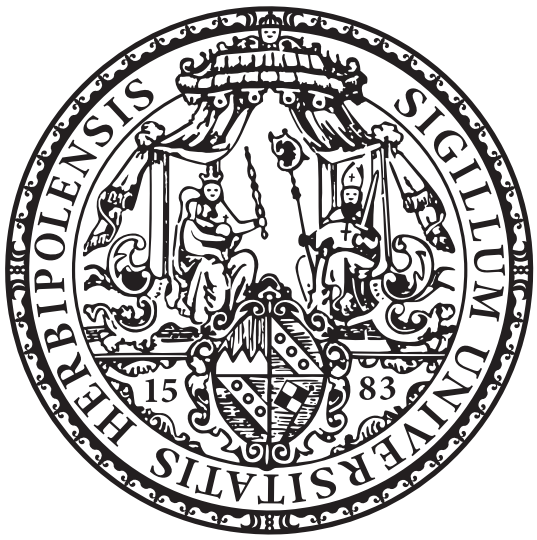
Praktikumsbericht

Chord-Diagramme mit partiellen Knotenvorgaben

Jan Sauer

Abgabedatum: 18. Januar 2026
Betreuer: Prof. Dr. Alexander Wolff
Tim Hegemann



Julius-Maximilians-Universität Würzburg
Lehrstuhl für Informatik I
Algorithmen und Komplexität

1. Einleitung

In modernen Datenanwendungen treten häufig Graphen auf, die global eher *sparsam*, lokal aber *sehr dicht* vernetzt sind. Diese finden sich etwa in sozialen, biologischen oder Ko-Autorenschaftsnetzwerken, in denen Gruppen (*Communities* oder *Cluster*) stark untereinander verbunden sind, jedoch nur wenige Verbindungen zwischen den Gruppen bestehen. Diese heterogene Struktur stellt besondere Herausforderungen an die Visualisierung, da klassische Darstellungsformen in den dicht vernetzten Teilbereichen unter Überlagerung, visueller Unübersichtlichkeit und kognitiver Überlastung leiden [DGDMT21].

Eine vielversprechende Lösung sind hybride Visualisierungsmethoden, die unterschiedliche graphische Metaphern kombinieren, um sowohl die globale Struktur als auch die lokalen Details – insbesondere innerhalb der Cluster – angemessen darzustellen. Ein prominentes Beispiel ist *Chord-Link*, welches von *Node-Trix* [HFM07] inspiriert wurde und dichte Cluster als Chord-Diagramme visualisiert, während die globalen Verbindungen zwischen Clustern und Knoten außerhalb weiterhin über Node-Link-Darstellungen realisiert werden [ADM+19].

Chord-Diagramme. Ein Chord-Diagramm visualisiert Knoten als Teilkreisbögen entlang eines Kreises, während Kanten als Sehnen oder glatte Kurven im Kreisinneren verlaufen und die Beziehungen zwischen diesen Knoten abbilden [ADM+19].

Ausgangspunkt dieser Arbeit ist die Idee, von außen kommende Verbindungen gebündelt in sogenannten *Gate-Abschnitten* mit den Kreisbögen der Chord-Diagramme zu verbinden. Die Positionierung der einzelnen Chord-Graphen bestimmt dabei die Lage der Gate-Abschnitte und unterteilt den Kreis in mehrere Segmente. In dieser Arbeit wird die daraus resultierende Problemstellung analysiert: die Knoten eines Clusters sollen auf die entstehenden Teilkreise verteilt, deren Gesamtgröße bestimmt und gleichzeitig eine übersichtliche Darstellung mit möglichst wenigen Kreuzungen erzeugt werden (siehe Abbildung 1).

Hierzu wird zunächst die Problemstellung präzise definiert und kurz auf die Komplexität der entstehenden Probleme eingegangen. Anschließend wird ein dynamisches Programm präsentiert, das sowohl die minimale Größe des Chord-Diagramms als auch dessen interne Struktur berechnet. In Abschnitt 3 folgt ein exakter Lösungsansatz auf Basis zweier kombinierter *ganzzahliger linearer Programme (ILPs)*. Dieser Ansatz wird experimentell ausgewertet und im Hinblick auf die praktische Anwendbarkeit untersucht. Schließlich wird eine praktikablere Lösung in Form einer Heuristik diskutiert.

2. Problemstellung

In dieser Arbeit betrachten wir die Konstruktion einzelner Chord-Diagramme. Sei G ein einfacher, ungerichteter Graph mit Knotenmenge $V(G)$ und Kantenmenge $E(G)$. Das zirkuläre Layout wird durch die Einführung von *Gate-Abschnitten* in Teilkreisstücke unterteilt, die im Folgenden als *Container* C bezeichnet werden. Die Gate-Abschnitte werden in dieser Arbeit zunächst ignoriert und nur die durch sie entstehenden Teilkreis-

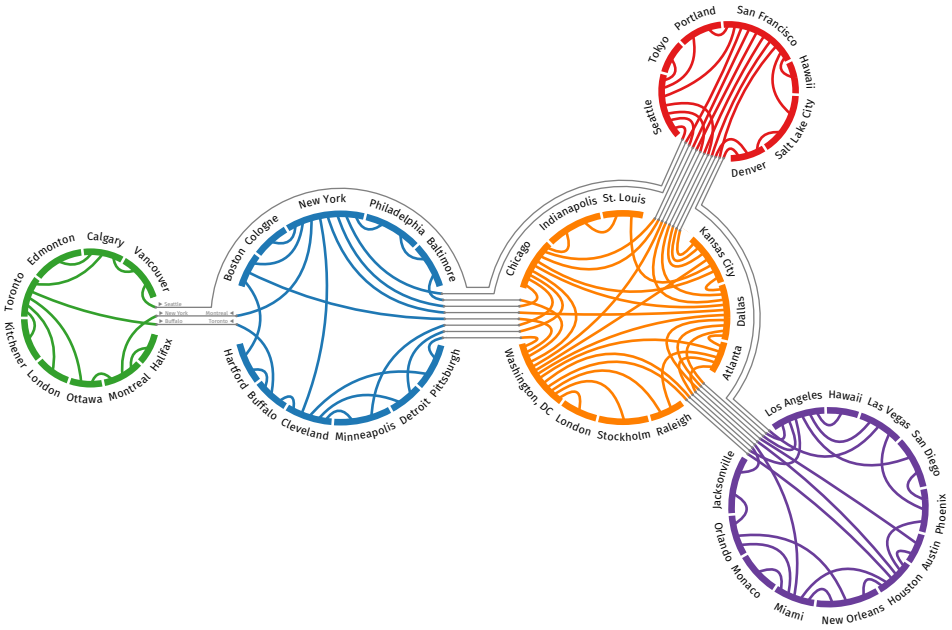


Abb. 1: Chord-Diagramme mit Gate-Abschnitten

stücke betrachtet. Eine mögliche Modellierung der Gate-Abschnitte wird in Abschnitt 6 weiter erläutert.

Jedem Container $c_i \in C$ ist eine Basisgröße f_i zugeordnet. Über einen gemeinsamen Skalierungsfaktor k ergibt sich die effektive Spannweite des Containers zu

$$\text{span}(c_i) = f_i \cdot k.$$

Jeder Knoten $v \in V(G)$ wird durch einen Teilkreisbogen repräsentiert, dessen Länge proportional zu seinem Gewicht w_v gewählt wird. Das Gewicht eines Knotens ist dabei gleich seines Knotengrades. Ziel ist es, die Knoten vollständig so auf die Container zu verteilen, dass die Gesamtlänge der Knotenbögen die Containerkapazität nicht überschreitet. Formal ergibt sich die Bedingung

$$\sum_{v \in c_i} w_v \;\leq\; f_i \cdot k \quad \forall c_i \in C.$$

Darüber hinaus sollen Kreuzungen zwischen den Kanten des Chord-Diagramms möglichst vermieden werden. Wir definieren zwei separate Optimierungsprobleme, die wir in einer Zielfunktion kombinieren, und die so die Minimierung des Faktors k, als auch die Reduktion der Kreuzungen erfasst.

Optimierung Skalierungsfaktor

Minimiere k unter den Nebenbedingungen:

$$\sum_{v \in c_i} w_v \;\leq\; f_i \cdot k \qquad \forall c_i \in C,$$
$$k \in \mathbb{N}_{>0},$$

- k ist der Skalierungsfaktor des Chord-Diagramms,
- w_v ist das Gewicht eines Knotens $v \in V$,
- f_i ist die relative Größe des Containers $c_i \in C$.

Optimierung Kreuzungsminimierung

Sei $\mathcal{S}(V)$ die Menge aller Permutationen der Knotenmenge V. Minimiere $\text{cross}(\pi)$ unter den Nebenbedingungen:

- $\pi \in \mathcal{S}(V)$ bezeichnet eine Permutation der Knoten, die deren Anordnung im Kreis beschreibt,
- $\text{cross}(\pi)$ gibt die Anzahl der Kantenkreuzungen für die Ordnung π an.

2.1. Kombinierte Zielfunktion

$\alpha, \beta > 0$ sind Gewichtungsfaktoren, die das Verhältnis von Größe und Kreuzungsvermeidung steuern.

$$\min_{k,\pi} \quad \alpha \cdot k \;+\; \beta \cdot \text{cross}(\pi)$$

2.2. Reduktion von Bin-Packing

Wir reduzieren das klassische Bin-Packing-Entscheidungsproblem auf die Entscheidungsvariante unseres Container-Zuweisungsproblems.

Instanz von Bin-Packing. Gegeben seien n Gegenstände mit Größen $s_1, \ldots, s_n \in \mathbb{N}$, eine Anzahl m von Behältern (Bins) und eine Bin-Kapazität $C \in \mathbb{N}$. Die Entscheidungsfrage lautet: Lässt sich die Menge der Gegenstände in höchstens m Bins so aufteilen, dass in jedem Bin die Summe der Größen höchstens C beträgt [KV18]?

Konstruktion der Instanz für das Container-Problem. Aus dieser Instanz konstruieren wir in polynomieller Zeit eine Instanz des Container-Problems wie folgt:

- Erzeuge m Container $c_1, \ldots, c_m$ und setze für alle i die Basisgröße $f_i := C$.
- Erzeuge n Knoten (Teilkreisbögen) $v_1, \ldots, v_n$ mit Gewichten $w_{v_j} := s_j$.

Korrektheit der Reduktion. Unter dieser Konstruktion gilt: Jede zulässige Packung der BIN-PACKING-Instanz in m Bins der Kapazität C entspricht genau einer Zuordnung der Knoten auf die m Container mit

$$\sum_{v \in c_i} w_v \leq f_i \cdot k = C \quad \text{für alle } i.$$

Umgekehrt liefert jede zulässige Container-Zuweisung mit $k = 1$ eine gültige Bin-Packung.

Daher ist die BIN-PACKING-Instanz genau dann lösbar, wenn die konstruierte Instanz des Container-Problems mit $k = 1$ lösbar ist.

Da BIN-PACKING NP-vollständig ist [GJ79], folgt damit, dass bereits die Entscheidung, ob für gegebene k und gegebene Containergrößen eine zulässige Zuweisung aller Knoten existiert, NP-schwer ist. Auch die Kreuzungsminimierung in kreisförmigen Layouts ist NP-vollständig [MKNF87]. Da beide Teilprobleme für sich bereits NP-vollständig sind, gilt dies für das gesamte Optimierungsproblem (Minimierung von k unter Kapazitätsbedingungen). Falls die Entscheidungsvariante in NP liegt (z. B. bei ganzzahligen Gewichten und Containergrößen ist eine Lösung durch die Zuordnung der Knoten verifizierbar), erhält man damit die NP-Vollständigkeit der Entscheidungsvariante.

3. Dynamisches Programm zur Bestimmung des minimalen Faktors k

Wie in Kapitel 2 beschrieben, ergibt sich die zentrale Problemstellung darin, für eine gegebene Menge von Knoten mit Gewichten w_v und eine Menge von Containern mit Basisgrößen f_i den minimalen Faktor k zu bestimmen, so dass alle Knoten vollständig auf die Container verteilt werden können.
Da bereits die Entscheidungsvariante dieses Problems (für festes k) NP-schwer ist (vgl. Reduktion auf BIN PACKING), wurde ein *Backtracking-Verfahren mit Memoisierung* entwickelt, das kleine Instanzen effizient lösen kann. Um den optimalen Faktor k zu bestimmen, wird dieses Verfahren mit einer *binären Suche* kombiniert.

3.1. Algorithmischer Ansatz

Das Verfahren besteht aus zwei verschränkten Komponenten:

1. **Machbarkeitsprüfung für ein festes k.** Die Knoten werden in absteigender Reihenfolge ihrer Gewichte betrachtet und sukzessive in die Container eingefügt. Ein rekursives Backtracking prüft, ob eine vollständige Belegung möglich ist. Zur Vermeidung exponentieller Wiederholungen werden Zwischenzustände in einer *Memoisierungstabelle* gespeichert. Symmetrische Containerkonfigurationen werden durch Sortieren der Restkapazitäten zusammengefasst.

2. **Binäre Suche über k.** Die Suche startet mit einer unteren Schranke

$$k_{\min} = \left\lceil \frac{\sum_{v \in V} w_v}{\sum_{c \in C} f_c} \right\rceil$$

und einer konservativen oberen Schranke

$$k_{\max} \;=\; \left\lfloor \frac{\max_{v \in V} w_v}{\min_{c \in C} f_c} \right\rfloor + 1.$$

In jedem Schritt wird für $k = \lfloor (k_{\min} + k_{\max})/2 \rfloor$ die Machbarkeit geprüft. Ist sie gegeben, wird $k_{\max}$ reduziert, andernfalls $k_{\min}$ erhöht. Dadurch ergibt sich am Ende der kleinste Faktor k, für den eine Lösung existiert.

3.2. Komplexität und Eigenschaften

Die Worst-Case-Laufzeit des Backtracking-Verfahrens ist exponentiell in der Anzahl der Knoten. In der Praxis werden jedoch durch folgende Mechanismen deutliche Verbesserungen erreicht:

- Sortierung der Knoten nach absteigendem Gewicht (Greedy-Heuristik),
- Summen-Bounds (Abbruch, falls Restgewichte größer als Restkapazitäten),
- Symmetrie-Reduktion durch sortierte Kapazitätsmultimengen,
- Memoisierung von nicht erfüllbaren Zwischenzuständen.

Durch die Kombination mit binärer Suche wird die Anzahl der Machbarkeitsprüfungen auf $\mathcal{O}(\log k_{\max})$ reduziert. Damit lässt sich das Verfahren für kleine bis mittlere Graphen in akzeptabler Zeit einsetzen.

3.3. Implementierung

Die Implementierung des Algorithmus erfolgt in *Java* unter Verwendung der Klassen `Graph`, `Container`, `Vertex` und `Edge`. Der vollständige Quelltext der Klasse `ChordPacking`, welche die beschriebene Logik enthält, ist im Anhang aufgeführt. Hier werden die Kapazitäten der Container und die Gewichte der Knoten direkt aus den Objekten extrahiert und für die Machbarkeitsprüfung verwendet.

4. ILP-Formulierung

Zur exakten Lösung von des Kombinationsproblems verwenden wir ein ganzzahliges lineares Programm (ILP), das die Containerzuordnung, die Skalierung des Faktors k sowie die Minimierung der Kreuzungen integriert. Grundlage ist die in [Goo22] vorgestellte Modellierung, die im Folgenden vollständig ausgeführt und erweitert wird.

Notation und Grundlagen

Sei G ein einfacher, ungerichteter Graph mit Knotenmenge $V(G)$ und Kantenmenge $E(G)$, sowie $n = |V(G)|$ und $m = |E(G)|$. Die Nachbarschaft eines Knotens $v \in V(G)$ sei definiert als $N(v) = \{u \in V(G) : \{u, v\} \in E(G)\}$.

Eine *zirkuläre Anordnung* (circular layout) von G ist gegeben durch eine Bijektion $\pi : V(G) \to \{0, \ldots, n-1\}$, die als Uhrzeiger-Reihenfolge der Knoten entlang des Kreises interpretiert wird.

Für einen Referenzknoten $s \in V(G)$ induziert π eine lineare Ordnung $\prec_s^\pi$ auf $V(G)$, definiert durch

$$u \prec_s^\pi v \iff \big(\pi(u) - \pi(s)\big) \bmod n \; < \; \big(\pi(v) - \pi(s)\big) \bmod n.$$

Wir führen die Variable γ ein. Wenn zwei Kanten $e_1 = \{u_1, v_1\}, e_2 = \{u_2, v_2\} \in E(G)$ sich in der Zeichnung Π kreuzen gilt:

$$\gamma_\Pi(\{u_1, v_1\}, \{u_2, v_2\}) = \begin{cases} 1 & \text{falls } u_1 \prec_s^\pi u_2 \prec_s^\pi v_1 \prec_s^\pi v_2, \\ 0 & \text{sonst.} \end{cases}$$

Ohne die Annahme $\pi(u_i) < \pi(v_i)$ für $i = 1, 2$ ergeben sich insgesamt acht mögliche Fälle, die Kreuzungen erzeugen [Goo22].

Analog lassen sich auch die Container ausgehend von s in einer Reihenfolge ordnen. Vereinfacht nehmen wir für spätere Notationen an, dass c_i vor c_j kommt, für alle $c_i, c_j \in C$ mit $i < j$.

Variable

Wir verwenden folgende Variablen:

- $\lambda_{v,c} \in \{0, 1\}$ für alle $v \in V(G)$ und $c \in C$: Indikatorvariable, die angibt, ob Knoten v Container c zugeordnet wird.
- $k \in \mathbb{N}$: Skalierungsfaktor zur Anpassung der Gesamtkapazität der Container. Aufgrund der späteren Implementierung beschränken wir uns auf ganzzahlige k.
- $\chi_{u,v} \in \{0, 1\}$ für alle $u, v \in V(G)$ und $u \neq v$: Ordnungvariable, die angibt, ob Knoten u im Kreis vor v platziert wird.
- $\gamma_{(a,b),(c,d)} \in \{0, 1\}$ für alle disjunkten Kantenpaare $(a, b), (c, d) \in E(G)$: Kreuzungsvariable, die angibt, ob sich die beiden Kanten in der gewählten Anordnung schneiden.
- $\delta_{v,c_i,u,c_j} \in \{0, 1\}$ für alle $c_i, c_j \in C$ mit $i \leq j$, und alle $v, u \in V(G)$ mit $u \neq v$: sorgt dafür, dass alle Knoten eines Containers direkt aufeinanderfolgen.

Formal ergibt sich:

$$\begin{aligned}
\lambda_{v,c} &\in \{0, 1\} && \forall v \in V(G), c \in C \\
k &\in \mathbb{N} \\
\chi_{u,v} &\in \{0, 1\} && \forall u, v \in V(G), u \neq v \\
\gamma_{(a,b),(c,d)} &\in \{0, 1\} && \forall (a, b), (c, d) \in E(G), \{a, b\} \cap \{c, d\} = \emptyset \\
\delta_{v,c_i,u,c_j} &\in \{0, 1\} && \forall c_i, c_j \in C, i < j, \; \forall v, u \in V(G), u \neq v
\end{aligned}$$

Zielfunktion

$$\min\ \alpha \cdot k\ +\ \beta \cdot \sum_{\substack{(a,b),(c,d)\in E \\ \{a,b\}\cap\{c,d\}=\emptyset}} \gamma_{(a,b),(c,d)}$$

mit Gewichtungsparametern $\alpha, \beta > 0$, die das Verhältnis zwischen minimalem Skalierungsfaktor und Kreuzungsminimierung steuern.

Nebenbedingungen

$\sum_{c\in C} \lambda_{v,c} = 1$	$\forall v \in V(G)$	jeder Knoten in genau einem Container
$\sum_{v\in V(G)} w_v \cdot \lambda_{v,c} \leq f_c \cdot k$	$\forall c \in C$	Kapazitäten der Container dürfen nicht überschritten werden
$\chi_{u,v} + \chi_{v,u} = 1$	$\forall u \neq v \in V(G)$	Kreisordnung
$\chi_{u,v} + \chi_{v,r} - \chi_{u,r} \leq 1$	$\forall u, v, r \in V(G)$	Transitivität
$\chi_{u,v} + \chi_{v,r} - \chi_{u,r} \geq 0$	$\forall u, v, r \in V(G)$	Transitivität
$\lambda_{v,c_i} + \lambda_{u,c_j} - \delta_{v,c_i,u,c_j} \leq 1$	$\forall c_i, c_j \in C,\ i < j,\ v \neq u$	Container zusammenhängend
$\sum_{v,u\in V(G)} \delta_{v,c_i,u,c_j} - \chi_{v,u} = 0$	$\forall c_i, c_j \in C,\ i < j,\ v \neq u$	Container zusammenhängend

Explizite Kreuzungsbedingungen

Für jedes disjunkte Kantenpaar $e_1 = \{a, b\}$ und $e_2 = \{c, d\}$ erzwingen wir, dass $\gamma_{(a,b),(c,d)} = 1$ genau dann wird, wenn die Endpunkte in der Kreisordnung abwechseln. Dies lässt sich durch die folgenden linearen Ungleichungen ausdrücken:

Bedingung	für Knotenordnung
$\gamma_{(a,b),(c,d)} \geq \chi_{a,c} + \chi_{c,b} + \chi_{b,d} - 2$	$\pi(a) < \pi(c) < \pi(b) < \pi(d)$
$\gamma_{(a,b),(c,d)} \geq \chi_{c,a} + \chi_{a,d} + \chi_{d,b} - 2$	$\pi(c) < \pi(a) < \pi(d) < \pi(b)$
$\gamma_{(a,b),(c,d)} \geq \chi_{a,d} + \chi_{d,b} + \chi_{b,c} - 2$	$\pi(a) < \pi(d) < \pi(b) < \pi(c)$
$\gamma_{(a,b),(c,d)} \geq \chi_{d,a} + \chi_{a,c} + \chi_{c,b} - 2$	$\pi(d) < \pi(a) < \pi(c) < \pi(b)$
$\gamma_{(a,b),(c,d)} \geq \chi_{b,c} + \chi_{c,a} + \chi_{a,d} - 2$	$\pi(b) < \pi(c) < \pi(a) < \pi(d)$
$\gamma_{(a,b),(c,d)} \geq \chi_{c,b} + \chi_{b,d} + \chi_{d,a} - 2$	$\pi(c) < \pi(b) < \pi(d) < \pi(a)$
$\gamma_{(a,b),(c,d)} \geq \chi_{b,d} + \chi_{d,a} + \chi_{a,c} - 2$	$\pi(b) < \pi(d) < \pi(a) < \pi(c)$
$\gamma_{(a,b),(c,d)} \geq \chi_{d,b} + \chi_{b,c} + \chi_{c,a} - 2$	$\pi(d) < \pi(b) < \pi(c) < \pi(a)$

4.1. Implementierung

Das ILP wurde in Java direkt mit der von Gurobi [Gur24] angebotenen API umgesetzt und getestet. Der Quellcode hierzu findet sich im Anhang.

5. Experimentelle Auswertung

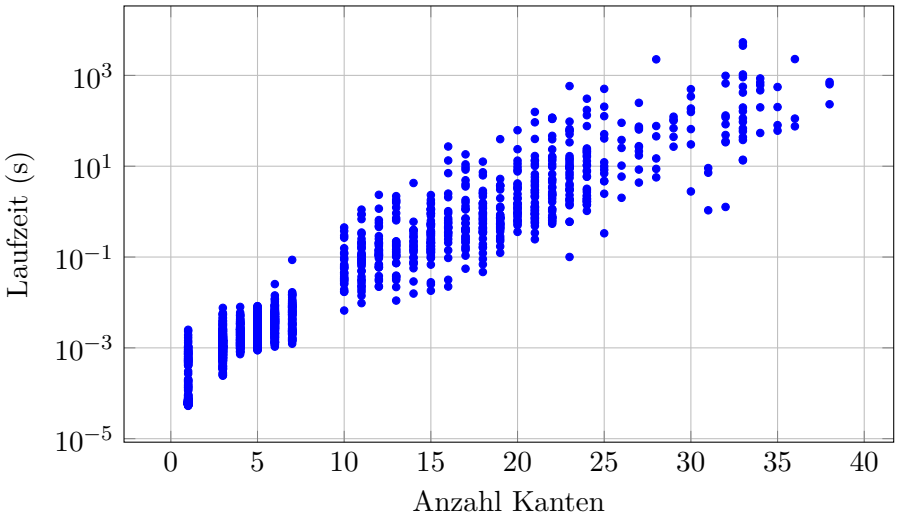


Abb. 2: Laufzeit in Abhängigkeit von der Kantenanzahl

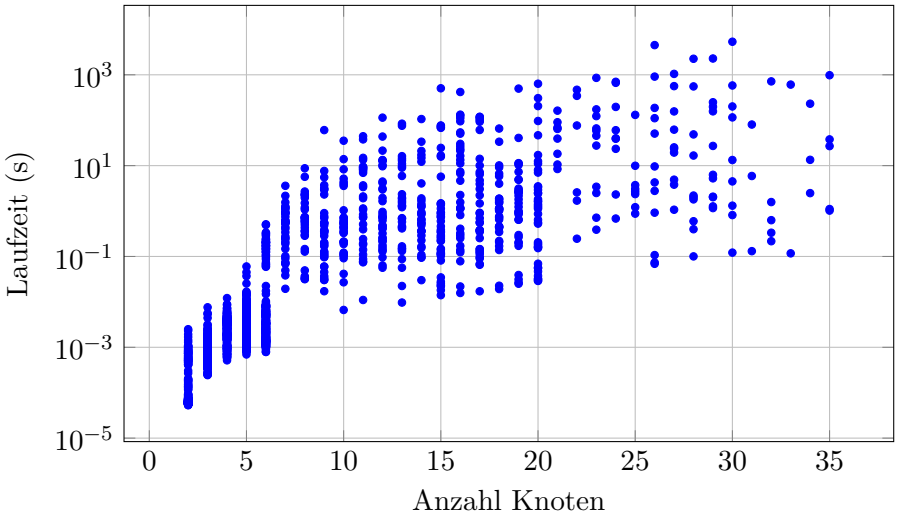


Abb. 3: Laufzeit in Abhängigkeit von der Knotenzahl

Ziel der experimentellen Auswertung ist es, die Laufzeit des entwickelten Lösungsverfahrens systematisch in Abhängigkeit der zentralen Eingabeparametern zu quantifizieren.

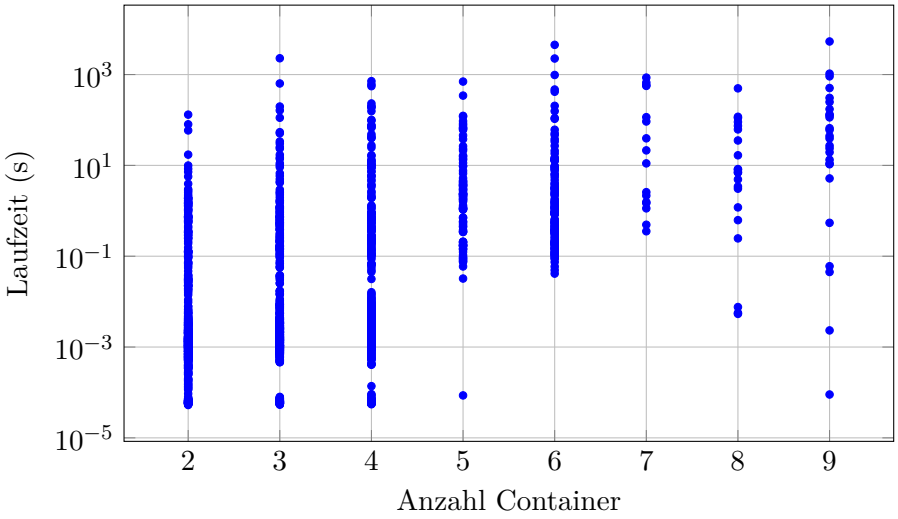


Abb. 4: Laufzeit in Abhängigkeit von der Containeranzahl

Dazu wurden 1.920 Instanzen generiert und ausgewertet. Zur Generierung wurden Graphen mit zufälligen Anzahlen an Knoten erzeugt, und eine zufällige Anzahl an Kanten zwischen zufällig gewählten Knoten eingefügt. Die Anzahl der Container wurde ebenfalls per Zufallsgenerator bestimmt. Die Zahl der Kreuzungen (k) sowie der Skalierungsfaktor (κ) ergeben sich aus der Optimierungsfunktion. Als Zielgröße diente die Laufzeit in Sekunden. Die Berechnungen erfolgten auf einem MacBook Air M3 mit 16 GB RAM unter Verwendung von Gurobi 12.0.2 sowie Java OpenJDK 24.0.1.

5.1. Zielfunktion und Gewichtung

Die Zielfunktion kombiniert Skalierungsfaktor und Kreuzungsanzahl linear ($\alpha = \beta = 1$). Eine differenzierte Gewichtung (z. B. $\alpha \gg \beta$ oder umgekehrt) wurde bisher nicht untersucht. Die vorliegenden Resultate sind daher im Kontext dieser Gleichgewichtung zu interpretieren.

5.2. Zusammenhang zwischen Laufzeit und Struktureigenschaften

Zur Korrelationsanalyse wurde der Pearson-Korrelationskoeffizient (r) der Eingabegrößen berechnet, der den linearen Zusammenhang zwischen zwei Variablen mit Werten zwischen -1 (perfekt negative Korrelation) und $+1$ (perfekt positive Korrelation) misst. Dieser weist auf deutliche Unterschiede zwischen den Eingabegrößen hin. Die Anzahl der Kanten zeigt mit $r = 0.955$ den stärksten linearen Zusammenhang zu $\log_{10}(t)$. Diese ist graphisch gut in Abbildung 2 zu erkennen. Steigt m, so nimmt die Laufzeit nahezu deterministisch zu. Auch die Knotenzahl besitzt einen starken Effekt ($r = 0.832$), ist aber etwas schwächer ausgeprägt. Auch hier ist eine lineare Steigung im Graphen grundsätzlich zu erkennen (siehe Abbildung 3). Der Containerfaktor ($r = 0.541$) beeinflusst die Laufzeit ebenfalls, jedoch in moderater Form wie in Abbildung 4 gezeigt.

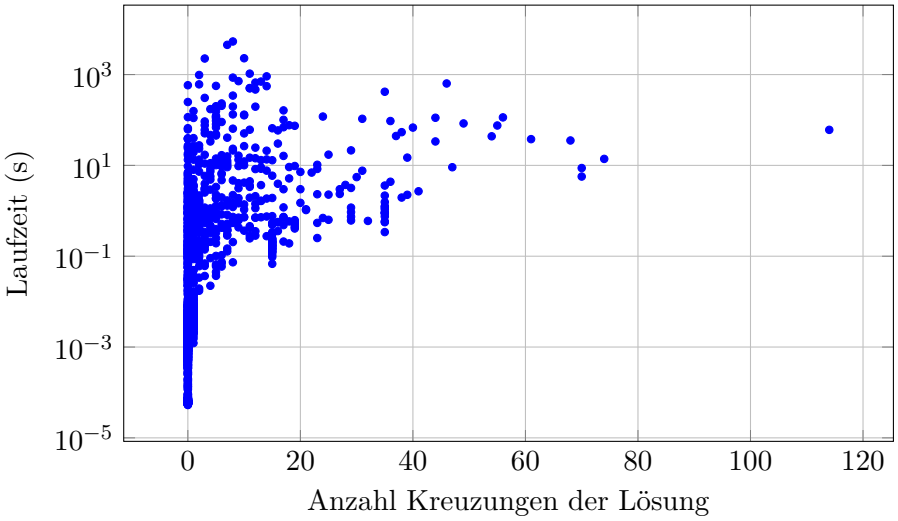


Abb. 5: Laufzeit in Abhängigkeit von der Kreuzungsanzahl

Für Instanzen mit weniger als fünf Knoten beträgt der Median der Laufzeit $1.22{\cdot}10^{-3}$ s. Bereits bei 15–20 Knoten steigt dieser Wert auf 1.14 s. Bei 20–25 Knoten verlängert sich die Laufzeit zu einem Median von 33.38 s. Analog führt ein Anwachsen der Kantenzahl zu sprunghaften Laufzeitsteigerungen. Im Intervall (20,25] Kanten liegt der Median bei 3.83 s, im Intervall (30,35] bereits bei 116.47 s. Diese sprunghaften Übergänge markieren die praxisrelevanten Skalierungsgrenzen des Ansatzes.

5.3. Kreuzungsanzahl und Skalierungsfaktor

Die Kreuzungsanzahl korreliert moderat mit der Laufzeit ($r = 0.514$); siehe Abbildung 5. Der Effekt ist signifikant, bleibt aber klar hinter dem Einfluss von der Kantenanzahl zurück. Methodisch ist zudem zu berücksichtigen, dass Kreuzungen mit steigender Kantenzahl typischerweise zunehmen. Die Hypothese einer fast vollständigen Abhängigkeit muss daher relativiert werden.

Der Kapazitätsfaktor k nimmt meist kleine Werte an (Median = 2, 75-Perzentil = 3). sSehe Abbildung 6.

Es ist zu beachten, dass Kreuzungen und Skalierungsfaktor in der Zielfunktion identisch gewichtet wurden. Das erklärt, warum beide Parameter in der Optimierung in ähnlicher Weise als Kostentreiber behandelt werden, auch wenn ihre empirische Relevanz für die Laufzeit unterschiedlich stark ausgeprägt ist. Für künftige Arbeiten bietet sich eine systematische Variation von α und β an, um den Effekt asymmetrischer Gewichtungen zu untersuchen.

5.4. Zusammenfassende Bewertung

Die Analyse legt nahe, dass die Kantenzahl der entscheidende Parameter für die Laufzeitkomplexität ist, gefolgt von der Knotenzahl. Containerzahl, Kreuzungen und Ska-

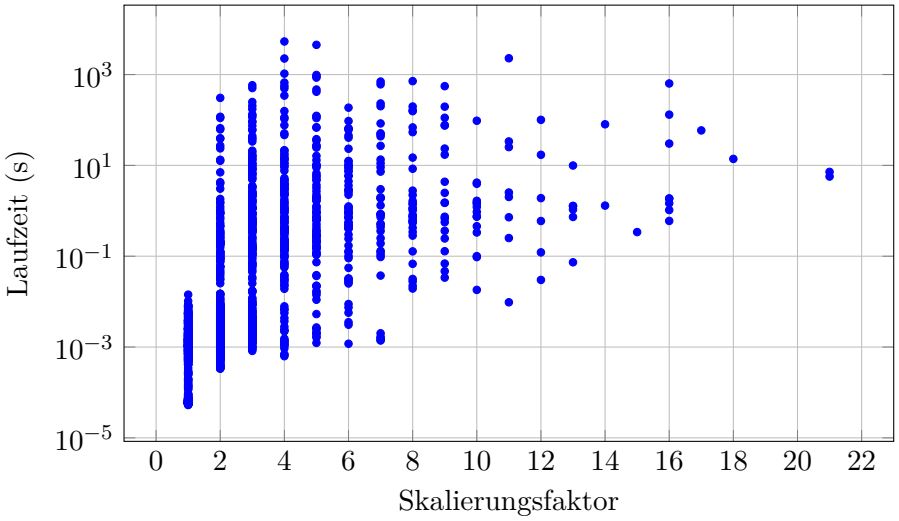


Abb. 6: Laufzeit in Abhängigkeit vom Kapaziätsfaktor

lierungsfaktor beeinflussen die Laufzeit ebenfalls, jedoch schwächer und meist indirekt. Praktisch bedeutet dies, dass Instanzen bis etwa $n \leq 15$ und $m \leq 15$ in interaktiven Szenarien problemlos lösbar sind (Median $t < 1\,\mathrm{s}$). Ab $m \geq 20$ treten Laufzeiten im zweistelligen Sekundenbereich auf, und ab $m \geq 30$ sind selbst mittlere Instanzen oft nur noch mit erheblichem Zeitbudget zu lösen.

Diese Erkenntnisse sind unmittelbar handlungsleitend für den Entwurf heuristischer Verfahren. Effektive Heuristiken sollten vorrangig darauf abzielen, die Zahl der relevanten Kanten zu reduzieren, z. B. durch Vorfilterung oder Strukturierung der Eingabe. Erst in zweiter Linie sind Kreuzungsreduktion oder Containerstrategien relevant. Dabei ist die gleichgewichtete Zielfunktion stets mitzudenken, da sie die Optimierungsausrichtung vorgibt.

5.5. Experimente mit großen Instanzen

Um die Skalierbarkeit des ILP-Ansatzes weiter auszuloten, wurden ergänzende Tests mit größeren Graphen durchgeführt. Die Instanzen umfassten bis zu 32 Knoten und 40 Kanten. Für jedes Experiment wurde eine maximale Laufzeit von zwei Stunden vorgegeben, um auch bei schwierigen Instanzen eine terminierende Auswertung zu gewährleisten.

Die Ergebnisse zeigen, dass das Modell für Graphgrößen oberhalb von $n = 32$ und $m = 40$ die Laufzeitbeschränkung regelmäßig überschritt (in 4 von 5 Tests) und keine optimalen Lösungen mehr berechnet werden konnten. Diese Beobachtung deckt sich mit den in Abschnitt 5 gewonnenen Erkenntnissen: Die Zahl der Kanten erweist sich als dominanter Laufzeittreiber, der ab einer bestimmten Schwelle selbst leistungsfähige ILP-Solver wie Gurobi überfordert.

Für die Praxis bedeutet dies, dass exakte Verfahren nur für kleine bis mittlere Graphen einsetzbar sind. Für größere Instanzen sind heuristische Ansätze, wie sie in Kapitel 6

entwickelt wurden, unverzichtbar.

6. Heuristik und Ausblick

6.1. Motivation

Die in Kapitel 5 präsentierten Ergebnisse verdeutlichen, dass die exakte Lösung mittels ILP nur für kleine bis mittlere Instanzen praktikabel ist. Bereits bei $m \geq 20$ treten Laufzeiten im zweistelligen Sekundenbereich auf, ab $m \geq 30$ sind selbst mittlere Instanzen nur noch mit erheblichem Zeitbudget lösbar. Eine praktikable Heuristik muss daher vorrangig die *dominanten Laufzeittreiber*, d. h. die Anzahl der Kanten und die Kreuzungsstruktur, adressieren. Containerzahl und Skalierungsfaktor sind zwar ebenfalls relevant, jedoch von geringerer Wirkung und größtenteils gekoppelt mit der Kantenanzahl.

6.2. Konstruktionsheuristik: Maximum-Adjacency

Für die Erzeugung einer Startlösung wird eine Ordnung nach dem Prinzip der *Maximum Adjacency Ordering* (MAO) verwendet [Nag02]. In jedem Schritt wird dabei der Knoten ausgewählt, der die größte Zahl von Kanten zu bereits platzierten Knoten aufweist. Dieses Greedy Verfahren führt zu einer Anordnung, in der stark verbundene Knoten frühzeitig und in unmittelbarer Nähe zueinander positioniert werden. Dadurch entstehen lokale Cluster mit kurzen Kanten, was im Kreislayout tendenziell zu weniger Kreuzungen führt. Obwohl MAO ursprünglich zur Analyse und Augmentierung der Konnektivität entwickelt wurde, zeigt sich ihre Wirksamkeit auch für das Kreuzungsproblem als heuristischer Startpunkt.

6.3. Lokale Verbesserung: Swapping nach Baur und Brandes

Auf die initiale Ordnung folgt eine lokale Optimierung, die auf *Swapping*-Operationen basiert. Methodisch wird hier das *circular sifting* nach Baur und Brandes [BB05] aufgegriffen. Dabei wird jeder Knoten sukzessive entlang der Kreisordnung verschoben, wobei jede mögliche Position auf ihre Auswirkung auf die Kreuzungsanzahl überprüft wird. Die Operation entspricht strukturell einem Swap oder 2-Opt, ist jedoch speziell für kreisförmige Layouts und Gruppierungszwänge entwickelt. Iterativ werden so Verbesserungen durchgeführt, bis ein lokales Optimum erreicht ist.

6.4. Zusammenfassung der Heuristik

Die vorgeschlagene Heuristik besteht somit aus zwei Phasen:

1. **Konstruktion:** Erzeugung einer Startordnung mit Maximum-Adjacency, um hochvernetzte Knoten zu clustern und so die Zahl der Kreuzungen von Beginn an niedrig zu halten.

2. **Verbesserung:** Anwendung des *circular sifting* nach Baur und Brandes, das mittels Swapping-Operationen sukzessive Optimierungen der Ordnung erzielt.

Diese Kombination adressiert die dominanten Laufzeittreiber direkt: die Kantenzahl wird implizit durch Clustering reduziert, die Kreuzungen werden durch lokale Optimierung gezielt minimiert.

6.5. Ausblick

Für weiterführende Arbeiten ergeben sich mehrere Forschungsrichtungen:

- **Visualisierung und Gate-Abschnitte**: Die Modellierung der Gate-Abschnitte wurde im ILP bisher nicht beachtet und könnte zusammen mit einer Visualisierung als Thema einer Arbeit dienen. Eine Idee zur Modellierung wäre es hier Containern bereits feste Dummy-Knoten zuzuweisen, die deren Kapazität vollständig ausfüllen und die entsprechenden Kanten nach außen mit ins ILP integrieren.
- **Gewichtete Zielfunktion:** Bisher wurden Kreuzungsanzahl und Skalierungsfaktor gleich gewichtet ($\alpha = \beta = 1$). Eine systematische Untersuchung asymmetrischer Gewichtungen könnte die Anwendbarkeit auf verschiedene Problemklassen erhöhen.
- **Problemreduktion:** Die Abbildung von Teilproblemen auf bekannte NP-schwere Probleme (z. B. Feedback Arc Set oder Crossing Minimization) verspricht theoretische Einsichten und neue algorithmische Heuristiken.

Fazit

Die Kombination von Maximum-Adjacency und Swap-basierter Optimierung bietet einen gangbaren Weg, große Instanzen in akzeptabler Zeit mit geringer Abweichung vom Optimum zu behandeln. Die hier vorgestellte Heuristik adressiert die entscheidenden Laufzeittreiber und bildet damit eine belastbare Grundlage für praktische Anwendungen sowie für weiterführende Forschungsarbeiten.

A. Quellcode

A.1. Repository und Datensatz

Der vollständige Quellcode, sowie der verwendete Datensatz ist im begleitenden ZIP enthalten.

Literatur

[ADM+19] Lorenzo Angori, Walter Didimo, Fabrizio Montecchiani, Daniele Pagliuca und Alessandra Tappini: ChordLink: A New Hybrid Visualization Model. In: Daniel Archambault und Csaba D. Tóth (Herausgeber): *Graph Drawing and Network Visualization (GD 2019)*, Band 11904 der Reihe *Lecture Notes in Computer Science*, Seiten 276–290. Springer, Cham, 2019, ISBN 978-3-030-35802-0, 10.1007/978-3-030-35802-0_21.

[BB05] Michael Baur und Ulrik Brandes: Crossing Reduction in Circular Layouts. In: Juraj Hromkovič, Manfred Nagl und Bernhard Westfechtel (Herausgeber): *Graph-Theoretic Concepts in Computer Science (WG 2004)*, Band 3353 der Reihe *Lecture Notes in Computer Science*, Seiten 332–343. Springer, Berlin, Heidelberg, 2005, ISBN 978-3-540-30559-0, 10.1007/978-3-540-30559-0_28. https://doi.org/10.1007/978-3-540-30559-0_28.

[DGDMT21] Emilio Di Giacomo, Walter Didimo, Fabrizio Montecchiani und Alessandra Tappini: A User Study on Hybrid Graph Visualizations. In: Helen C. Purchase und Ignaz Rutter (Herausgeber): *Graph Drawing and Network Visualization (GD 2021)*, Band 12868 der Reihe *Lecture Notes in Computer Science*, Seiten 21–38. Springer, Cham, 2021, ISBN 978-3-030-92930-5, 10.1007/978-3-030-92931-2_2.

[GJ79] Michael R. Garey und David S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1979, ISBN 0716710447.

[Goo22] Arash Torabi Goodarzi: Crossing Reduction in Circular Layouts under Grouping Constraints. Master's Thesis, Julius-Maximilians-Universität Würzburg, Würzburg, Germany, 2022. https://www1.pub.informatik.uni-wuerzburg.de/pub/theses/2022-torabi-goodarzi-bachelorarbeit.pdf.

[Gur24] Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual, 2024. https://www.gurobi.com.

[HFM07] Nathalie Henry, Jean Daniel Fekete und Michael J. McGuffin: NodeTrix: A Hybrid Visualization of Social Networks. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1302–1309, 2007, 10.1109/TVCG.2007.70582.

[KV18] Bernhard Korte und Jens Vygen: *Bin-Packing*, Band 21 der Reihe *Algorithms and Combinatorics*, Seiten 489–507. Springer, 6th Auflage, 2018, 10.1007/978-3-662-56039-6_18.

[MKNF87] Sumio Masuda, Toshinobu Kashiwabara, Kazuo Nakajima und Toshio Fujisawa: On the NP-completeness of a computer network layout problem. In:

Proc. IEEE Intl. Symp. Circuits and Systems, Seiten 292–295, 1987. Technical report available at `https://api.drum.lib.umd.edu/server/api/core/bitstreams/7e6cd544-c6ea-4093-a51c-a4a1ddb03f86/content`.

[Nag02] Hiroshi Nagamochi: Graph connectivity and its augmentation: Applications of maximum adjacency ordering. *Discrete Applied Mathematics*, 126(1):67–86, 2002, 10.1016/S0166-218X(01)00349-3.