch weiter zu reduzieren.

Die Auswirkungen dieser Kraft zeigt Abb. 3.7a. Die gestrichelten Pfeile zeigen die Kräfte, die auf die Paare von Knoten wirken. Wie man sieht, sind die Kräfte, die die Paare $v_1, v_2$ und $v_1, v_3$ aufeinander ausüben, deutlich kleiner als die Kräfte zwischen $v_2$ und $v_3$, da die Knoten $v_2$ und $v_3$ näher beieinander liegen. Die durchgängig gezeichneten Pfeile zeigen dann die addierten Kräfte, die insgesamt auf die Knoten wirken. Abb. 3.7b zeigt die Situation, nachdem die Kräfte angewandt wurden, wodurch die Knoten nun weiter voneinander entfernt liegen.




**(a)** Knoten vor Anwendung der Kräfte **(b)** Knoten nach Anwendung der Kräfte

**Abb. 3.7:** Einfaches Beispiel für die abstoßenden Kräfte auf Paare von Knoten

Um die Länge der Kanten regulieren zu können, wird für jede Kante $(u, v)$ eine gewünschte Kantenlänge $\delta(u, v)$ festgelegt. Damit alle Kanten möglichst gleich lang gezeichnet werden, wird für jede Kante die gleiche optimale Kantenlänge $l_{\text{unit}}$ festgelegt. Damit die Kanten ungefähr so lang gezeichnet werden wie in der Eingabe, ist $l_{\text{unit}}$ definiert als die durchschnittliche Kantenlänge im Eingabegraph.

Sei $\{u, v\}$ ein beliebiges Knotenpaar. Der gewünschte Abstand von $u$ und $v$ sei $\delta(u, v)$ und der tatsächliche Abstand sei $d(u, v)$. Wenn $u$ und $v$ benachbart sind, dann sei $k_{uv}$ die Anzahl von Stationen, die auf der Kante $(u, v)$ gespeichert sind. Ansonsten sei $k_{uv} = 0$. Dann gilt $\delta(u, v) = (k_{uv} + 1) \cdot l_{\text{unit}}$. Dann lautet die abstoßende Kraft, die von einem Knoten $u$ auf den Knoten $v$ wirkt, so wie bereits von Bertault [Ber99] definiert (erweitert um eine obere Schranke $c \in \mathbb{R}^+$ für den Abstand):

$$F^{\text{r}}(u, v) = \begin{cases} \frac{\delta(u,v)^2}{d(u,v)^2} \cdot \overrightarrow{uv}, & \text{wenn } d(u, v) < c \\ 0, & \text{sonst} \end{cases}$$

### 3.3.2 Anziehende Kraft für benachbarte Knoten

Die abstoßenden Kräfte zwischen den Knoten sorgen dafür, dass sich die Knoten voneinander entfernen. Damit die Ausgabe nicht zu viel Platz benötigt, wird eine Kraft eingeführt, die die Knoten und somit den Graph beisammen hält. Außerdem sollten benachbarte Knoten immer relativ nah nebeneinander gezeichnet werden, damit der Betrachter alle von einem Knoten aus direkt erreichbaren Nachbarknoten leicht erkennen kann.

Diese zwei Ziele erfüllt eine Kraft, die nur zwischen benachbarten Knoten wirkt und durch die sich benachbarte Knoten gegenseitig anziehen. Die Kraft ist umso stärker, je größer die aktuelle Kantenlänge im Vergleich zur gewünschten Kantenlänge ist.

In Abb. 3.8 sieht man die Auswirkungen der anziehenden Kräfte. Dabei entsprechen die Pfeile auf den Kanten den einzelnen Kräften auf benachbarte Knoten und die roten Pfeile zeigen die addierten Kräfte und somit die Auswirkungen auf die Knoten. Isoliert betrachtet macht diese Kraft keinen Sinn, weil sich dadurch jeder zusammenhängende Graph immer mehr zusammenziehen würde. Gemeinsam mit der abstoßenden Kraft für Knoten ergibt sich aber über mehrere Iterationen ein Kräftegleichgewicht, das zu einer schönen Zeichnung führt (siehe Abb. 3.9).

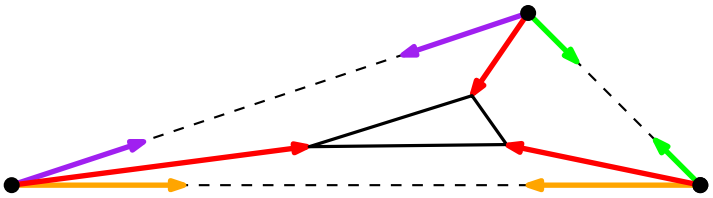
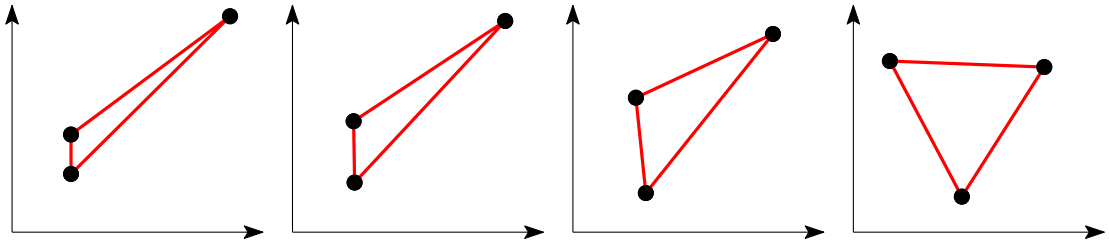


**Abb. 3.8:** Beispiel für die anziehenden Kräfte von benachbarten Knoten



**(a)** Die Eingabe **(b)** Nach einer Iteration **(c)** Nach fünf Iterationen **(d)** Nach 30 Iterationen

**Abb. 3.9:** Auswirkungen der abstoßenden und anziehenden Kräfte

Die anziehende Kraft, die von einem Knoten $u$ auf einen Knoten $v$ wirkt, ist folgendermaßen definiert:

$$F^{\mathrm{a}}(u,v) = \begin{cases} \frac{d(u,v)}{\delta(u,v)} \cdot \overrightarrow{vu}, & \text{wenn } u, v \text{ benachbart} \\ 0, & \text{sonst} \end{cases}$$

Die Stärke der Kraft ist so gewählt, dass sich die anziehenden und abstoßenden Kräfte genau ausgleichen, wenn eine Kante exakt die gewünschte Länge hat. Denn für $\delta(u,v) = d(u,v)$ gilt:

$$F^{\mathrm{r}}(u,v) + F^{\mathrm{a}}(u,v) = \frac{\delta(u,v)^2}{d(u,v)^2} \cdot \overrightarrow{uv} + \frac{d(u,v)}{\delta(u,v)} \cdot \overrightarrow{vu} = \overrightarrow{uv} + \overrightarrow{vu} = 0$$

Abb. 3.10 zeigt das Verhältnis von anziehender und abstoßender Kraft für eine gewünschte Kantenlänge von $\delta(u,v) = 1$. Für $d(u,v) < 1$ liegen die Knoten zu nahe beisammen und es überwiegt die abstoßende Kraft $F^{\mathrm{r}}(u,v)$, wodurch sich die Knoten voneinander entfernen. Für $d(u,v) > 1$ sind die Knoten zu weit entfernt, es überwiegt die anziehende Kraft $F^{\mathrm{a}}(u,v)$ und die Knoten ziehen sich gegenseitig an.

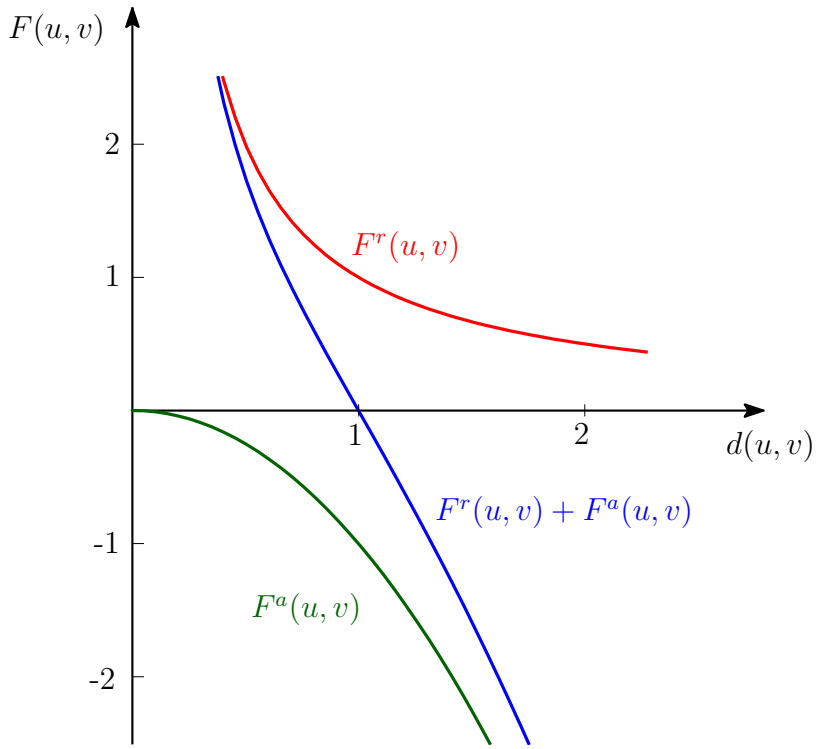



**Abb. 3.10:** Vergleich von anziehenden und abstoßenden Kräften für eine gewünschte Kantenlänge von $\delta(u,v) = 1$ und eine obere Schranke für die Kraft $F^{\mathrm{r}}(u,v)$ von $c \gg 2$.

### 3.3.3 Kraft zu der Ursprungsposition des Knotens

Da in dieser Arbeit die Knoten reale Orte repräsentieren, ist es wünschenswert, dass alle Stationen möglichst nahe an ihrer tatsächlichen Position gezeichnet werden. Diese Positionen sind jedoch nicht verfügbar, da die Eingabe bereits ein U-Bahn-Linienplan ist, in dem die Stationen nicht mehr an ihrer echten Position liegen. Deshalb wird ersatzweise

eine Kraft eingesetzt, die alle Knoten an ihre Position in der Eingabe zieht. Diese Kraft wird größer, je weiter ein Knoten von seiner Ursprungsposition entfernt ist.

Eine Sonderrolle nehmen hierbei Knoten mit Grad 1 ein. Diese liegen in den meisten Fällen am äußeren Rand des Graphen, wo die relative Position zu den anderen Knoten nicht so wichtig ist. Deshalb wird bei diesen Knoten die Kraft zur Ursprungsposition reduziert, weil mehr Wert auf eine schöne Zeichnung gelegt werden soll.

Ein einfaches Beispiel für diese Kraft zeigt Abb. 3.11. Die Kreise entsprechen den Ursprungspositionen der Knoten und die roten Pfeile zeigen die Kräfte, die zur Ursprungsposition wirken.

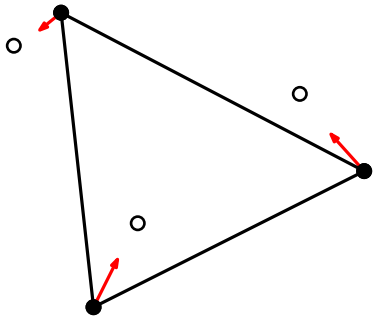
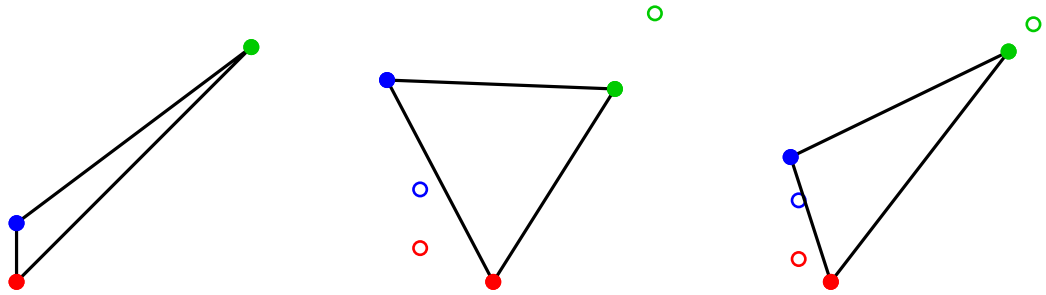


**Abb. 3.11:** Beispiel für die Kräfte zu den Ursprungspositionen

In Abb. 3.12 sieht man, wie sich die Ausgabe des Algorithmus verändert, wenn man die Kraft zu Ursprungsposition hinzunimmt. Die Eingabe ist in Abb. 3.12a zu sehen. Abb. 3.12b zeigt die Situation nach 30 Iterationen ohne die Ursprungskraft. Hier ist ein annähernd gleichseitiges Dreieck entstanden, was aufgrund der Graphstruktur auch zu erwarten war. Anders sieht das Ergebnis aus, wenn man die Ursprungskraft mit hinzunimmt (Abb. 3.12c). Hier ist die Ausgabezeichnung noch deutlich näher an der Eingabe, weil die Knoten in der Nähe ihrer Eingabeposition gehalten werden.



**(a)** Die Eingabe

**(b)** 30 Iterationen ohne Ursprungskraft

**(c)** 30 Iterationen mit Ursprungskraft

**Abb. 3.12:** Auswirkungen der Kraft zur Ursprungsposition

Die Ursprungskraft ist wie folgt definiert. Sei $u$ ein Knoten und $u_{\text{org}}$ die Position von $u$ in der Eingabe. Sei außerdem $\alpha$ der Faktor, um den die Ursprungskraft für Grad-1-

Knoten reduziert wird (In der Praxis hat sich $\alpha = \frac{1}{50}$ als sinnvoll erwiesen). Dann lautet die Ursprungskraft, die auf $u$ wirkt:

$$F^{\mathrm{o}}(u) = \begin{cases} d(u, u_{\mathrm{org}}) \cdot \overrightarrow{uu_{\mathrm{org}}}, & \text{wenn } \deg(u) \geq 2 \\ \alpha \cdot d(u, u_{\mathrm{org}}) \cdot \overrightarrow{uu_{\mathrm{org}}}, & \text{sonst} \end{cases}$$

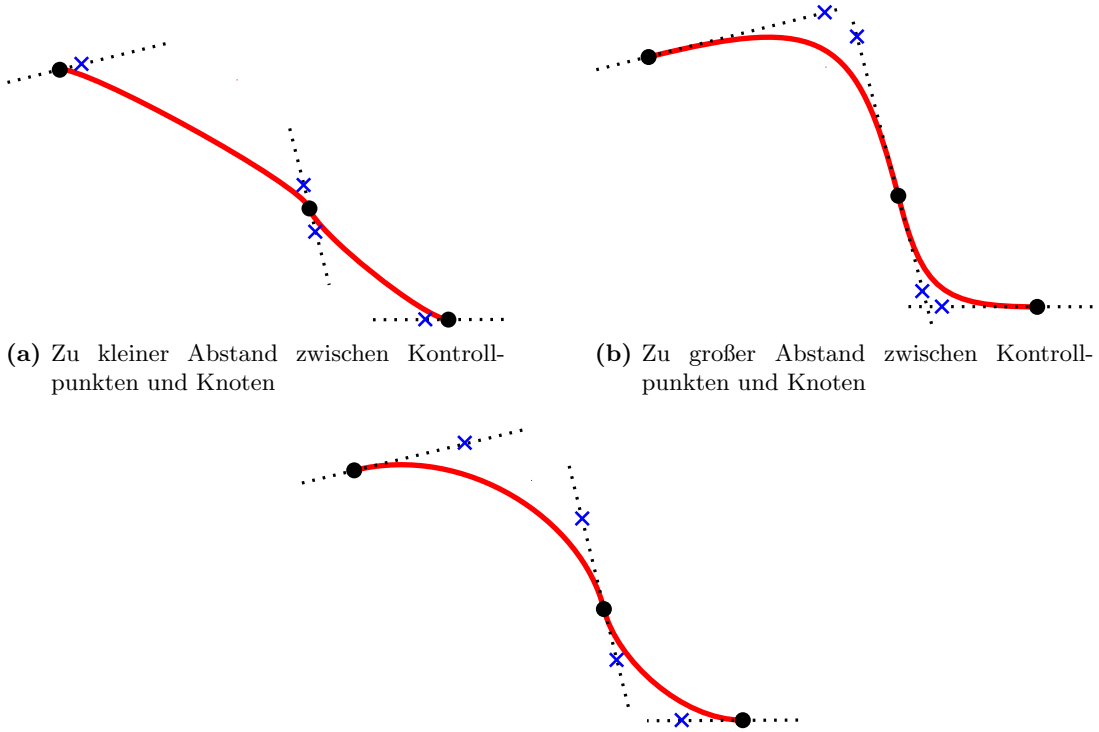
### 3.3.4 Kraft auf die Kontrollpunkte

Die ersten beiden vorgestellten Kräfte gehören zu den Standard-Kräften in kräftebasierten Algorithmen. Da in dieser Arbeit allerdings die Kanten durch Bézierkurven dargestellt werden, sind nun noch einige speziell dafür entwickelte Kräfte nötig. So wirkt auf die Kontrollpunkte der Kanten eine Kraft, die dafür sorgt, dass der Abstand von Kontrollpunkt und dazugehörigem Knoten so groß ist, dass sich schöne Kurven ergeben. Denn die Position der Kontrollpunkte ist maßgeblich dafür verantwortlich, wie die Bézierkurven aussehen. Abb. 3.13 zeigt, wie die Kontrollpunkte die Form der Kurven bestimmen. Die gepunkteten Linien sind die Tangenten der Kurven in den Knoten und die blauen Kreuze entsprechen den Kontrollpunkten der Kurven. Ist der Abstand zwischen Kontrollpunkt und dazugehörigem Knoten zu klein, ähnelt die Kurve stark einer Gerade (Abb. 3.13a), ist der Abstand zu groß, dann ist die Kurve zu stark gebogen (Abb. 3.13b). In der Praxis hat sich für den Abstand zwischen Kontrollpunkt und Knoten ein Wert von ungefähr einem Drittel des Abstands der beiden Endpunkte als sinnvoll erwiesen. So entstehen „schöne“ Bézierkurven wie in Abb. 3.13c. Der optimale Abstand zwischen einem Kontrollpunkt $c_u(u, v)$ und dem entsprechenden Endpunkt $u$ lautet also: $\delta(c_u(u, v), u) = \frac{d(u,v)}{3}$.

Die Definition der Kraft ähnelt den Definitionen der abstoßenden und anziehenden Kräfte aus den vorherigen Kapiteln. Ein Kontrollpunkt wird also gleichzeitig von dem entsprechenden Knoten abgestoßen und angezogen. Als optimaler Abstand zwischen Kontrollpunkt und Knoten wird ein Drittel des Abstands der beiden Endpunkte gewählt. Ist dieser zu groß, wird der Kontrollpunkt insgesamt angezogen, ist er zu klein, wird er insgesamt abgestoßen. Der Unterschied zu den beiden vorherigen Kräften ist jedoch, dass die Kraft nicht zwei Dimensionen hat, sondern nur eine, da sie nur auf den Abstand der Kontrollpunkte von den zugehörigen Knoten wirkt.

Sei $(u, v)$ eine beliebige Kante und sei $c_u(u, v)$ der Kontrollpunkt der Kante $(u, v)$ auf der Seite von $u$. Dann ist die Kraft, die auf den Abstand von $c_u(u, v)$ und $u$ wirkt, wie folgt definiert:

$$\begin{aligned} F^{\mathrm{c}}(d(c_u(u, v), u)) &= \underbrace{\frac{\delta(c_u(u, v), u)^2}{d(c_u(u, v), u)}}_{\text{stößt den Kontrollpunkt ab}} - \underbrace{\frac{d(c_u(u, v), u)^2}{\delta(c_u(u, v), u)}}_{\text{zieht den Kontrollpunkt an}} \\ &= \frac{\left(\frac{d(u,v)}{3}\right)^2}{d(c_u(u, v), u)} - \frac{d(c_u(u, v), u)^2}{\frac{d(u,v)}{3}} \end{aligned}$$



**(a)** Zu kleiner Abstand zwischen Kontrollpunkten und Knoten

**(b)** Zu großer Abstand zwischen Kontrollpunkten und Knoten

**(c)** Guter Abstand zwischen Kontrollpunkten und Knoten

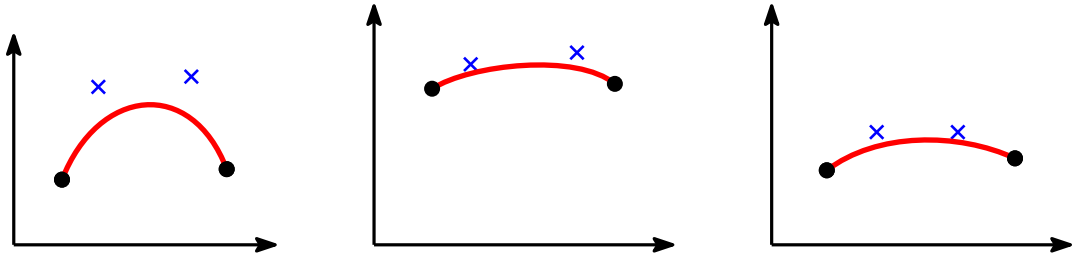
**Abb. 3.13:** Auswirkungen der Kontrollpunkte auf die Form von Bézierkurven

### 3.3.5 Kraft zum Geradedrücken der Kanten

Obwohl Kanten als Bézierkurven dargestellt werden, sollen die Kanten nicht zu stark gebogen sein, damit der der Betrachter des Plans dem Verlauf einer Linie leichter folgen kann. Um das zu erreichen, gibt es im Wesentlichen zwei Möglichkeiten. Angenommen, man hat eine stark gebogene Bézierkurve (Abb. 3.14a). Dann ist es zum einen möglich, die Knoten zu verschieben, um die Biegung zu reduzieren (Abb. 3.14b). Zum anderen kann man die Kontrollpunkte ändern (Abb. 3.14c). In dieser Arbeit werden beide Kräfte verwendet. Das hat den Vorteil, dass, wenn die Knoten durch andere Kräfte sehr stark an ihrer Position gehalten werden, die Tangenten und damit die Kontrollpunkte geändert werden können. Andererseits kann es vorkommen, dass die Tangenten nur wenig geändert werden können, um eine gute Winkelauflösung zu behalten. Dann können trotzdem noch die Knoten verschoben werden.

#### Kraft auf Knoten

Sei $(u, v)$ eine beliebige Kante. Um die Kraft auf $v$ zu berechnen, muss die „gewünschte“ Position des Knotens berechnet werden. Diese Position liegt so, dass die Kurve, die $(u, v)$ repräsentiert, eine möglichst schwache Biegung hat und liegt damit auf der zugehörigen



**(a)** Zu stark gebogene Kante

**(b)** Verschieben der Knoten

**(c)** Verschieben der Kontrollpunkte

**Abb. 3.14:** Möglichkeiten, die Biegung einer Kante zu reduzieren

Tangente. Der Abstand vom Knoten $u$ ist dabei $\delta(u, v)$ (die gewünschte Kantenlänge). Abb. 3.15 zeigt das an einem Beispiel. Dabei ist $v_g(u, v)$ die gewünschte Position von $v$ und der Pfeil zeigt die Kraft, die damit auf $v$ wirkt.

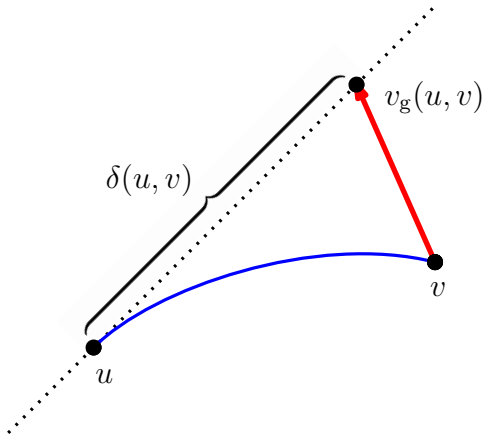


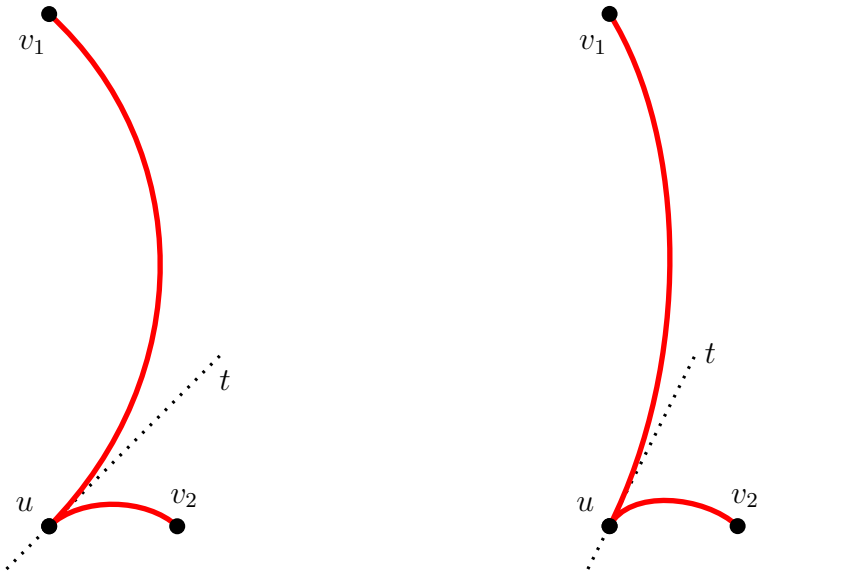

**Abb. 3.15:** Beispiel für die gewünschte Position eines Knotens $v$

Sei $(u, v)$ eine beliebige Kante und sei $v_g(u, v)$ die gewünschte Position von $v$. Dann ist $F^{sv}(u, v) = \overrightarrow{vv_g(u, v)}$ die Kraft auf $v$ zum Geradedrücken der Kante $(u, v)$. Sie ist also der Vektor von der aktuellen Position von $v$ zu der gewünschten Position $v_g(u, v)$ von $v$.

### Kraft auf Tangenten

Die Kraft auf eine Tangente ist eine Drehkraft. Es wird also ein Winkel berechnet, um den die Tangente gedreht wird. Um die Kraft von einer Kante $(u, v)$ auf die entsprechende Tangente im Knoten $u$ zu berechnen, wird der Winkel zwischen der Tangente und dem Vektor von $u$ nach $v$ berechnet. Dieser Winkel wird mit dem Abstand von $u$ zum zugehörigen Kontrollpunkt $c_u$ auf $(u, v)$ gewichtet. Die Kraft wirkt somit wie eine Art Hebel. Der Sinn dahinter ist, dass längere Kanten (deren Kontrollpunkte einen größeren Abstand von den Knoten haben) einen größeren Einfluss auf die Tangente bekommen, da

eine Tangente zu mehreren Kanten gehören kann. Abb. 3.16 zeigt ein Beispiel, bei dem zwei Kanten eine Kraft auf die selbe Tangente ausüben. In Abb. 3.16a wird keine Gewichtung vorgenommen, wodurch die Tangente zu stark in Richtung $v_2$ zeigt. Abb. 3.16b zeigt das Beispiel mit Gewichtung. Hier hat die längere Kante einen größeren Einfluss auf die Tangente, was dazu führt, dass die längere Kante keinen zu großen Bogen machen muss.




**(a)** Ergebnis ohne Gewichtung **(b)** Ergebnis mit Gewichtung

**Abb. 3.16:** Beispiel für die Gewichtung der Kräfte auf Tangenten

Seien $(u, v_1), \ldots, (u, v_l)$ die Kanten, die im Knoten $u$ die Tangente $t$ benutzen, sei $c_u(u, v_k)$ für $k = 1, \ldots, l$ der Kontrollpunkt von $(u, v_k)$ auf der Seite von $u$ und sei $\alpha_k$ der Winkel mit Vorzeichen zwischen $\overrightarrow{uv_{k_g}}$ und $\overrightarrow{uv_k}$, so dass $|\alpha| \leq 180°$. Dann lautet die Kraft $F^{\text{st}}(t, u, v_1, \ldots, v_l)$, die von $v_1, \ldots, v_l$ auf $t$ wirkt:

$$F^{\text{st}}(t, u, v_1, \ldots, v_l) = \frac{\sum_{k=1}^{l} \alpha_k \cdot d(u, c_u(u, v_k))}{\sum_{k=1}^{l} d(u, c_u(u, v_k))}$$

Abb. 3.17 veranschaulicht die Definition an einem Beispiel. Sowohl die Kante $(u, v_1)$ als auch die Kante $(u, v_2)$ üben eine Kraft auf die Tangente $t$ aus. In diesem Beispiel sind $d(u, c_{u,1}) = 1$ und $d(u, c_{u,2}) = 2$ sowie $\alpha_1 = -45°$ und $\alpha_2 = 45°$. Deshalb ist die Kraft, die die Kante $(u, v_2)$ ausübt, stärker. Die Kraft auf $t$ lautet dann:

$$F^{\text{st}}(t,u,v_1,v_2) = \frac{\sum_{k=1}^{2} \alpha_k \cdot d(u, c_u(u,v_k))}{\sum_{k=1}^{2} d(u, c_u(u,v_k))} = \frac{\alpha_1 \cdot d(u, c_u(u,v_1)) + \alpha_2 \cdot d(u, c_u(u,v_2))}{d(u, c_u(u,v_1)) + d(u, c_u(u,v_2))}$$
$$= \frac{-45° \cdot 1 + 45° \cdot 2}{1+2} = \frac{45°}{3} = 15°$$

Die Tangente $t$ würde also um 15° in Richtung von $v_2$ gedreht werden.

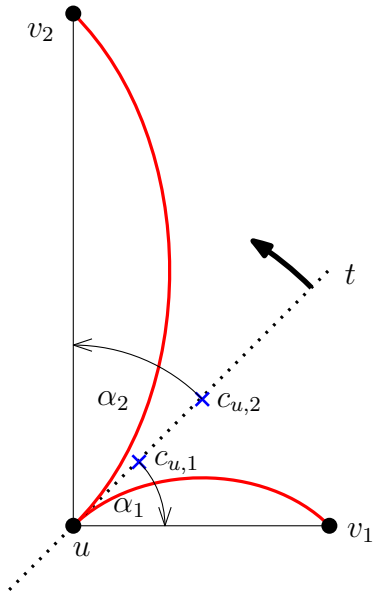


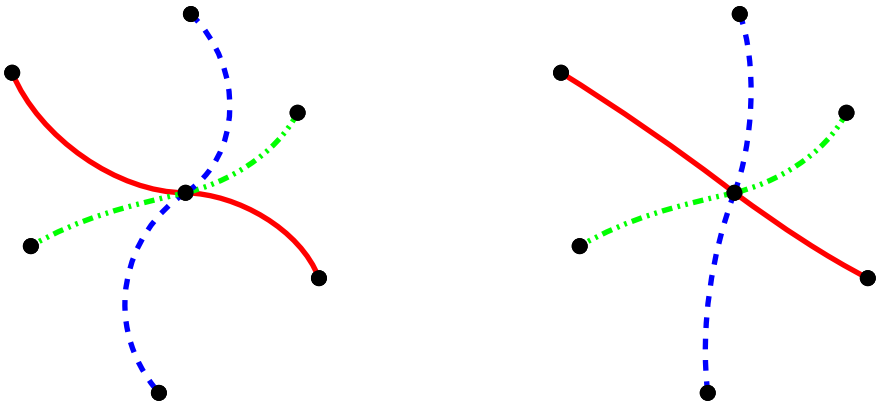

**Abb. 3.17:** Berechnung der Kraft auf eine Tangente $t$

### 3.3.6 Kraft für bessere Winkelauflösung in Knoten

Stationen mit vielen verschiedenen Linien sollen möglichst übersichtlich gezeichnet werden. Wenn in einer Station also mehrere unterschiedliche Tangenten existieren, soll der Winkel zwischen den Tangenten möglichst groß sein. Das macht die Zeichnung übersichtlicher und macht es dem Betrachter leichter, einer Linie zu folgen. Abb. 3.18 zeigt den Unterschied zwischen einer guten und einer schlechten Winkelauflösung.

Um eine gute Winkelauflösung zu erhalten, stoßen sich in einem Knoten alle Tangenten gegenseitig ab. Die abstoßende Kraft zwischen zwei Tangenten ist umso größer, je kleiner der minimale Winkel zwischen ihnen ist.

Seien $t_1$ und $t_2$ zwei Tangenten und sei $\alpha$ der Winkel mit Vorzeichen von $t_2$ nach $t_1$, so dass $|\alpha| \leq 90°$. Sei außerdem $c$ eine Konstante, die die Stärke der Kraft beeinflusst (in der Praxis hat sich $c = 300$ als sinnvoll erwiesen). Dann ist die Kraft, die von $t_2$ auf $t_1$ wirkt, folgendermaßen definiert:



**(a)** Schlechte Winkelauflösung **(b)** Gute Winkelauflösung

**Abb. 3.18:** Beispiel für unterschiedliche Winkelauflösungen

$$F^{\mathrm{rt}}(t_1, t_2) = \frac{c}{\alpha}$$

Abb. 3.19 zeigt ein Beispiel für $c = 300$. Der Winkel $\alpha$ zeigt wie in der Definition von $t_2$ nach $t_1$ und beträgt $-30°$. Die resultierende Winkeldrehung für $t_1$ lautet dann:

$$F^{\mathrm{rt}}(t_1, t_2) = \frac{300}{\alpha} = \frac{300}{-30} = -10°$$

Die Tangente $t_1$ würde also, wie in der Zeichnung zu sehen, um 10° im Uhrzeigersinn gedreht werden.

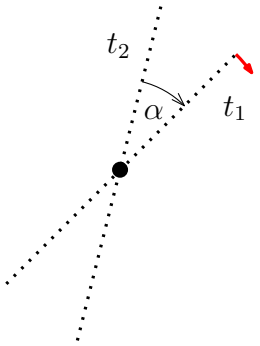



**Abb. 3.19:** Beispiel für die Kraft, die von $t_2$ auf $t_1$ wirkt

## 3.4 Begrenzen der Kräfte zum Verhindern von Überschneidungen

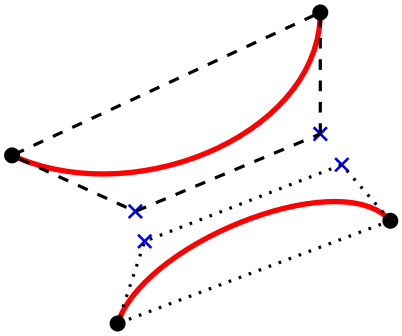
Durch die Anwendung der berechneten Kräfte kommt es häufig dazu, dass Kantenüberschneidungen entstehen. Darunter leidet die Übersichtlichkeit der Zeichnung. Außerdem kann es passieren, dass sich durch die Anwendung der Kräfte die Einbettung verändert. Das soll vermieden werden, weil sonst der Plan nicht mehr mit der realen Topologie des U-Bahn-Netzes übereinstimmt. Um diese Probleme zu verhindern, müssen die Beträge der Kräfte auf manche Knoten, Kanten und Tangenten reduziert werden oder sogar komplett auf Null gesetzt werden. Deshalb muss der Algorithmus während der Ausführung erkennen, wann es Überschneidungen oder Änderungen der Einbettung gibt, und die entsprechenden Kräfte anpassen. Das passiert folgendermaßen.

Zuerst werden alle berechneten Kräfte angewandt. Anschließend wird getestet, ob dadurch Kantenüberschneidungen entstanden sind oder sich die Einbettung geändert hat. Falls das der Fall ist, werden die Beträge der Kräfte, die auf diese Kanten angewandt wurden, so lange halbiert, bis sich die Überschneidungen auflösen und die vorherige Einbettung wiederhergestellt wurde. Das ist immer möglich, da die Zeichnung vor dieser Iteration überschneidungsfrei war.

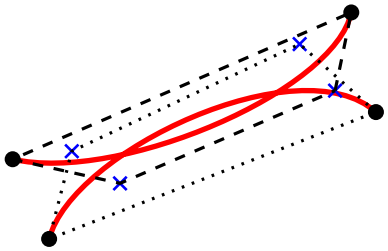
Um zwei nicht inzidente Kanten auf Überschneidungen zu testen, werden die konvexen Hüllen der beiden Kanten betrachtet. Wenn sich diese nicht überschneiden, dann überschneiden sich auch die Kanten nicht, weil Bézierkurven immer innerhalb der konvexen Hülle aus Kontrollpunkten und Endpunkten liegen. Abb. 3.20 zeigt ein Beispiel. Ein Fall wie in Abb. 3.20a ist erlaubt, da sich hier die konvexen Hüllen und damit auch die Kurven nicht überschneiden. Abb. 3.20b zeigt einen Fall, in dem sich die konvexen Hüllen und die Kurven überschneiden. Das wird im Algorithmus verhindert. Abb. 3.20c zeigt ein Beispiel, in dem sich die konvexen Hüllen überschneiden, die Kurven aber nicht. Das müsste im Algorithmus nicht verhindert werden. Da es aber einfacher ist, die konvexen Hüllen auf Überschneidungen zu testen als die tatsächlichen Bézierkurven, wird auch ein solcher Fall verhindert. Das schränkt die Flexibilität der Zeichnung zwar etwas ein, aber nicht übermäßig stark.

Für zwei inzidente Kanten kann dieses Verfahren nicht angewandt werden, da sich die konvexen Hüllen immer überschneiden, weil die Kanten einen gemeinsamen Endpunkt haben. Hier wird getestet, ob die Kanten ihre relative Position zueinander beibehalten haben oder ob Kanten über inzidente Kanten „gesprungen“ sind. Das muss verhindert werden, weil sich sonst die Einbettung ändert (Abb. 3.21). Die Situation vor Anwendung der Kräfte zeigt Abb. 3.21a. Durch die Kräfte „springt“ die Kante $e_1$ über $e_2$, wodurch sich die Einbettung ändert (Abb. 3.21b). Der Algorithmus erkennt das und reduziert die Beträge der Kräfte so lange, bis die ursprüngliche Einbettung wiederhergestellt ist (Abb. 3.21c).

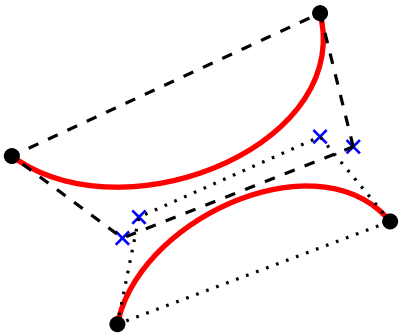
Algorithmus 2 zeigt das gesamte Vorgehen zum Verhindern von Kantenüberschneidungen und Beibehalten der Einbettung in Pseudocode.



**(a)** Keine Überschneidung der konvexen Hüllen und Kurven

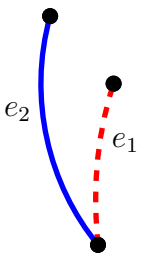


**(b)** Überschneidung der konvexen Hüllen und Kurven

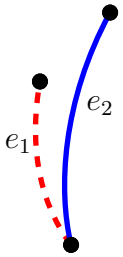


**(c)** Überschneidung der konvexen Hüllen, aber keine Überschneidung der Kurven

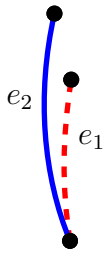
**Abb. 3.20:** Beispiele für den Test auf Überschneidungen für nicht inzidente Kanten




**(a)** Vor Anwendung der Kräfte




**(b)** Nach Anwendung der Kräfte




**(c)** Nach Anwendung der reduzierten Kräfte

**Abb. 3.21:** Beispiel für den Test zur Beibehaltung der Einbettung

## 3.5 Zusammenlegen von Bézierkurven während des Algorithmus

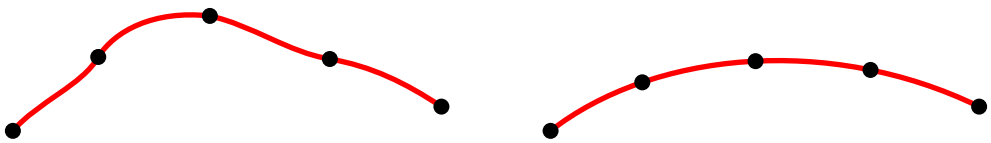
Am Ende des Algorithmus soll jede Linie möglichst einheitlich gekrümmt sein, damit die Zeichnung so übersichtlich wie möglich ist. Um das zu erreichen, werden während

**Algorithmus 2:** Verhindern von Überschneidungen und Beibehaltung der Einbettung

**Eingabe** : Bézierzeichnung $Z$, Kräfte für Knoten, Kanten und Tangenten
**Ausgabe** : Bézierzeichnung $Z'$ ohne Kantenüberschneidungen und mit gleicher Einbettung wie $Z$

```
1  wende alle Kräfte auf Z an, erhalte dadurch Z'
2  while es existieren Überschneidungen in Z' or Einbettungen von Z und Z'
   unterscheiden sich do
3      foreach e_1 ∈ E do
4          foreach e_2 ∈ E do
5              if e_1 inzident zu e_2 then
6                  if die Einbettung hat sich an dieser Stelle verändert then
7                      halbiere die Kräfte auf e_1 und e_2 sowie die beteiligten Knoten
                       und Tangenten so lange, bis die ursprüngliche Einbettung
                       wiederhergestellt ist
8              else if  die konvexen Hüllen von e_1 und e_2 überschneiden sich then
9                  halbiere die Kräfte auf e_1 und e_2 und die beteiligten Knoten sowie
                   deren Tangenten so lange, bis die Überschneidung aufgelöst ist
10 return Z'
```

des Algorithmus mehrere Kanten zu einer Kante zusammengelegt, wenn dies möglich ist. Das führt dazu, dass lange Teile einer Linie gleichmäßig gekrümmt sind. Abb. 3.22 zeigt, wie das Zusammenlegen zu einer einheitlichen Krümmung und damit zu einem schöneren Verlauf der Linie führen kann.



**(a)** Vor der Zusammenlegung **(b)** Nach der Zusammenlegung

**Abb. 3.22:** Beispiel für das Zusammenlegen von Kanten

Damit zwei Kanten $e_1$ und $e_2$ zusammengelegt werden können, müssen einige Bedingungen erfüllt sein:

1. Die Kanten $e_1$ und $e_2$ müssen inzident sein.
2. Die Linien, die auf $e_1$ verlaufen, müssen dieselben sein wie auf $e_2$.
3. Der Knoten $v$, der $e_1$ und $e_2$ verbindet, muss Grad 2 haben.
4. Durch das Zusammenlegen wird die Einbettung des Graphen nicht verändert.

5. Durch das Zusammenlegen entstehen keine parallelen Kanten, die aufeinander gezeichnet würden

Die letzten beiden Bedingungen werden nun noch kurz erläutert:
**zu 4:** Die Einbettung soll beibehalten werden, weil sonst die Topologie des Plans nicht mehr der Realität entsprechen würde. In Abb. 3.23 sieht man ein Beispiel, in dem die Kanten $e_1$ und $e_2$ nicht zusammengelegt werden können, da sonst die Einbettung verändert würde.

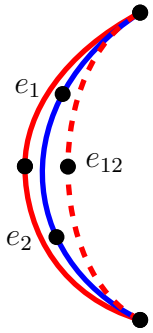


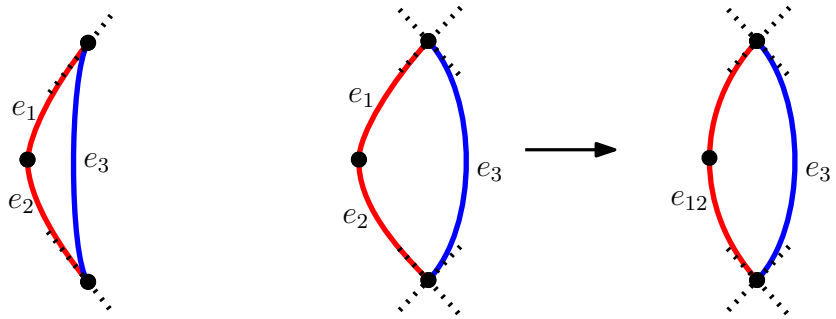

**Abb. 3.23:** Einbettung würde verändert

**zu 5**: Es soll verhindert werden, dass zwei unterschiedliche Kanten (mit eventuell unterschiedlichen Stationen auf ihnen) aufeinander gezeichnet werden. Es dürfen also nur dann parallele Kanten existieren, wenn deren Tangenten sich in mindestens einem Knoten unterscheiden. Abb. 3.24a zeigt einen Fall, in dem die Kanten $e_1$ und $e_2$ nicht zusammengelegt werden können, da die neue Kante $e_{12}$ die gleichen Tangenten hätte wie die parallele Kante $e_3$. In Abb. 3.24b hingegen können $e_1$ und $e_2$ zusammengelegt werden, da $e_{12}$ und $e_3$ in mindestens einem Knoten eine unterschiedliche Tangente haben. Man beachte dabei, dass der Knoten auf $e_{12}$ kein echter Knoten mehr ist, für den Kräfte berechnet werden, sondern nur noch eine Station auf $e_{12}$ darstellt.

Wenn alle diese Bedingungen erfüllt sind, können die Kanten $e_1$ und $e_2$ zu einer neuen Kante $e_{12}$ zusammengelegt werden. Dann wird der gemeinsame Knoten $v$ aus dem Graphen entfernt. Er wird allerdings auf der Kante $e_{12}$ gespeichert, um die Station später zeichnen zu können und die gewünschte Kantenlänge von $e_{12}$ zu berechnen. Die Kontrollpunkte der neuen Kante entsprechen den Kontrollpunkten der alten Kanten an den jeweiligen Knoten, das heißt die Tangenten und die Abstände zu den Kontrollpunkten bleiben erhalten.

## 3.6 Abwägen der Stärke der Kräfte

Nachdem die Kräfte jetzt definiert sind, müssen diese noch aufeinander abgestimmt werden. Die Stärke der einzelnen Kräfte muss so gewählt werden, dass sich ein gutes Gleichgewicht ergibt. Jede Kraft muss so stark sein, dass sie sich spürbar auf die Zeichnung auswirkt, aber nicht so groß, dass andere Kräfte dadurch kaum noch einen Effekt haben.




**(a)** Parallele Kanten mit gleichen Tangenten würden entstehen

**(b)** Zusammenlegen möglich, da die parallelen Kanten unterschiedliche Tangenten haben

**Abb. 3.24:** Beispiele für die Bedingungen, die beim Zusammenlegen erfüllt sein müssen

Zusammen mit den Gewichtungsfaktoren $f_\mathrm{r}, f_\mathrm{a}, f_\mathrm{o}, f_\mathrm{sv}$, die die Stärken der Kräfte anpassen, wirkt auf einen Knoten $v$ des Graphen folgende Kraft:

$$F(v) = \underbrace{\sum_{u \in V} f_\mathrm{r} \cdot F^\mathrm{r}(u,v)}_{\text{abstoßende Kräfte}} + \underbrace{\sum_{(u,v)\in E} f_\mathrm{a} \cdot F^\mathrm{a}(u,v)}_{\text{anziehende Kräfte}} + \underbrace{\sum_{(u,v)\in E} f_\mathrm{sv} \cdot F^\mathrm{sv}(u,v)}_{\text{Geradedrücken der Kanten}} + \underbrace{f_\mathrm{o} \cdot F^\mathrm{o}(v)}_{\text{Ursprungskraft}}$$

Auf einen Kontrollpunkt $c_u$, der zum Knoten $u$ gehört, wirkt nur die Kraft $F^\mathrm{c}(c_u) = f_\mathrm{c} \cdot F^\mathrm{c}(d(c_u, u))$ mit Gewichtungsfaktor $f_\mathrm{c}$, die für einen sinnvollen Abstand zwischen Knoten und Kontrollpunkten sorgt.

Sei $t$ eine Tangente in einem Knoten $u$. Sei zudem $l$ die Anzahl von Kanten, die $t$ in $u$ benutzen, und $p$ die Anzahl von Tangenten in $u$ ohne $t$. Dann ist $\alpha_k$ jeweils der Winkel zwischen $t$ und dem Vektor $\overrightarrow{uv_k}$ für jede Kante $(u, v_k)$, die $t$ benutzt. Die Kraft, die im Knoten $u$ auf die Tangente $t$ wirkt, lautet dann mit den Gewichtungsfaktoren $f_\mathrm{st}$ und $f_\mathrm{rt}$:

$$F(t) = \frac{\sum_{k=1}^{l} f_\mathrm{st} \cdot \alpha_k \cdot d(u, c_u(u, v_k)) + \sum_{k=1}^{p} F^\mathrm{rt}(t, t_k) \cdot f_\mathrm{rt}}{\sum_{k=1}^{l} d(u, c_u(u, v_k)) + \sum_{k=1}^{p} f_\mathrm{rt}}$$

Um die Stärke der Kräfte gut auszubalancieren, müssen die Gewichtungsfaktoren $f_\mathrm{r}, f_\mathrm{a}, f_\mathrm{o}, f_\mathrm{sv}, f_\mathrm{c}, f_\mathrm{st}, f_\mathrm{rt}$ richtig gewählt werden. Dazu ist viel Test-Arbeit nötig. Wie die Faktoren in dieser Arbeit gewählt sind, wird im nächsten Kapitel näher erläutert.

# 4 Implementierung und Tests

Zur Implementierung des Algorithmus wurde Java verwendet und zum Speichern des Graphen wurde die Java-Graphen-Bibliothek JUNG[1] benutzt. Diese bietet alle grundlegenden Funktionen, die für diese Arbeit nötig sind. Das ist natürlich primär die Speicherung des Graphen, aber auch andere Funktionen wie das Finden von Nachbarknoten oder inzidenten Kanten werden bei der Implementierung benötigt. Eine wichtige Eigenschaft dieser Bibliothek ist auch, dass man eigene Knoten- und Kantenklassen definieren kann, die dann im Graph benutzt werden. Damit ist es sehr leicht möglich, alle benötigten Informationen wie Knoten-Positionen oder Kräfte in den Knoten und Kanten zu speichern.

Durch die in Java vorhandenen DOM-Funktionen[2] kann die Eingabe unkompliziert eingelesen werden. Diese besteht aus sogenannten GraphML-Dateien[3]. Das sind XML-Dateien mit einer festgelegten Syntax, um die Struktur des Graphen zu definieren. Der Vorteil dieses Formats ist vor allem, dass man die Grundstruktur sehr flexibel erweitern kann. Damit können alle wichtigen Informationen in die Eingabe geschrieben werden, wie zum Beispiel die Position aller Knoten oder die Kantenannotationen.

Die Ausgabe des Algorithmus wird in eine Ipe-Datei geschrieben. Auch dieses Format basiert auf XML und kann mit dem Zeichenprogramm Ipe[4] bearbeitet werden. Der entscheidende Vorteil von Ipe-Dateien ist, dass sie sich sehr einfach in PDF-Dateien umwandeln lassen.

## 4.1 Aufbau des Programms

Die wichtigsten Klassen des Programms sind folgende:

- **MyVertex** und **MyEdge**: Diese Klassen speichern die Informationen der Knoten und Kanten, aus denen der Graph aufgebaut ist. Dazu gehören unter anderem die aktuellen Positionen der Knoten, die Information, welche Kante welche Tangente benutzt, oder die Stationen, die auf den Kanten liegen. Hier werden auch die Kräfte für Knoten und Kanten gespeichert, bis sie am Ende einer Iteration angewandt werden.

- **GraphMLReader**: Liest, wie der Name schon sagt, die GraphML-Eingaben ein. Hier wird der Graph erstellt und es werden für jeden Knoten die Tangenten fest-

[1] http://jung.sourceforge.net/
[2] http://w3c.org/DOM
[3] http://graphml.graphdrawing.org/
[4] http://ipe7.sourceforge.net/

gelegt, die die inzidenten Kanten benutzen. Das wird für die einfachen Fälle automatisch erledigt, wie in Kapitel 3.2 beschrieben. Für komplexere Fälle werden hier die Kantenannotationen ausgewertet. Anschließend werden die Kontrollpunkte so gesetzt, dass keine Kantenüberschneidungen entstehen. Damit ist der Graph vollständig erstellt und das kräftebasierte Verfahren kann beginnen.

- **IPEWriter**: Enthält Methoden, um die aktuelle Zeichnung in eine Ipe-Datei zu schreiben. Hier können auch die Farben der Linien und die Größe der Zeichnung festgelegt werden.

- **ForceDirectedAbstract**: Das ist die Hauptklasse des Algorithmus. Hier sind die Kräfte definiert und hier ist das kräftebasierte Verfahren implementiert. Außerdem sind hier die Methoden implementiert, die Kantenüberschneidungen verhindern und die Einbettung des Graphen beibehalten. Hier wird auch nach jeder Iteration getestet, ob mehrere Kanten zusammengelegt werden können. Die Klasse ist als abstrakte Klasse definiert. Um den Algorithmus verwenden zu können, muss von dieser Klasse abgeleitet werden und die Funktion, die bestimmt, welche Kräfte berechnet werden sollen, muss in der abgeleiteten Klasse implementiert werden.

- **ForceDirected**: Diese Klasse ist von ForceDirectedAbstract abgeleitet und führt den Algorithmus aus. Der Vorteil des Ableitens ist, dass so „verschiedene“ Algorithmen definiert werden können. Denn wenn man eine neue Klasse schreibt, die auch von ForceDirectedAbstract ableitet, können hier zum Beispiel manche Kräfte ausgestellt werden oder die Gewichtungsfaktoren können anders gesetzt werden.

- **Test**: Diese Klasse enthält die Main-Methode, die das Programm startet. Hier können außerdem viele Einstellungen für den Algorithmus festgelegt werden. Es kann eine Schranke festgelegt werden, die dafür sorgt, dass, wenn in einer Iteration die größte Verschiebung eines Knotens kleiner als diese Schranke ist, der Algorithmus abbricht. Da das Testen auf Kantenüberschneidungen aufwändig ist (siehe Tab. 4.1), kann es ausgestellt werden, wenn es nicht benötigt wird, um schneller Ergebnisse zu erhalten.

## 4.2 Laufzeiten

Um die Laufzeiten des Algorithmus zu bestimmen, wurden Tests mit jeweils 200 Iterationen durchgeführt. Hier ergeben sich meist schon sehr gute Zeichnungen und später ändert sich nicht mehr viel. Der Test-Rechner war ein Dual-Core mit 3 GHz, 4 GB Arbeitsspeicher und Windows 7. Tabelle 4.1 zeigt die Laufzeiten für 4 verschiedene Städte. Wie man sieht, ist der Algorithmus für kleine Eingaben sehr schnell, auch deshalb, weil hier viele Kanten zusammengelegt werden können und somit die Anzahl der berechneten Kräfte pro Iteration klein ist. Auch das Testen auf Überschneidungen geht hier sehr schnell. Für große Eingaben wie London hingegen braucht der Algorithmus recht lange. Das liegt vor allem daran, dass in London weniger Kanten zusammengelegt werden können, weil der Graph sehr stark vernetzt ist, und der Graph wesentlich mehr

Facetten hat. Deshalb müssen pro Iteration mehr Kräfte berechnet werden als bei den anderen Eingaben. Dadurch, dass viele Kanten bestehen bleiben, braucht auch der Test auf Überschneidungen sehr lange.

| | Eingabe | | Nach dem Zusammenlegen der Kanten | | Facetten |
|---|---|---|---|---|---|
| | Knoten | Kanten | Knoten | Kanten | |
| Montreal | 69 | 70 | 10 | 11 | 3 |
| Wien | 90 | 96 | 19 | 25 | 8 |
| Sydney | 205 | 214 | 35 | 44 | 11 |
| London | 402 | 455 | 100 | 153 | 55 |

| | Laufzeit ohne Test auf Kantenüberschneidungen | Laufzeit mit Test auf Kantenüberschneidungen |
|---|---|---|
| Montreal | 2 Sek. | 7 Sek. |
| Wien | 2 Sek. | 30 Sek. |
| Sydney | 9 Sek. | 2 Min. 24 Sek. |
| London | 1 Min. 14 Sek. | 27 Min. 40 Sek. |

**Tab. 4.1:** Laufzeit des Algorithmus für 200 Iterationen

In Tabelle 4.2 sieht man die asymptotischen Laufzeiten für die Berechnung der Kräfte für eine Iteration. Am aufwändigsten sind die abstoßenden Kräfte auf Paare von Knoten, weshalb die Berechnung aller Kräfte $O(|V|^2)$ Zeit benötigt. Das Vermeiden von Kantenüberschneidungen benötigt $O(|E|^2)$ Zeit, da alle Paare von Kanten getestet werden müssen. Insgesamt hat der Algorithmus also eine asymptotische Laufzeit in $O(|V|^2 + |E|^2)$ für eine Iteration, wobei hier $V$ und $E$ die Knoten und Kanten sind, die nicht durch Zusammenlegen von Kanten aus dem Graph entfernt werden können.

| | $F^{\mathrm{r}}$ | $F^{\mathrm{a}}$ | $F^{\mathrm{o}}$ | $F^{\mathrm{c}}$ | $F^{\mathrm{sv}}$ | $F^{\mathrm{st}}$ | $F^{\mathrm{rt}}$ |
|---|---|---|---|---|---|---|---|
| Laufzeit | $O(\|V\|^2)$ | $O(\|E\|)$ | $O(\|V\|)$ | $O(\|E\|)$ | $O(\|V\|)$ | $O(\|V\|)$ | $O(\|V\|)$ |

**Tab. 4.2:** Asymptotische Laufzeiten der Kräfte für eine Iteration

## 4.3 Abwägen der Stärke der Kräfte

In Kapitel 3.6 ist zusammengefasst, welche Kräfte auf Knoten, Kontrollpunkte und Tangenten wirken. Es ist nun nötig, die richtigen Werte für die Gewichtungsfaktoren $f_{\mathrm{r}}, f_{\mathrm{a}}, f_{\mathrm{o}}, f_{\mathrm{sv}}, f_{\mathrm{c}}, f_{\mathrm{st}}, f_{\mathrm{rt}}$ zu finden, denn für falsche Gewichtungsfaktoren ergeben sich keine schönen Zeichnungen. Außerdem dürfen die Faktoren für die Knoten-Kräfte nicht zu groß sein, weil es sonst passieren kann, dass die Verschiebungen durch die Kräfte mit jeder Iteration größer werden und die Zeichnung unbrauchbar wird. Abb. 4.1 zeigt ein Beispiel, wie es zu solch einem Aufschaukeln der Kräfte kommen kann. Die Kräfte, dargestellt durch die Pfeile, sollen eigentlich dafür sorgen, dass sich die Zeichnung

zusammenzieht. Weil sie aber zu stark sind, vergrößert sich der Abstand zwischen den Knoten sogar, woraufhin die Kräfte noch stärker werden.

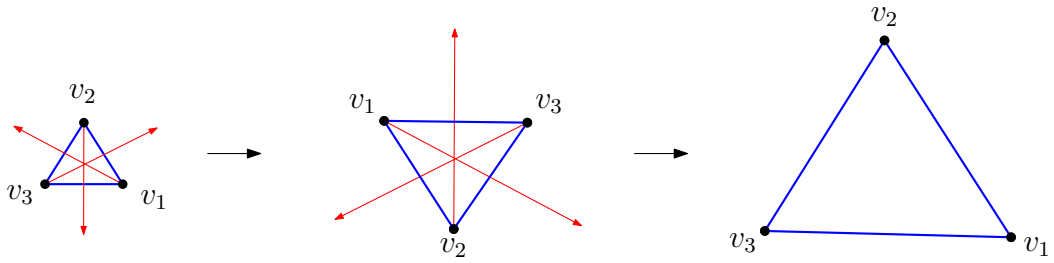
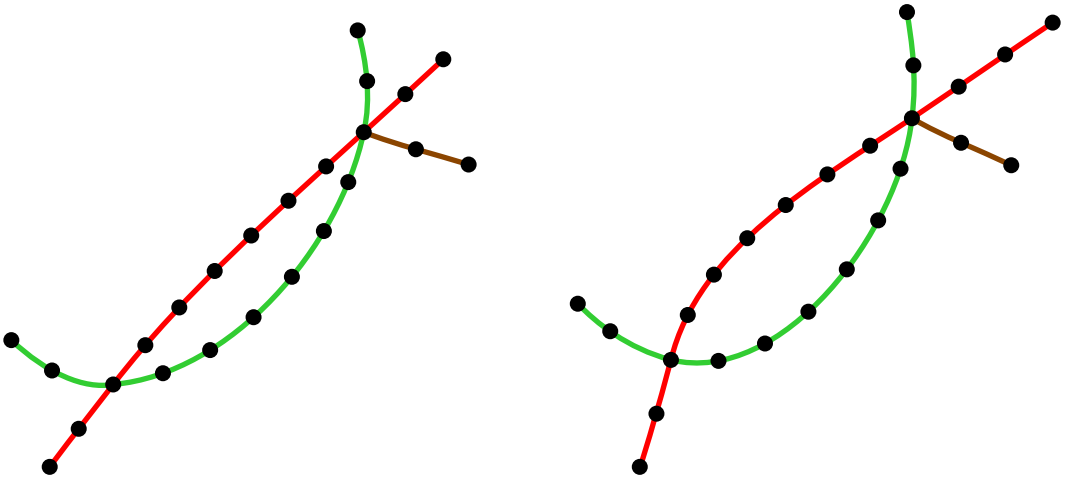



**Abb. 4.1:** Beispiel für das Aufschaukeln, wenn die Kräfte zu stark sind

Die Gewichtungsfaktoren, die sich in der Praxis als sinnvoll erwiesen haben, sind folgende: $f_\mathrm{r} = 0,01$; $f_\mathrm{a} = 0,01$; $f_\mathrm{o} = 0,05$; $f_\mathrm{sv} = 0,03$; $f_\mathrm{c} = 0,05$; $f_\mathrm{st} = 0,3$; $f_\mathrm{rt} = 1$. Was passiert, wenn zum Beispiel der Faktor $f_\mathrm{rt}$ für die optimale Winkelauflösung zu groß ist, zeigt Abb. 4.2. In Abb. 4.2a sieht man das Ergebnis mit den Standardeinstellungen. Für Abb. 4.2b wurde $f_\mathrm{rt}$ um den Faktor 10 erhöht. Deshalb wird die rote Linie weniger gerade gedrückt und es wird mehr Wert auf die Kraft zu Winkelauflösung gelegt, wodurch diese besser wird.



**(a)** Standardeinstellungen

**(b)** Stark erhöhte Kraft zur besseren Winkelauflösung

**Abb. 4.2:** Vergleich von unterschiedlichen Werten für $f_\mathrm{rt}$

Das Testen des Algorithmus war sehr aufwendig. Vor allem die Kräfte mussten lange getestet werden, bis sie für alle Eingaben die gewünschte Wirkung hatten und auch das Zusammenspiel zwischen ihnen einwandfrei funktionierte. Dabei war vor allem problematisch, dass für kleine Eingaben sehr viele unterschiedliche Konfigurationen der Kräfte sehr gut funktionierten. Für große Eingaben hingegen funktionierten viele Konfigurationen nicht sehr gut. Es musste zum Testen also immer London oder mindestens Sydney

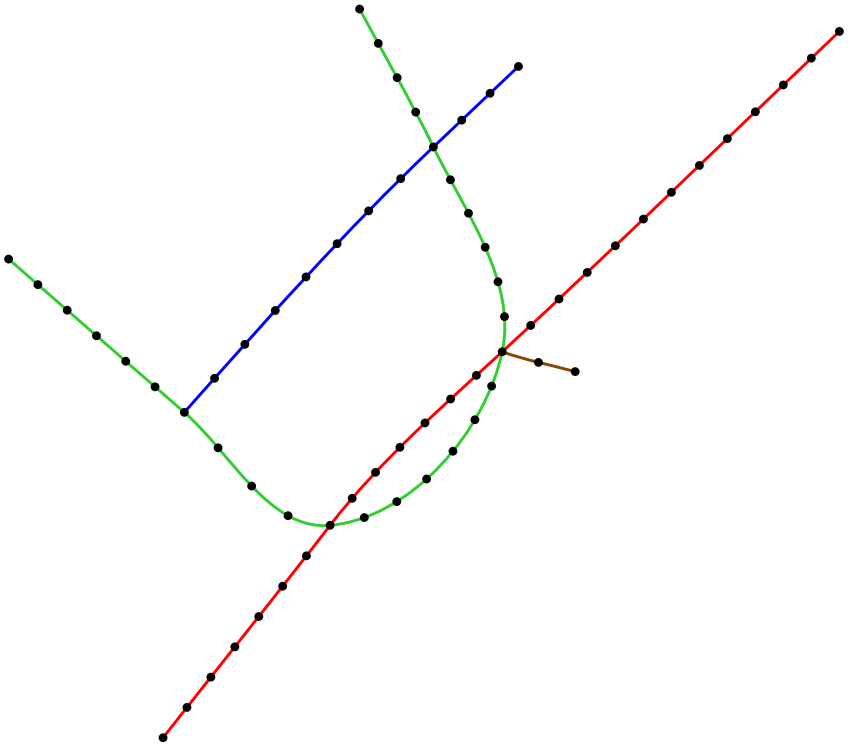
verwendet werden, um zu erkennen, ob die Einstellungen die gewünschten Auswirkungen auf die Zeichnungen haben. Da die Laufzeit für die großen Eingaben aber recht hoch ist, benötigte das Testen deshalb sehr viel Zeit.

Die abstoßenden und anziehenden Kräfte auf Knoten, die für die optimale Kantenlänge sorgen, funktionierten von Anfang an sehr gut. Die Definitionen dieser Kräfte waren bereits bekannt und die Implementierung bereitete keine großen Probleme. Auch die Kraft auf Kontrollpunkte war schnell eingebaut und funktionierte sofort sehr gut. Für diese Kraft ließ sich auch der richtige Gewichtungsfaktor leicht finden, da es die einzige Kraft ist, die auf Kontrollpunkte wirkt, und somit die Abstimmung mit anderen Kräften wegfiel.

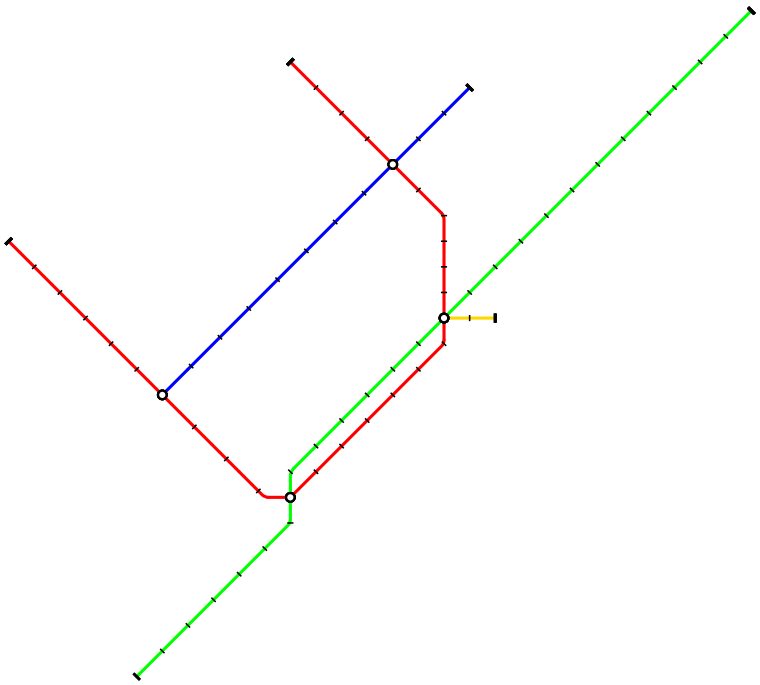
Die Implementierung der Kräfte auf Tangenten hingegen war sehr kompliziert. Da die Kräfte zuvor noch in keinem kräftebasierten Verfahren verwendet wurden, mussten zuerst die Definitionen gefunden werden, mit denen die Kräfte die gewünschte Wirkung haben. Diese wurden im Verlauf der Implementierung des Algorithmus mehrfach angepasst, bis die jetzigen, gut funktionierenden Definitionen gefunden waren. Auch die Hebelwirkung der Tangentenkräfte wurde erst nachträglich eingebaut, als sich herausstellte, dass manche Kräfte auf Tangenten wichtiger sind als andere, um schöne Zeichnungen zu erhalten.

## 4.4 Ergebnisse des Algorithmus

Die Abbildungen 4.3, 4.4, 4.6 und 4.8 zeigen anhand der U-Bahn-Linienpläne von Montreal, Wien, Sydney und London die Ausgabe des Algorithmus mit den Standardeinstellungen. Zusätzlich ist jeweils die oktilineare Eingabe mit angegeben. Das Ergebnis für Montreal sieht sehr gut aus. Es ist noch nicht optimal, weil die grüne Linie nicht komplett einheitlich gekrümmt ist. Trotzdem funktioniert der Algorithmus für diese Instanz sehr gut. Auch für Wien ergibt sich eine schöne Ausgabe. Verglichen mit der Wunsch-Ausgabe (Abb. 4.5) aus der Einleitung wirkt das Ergebnis noch etwas unrund, da die Linien nicht gleichmäßig gebogen sind. Das ist aber nicht verwunderlich, da die Linien durch mehrere Bézierkurven dargestellt werden und noch keine Kraft existiert, die für eine gleichmäßige Krümmung sorgt. Für Sydney funktioniert der Algorithmus ebenfalls gut. Das Finden einer schönen Zeichnung für London ist der Härtetest für den Algorithmus. In dieser Ausgabe liegt noch am meisten Verbesserungspotenzial. Hier existieren sehr viele lange Linien, die sich häufig kreuzen. Deshalb kann der Algorithmus an vielen Stellen kaum Kanten zusammenlegen, und es werden manche Linien sehr wellig gezeichnet. Hier könnte man die Zeichnung durch Kräfte, die für eine gleichmäßige Krümmung sorgen, noch stark verbessern.

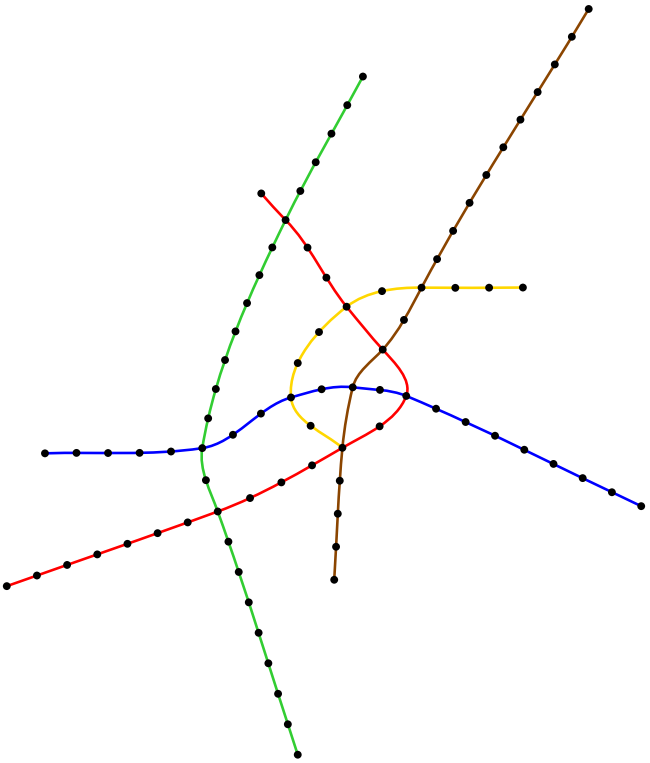


**(a)** Montreal mit Standardeinstellungen

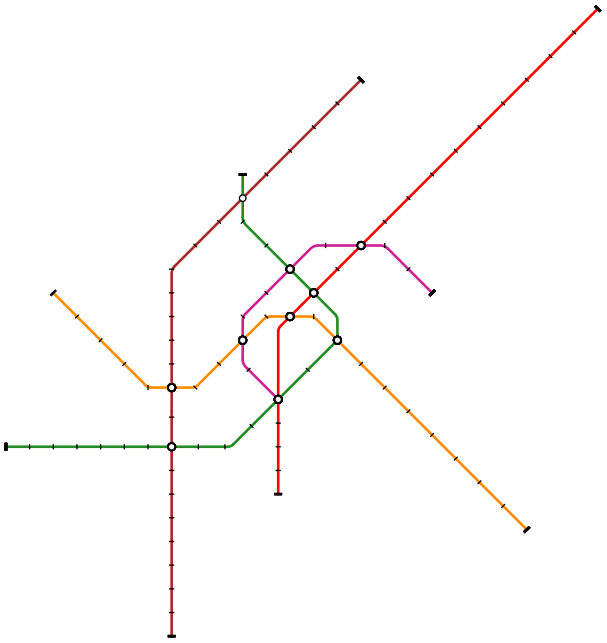


**(b)** Oktilineare Eingabe

**Abb. 4.3:** Eingabe und Ausgabe des Algorithmus für Montreal



**(a)** Wien mit Standardeinstellungen



**(b)** Oktilineare Eingabe für Wien

**Abb. 4.4:** Eingabe und Ausgabe des Algorithmus für Wien

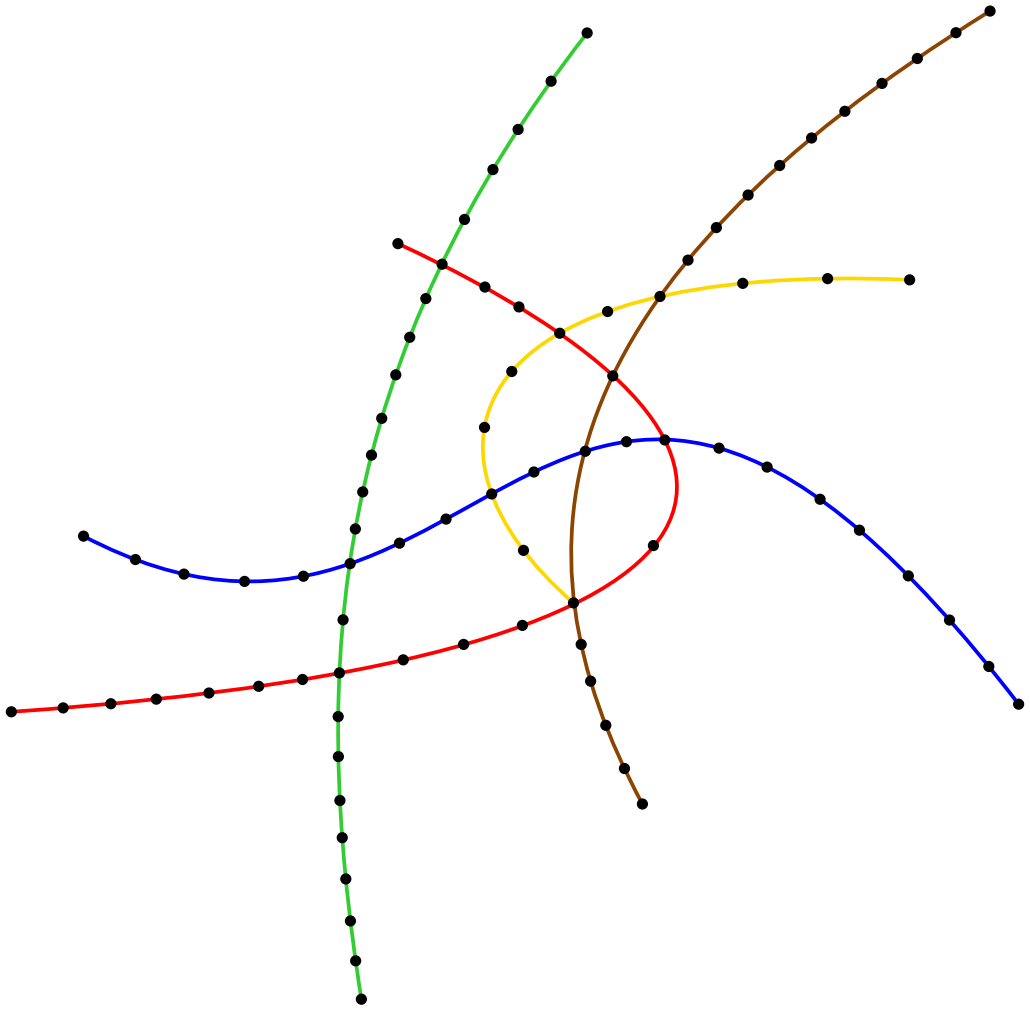


**Abb. 4.5:** Wunsch-Ausgabe aus der Einleitung

**Abb. 4.6:** Sydney mit Standardeinstellungen

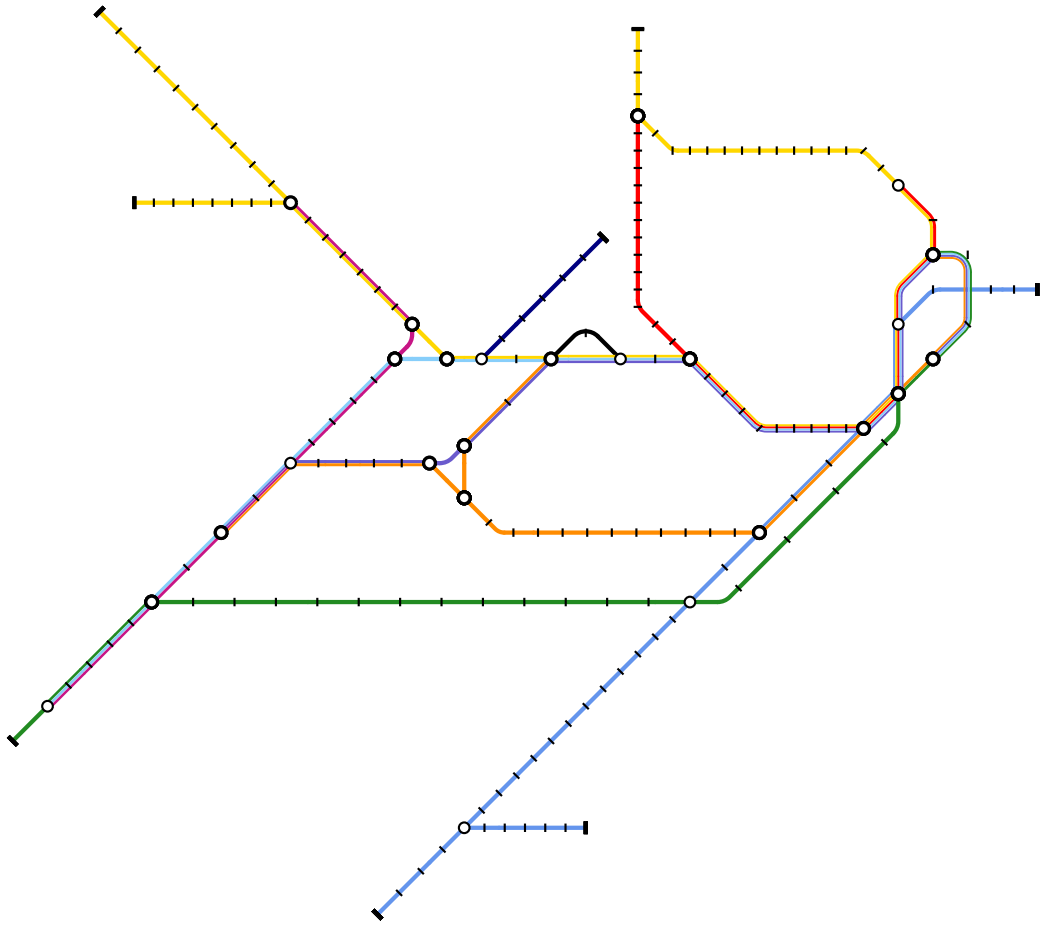


**Abb. 4.7:** Oktilineare Eingabe für Sydney

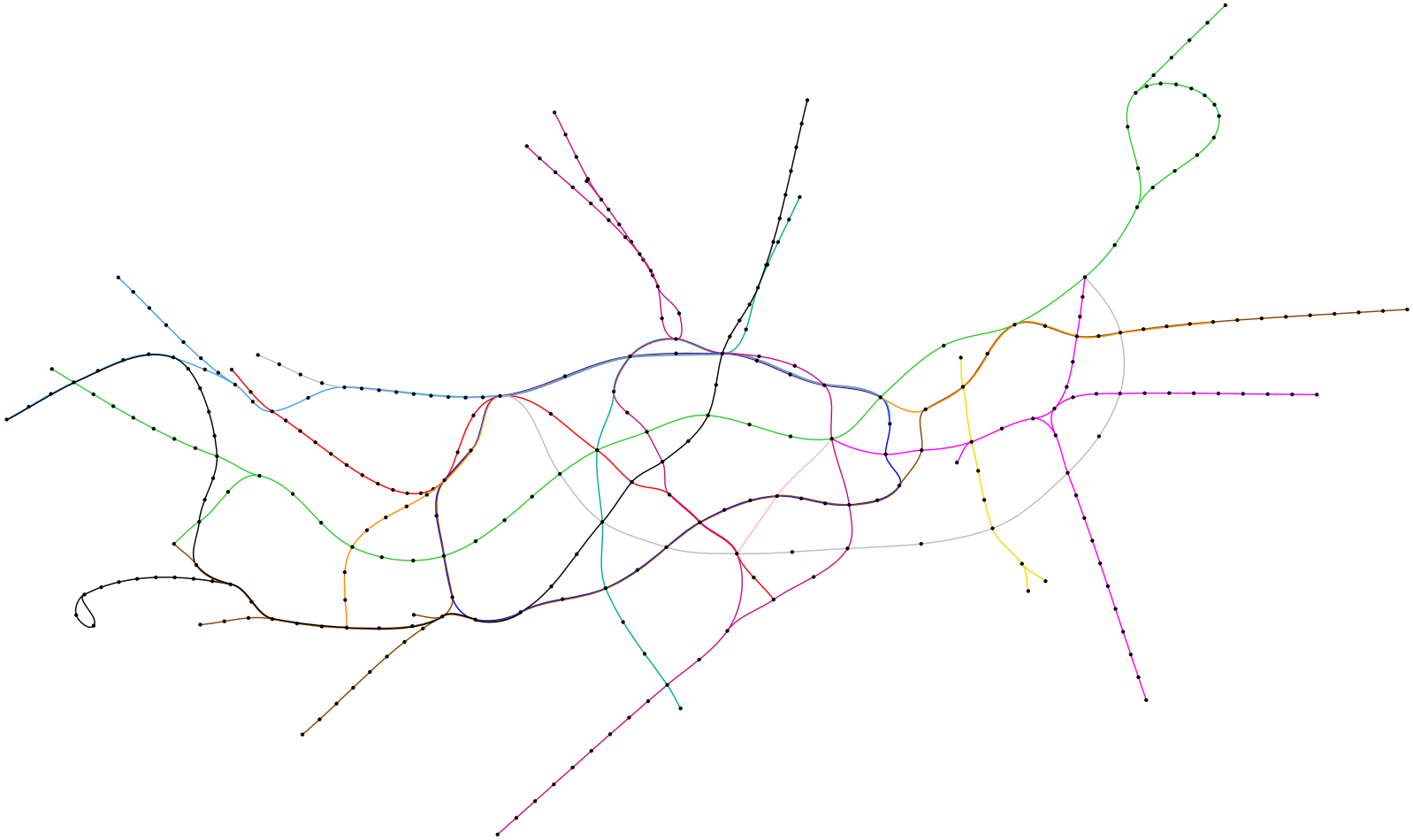


**Abb. 4.8:** London mit Standardeinstellungen

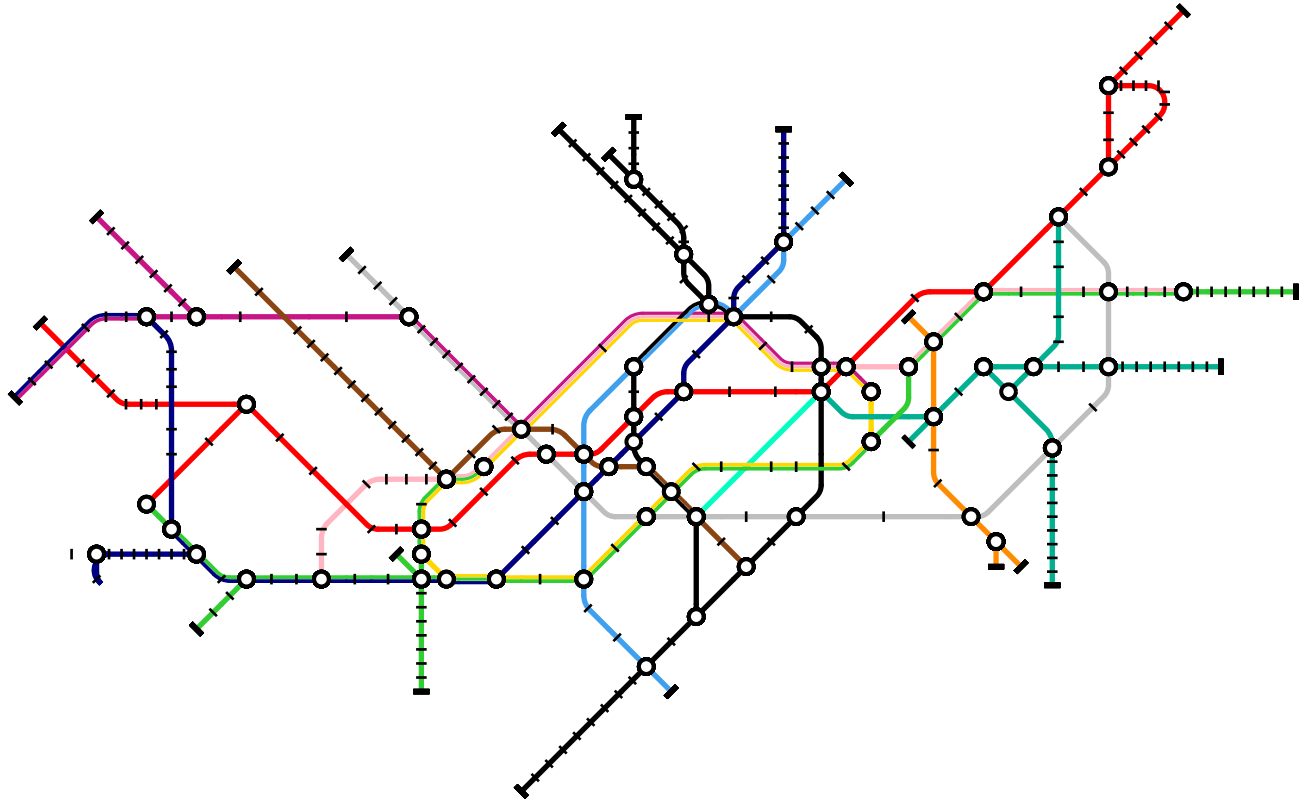


**Abb. 4.9:** Oktilineare Eingabe für London

# 5 Fazit und Ausblick

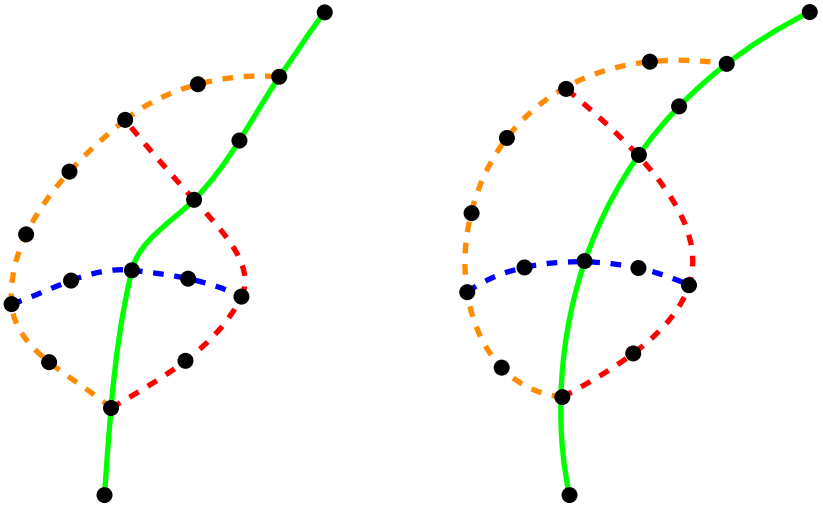
Aufbauend auf bekannten kräftebasierten Verfahren wurde ein Algorithmus entwickelt, der aus einem gegebenen oktilinearen U-Bahn-Linienplan einen Plan erstellt, in dem alle Linien durch möglichst wenige Bézierkurven dargestellt werden. Um übersichtliche Pläne zu erhalten, war es nötig, neue Kräfte einzuführen, die speziell darauf ausgelegt sind, das Aussehen von Bézierkurven zu verbessern. Des Weiteren wurde eine Methode entwickelt, um Kantenüberschneidungen zu verhindern und die ursprüngliche Einbettung beizubehalten. Anschließend wurden in ausgiebiger Testarbeit die Stärken der Kräfte aufeinander abgestimmt.

Wie man an den Ausgaben im letzten Kapitel gesehen hat, liefert der Algorithmus für die kleinen Eingaben schon gute Ergebnisse und auch die Ausgaben für die großen Pläne sehen gut aus. Trotzdem gibt es noch einige Möglichkeiten wie man den Algorithmus erweitern könnte, um die Ergebnisse noch zu verbessern.

Um schönere Kurven zu erhalten, werden im Algorithmus möglichst lange Kanten erzeugt, damit eine Linie gleichmäßig gekrümmt ist. Das ist aber nicht immer möglich, zum Beispiel wenn die Linie von vielen anderen Linien gekreuzt wird. Deshalb wäre es sinnvoll, eine Kraft einzuführen, die dafür sorgt, dass Linien auch über mehrere Knoten hinweg gleichmäßig gekrümmt sind. Noch besser wäre es allerdings, wenn mehrere Kanten durch eine Kurve dargestellt würden, auch wenn auf der Linie Knoten liegen, die man nicht entfernen kann. Abb. 5.1 zeigt einen Ausschnitt aus dem Plan von Wien. In Abb. 5.1a sieht man das Ergebnis des aktuellen Algorithmus. Schöner wäre hingegen eine Ausgabe wie in Abb. 5.1b. Hier wird jede Linie durch eine einzige Bézierkurve dargestellt, wodurch besonders die grüne (durchgezogene) Linie schöner aussieht, da sie auch über mehrere Knoten hinweg ihre Krümmung beibehält.

Ein weiterer Punkt, an dem der Algorithmus verbessert werden könnte, ist die Begrenzung der Kräfte. Momentan werden, sobald es eine Überschneidung gibt, alle beteiligten Kräfte begrenzt. Das muss aber nicht immer nötig sein. Beispielsweise könnten Kräfte auf Kontrollpunkte trotzdem angewandt werden, ohne dass diese zu Überschneidungen führen. Hier könnte man Prioritäten einführen, welche Kräfte zuerst reduziert werden sollen. Genauso könnte man darüber nachdenken, ob es sinnvoll wäre, nur die Kräfte auf eine der beiden Kanten zu reduzieren. So könnte man zum Beispiel die Kräfte auf längere Kanten weniger reduzieren, um diesen wichtigeren Kanten mehr Freiheit zu gewähren. Um die Freiheiten für die Kurven noch mehr zu erhöhen, könnte man anstatt der konvexen Hüllen die tatsächlichen Kurven auf Überschneidungen testen. Das würde zwar wahrscheinlich zu Lasten der Performanz gehen, könnte aber bessere Ergebnisse liefern.

In der Kraft zur Ursprungsposition werden momentan nur die Positionen der Knoten in der Eingabe verwendet. Da diese aber nicht den echten Orten entsprechen, könnte man



**(a)** Linien werden mit mehreren Bézierkurven dargestellt

**(b)** Linien werden durch eine Bézierkurve dargestellt

**Abb. 5.1:** Beispiel für eine Kraft, die zur gleichmäßigen Krümmung führt

die Eingabe erweitern und die geographischen Positionen mit hinein nehmen. Damit hätte die Ursprungskraft sinnvollere Auswirkungen auf die Zeichnung.

Um die allgemeine Übersichtlichkeit der Zeichnung noch zu erhöhen, könnte eine zusätzliche Kraft eingeführt werden, die dafür sorgt, dass Knoten von Kanten abgestoßen werden. In Abb. 5.2 sieht man ein Beispiel, in dem eine solche Kraft helfen würde. Da sich beide Kanten die gleiche Tangente teilen, werden sie momentan aufgrund der Kraft zum Geradedrücken der Kanten sehr nah beieinander gezeichnet (Abb. 5.2a). Mit einer Kraft, die die Knoten von den Kanten abstößt, würde sich eine schönere Zeichnung ergeben (Abb. 5.2b).

**(a)** Knoten wird nicht von der Kante abgestoßen

**(b)** Knoten wird von der Kante abgestoßen

**Abb. 5.2:** Beispiel für eine neue Kraft, die Knoten von Kanten abstößt

# Literaturverzeichnis


[Ber99] François Bertault: A force-directed algorithm that preserves edge crossing properties. In: Jan Kratochvíl (Herausgeber): *Proc. 7th Int. Symp. Graph Drawing (GD'99)*, Band 1731 der Reihe *Lecture Notes in Computer Science*, Seiten 351–358. Springer, 1999.

[BFK84] Wolfgang Böhm, Gerald E. Farin und Jürgen Kahmann: A survey of curve and surface methods in CAGD. *Computer Aided Geometric Design*, 1(1):1–60, 1984.

[BW00] Ulrik Brandes und Dorothea Wagner: Using graph layout to visualize train interconnection data. *J. Graph Algorithms Appl.*, 4(3):135–155, 2000.

[Ead84] Peter Eades: A heuristic for graph drawing. *Congressus numerantium*, 42:149–160, 1984.

[FR91] Thomas M. J. Fruchterman und Edward M. Reingold: Graph Drawing by Force-directed Placement. *Softw. Pract. Exper.*, 21(11):1129–1164, 1991.

[FT05] Benjamin Finkel und Roberto Tamassia: Curvilinear graph drawing using the force-directed method. In: János Pach (Herausgeber): *Proc. 12th Int. Symp. Graph Drawing (GD'04)*, Band 3383 der Reihe *Lecture Notes in Computer Science*, Seiten 448–453. Springer, 2005.

[HMdN06] Seok Hee Hong, Damian Merrick und Hugo A. D. do Nascimento: Automatic visualisation of metro maps. *J. Vis. Lang. Comput.*, 17(3):203–224, 2006.

[Kan92] Goos Kant: Drawing planar graphs using the lmc-ordering. In: *Proc. 33rd Annu. IEEE Conf. Foundat. Comput. Sci. (FOCS'92)*, Seiten 101–110, 1992.

[KN09] Sven Oliver Krumke und Hartmut Noltemeier: *Graphentheoretische Konzepte und Algorithmen*. Vieweg+Teubner Verlag / GWV Fachverlage GmbH, Wiesbaden, 2009.

[MM96] Kurt Mehlhorn und Petra Mutzel: On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm. *Algorithmica*, 16(2):233–242, 1996.

[NW11] Martin Nöllenburg und Alexander Wolff: Drawing and Labeling High-Quality Metro Maps by Mixed-Integer Programming. *IEEE Trans. Vis. Comput. Graph.*, 17(5):626–641, 2011.

[RND+11] Maxwell J. Roberts, Elizabeth J. Newton, Fabio D.Lagattolla, Simon Hughes und Megan C. Hasler: Objective versus Subjective Measures of Metro Map Usability: Investigating the Benefits of Breaking Design Rules (unveröffentlichtes Manuskript), 2011.

[SRMW11] Jonathan M. Stott, Peter Rodgers, Juan Carlos Martinez-Ovando und Stephen G. Walker: Automatic Metro Map Layout Using Multicriteria Optimization. *IEEE Trans. Vis. Comput. Graph.*, 17(1):101–114, 2011.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen Hilfsmittel und Quellen als die angegebenen benutzt habe. Weiterhin versichere ich, die Arbeit weder bisher noch gleichzeitig einer anderen Prüfungsbehörde vorgelegt zu haben.

Würzburg, den _____ , _____

(Julian Schuhmann)