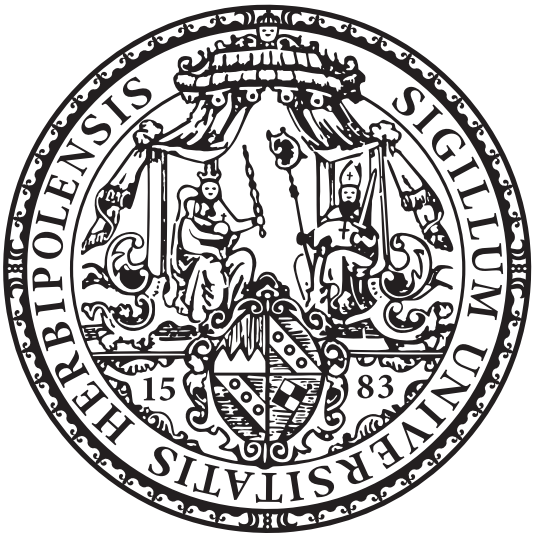
Bachelorarbeit

Automatisiertes Zeichnen von Graphen unter Verwendung von neuronalen Netzen

Timo Säckl

Abgabedatum:	8. August 2024
Betreuer:	Prof. Dr. Alexander Wolff
	Dr. Johannes Zink



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

Zusammenfassung

Ein Graph ist eine abstrakte Datenstruktur, die Zusammenhänge in Systemen repräsentiert. In der Graphenvisualisierung werden Zeichnungen von Graphen generiert. Die Suche nach einer optimalen Zeichnung des Graphen kann allgemein als ein Optimierungsproblem angesehen werden. Im Graphzeichnen werden Methoden verwendet, die Zeichnungen von Graphen nach bestimmten Metriken analysieren und optimieren. Ferner approximieren neuronale Netzwerke durch gegebene Erfahrungswerte Optimierungsprobleme. In den letzten Jahren wurden neuronale Netzwerke entwickelt, welche aus der Eingabe eines Graphen eine dazugehörende Zeichnung generieren und dabei Metriken der Graphenvisualisierung optimieren. Diese Methoden erzielen stets verbesserte Ergebnisse. Grötschla et al. [GMVW24] haben ein neuronales Netzwerk CoRe-GD entwickelt, welches den Stress einer Graphzeichnung optimiert.

In dieser Arbeit wird CoRe-GD mit klassischen Methoden der Graphenvisualisierung kombiniert, um eine verbesserte Zeichnung zu liefern. Wir haben die Methoden so ausgewählt, dass diese initiale Positionen aufnehmen können. Daher verwenden wir das neuronale Netzwerk und die klassischen Methoden sowohl als Postprocessing-Methoden als auch als Preprocessing-Methoden. Die Preprocessing-Methode liefert Positionen und diese werden von der Postprocessing-Methode als initiale Werte aufgenommen. Die Postprocessing-Methode liefert das finale Ergebnis der Kombination. Wir verwenden mehrere Metriken der Graphenvisualisierung, um die Güte der Zeichnungen der einzelnen Methoden und der Kombinationen bewerten zu können.

Unsere Experimente umfassen zwei Arten von Kombinationen. Die Kombinationen, wobei die klassische Methode der Graphenvisualisierung als Preprocessing-Methode und CoRe-GD als Postprocessing-Methode aufgefasst werden, liefern keinen Vorteil gegenüber einem einzelnen CoRe-GD. Allerdings lassen sich die Zeichnungen von Spring von Fruchterman und Reingold [FR91] und des ForceAtlas2 von Jacomy et al. [JVHB14] bezüglich fast aller hier benutzten Metriken durch CoRe-GD optimieren. In der zweiten Kombination ist das neuronale Netzwerk als Preprocessing-Methode ausgewählt. Der klassische Algorithmus definiert die Postprocessing-Methode. Unsere Experimente zeigen, dass Zeichnungen bezüglich bestimmter Metriken von CoRe-GD durch den Algorithmus ForceAtlas2 optimiert werden können, wobei die Kantenkreuzungen um ca. 23% und die durchschnittliche Abweichung des minimalen Winkels vom optimalen Winkel um ca. 13% gegenüber einem einzelnen CoRe-GD verbessert werden können. Gegenüber dem ForceAtlas2 optimiert die Kombination mehrere Metriken um ca. 6–37%. Allerdings werden andere Metriken der Graphzeichnungen der Kombination negativ beeinflusst.

Mit den hier in den Experimenten genutzten getesteten Algorithmen $(sgd)^2$ von Zheng et al. [ZGP17] und der multidimensionalen Skalierung von Kamada und Kawai [KK89] ist bei deren Kombinationen mit CoRe-GD kaum eine Verbesserung der Zeichnungen festzustellen. Durch die Kombination mit CoRe-GD werden die Laufzeiten der klassischen Methoden außerordentlich erhöht.

Abstract

A graph is an abstract data structure that represents relationships within systems. In graph drawing, drawings of graphs are generated. The search for an optimal graph drawing can generally be considered as an optimization problem. Graph drawing methods are used to analyze and optimize drawings using particular metrics. Furthermore, neural networks approximate optimization problems by given empirical values. In recent years, neural networks have been developed, which generate a drawing of a given graph by optimizing the graph drawing metrics. These methods provide more and more better results. Grötschla et al. [GMVW24] have developed a neural network CoRe-GD, which optimizes the stress of a graph drawing.

In this work, CoRe-GD is combined with classical graph drawing algorithms to provide a better drawing. We have chosen methods, which can use initial positions. Therefore, we use the neural network and classical methods as preprocessing and postprocessing methods. The preprocessing method provides the positions, which are used from the postprocessing method to provide the final positions of the combination. We use several metrics of graph drawing to evaluate the quality of the drawings of each method and the combinations.

Thus, our experiments include two directions of combinations. The combinations, where the classical method as a preprocessing method and CoRe-GD as a postprocessing method, provide no advantage over using CoRe-GD only. However, the drawing of Spring by Fruchterman and Reingold [FR91] and of ForceAtlas2 from Jacomy et al. [JVHB14] can be optimized by CoRe-GD in almost all metrics used here. In the second type of combinations, the neural network is selected as the preprocessing method. The classical method defines the postprocessing method. Our experiments show that combining CoRe-GD and ForceAtlas2 reduces edge crossings by approx. 23% and the average deviation of the minimum angle from the optimum angle by approx. 13% compared to the classical CoRe-GD. However, in other aesthetic metrics, the drawings are negatively affected compared with CoRe-GD. Compared to a single a single run of ForceAtlas2, the combination improves several metrics by approx. 6–37%.

With the combinations of the remaining algorithms, $(sgd)^2$ by Zheng et al. [ZGP17] and the multidimensional scaling by Kamda and Kawai [KK89], nearly no improvements of the drawings can be observed. Through combinations with the neural network, the runtimes of the classical methods are extraordinarily increased.

Inhaltsverzeichnis

1. Einleitung

Ein *Graph* ist eine abstrakte Datenstruktur, die Zusammenhänge in Systemen repräsentiert. Objekte, die in diesen Systemen agieren, werden abstrakt *Knoten* genannt. Eine *Kante* zwischen den Knoten bezeichnet eine Relation zwischen den Objekten.

Eine Zeichnung eines Graphen ermöglicht ein besseres Verständnis der Wechselwirkungen in dem System. Es gibt keine *beste* Zeichnung eines Graphen, denn, je nach Anwendungszweck, muss die Zeichnung bestimmte Merkmale betonen. In verschiedenen Studien ist gezeigt worden, dass durch das Hervorheben bestimmter Merkmale der Zeichnung eines Graphen eine bessere Erfassung der Struktur durch einen Menschen erfolgen kann. Hier sind zu nennen:

- Je größer der Winkel zwischen zwei adjazenten Kanten ist, desto leichter ist es, sie zu unterscheiden [CP96][WPCM02].
- Die Kanten sollten einheitlich lang sein [CP96].
- Kantenkreuzungen erschweren das visuelle Verfolgen der Kanten [Pur97].

In der Graphenvisualisierung werden Methoden benutzt, die es einem ermöglichen die Zeichnung eines Graphen bezüglich bestimmter Merkmale zu optimieren. Der *Shift*-Algorithmus von de Fraysseix et al. [DFPP90] optimiert die Zeichenfläche einer planaren Graphzeichnung. Zink et al. [ZWBW22] kombinierten das Sugiyama-Framework [STT81] mit einem kräftebasierten Algorithmus, um einen ungerichteten Graphen zu orientieren und so eine gute Ausgangslage für eine lagenbasierte Zeichnung mit wenigen Kreuzungen zu schaffen. Shiono et al. [SYT22] optimieren Zeichnungen von Fuzzy-Graphen mithilfe von Partition-Trees. Xu und Zhang [XZ19] verbessern die Zeichnung eines kräftebasierten Algorithmus mithilfe des PageRank-Algorithmus, um Knoten abhängig des Knotengrades zu gewichten. Walshaw [Wal01] kombiniert einen kräftebasierten Algorithmus mit einer Multilevel-Technik, in welcher Cluster von Knoten gebildet und rekursiv verfeinert werden.

Motiviert von diesen Arbeiten untersuchen wir in unserer Arbeit, ob eine Kombination eines neuronalen Netzwerks mit klassischen Algorithmen zum Graphzeichnen eine bessere Zeichnung liefert als die Methoden einzeln. Als neuronales Netzwerk benutzen wir das Modell von Grötschla et al. [GMVW24] *CoRe-GD*, welches aus der Eingabe eines Graphen eine Zeichnung generiert. Dieses Modell wurde darauf trainiert, den *Stress* einer Zeichnung zu optimieren. Der Stress ist ein Maß, das bestimmt, wie hoch die Diskrepanz zwischen der graphentheoretischen Distanz und der euklidischen Distanz zweier Knoten ist [Kru64]. Als einen klassischen kräftebasierten Algorithmus wählen wir den Spring-Embedder (kurz *Spring*) von Fruchterman und Reingold [FR91], deren Methode auf dem

Algorithmus von Eades [Ead84] basiert. Ferner verwenden wir den kräftebasierten Algorithmus *ForceAtlas2* von Jacomy et al. [JVHB14] und die multidimensionale Skalierung von Kamada und Kawai [KK89]. Zusätzlich verwenden wir als klassische Methode den $(\text{sgd})^2$ von Zheng et al. [ZGP17], welche durch den Gradientenabstieg den Stress einer Zeichnung optimiert.

Das neuronale Netzwerk und die klassischen Methoden liefern uns Positionen und diese können von den Methoden als initiale Positionen aufgenommen werden. Aus diesem Grund werden die Kombinationen in beiden Richtungen betrachtet. Dies bedeutet, dass CoRe-GD und die klassischen Methoden sowohl als Postprocessing-Methode als auch als Preprocessing-Methode verwendet werden. Die Preprocessing-Methode liefert Positionen, die die Postprocessing-Methode als initiale Werte aufnimmt, um die finale Zeichnung der Kombination zu generieren.

Wir verwenden mehrere Metriken der Graphenvisualisierung, um eine Aussage über die Güte der Zeichnungen aller Methoden treffen zu können. Neben dem Stress analysieren wir die Graphzeichnung nach u. a. Kantenkreuzungen, Kantenlängenverhältnis und Winkelauflösung.

In Kapitel 2 werden die Grundlagen für diese Arbeit eingeführt. Im ersten Abschnitt werden die Grundzüge der Graphenvisualisierung und ein Teil der hier benutzten Algorithmen vorgestellt. Wir verwenden den *kräftebasierten* Algorithmus Spring von Fruchterman und Reingold [FR91], welcher auf den Spring-Embedder von Eades [Ead84] zurückgeht. Zusätzlich wird die multidimensionale Skalierung von Kamada und Kawai [KK89] vorgestellt. In kräftebasierten Algorithmen werden physikalische Kräfte zwischen Knotenpaaren modelliert. Darauffolgend werden klassische neuronale Netzwerke eingeführt, welche mithilfe von gegebenen Erfahrungswerten Optimierungsprobleme approximieren. Im letzten Unterabschnitt werden wir graphbasierte neuronale Netzwerke vorstellen, welche Graphen als Eingabe verarbeiten können. Der Abschnitt danach soll als eine Verbindung zwischen Graphenvisualisierung und graphbasierten neuronalen Netzwerken dienen. Anschließend wird das neuronale Netzwerk CoRe-GD von Grötschla et al. [GMVW24] eingeführt. Hierbei werden zuerst die verschiedenen Module des Algorithmus und die Architektur dargestellt. In den letzten Unterabschnitten werden die Fehlerfunktion und die Komplexität von CoRe-GD beleuchtet.

Als Methodik für diese Arbeit wurde die Analyse der Graphzeichnungen der Kombinationen und der einzelnen Methoden nach bestimmten Metriken ausgewählt. Diese wird in Kapitel 3 dargestellt. Im ersten Abschnitt dieses Kapitels werden die Algorithmen vorgestellt, welche wir ebenfalls mit CoRe-GD kombinieren: ForceAtlas2 von Jacomy et al. [JVHB14] und $(\text{sgd})^2$ von Zheng et al. [ZGP17]. Im zweiten Abschnitt wird die Funktionsweise der Positionsübergabe eines klassischen Algorithmus an CoRe-GD erläutert. Zuletzt werden wir die Metriken und deren Implementierung beschreiben.

In Kapitel 4 werden wir unsere Untersuchungsergebnisse vorstellen und in Kapitel 5 diskutieren. Ferner werden ein Fazit gezogen und Ziele für weitere Forschungsarbeiten definiert.

In Kapitel 6 wird die Arbeit zusammengefasst. Im Anhang befinden sich eine Auswahl von Zeichnungen von Graphen, die von CoRe-GD, den klassischen Methoden und

deren Kombinationen erstellt wurden. Ferner sind die Ergebnisse der Metriken zu den Zeichnungen aufgeführt.

2. Grundlagen

Zur Einführung der Begriffe der graphentheoretischen Objekte stützen wir uns auf Krumke und Noltemeier [SOK12]. Ein *ungerichteter Graph* $G = (V, E)$ ist ein geordnetes Paar, das aus einer endlichen Menge von *Knoten* V und einer endlichen Menge von *Kanten* E besteht. Eine Kante e aus der Menge E ist ein ungeordnetes Paar $\{u, v\}$, wobei u und v aus V stammen. Je zwei Knoten u und v sind *adjazent*, falls $\{u, v\}$ ein Element von E ist. Eine Kante $\{u, v\}$ ist *inzident* zu den Endknoten der Kante u und v. Zwei Kanten $\{u, v\}$ und $\{v, w\}$ sind adjazent zueinander, wenn sie zu demselben Knoten v inzident sind. Die *Nachbarschaft* $N(v)$ des Knoten v ist die Menge aller Knoten, für die gilt, dass sie adjazent zu v sind. Eine *Adjazenzmatrix* $A = (a_{ij})$ ist eine quadratische Matrix, wobei für jeden Eintrag gilt

$$a_{ij} = \begin{cases} 1, & \text{falls } \{i, j\} \in E \\ 0, & \text{sonst} \end{cases}$$

Der *Grad* $\deg(v)$ eines Knotens v ist die Anzahl der Nachbarn von v, d. h. $|N(v)|$. Ein *Pfad* $d_{v_0 v_n}$ ist eine Menge von wohlunterscheidbaren Knoten $\{v_0, v_1, \ldots, v_n\} \in V$, wobei $|V| \geq 2$, so dass v_0 und v_n durch eine Sequenz aus adjazenten Kanten d. h. $\{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{n-1}, v_n\} \in E$. Ein Pfad d_{uv} wird *kürzester Pfad* genannt, wenn die Anzahl der Kanten der Sequenz unter alle Pfaden zwischen u und v minimal ist. Ein Pfad d wird *Zyklus* genannt, wenn für diesen gilt $\{v_0, v_1\}, \ldots, \{v_{n-1}, v_n\}, \{v_n, v_0\} \in E$, d. h. Anfangs- und Endkante sind zum selben Knoten inzident. Andernfalls wird der Pfad d *azyklisch* genannt. Die *Länge* l eines Pfads d_{uv} ist die Anzahl der Kanten. Ein Graph G, der keinen Zyklus enthält, wird *azyklisch* genannt. Ein Graph G heißt *zusammenhängend*, falls für alle Knoten $u, v \in V$ gilt, dass ein Pfad $d_{uv} \in G$ existiert. Ein gewurzelter *Baum* T ist ein azyklischer, zusammenhängender Graph mit einem ausgezeichneten Knoten v, der *Wurzel* des Baumes genannt wird. Die Knoten des Baumes, die nur zu *einem* Knoten adjazent sind und nicht die Wurzel sind, werden *Blätter* des Baumes genannt. Die *Tiefe* l eines Baumes T wird durch die Länge des längsten Pfads d_{vb} von der Wurzel v zu einem Blatt b gekennzeichnet. Die *Breitensuche* ist ein Verfahren, bei dem die kürzesten Pfade aller Knoten ausgehend von einem Startknoten wellenförmig gefunden werden.

2.1. Graphenvisualisierung

Zur Einführung stützen wir uns auf Kobourov [Kob14]. Sei G ein Graph. Die Graphenvisualisierung beschäftigt sich mit geometrischen Repräsentationen von Graphen, sog. *Zeichnungen*.

Eine Zeichnung Γ ist definiert als

$$\Gamma : V \to \mathbb{R}^d, v \mapsto \Gamma(v)$$

wobei Γ jeden Knoten v aus V auf eine Position des Knoten $\Gamma(v)$ der Zeichnung abbildet und d stellt die Dimension der Repräsentation dar. Eine Kante e aus E in G wird in der Zeichnung als eine Verbindungsstrecke mit den Endknoten $\Gamma(u)$ und $\Gamma(v)$ dargestellt, gekennzeichnet als $(\Gamma(v), \Gamma(u))$. Wir beschräken uns hier auf gradlinige Abbildungen der Kanten.

2.1.1. Kräftebasierte Algorithmen zum Graphzeichnen

Ob eine Zeichnung als „gut“ wahrgenommen wird, hängt davon ab, welche Merkmale der Zeichnung Γ betont werden sollen. *Kräftebasierte Algorithmen* entfalten eine initiale Zeichnung eines Graphen mithilfe von modellierten physikalischen Kräften. Eine gleichmäßige Verteilung aller Knoten auf der Zeichnungsfläche ist unter anderem das Ziel von Algorithmen, die auf Kräfte basieren. Vor allem, dass adjazente Knoten nah beieinander liegen, während nicht-adjazente Knoten weit voneinander entfernt liegen. Die Methode von Eades [Ead84] modelliert solche physikalischen Kräfte. In dieser werden iterativ abstoßende Kräfte $f_{\text{rep}}(u, v)$ zwischen nicht-adjazenten Knoten und anziehende Kräfte $f_{\text{attr}}(u, v)$ zwischen adjazenten Knoten simuliert. Zur Veranschaulichung der Modellierung physikalischer Kräfte in Algorithmen wollen wir die Variante des *Spring-Embedders* von Fruchterman und Reingold [FR91] vorstellen, welcher wiederum auf dem Algorithmus von Eades [Ead84] basiert.

Seien ein Graph $G = (V, E)$, eine initiale Zeichnung Γ von G, eine Zahl $\varepsilon > 0$ und eine Zahl $K \in \mathbb{N}$, die die maximale Iteration der Entfaltungsschritte bezeichnet, gegeben. Sei t die Iterationsvariable und initial $t := 1$. Vor jedem Iterationsschritt wird überprüft, ob $t < K$ und ob die maximale Kraft, die auf einen Knoten wirkt, größer als ε ist.

Seien $t \geq 1$, die Zahl ℓ bezeichnet die ideale Kantenlänge für Kanten und $\|\cdot\|_2$ die euklidische Norm. Für jeden Knoten u aus V wird eine resultierende Kraft $F_u(t)$, die sich aus der Summe aller abstoßenden Kräfte $f_{\text{rep}}(u, v)$ zu jedem Knoten v aus V und der Summe aller anziehenden Kräfte $f_{\text{attr}}(u, v)$, wobei $v \in N(u)$ gilt, zusammensetzt. Für die abstoßende Kraft $f_{\text{rep}}(u, v)$ gilt:

$$f_{\text{rep}}(u, v) = \frac{\ell^2}{\|\Gamma(v) - \Gamma(u)\|_2} \cdot \overrightarrow{\Gamma(u)\Gamma(v)}.$$

Das bedeutet, je größer die euklidische Distanz $\|\Gamma(v) - \Gamma(u)\|_2$, desto schwächer ist die abstoßende Kraft $f_{\text{rep}}(u, v)$. Für die anziehende Kraft $f_{\text{attr}}(u, v)$ bezüglich der Nachbarknoten Knoten v aus $N(u)$ gilt:

$$f_{\text{attr}}(u, v) = \frac{\|\Gamma(v) - \Gamma(u)\|_2^2}{\ell} \cdot \overrightarrow{\Gamma(u)\Gamma(v)}.$$

Dies bedeutet, je größer die euklidische Distanz $\|\Gamma(v) - \Gamma(u)\|_2$, desto stärker ist die anziehende Kraft $f_{\text{attr}}(u, v)$ zwischen dem adjazenten Knotenpaar u und v.

Andere kräftebasierte Algorithmen sind u. a. der Algorithmus von Tutte [Tut63], welcher Schwerkraftzentren bzw. *Baryzentren* modelliert.

2.1.2. Multidimensionale Skalierung zum Graphzeichnen

Multidimensionale Skalierung (MDS) bezieht sich auf Methoden der Reduktion der Dimension eines Raumes $\mathbb{R}^n$ auf einen Raum niedriger Dimension $\mathbb{R}^d$, wobei $n \gg d$, und analysiert die *Ähnlichkeit* der Objekte in diesen Räumen, z. B. siehe Shepard [She62]. Ziel ist es, dass die Ähnlichkeit, d h. der Abstand, δ_{ij} der Objekte i, j im ursprünglichen Raum $\mathbb{R}^n$ der Ähnlichkeit der Objekte im niedrigdimensionalen Raum $\mathbb{R}^d$ möglichst genau entspricht. Bei der multidimensionalen Skalierung im Graphzeichnen wird üblicherweise die Ähnlichkeit eines Knotenpaares $u, v \in V$ für ein Graphen G durch die Distanz des kürzesten Pfads $\delta_{uv} := d_{uv}$ charakterisiert [GHN13].

Sei G ein Graph und eine Zeichnung $\Gamma \subset \mathbb{R}^d$ für $1 \leq i, j \leq n$ und $n \in \mathbb{N}$, wobei ohne Beschränkung der Allgemeinheit gilt $d := 2$. Seien ferner eine Menge von Knoten $\{v_1, \ldots, v_n\} \in V$, eine Distanzmatrix $\Delta := (d_{v_i v_j})$, wobei jeder Eintrag die Distanz des kürzesten Pfads $d_{v_i v_j}$ zwischen den Knoten v_i und v_j im Graphen G bezeichnet, und die Positionen der Knoten in der Zeichnung $\Gamma(v_i) \in \mathbb{R}^2$ gegeben. Eine bevorzugte MDS-Variante für das Graphzeichnen ist es, den *Stress* einer Zeichnung eines Graphen zu optimieren [KB13][GKN04]. Eine Optimierung der Stress-Funktion bedeutet, die geometrischen Repräsentationen $\Gamma(v_i)$ und $\Gamma(v_j)$ räumlich auf der Zeichenfläche so anzuordnen, dass die euklidische Distanz $\|\Gamma(v_i) - \Gamma(v_j)\|_2$ zwischen den Koordinaten $\Gamma(v_i)$ und $\Gamma(v_j)$ der graphentheoretischen Distanz $d_{v_i v_j}$ zwischen v_i und v_j im Graphen G möglichst genau entspricht.

Der Spring-Embedder von Eades [Ead84] hat das Ziel, die Knoten so anzuordnen, dass adjazenten Knoten nah beieinander, während nicht-adjazente Knoten weit voneinander entfernt liegen sollen. In der Methode von Kamada und Kawai [KK89] wird ein anderer Ansatz gewählt: Die graphentheoretische Distanz zwischen zwei Knoten u und v im Graphen G und die euklidische Distanz der Knoten $\Gamma(u)$ und $\Gamma(v)$ in der Zeichnung Γ sollen möglichst genau entsprechen. Laut Kamada und Kawai [KK89] ist die optimale Anordnung der Knoten in der Zeichnung Γ erreicht, wenn die „Energie“ des Systems minimal ist, d. h. wenn die graphentheoretische Distanz der euklidischen Distanz möglichst genau entspricht. Wir bezeichnen diese Energie von Kamada und Kawai als Stress, der von Kruskal [Kru64] vorgestellt wurde. Der Stress wird als Anlehnung an Grötschla et al. [GMVW24] für diese Arbeit folgendermaßen definiert: Die optimale Zeichnung Γ des Graphen G minimiert dann die Stressfunktion

$$\text{stress}(G, \Gamma) := \sum_{(u,v) \in V, u \neq v} d_{uv}^{-2} (\|\Gamma(u) - \Gamma(v)\|_2 - d_{uv})^2, \tag{2.1}$$

wobei d_{uv} die Länge des kürzesten Pfads in G zwischen den Knoten u und v bezeichnet. Der Ausdruck $\|z\|_2$ bezeichnet die euklidische Norm von z.

2.1.3. Qualitätsmetriken einer Zeichnung

In der Graphenvisualisierung spielen *Qualitätsmetriken* einer Zeichnung eine große Rolle. Eine Qualitätsmetrik einer Zeichnung betont ein bestimmtes Merkmal der Zeichnung Γ. Neben der Minimierung des Stresses gibt es weitere Qualitätsmetriken. Zu nennen sind:

- Die Minimierung der Kantenkreuzungen ist die minimale Anzahl an Überschneidungen zweier nicht-adjazenten Kanten $(\Gamma(u), \Gamma(v))$ und $(\Gamma(r), \Gamma(s))$ in der Zeichnung Γ [BMRW98].
- Die Kanten sollten einheitlich lang sein [FR91].
- Die kürzeste Distanz zwischen einem Knoten $\Gamma(v)$ und einer nicht-inzidenten Kante $(\Gamma(u), \Gamma(w))$ in der Zeichnung Γ, d. h. $v \neq u, w$, soll maximal sein [DH96].
- Der Abstand zwischen einem Knoten $\Gamma(u)$ und einem nicht-adjazenten Knoten $\Gamma(v)$ sollte maximal sein [DH96].
- Die Winkelauflösung, d. h. der kleinste Winkel α zwischen zwei adjazenten Kanten $(\Gamma(u), \Gamma(v))$ und $(\Gamma(v), \Gamma(w))$ in der Zeichnung Γ, sollte groß sein [TR05].
- Der minimale Kreuzungswinkel, das ist der kleinste Winkel α des Graphen G, der entsteht, wenn sich zwei Kanten $(\Gamma(u), \Gamma(v)), (\Gamma(r), \Gamma(w))$ in der Zeichnung Γ schneiden, sollte groß sein [WPCM02].
- Die Abweichung des minimalen Winkels α vom an einem Knoten v vom optimalen Winkel $360/\deg(v)$ sollte minimal sein.

Die Anwendungen, die in der Graphenvisualisierung entwickelt werden, optimieren die Zeichnung Γ des Graphen G bezüglich bestimmter Qualitätsmetriken.

2.2. Neuronale Netzwerke

Maschinelles Lernen ist ein Teilgebiet der künstlichen Intelligenz und verwendet u. a. *Lernalgorithmen*, die durch gegebene Erfahrungswerte ein Problem *lernen* bzw. approximieren.

2.2.1. Klassische neuronale Netzwerke

Zur Einführung des neuronalen Netzwerks als mathematische Funktion nutzen wir Bauckhage et al. [BOS+18] und zur weiteren Einführung Smets [Sme21]. Ein *neuronales Netzwerk* ist eine Funktion $f : X \to Z$ mit $x \mapsto z$, die eine Menge von *Beobachtungen* X erhält und diese auf eine Menge von *Vorhersagen* Z abbildet. Seien x aus $X \subseteq \mathbb{R}^{d_1}$ und z aus $Z \subseteq \mathbb{R}^{d_2}$, wobei d_1 die Dimension der Beobachtungen und d_2 die Dimension der Vorhersagen sind. Ein neuronales Netzwerk f berechnet eine Komposition von Funktionen $f^{(l)}$, die den Parameter x auf den Funktionswert z abbildet, d. h.

$$f(x) = f^{(L)}(\dots f^{(2)}(f^{(1)}(x))) = f^{(L)} \circ \dots f^{(2)} \circ f^{(1)}(x) = z,$$

wobei jede Funktion $f^{(l)}$ als *Layer* l mit $l \in [1, \dots L+1]$ bezeichnet wird. Jedes Layer l enthält mehrere Berechnungseinheiten, die sog. *Neuronen* $f_i^{(l)}$, mit $i \in [0, \dots n]$ und $n \in \mathbb{N}$. Seien l fixiert, wobei $l \in [1, \dots, L+1]$, und $z^{(0)} := x$. Seien ferner i,j fixiert

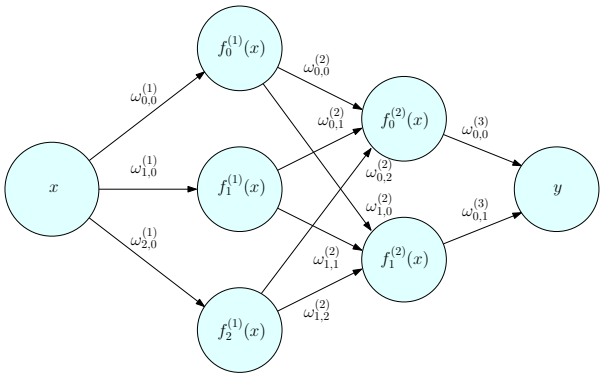


Abb. 2.1.: Eine schematische Darstellung eines neuronalen Netzwerks f mit zwei *Hidden-Layern* (Eigene Darstellung).

und natürliche Zahlen, die die Neuronen in den Layern eindeutig kennzeichnen. Es gilt $j \in [0, \ldots, m]$ mit $m \in \mathbb{N}$. Jedes Neuron $f_i^{(l)}$ des aktuellen Layers (l) erhält als Eingabe die Ausgabe $z^{(l-1)}$ aller Neuronen des vorherigen Layers $(l-1)$ als Vektor der Länge m mit $f^{(l)}(z^{(l-1)}) := \left[f_0^{(l)}(z^{(l-1)}) \ldots f_j^{(l)}(z^{(l-1)}) \ldots f_m^{(l)}(z^{(l-1)})\right]^\top = z^{(l)}$ und einen Vektor der Länge m mit $\Omega_i^{(l)} = \left[\omega_{i,0}^{(l)} \ldots \omega_{i,m}^{(l)}\right]^\top$. Eine Zahl $\omega_{i,j}^{(l)}$ aus $\Omega_i^{(l)}$ wird *Gewicht* genannt und bezeichnet die Relevanz der Ausgabe des Neurons $f_j^{(l-1)}$ als Eingabe für das Neuron $f_i^{(l)}$. Der Berechnungsablauf innerhalb eines Neurons $f_i^{(l)}$ sieht folgendermaßen aus:

$$f_i^{(l)}(z^{(l-1)}) = \sigma(\Omega_i^{(l)} z^{(l-1)} - b) = \sigma(\sum_{j=0}^{n} \omega_{i,j}^{(l)}(z_j^{(l-1)}) - b_i) = z^{(l)}, \tag{2.2}$$

wobei die Funktion σ die sog. *Aktivierungsfunktion* bezeichnet, welche die Ausdruckskraft von $f^{(l)}$ verstärken und das Neuron „aktivieren“ können. Die Funktion $f_i^{(l)}$ ist das i-te Neuron im Layer (l) und $\Omega^{(l)}$ der Gewichtsvektor des i-ten Neurons. Die Aktivierung eines Neurons bedeutet, eine bestimmte *Schwelle*, d. h. einen bestimmten Wert der Aktivierungsfunktion σ zu erreichen, damit die Ausgabe des Neurons $f_i^{(l)}$ als äußerst aussagekräftig für folgende Neuronen gilt. Die Zahl b, der sog. *Bias*, kann allerdings den Ausdruck $\Omega^{(l)} z^{(l-1)}$ herabsetzen. Im *Input-Layer* $l = 0$ werden den Neuronen die Beobachtungen x übergeben. Die Ausgaben der Neuronen des *Output-Layers* $l = L + 1$ werden als Vorhersagen des neuronalen Netzwerks betrachtet. In der Funktion eines Neurons zeigt sich die rekursive Weiterreichung der Werte, die sog. *Propagation* [Ant01]. Eine Veranschaulichung eines neuronalen Netzwerks ist in Abbildung 2.1 dargestellt.

Ein neuronales Netzwerk f approximiert durch Berechnung ein Optimierungsproblem. Die Aufgabe eines neuronalen Netzwerks besteht darin Vorhersagen $f(x)$ an exakte Werte y für gegebene Beobachtungen x im Rahmen einer Fehlerfunktion $\mathcal{L}$ anzunähern, d. h.

$$\min_{\Omega} \mathcal{L}(y, f(x)) \text{ für alle } x \in \mathbb{R}^{d_1},$$

abhängig von den Gewichten Ω. Demnach wird eine optimale Belegung der Gewichte Ω gesucht, sodass sich $f(x)$ so gut wie möglich y annähert.

Ein neuronales Netzwerk durchläuft mehrere Phasen. In der *Trainingsphase* wird das neuronale Netz f *trainiert*, indem f mehrfach eine Beobachtung x erhält und eine Vorhersage z trifft. Seien $y \in \mathbb{R}^{d_2}$ die exakten Werte, die das neuronale Netzwerk versucht optimal zu approximieren. Die Dimension der optimalen Werten müssen der Dimension der Vorhersagen entsprechen. Die Vorhersage z ist von den Gewichten Ω abhängig. Die Fehlerfunktion $\mathcal{L}$ bestimmt die Diskrepanz zwischen den Vorhersagen z bzw. $f(x)$ und den optimalen Werten y, d. h.

$$\mathcal{L}(x) = \|y - f(x)\|^2, \text{ für alle } x \in \mathbb{R}^{d_1},$$

abhängig von den Gewichten Ω des neuronalen Netzwerks. Durch Differenzierung der Fehlerfunktion $\mathcal{L}$ nach den Gewichten Ω wird eine Aktualisierung der Gewichte innerhalb des neuronalen Netzwerks erreicht. Dieses Verfahren wird *Backpropagation* genannt und ist der *Lernprozess* des neuronalen Netzwerks. In dieser wird „rückwärts“ durch das neuronale Netzwerk eine Aktualisierung aller Gewichte ausgeführt. Diesen Schritt werden wir in dieser Arbeit aber nicht weiter erläutern.

Im erwünschten Fall konvergiert die Fehlerfunktion und die Werte y sind durch $f(x)$ optimal approximiert. In der *Validierungsphase* werden die sog. *Hyperparameter* des neuronalen Netzwerks leicht verändert. Ein Hyperparameter ist eine Variable, die der Benutzer vor dem Trainingsablauf festsetzt, um das neuronale Netzwerk zu steuern, z. B. die Anzahl der *Epochen* des neuronalen Netzwerks bzw. wie oft die Gewichte aktualisiert werden. In der *Testphase* sind die optimierten Gewichte des neuronalen Netzwerks f fixiert. Diesem werden dann ausschließlich Beobachtungen x übergeben und das neuronale Netzwerk berechnet die Vorhersagen $f(x)$.

Im obigen gehen wir von dem *supervised* Fall aus, wobei die exakten Werte y für die Trainingsphase bekannt sind. Im *unsupervised* Trainingsfall haben die Werte kein Label, d. h. es ist dem Lernalgorithmus „unbekannt“, wie die Ergebniswerte y einzuordnen sind. Dann versucht dieser Muster und Abhängigkeiten in den Daten zu erkennen, um für unbekannte Daten Vorhersagen zu treffen, z. B. mittels Clustering [Mis17].

2.2.2. Encoder-Decoder-Framework

Da die Objekte eines Graphen abstrakt sind und keine numerischen Informationen enthalten, können diese nicht durch Lernalgorithmen verarbeitet werden. Bevor ein Graph als *Datenobjekt* für die Verarbeitung durch ein neuronales Netzwerk verwendet werden kann, muss vorher definiert werden, welche Graphenobjekte Informationen tragen sollen,

z. B. Knoten, Kanten oder Graphen. Für diese Arbeit werden den Knoten Informationen zugewiesen, da wir an deren Anordnung im Raum interessiert sind.

Sei $G = (V, E)$ ein Graph und Γ eine Zeichnung des Graphen. Bei der multidimensionalen Skalierung der Graphenvisualisierung wird oft die graphentheoretische Distanz d_{uv} als die Ähnlichkeitsfunktion verwendet, z. B. siehe Gansner et al. [GHN13] (siehe dazu Abschnitt 2.1.2). Intuitiv verschieben die Methoden der multidimensionalen Skalierung iterativ die Positionen der Knoten in der Zeichnung, um den Stress zu minimieren, z. B. in der multidimensionalen Skalierung von Kamada und Kawai [KK89]. Im Gegensatz dazu *lernt* die abbildende Funktion in neuronalen Netzwerken wie sie Objekte im niedrigdimensionalen Raum anordnet, damit die Ähnlichkeitsfunktion des höherdimensionalen Raumes der Ähnlichkeit des niedrigdimensionalen Raumes möglichst genau entspricht. Laut Hamilton et al. [HYL17] sind sog. *Shallow-Embedder*, welche die Einbettung im niedrigdimensionalen Raum lernen, weitgehend von der multidimensionalen Skalierung inspiriert.

Hamilton et al. [HYL17] haben ein allgemeines *Encoder-Decoder-Framework* vorgeschlagen, welches den Zusammenhang zwischen abbildender Funktion und Ähnlichkeitsfunktion erklärt. Dieses wird hier skizziert.

Sei G ein Graph und d die Anzahl der Merkmale, genannt *Features*, die jedem Knoten im Graphen G zugewiesen werden. Ein Feature eines Knoten kann unter anderem strukturelle und positionelle Eigenschaften eines Knoten enthalten, wie den Grad eines Knoten, Informationen über die lokale Nachbarschaft, oder inhaltliche Merkmale, um z. B. Personen in einem sozialen Netzwerk zu repräsentieren.

Der *Encoder* f_{enc} ist eine Funktion mit trainierbaren Gewichten und bildet die Knoten eines Graphen $v \in V$ in einem *Embbeding-Space* $\mathbb{R}^d$ auf eine Vektorrepräsentation oder ein *Node-Embedding* $z_v \in \mathbb{R}^d$ ab:

$$f_{\text{enc}} : V \to \mathbb{R}^d.$$

Dann ist ein kodierter Knoten z_v ein Vektor der Größe d mit d Features. Im Hinblick auf die multidimensionale Skalierung ordnet der Encoder jedem Knoten v aus einem abstrakten Raum ein Node-Embedding z_v in einem niedrigdimensionalen Raum $\mathbb{R}^d$ zu.

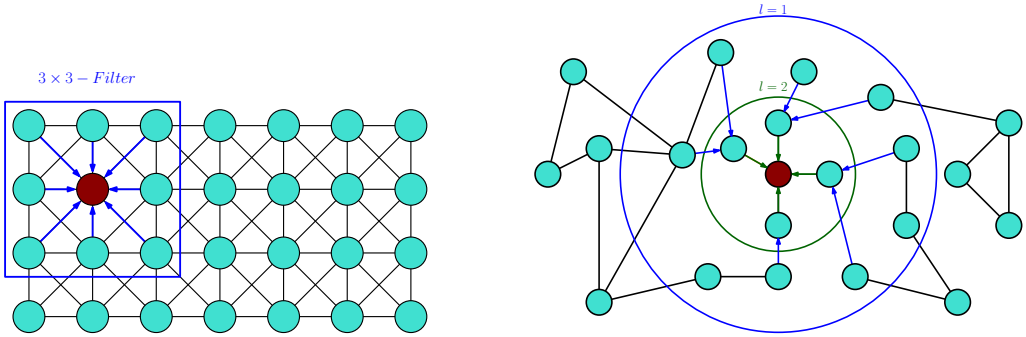
Der *Decoder* f_{dec} ist eine Funktion, die die Struktur des originalen Graphen G aus den berechneten Vektorrepräsentationen z_v aller Knoten $v \in V$ des Encoders rekonstruiert:

$$f_{\text{dec}} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+.$$

Der Wert des Decoders beschreibt für je zwei Node-Embeddings z_v und z_u deren Ähnlichkeit und Beziehung im Embedding-Space [Ham20].

Üblicherweise besitzt der Decoder keine trainierbaren Gewichte und diesem werden meistens zwei gelernte Node-Embeddings[1] übergeben.

[1] Dies hängt aber von der Definition des Decoders ab, z. B. wird dem Decoder in CoRe-GD nur ein Node-Embedding übergeben und dieser besitzt trainierbare Gewichte [GMVW24]: https://github.com/floriangroetschla/CoRe-GD



(a) (b)

Abb. 2.2.: (a) zeigt den Convolution-Prozess auf einem Bild durch ein CNN und (b) zeigt den Graph-Convolution-Prozess auf einem Graphen durch ein GNN

Der *Reconstruction-Loss* beschreibt die Diskrepanz zwischen der Ähnlichkeit der Knoten u und v im originalen Graphen $f_{\text{graph}}(u, v)$ und der Beziehung der gelernten Vektorrepräsentation im Embedding-Space $f_{\text{dec}}(z_u, z_v)$. Für alle Knoten u, v des Graphen G gilt dann

$$f_{\text{dec}}(f_{\text{enc}}(u), f_{\text{enc}}(v)) = f_{\text{dec}}(z_u, z_v) \approx f_{\text{graph}}(u, v).$$

Durch eine Optimierung des Encoders und Decoders wird der Reconstruction-Loss $\mathcal{L}$ minimiert:

$$\mathcal{L} = \sum_{(u,v)\in\mathcal{G}} \ell(f_{\text{dec}}(z_u, z_v), f_{\text{graph}}(u, v)).$$

Das Ziel besteht darin, den Encoder f_{enc} so zu trainieren, dass die Node-Embeddings z_v und z_u im Embedding-Space so positioniert werden, dass die Ähnlichkeit $f_{\text{dec}}(z_u, z_v)$ zwischen den beiden Node-Embedding z_u und z_v möglichst genau der graphentheoretischen Ähnlichkeit $f_{\text{graph}}(u, v)$ entspricht. Die Funktion $\ell : \mathbb{R}\times\mathbb{R} \to \mathbb{R}$ beschreibt die Diskrepanz zwischen $f_{\text{dec}}(z_u, z_v)$ und $f_{\text{graph}}(u, v)$ für je zwei Knoten u und v des Graphen G.

Beispielsweise gilt laut Hamilton et al. für den *Skalarprodukt*-Decoder, dass das Skalarprodukt der Vektorrepräsentationen z_u und z_v proportional zu der Stärke der Beziehung u und v in dem Graph G ist. Als Beispiel haben Ahmed et al. [ASN+13] einen Skalarprodukt-Decoder vorgestellt, für den gilt

$$f_{\text{dec}}(z_u, z_v) = z_u^\top z_v \approx (a_{uv}),$$

wobei die Ähnlichkeit der Knoten u und v im Graphen $f_{\text{graph}}(u, v) := (a_{uv})$ auf der Adjazenzmatrix bezeichnet.

2.2.3. Graphbasierte neuronale Netzwerke

Die strukturelle Repräsentation eines Graphen, die Adjazenzmatrix, kann nicht mit einem *klassischen* neuronalen Netzwerk verarbeitet werden. Ein Grund ist hier, dass

u. a. die Adjazenzmatrix nicht permutations-invariant bezüglich der Knotenanordnung ist [Ham20]. Man verwendet die Idee der *Convolution-Neural-Networks (CNN)*, welche für die Bildverarbeitung eingesetzt werden [ON15]. In diesen wird ein Bild als ein Gittergraph interpretiert. In einem sich bewegenden *Filter* aggregiert ein Zentralknoten Pixelinformationen der Nachbarknoten und komprimiert diese zu einem abstrakteren Pixel. Dieser Prozess wird in der Abbildung 2.2a dargestellt.

Ein graphbasiertes neuronales Netzwerk, ein sog. *Graph-Neural-Network (GNN)*, kann Graphen allgemeiner Form verarbeiten und deren komplexe Struktur erfassen (siehe z. B. [SGT$^+$09] [KW16][XHLJ19]). Zur Einführung dieses speziellen Typs der neuronalen Netzwerke stützen wir uns auf Hamilton [Ham20] und Wu et al. [WCPZ22]. Im Gegensatz zum vorherigen Abschnitt 2.2.2 wird hier ein trainierbarer Encoder definiert, der die Vektorrepräsentationen der Knoten in Abhängigkeit von der umgehenden Nachbarschaft generiert. Solche Encoder werden *Neighbor-Aggregation*-Encoder und der interne Prozess als der *Message-Passing*-Algorithmus bezeichnet [HYL17].

Sei $G = (V, E)$ ein Graph, v ein Knoten und dessen Nachbarschaft $N(v)$. Der Informationstransfer eines Knoten v mit einem Knoten u aus seiner Nachbarschaft $N(v)$ geschieht in einem *Graph-Convolution*-Layer mithilfe des Message-Passing-Algorithmus [GSR$^+$17]. Jeder Knoten v konstruiert einen individuellen sogenannten *Computation-Graph*, wobei die Nachfolger-Knoten von v die Nachbarknoten aus $N(v)$ sind. Der Computation-Graph ist ein gerichteter Baum T, wobei v die Wurzel und die Nachbarknoten $N(v)$ die Blätter bilden. Die Richtung von T verläuft von den Blättern zur Wurzel v, wobei Informationen von den Nachbarn $N(v)$ zu der Wurzel v übertragen werden. Der Computation-Graph von v kann in die Tiefe l wachsen, indem für jeden Knoten $u \in N(v)$ die Nachbarknoten $N(u)$ ebenfalls betrachtet. Eine Ebene des Baumes wird ebenfalls Layer l genannt. Der Zweck der Erweiterung des Computation-Graphen des Knotens v besteht darin, dass dieser Knoten Information von entfernten Knoten erhält.

Intuitiv wird vom Zentralknoten ausgehend eine *Breitensuche* ausgeführt, um den Computation-Graph zu bilden. Anschließend wird im Message-Passing-Algorithmus eine „invertierte“ Breitensuche durchgeführt, indem wellenförmig Informationen von den äußeren Knoten ins Zentrum aggregiert werden, siehe dazu Abbildung 2.2b.

Sei v ein Knoten und u ein Knoten aus der Nachbarschaft $N(v)$. Sei l die Tiefe des Computation-Graphen von v und für jeden Knoten $v \in V$ gilt $h_v^{(0)} := z_v$. Dann ist der Message-Passing-Algorithmus rekursiv definiert als

$$h_v^{(l)} = f_{\text{update}} \left(h_v^{(l-1)}, \bigoplus_{u \in N(v)} f_{\text{msg}}\Big(h_v^{(l-1)}, h_u^{(l-1)}\Big) \right). \tag{2.3}$$

Die Funktionen f_{msg} und f_{update} können neuronale Netzwerke sein. Eine Nachricht eines Knoten u zu v ist definiert als

$$m_{u,v}^{(l)} := f_{\text{msg}}\Big(h_v^{(l-1)}, h_u^{(l-1)}\Big),$$

wobei das Node-Embedding des Wurzelknotens $h_v^{(l-1)}$ und die Vektorrepräsentation des Blattes $h_u^{(l-1)}$ in eine Nachricht $m_{u,v}^{(l)}$ transformiert werden. Eine Nachricht ist eine Funktion, die eine Transformation von $h_v^{(l-1)}$ in Abhängigkeit des Nachbarn $h_u^{(l-1)}$ erwirkt.

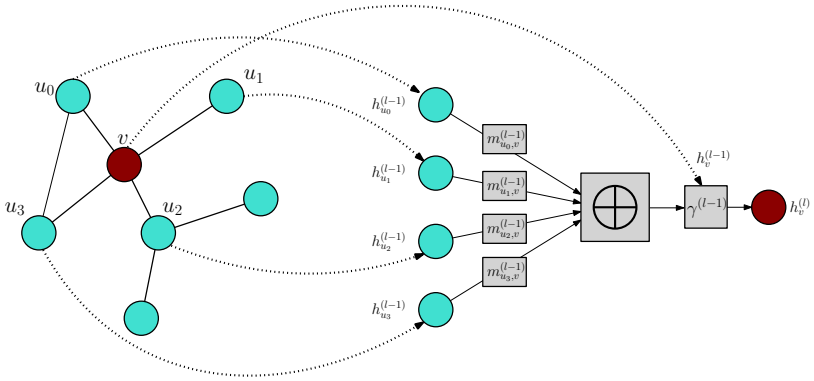


Abb. 2.3.: Schematische Abbildung des Computation-Graphen des Knotens v und des Message-Passing-Algorithmus um die Vektorrepräsentation $h_v^{(l)}$ auf das Node-Embedding zu $h_v^{(l+1)}$ aktualisieren. Das Symbol $\bigoplus$ kennzeichnet die Aggregator-Funktion f_{agg}. Die Funktion $\gamma^{(l)}$ bezeichnet hier ein neuronales Netzwerk als Update-Funktion f_{update}. (Eigene Darstellung)

Die Aggregator-Funktion f_{agg} aggregiert die Nachrichten der Nachbarschaft $N(v)$ zum Knoten v und wird hier als $f_{\text{agg}} := \bigoplus$ gekennzeichnet,

$$m_{N(v)}^{(l)} := \bigoplus_{u \in N(v)} m_{u,v}^{(l)}.$$

Die Aggregator-Funktion muss differenzierbar und invariant gegenüber Permutationen bezüglich der Knotenanordnung sein, z. B. eine Summe oder eine Durchschnittsberechnung.

Zuletzt werden in der Update-Funktion die aggregierte Nachricht $m_{N(v)}^{(l)}$ und $h_v^{(l-1)}$ kombiniert zu

$$h_v^{(l)} = f_{\text{update}}\Big(h_v^{(l-1)}, m_{N(v)}^{(l)}\Big).$$

Dann besitzt der Knoten v eine aktualisierte Vektorrepräsentation $h_v^{(l)}$. Diese wird aufgrund der rekursiven Definition im Schritt $(l+1)$ als Vektorrepräsentation der Nachbarn einbezogen. Bei $l := L$ terminiert der Message-Passing-Algorithmus, wobei die Vektorrepräsentation des Wurzelknotens $h_v^{(L)}$ die Informationen aller Blätter und Zwischenknoten enthält, und auf das finale Node-Embedding $z_v := h_v^{(L)}$ aktualisiert wird. Dieser Prozess wird *Graph-Convolution* genannt und wird schematisch in Abbildung 2.3 gezeigt.

Um ein Verständnis des Message-Passing-Algorithmus zu erlangen, wurde dieser für jeden Knoten v mit h_v eingeführt. Allerdings besitzt dieser auch eine Matrixschreibweise, wobei dann $H = [h_{v_1}, \ldots, h_{v_i}, \ldots, h_{v_n}]$, wobei $v_i \in V$, $1 \leq i \leq n$ und $n := |V|$ gilt. Die Matrix H besitzt die Dimension $\mathbb{R}^{n \times d}$, wobei d die Dimension der Node-Embeddings h_v für alle Knoten $v \in V$ ist.

Ein Graph-Neural-Network ist ein neuronales Netzwerk mit Graph-Convolution-Layers. Jeder Knoten v eines Graphen G wird mithilfe eines Encoders $f_{\text{enc}}(v)$ in eine Vektorre-

präsentationen z_v transformiert. Die Vektorrepräsentation z_v wird mithilfe von Graph-Convolution-Layern in aktualisierte Vektorrepräsentationen gebracht. Der Wert des Decoders $f_{\text{dec}}(z_v)$ ist dann die Vorhersage des Graph-Neural-Networks. Abhängig davon, wie die Loss-Funktion $\mathcal{L}$ definiert ist, wird die Diskrepanz zwischen dem Wert $f_{\text{dec}}(z_v)$ und dem exakten Wert y berechnet. Durch die Backpropagation werden die trainierbaren Gewichte Ω der klassischen neuronalen Netzen γ, ϕ und f_{enc} aktualisiert.

2.3. Graphenvisualisierung mit neuronalen Netzwerken

Graphbasierte neuronale Netzwerke sind in der Lage die Struktur eines Graphen zu erfassen [ZCZ+20].

Durch eine geeignete Auswahl der Features der Vektorrepräsentationen ist es möglich, mithilfe des Message-Passing-Algorithmus sowohl globale als auch lokale Informationen innerhalb des Graphen zu verbreiten. Dadurch wird die positionelle und strukturelle Wahrnehmung der Knoten erhöht, z. B. Distanz eines Knoten zu dem Graph-Centroid, Distanz zum Cluster-Center, relative Distanzen zwischen Knoten, Knotengrad, Substrukturen des Graphen wie Kreisen und Dreiecken [YFK+14][RGD+22][ZJAS22]. Im folgenden Abschnitt werden die neuronalen Netzwerke zum Zeichnen von Graphen aus der Literatur vorgestellt. Die vorgestellten neuronalen Netzwerke optimieren direkt oder indirekt eine Metrik des Graphzeichnens.

2.3.1. A Deep Generative Model von Kwon et al.

Kwon et al. [KM20] haben ein generatives Modell für Zeichnungen von Graphen vorgestellt, das auf einem Encoder-Decoder-Framework basiert. Dieses Modell nimmt eine Zeichnung Γ eines Graphen G in Form von Positionen P der Knoten auf. Dem Encoder werden eine Zeichnung des Graphen und eine Adjazenzmatrix übergeben. Der Encoder liefert eine Vektorrepräsentation z_L der Zeichnung des Graphen. In einem sog. „Fusion-Layer“ wird die Repräsentation des Graphen z_L und eine Feature-Matrix der Zeichnung Γ_V übertragen. Als Features haben Kwon et al. eine Zeichnung mit den euklidischen Distanzen zwischen den Knoten versehen. Der aus Graph-Convolution-Layern bestehender Decoder transformiert das fusionierte Objekt in neue Positionen P'. Diese werden dann in eine Zeichnung Γ' umgewandelt. Das Ziel des Trainingsprozesses ist es, die Diskrepanz zwischen Eingabezeichnung und Ausgabezeichnung des Modells zu minimieren, d. h. den Reconstruction-Loss zu minimieren. Laut Kwon et al. kann das Modell nur für einen Graph verschiedene Zeichnungen generieren und muss für jeden neuen Graph neu trainiert werden [KM20]. Das Modell kann nicht eigenständig ein ästhetisches Kriterium minimieren, sondern versucht eine Ähnlichkeit zwischen Eingabe- und Ausgabezeichnung herzustellen.

2.3.2. DeepDrawing von Wang et al.

Das Modell *DeepDrawing* von Wang et al. [WJW+20] ist ein Graph-LSTM-basierter Ansatz um einen Graphen G direkt auf eine Zeichnung Γ abzubilden. Eine *LSTM*-Zelle

(Long-Short-Term-Memory) von Hochreiter und Schmidhuber [HS97] ist eine Gedächtniszelle, welche aus einem neuronalen Netzwerk besteht. Mit dieser lassen sich über große Entfernungen im Graph Abhängigkeiten in der Eingabesequenz erkennen [GSC00]. Da eine LSTM-Zelle nur Sequenzen verarbeiten kann [HS97][WJW+20], wird eine Sequenz aus Adjazenzvektoren erstellt. Derer Reihenfolge wird von einer Breitensuche auf dem Graphen G bestimmt. Jeder Knoten v des Graphen G wird als eine LSTM-Zelle repräsentiert. Die LSTM-Zellen werden durch zusätzliche Kanten verbunden, um die Struktur des Graphen zu modellieren. Die LSTM-Zellen generieren den Zustand eines Knotens und zuletzt wird eine Zeichnung des Graphen generiert. Laut Gionvannageli et al. [GLA+22] wird DeepDrawing trainiert eine „Goldstandard"-Zeichnung zu reproduzieren anstatt die Graphenstruktur zu lernen. Laut Wang et al. [WJW+20] muss jede mögliche Anordnung von Knoten der Breitensuche berechnet werden, um einen sinnvollen Trainingsprozess zu ermöglichen. Die Performanz von DeepDrawing wird schlechter, falls ein Eingabegraph eine deutlich andere Graphenstruktur als die Graphen aus dem Trainingssatz aufweist. Laut Wang et al. besitzt DeepDrawing Schwierigkeiten bezüglich der Interpretierbarkeit, da nicht ersichtlich ist, welche Merkmale des Graphen gelernt werden.

2.3.3. $(\mathrm{DNN})^2$ von Giovannageli et al.

Das neuronale Netzwerk *(DNN)*^{2} von Giovannangeli et al. [GLA+22] basiert auf der *ResNet*-Architektur von He et al. [HZRS16], welche zur Klassifizierung von Bildern genutzt wird. DNN^2 ist ein neuronales Netz, das aus 52 *spektralen* Graph-Convolution-Layers [KW16] besteht. Als Eingabe erhält $(\mathrm{DNN})^2$ die Feature-Matrix der Knoten und eine Datenstruktur, die die strukturelle Information des Graphen beinhaltet. Die Eingabe wird kodiert und durch spektrale Graph-Convolution-Layers propagiert. Schlussendlich generiert $(\mathrm{DNN})^2$ dadurch eine Zeichnung. Im Gegensatz zu DeepDrawing von Wang et al. verwendet $(\mathrm{DNN})^2$ eine strukturabhängige Fehlerfunktion. Das Modell kann verschiedene Metriken optimieren, u. a. den Stress der Zeichnung (siehe Gleichung (2.1)). Ein Nachteil von $(\mathrm{DNN})^2$ ist, dass topologisch äquivalente Knoten des Graphen in der Zeichnung fälschlicherweise gruppiert werden können, wobei u. U. der Stress der Zeichnung hoch sein kann, z. B. in einer Zeichnung eines Gittergraphen. Aus diesem Grund haben die Autoren [GLA+22] eine Modifikation von $(\mathrm{DNN})^2$ vorgestellt, welche dieses Problem nicht besitzt. Laut Giovannageli et al. benötigt $(\mathrm{DNN})^2$ und das modifizierte Modell eine große Anzahl von Ressourcen.

2.3.4. DeepGD von Wang et al.

Das Modell *DeepGD* von Wang et al. [WYHS21] benutzt eine Fehlerfunktion, wobei das Modell mehrere Metriken der Graphenvisualisierung gleichzeitig optimieren kann. DeepGD kann für unterschiedliche Typen von Graphen Zeichnungen anfertigen, und muss nicht nach jedem neuen Graphentypen neu trainiert werden. Das Modell besteht aus mehreren Graph-Convolution-Layern, wobei im Message-Passing-Algorithmus u. a. die Richtung der Kante und die Längendifferenz zwischen zwei Knoten übertragen wird. Das

Modell wurde auf dem Rome-Graphen-Datensatz[2], welches aus einer Menge von ungerichteten Graphen besteht, trainiert. Jeder Graph wird zu einem *vollständigen* Graphen ohne Schleife erweitert, d. h. jedes unverbundene Knotenpaar wird durch eine „virtuelle“ Kante verbunden [GSR+17]. Laut Wang et al. ist der Grund dafür, dass der Message-Passing-Algorithmus bei großer graphentheoretischer Distanz viele Schritte benötigt, um einen Knoten zu erreichen, der in der Zeichnung aktuell zu nah an einem anderen Knoten platziert wurde (siehe Gleichung (2.1) und Gleichung (2.3)). Zusätzlich erhält jeder Rome-Graph anstatt einer ungerichteter Kante, zwei gerichtete Kanten entgegengesetzter Richtung, um eine asymmetrische Informationsübertragung zu vermeiden. In DeepGD werden Kanten durch ein *Edge-Feature-Network* gewichtet, um die Relevanz einer Nachricht für einen Knoten zu bestimmen. Die Vorhersage des Edge-Feature-Network ist eine Kanten-Feature-Matrix. In DeepGD werden die Adjazenzmatrix, die Feature-Matrix der Knoten und die Kanten-Feature-Matrix kodiert und durch die Graph-Convolution-Layer propagiert. Schließlich wird eine Zeichnung ausgegeben. Die Fehlerfunktion von DeepGD besteht aus mehreren Fehlerfunktionen. In jeder dieser Fehlerfunktion ist eine Metrik der Graphenvisualisierung direkt implementiert. Jede Metrik in der Fehlerfunktion wird mit einem Parameter versehen, welche das Verhältnis dieser Metrik gegenüber den anderen Metriken in der kombinierten Fehlerfunktion beschreibt. DeepGD kann u. a. die Abweichung des minimalen Winkels vom optimalen Winkel, den Stress oder die Kantenlängenvarianz optimieren.

2.3.5. SmartGD von Wang et al.

Die Differenzierung der Fehlerfunktion ist für neuronale Netzwerke essentiell, denn die Fehlerfunktion wird nach den Gewichten differenziert und mithilfe der Backpropagation die Gewichte aktualisiert. Durch diesen Prozess gelingt ein Lernen des neuronalen Netzwerks. Nach Wang et al. [WYHS23] können die Metriken der Graphenvisualisierung in zwei Kategorien unterteilt werden: differenzierbare Kriterien, z. B. den Stress, und nicht-differenzierbare Kriterien, z. B. Anzahl der Kantenkreuzungen [BMRW98]. Das neuronale Netzwerk *SmartGD* von Wang et al. [WYHS23] ist ein *Generative-Adversial-Network* [Jol18], welches nicht-differenzierbare ästhetische Kriterien optimieren kann. SmartGD besteht aus zwei neuronalen Netzwerken: dem *Diskriminator* und dem *Generator*. Beide Sub-Module bestehen aus Graph-Convolution-Layers, um die Struktur des Graphen zu erfassen. Im Trainingsprozess erhält der Generator einen Graphen und generiert daraus eine Zeichnung. Der Diskriminator erhält als Eingabe denselben Graph wie der Generator, aber zusätzlich eine gute Zeichnung des Graphen, die die gewünschten Kriterien erfüllt. In der Fehlerfunktion des Diskriminators wird die Zeichnung des Generators anhand der erhaltenen guten Zeichnung mit diesem Graphen verglichen, um zu bewerten, in welchem Maße die generierte Zeichnung besser ist als die gute Zeichnung. Diese Diskrepanz zwischen generierter und guter Zeichnung wird dem Generator mitgeteilt und dessen Gewichte werden entsprechend angepasst. Der Generator wird trainiert, eine Zeichnung ähnlicher Qualität wie die gute Zeichnung zu generieren, die der Diskri-

[2] http://www.graphdrawing.org/data.html

minator erhalten hat. Das Ziel besteht darin, die gute Zeichnung mit einer generierten Zeichnung zu übertreffen. Falls dies dem Generator gelingt, besitzt SmartGD einen *Self-Challenging*-Mechanismus, bei dem die gute Zeichnung durch die Generierte ersetzt wird. Die Fehlerfunktionen beider Modelle sind nicht von der Differenzierbarkeit der Metriken abhängig, und damit kann SmartGD sowohl differenzierbare Metriken, z. B. den Stress, als auch nichtdifferenzierbare Metriken, z. B. die Anzahl der Kantenkreuzungen, optimieren.

2.3.6. CoRe-GD von Grötschla et al.

Für unsere Experimente in dieser Arbeit verwenden wir das neuronale Netzwerk *CoRe-GD*, „a scalable **Co**arsening and **R**ewiring Framework for **G**raph **D**rawing“, von Grötschla, Mathys, Veres und Wattenhofer [GMVW24], das wir daher hier ausführlicher diskutieren. CoRe-GD besteht aus mehreren Graph-Convolution-Layers und übertrifft das Modell DeepGD von Wang et al. [WYHS21] und $(\text{DNN})^2$ von Giovannageli et al. [GLA+22] bezüglich des Stress-Wertes; Siehe dazu [GMVW24]. Das Modell CoRe-GD ist aufgrund eines *Verfeinerungsmechanismus* skalierbar und benutzt eine *Rewiring*-Methode, die die Struktur des Graphen verändert, um den Fluss des Message-Passings zu verändern. In der Arbeit von Grötschla et al. bezüglich CoRe-GD wurden ausschließlich die Stressfunktion und Varianten davon verwendet.

Zuerst wird erläutert, welche Module CoRe-GD benutzt, um aus Graphen Zeichnungen zu generieren. Der Verfeinerungsmechanismus von CoRe-GD ermöglicht es dem neuronalen Netz große Graphen verarbeiten zu können. Die internen Encoder und Decoder von CoRe-GD werden nicht mit der Fehlerfunktion in Zusammenhang gebracht, wie das in Abschnitt 2.2.2 geschieht. Der Zweck des Encoders in CoRe-GD ist laut Grötschla et al., dass Knoten innerhalb des Algorithmus mehr strukturelle Informationen tragen sollen, anstatt nur deren Positionen [GMVW24]. Die Initialisierung der Knoten ist für eine Unterscheidbarkeit der Knoten für den Message-Passing-Algorithmus wichtig und wird in diesem Abschnitt erklärt. Im Folgenden wird der Graph-Convolution-Layer von CoRe-GD erläutert. Danach wird die Rewiring-Methode von CoRe-GD erklärt. Als Nächstes werden die Architektur und die Fehlerfunktion von CoRe-GD beleuchtet. Abschließend wird die Komplexität von CoRe-GD beschrieben. Für den folgenden Abschnitt sei $G = (V, E)$ ein Graph und Γ eine Zeichnung.

Verfeinerungsmechanismus Der Vollständigkeit halber wird der Verfeinerungsmechanismus von CoRe-GD erklärt, da dieser eine Besonderheit von CoRe-GD gegenüber den verwandten Arbeiten darstellt. Für unsere Experimente wird dieser aber nicht verwendet.

Der Verfeinerungsmechanismus von CoRe-GD basiert auf iterativen *Kontraktionen* eines Graphen G. Sei U eine Teilmenge der Knotenmenge V, d. h. $\{v_i, \dots, v_k\} = U \subseteq V$ wobei $1 \leq i, k \leq n$ mit $n = |V|$. Eine Kontraktion ist eine Abbildung $K : V \to V \cup \{U\}$, wobei die Knotenmenge $\{v_i, \dots, v_k\}$ auf einen Knoten u abgebildet wird und alle Kanten, deren Endknoten beide in U liegen, entfernt werden. Jede Kante $\{v, w\} \in E$, wobei

$v \in U$ und $w \notin U$ wird durch die Kante $\{u, w\}$ ersetzt. Wir schreiben $V_{G/U}$ für $V \cup U$, wobei $V_{G/U}$ die Knotenmenge V des kontrahierten Graphen G/U bezeichnet. In der Arbeit [GMVW24] von Grötschla et al. wird die *Spectral-Preserving-Coarsening*-Methode von Jin und Jájá [JJ20] mit einem Reduktionsfaktor c genutzt, wobei schließlich eine Sequenz von kontrahierten Graphen $\{G_1, \ldots, G_c\}$ entsteht. Bevor eine Zeichnung Γ eines Graphen erstellt wird, wird der Verfeinerungsmechanismus durchgeführt. Nach der Verfeinerung werden die Knoten des gröbsten Graphen G_1 durch einen Encoder f_{enc} zu Vektorrepräsentationen transformiert und nach mehreren Schritten werden die Positionen der Knoten optimiert. Mithilfe einer inversen Funktion $K^{-1} : V_{G_1} \to V_{G_2}$ wird nach der Optimierung die Kontraktion K von G_1 rückgängig gemacht, wobei allerdings ein Knoten v nicht sofort dekodiert wird, sondern ein Node-Embedding z_v bleibt, um diesen möglicherweise für weitere Graph-Convolutions zu verwenden. Sei i die Iterationsvariable mit $1 \leq i \leq c$. Die inverse Funktion $K^{-1} : V_{G_i} \to V_{G_{i+1}}$ wird *Expansion* genannt. Die Urbilder $\{h_{v_i}, \ldots, h_{v_k}\}$ von $h_{v_u} \in G_i$ erhalten ein zufälliges Rauschen, um diese für den Message-Passing-Algorithmus unterscheidbar zu machen.

Encoder-Decoder Der Encoder $f_{\text{enc}} := \text{enc}$ und Decoder $f_{\text{dec}} := \text{dec}$ von CoRe-GD[3] sind *Multi-Layer-Perceptrons*, d. h. klassische neuronale Netzwerke mit mehreren Layers. Encoder und Decoder lassen sich nicht in den Zusammenhang bezüglich einer Fehlerfunktion bringen, wie in Abschnitt 2.2.2. In CoRe-GD wird jeder Knoten v durch den Encoder $\text{enc}(v) = h_v$ in Vektorrepräsentation transformiert. Laut den Autoren von [GMVW24], werden intern von CoRe-GD Node-Embeddings übertragen, da die Knoten mehr Informationen tragen sollen als nur ihre aktuelle Position, u. a. auch strukturelle Information im Graphen. Nach einem Ablauf von CoRe-GD werden die Node-Embeddings h_v durch den Decoder dec auf Positionen $\Gamma(v)$ abgebildet. Diese sind die Vorhersagen von CoRe-GD und werden in die Fehlerfunktion $\text{stress}(G, \Gamma)$ (siehe Gleichung (2.1)) eingesetzt. Durch Backpropagation werden die Gewichte der trainierbaren Funktionen von CoRe-GD aktualisiert.

Initialisierung der Knoten Die Problematik des Message-Passing-Algorithmus ist dessen Ununterscheidbarkeit von *strukturgleichen* Knoten. Der Message-Passing-Algorithmus kann z. B. die Knoten eines Dreiecks nicht unterscheiden, da die Knoten strukturgleich sind [FCL+23]. Die Node-Embeddings werden mit positionellen oder strukturellen Informationen versehen, um die Unterscheidbarkeit der Knoten für den Message-Passing-Algorithmus zu erhöhen [RGD+22]. In CoRe-GD werden die Node-Embeddings mit drei Features versehen, um eine bessere Unterscheidbarkeit zu gewährleisten und den Informationsgehalt der Vektorrepräsentationen der Knoten zu erhöhen: *Laplace Eigenvektoren* [BN03] [DJL+22], relative Entfernung zu *Beacon*-Nodes [ZLWH24] und zufälliges Rauschen. Nach Dwivedi et al. [DJL+22] bilden die Laplace Eigenvektoren ein lokales Koordinatensystem, welches die globale Struktur des Graphen erhält. Die Beacon-Nodes $U_b \subset V$ sind Knoten des Graphen G und $k := |U_b|$, wobei k konstant. Jeder

[3] In der Implementierung von CoRe-GD ersichtlich: https://github.com/floriangroetschla/CoRe-GD

Knoten v erhält Feature-Informationen über den kürzesten Pfad zu jedem Beacon-Knoten $v_b \in U_b$ und diese Pfade werden mit einer Breitensuche gefunden. Das Rauschen als Feature verbessert die Unterscheidbarkeit.

Graph-Convolution In der Arbeit [GMVW24] werden drei Graph-Convolution-Layer für CoRe-GD beschrieben. Allerdings benutzt CoRe-GD in der Praxis nur eines der Graph-Convolution-Layers, welches vom Benutzer ausgewählt werden kann. Wir beschreiben das *Graph-Isomorphism-Network (GIN)* von Xu et al. [XHLJ19] detaillierter, um eine Anwendung des Message-Passing-Algorithmus zu illustrieren. In dieser Arbeit werden die anderen Convolution-Layer, die *Gated-Recurrent-Unit (GRU)* von Huang und Carley [HC19] und das *Graph-Attention-Network (GAN)* von Veličković et al. [VCC+18], nicht weiter erläutert.

Sei G ein Graph und sei v der zentrale Knoten. Sei für jeden Knoten v das Node-Embedding h_v, dann ist die *(GIN)*-Convolution von Xu et al. in CoRe-GD definiert als

$$h_v^{(l)} = \gamma\left((1+\epsilon)\cdot h_v^{(l-1)} + \sum_{u \in N(v)} \phi(h_v^{(l-1)} \parallel h_u^{(l-1)})\right),$$

wobei γ, ϕ neuronale Netzwerke sind, die Zahl ϵ ein Parameter ist und jede Nachricht des Nachbarn $u \in N(v)$ ist eine Konkatenation das Node-Embeddings des Zentralknotens h_v und des Nachbarknoten h_u, die an ein neuronales Netzwerk ϕ übertragen wird. In dem folgenden Abschnitt zeigen wir, wie die hier vorgestellte Definition des Message-Passing-Algorithmus mit der GIN-Convolution in Einklang gebracht wird (siehe die Definition Arbeit in Abschnitt 2.2.3). Die Nachricht des GIN-Convolution-Layers ist definiert als

$$m_{u,v}^{(l)} := f_{\text{msg}}\left(h_v^{(l-1)}, h_u^{(l-1)}\right) := \phi(h_v^{(l-1)} \parallel h_u^{(l-1)}),$$

wobei ϕ ein neuronales Netzwerk ist und der Operator $\parallel$ bezeichnet die Konkatenation der Embeddings $h_v^{(l-1)}$ und $h_u^{(l-1)}$. Eine Adaption von Grötschla et al. ist, dass f_{msg} ein neuronales Netzwerk ϕ ist. Der Aggregator-Operator der GIN-Convolution ist definiert als

$$m_{N(v)}^{(l)} := \bigoplus_{u \in N(v)} m_{u,v}^{(l)} := \sum_{u \in N(v)} m_{u,v}^{(l)},$$

wobei die Aggregator-Funktion $\bigoplus$ eine einfache Summe der Nachrichten der Nachbarknoten von $N(v)$ darstellt. Die Update-Funktion der GIN-Convolution ist definiert als

$$h_v^{(l)} = f_{\text{update}}\left(h_v^{(l-1)}, m_{N(v)}^{(l)}\right) := \gamma\left((1+\epsilon)\cdot h_v^{(l-1)} + m_{N(v)}^{(l)}\right),$$

wobei γ ein neuronales Netzwerk, die Zahl ϵ ein Parameter und $m_{N(v)}^{(l)}$ die aggregierte Nachricht bezeichnet.

Rewiring-Methode Sei G ein Graph. *Rewiring* in Graphen bedeutet, neue Kanten zwischen Knoten entstehen und andere entfernen zu lassen. Durch diesen Prozess verändert sich die Struktur des Graphen. Grund für die Einführung der Rewiring-Methode in CoRe-GD ist, die Effizienz des Message-Passing-Algorithmus zu erhöhen. Der Stress einer Zeichnung ist hoch, falls ein Knoten $\Gamma(v)$ in der unmittelbaren Nähe eines anderen Knoten $\Gamma(u)$ liegt und der Pfad d_{uv} im Graph G entsprechend lang ist (siehe Abschnitt 2.1.2). Der Message-Passing-Algorithmus wird iterativ knotenweise durchgeführt, und daher ist die Anzahl der Iterationen hoch, falls der Pfad d_{uv} im Graphen entsprechend lang ist, d. h. viele Knoten auf dem Pfad liegen (siehe Gleichung (2.3)).

Vor der Rewiring-Methode werden alle Vektorrepräsentation h_v durch den internen Decoder von CoRe-GD als eine Punktwolke abgebildet. Jeder Punkt $\Gamma(v)$ in der Zeichnung bildet eine Kante zu seinen *nächsten* Nachbarknoten, unabhängig davon, ob diese im originalen Graphen G verbunden sind oder nicht. Durch das Legen „neuer“ Kanten im Graphen, wird es dem Message-Passing-Algorithmus ermöglicht, nur eine Iteration durchzuführen, um den Node-Embedding h_v zu aktualisieren [GMVW24]. In CoRe-GD sind in der Rewiring-Methode *K-Nearest-Neighbor*-Graphen, *Radius*-Graphen und die *Delaunay-Triangulierung* von Delaunay et al. [Del34] implementiert.

Anzumerken ist, dass auch Wang et al. [WYHS21] für DeepGD eine Veränderung des Message-Passing-Flusses ausgeführt haben. Dies war im Preprocessing-Schritt als die Rome-Graphen zu vollständigen, gerichteten Graphen erweitert wurden.

Architektur Da wir den Verfeinerungsmechanismus von CoRe-GD nicht benutzen, wird der Algorithmus für einen einzelnen Graphen G erklärt, d. h. der Reduktionsfaktor $c := 1$ ist fixiert.

Sei $G = (V, E)$ und Γ eine Zeichnung. Sei H die kodierte Knotenmatrix aus $\mathbb{R}^{n \times d}$, d. h. die Zeilen der Matrix die Vektorrepräsentationen der Knoten v: $\text{enc}(v) = h_v$, wobei $n := |V|$ und d die Anzahl der Features der Knoten bezeichnen (siehe Abbildung 2.4a). Sei ferner Conv_{E} eine Graph-Convolution auf dem originalen Graphen G und Conv_{rew} eine Graph-Convolution auf dem Graphen $G_{\text{rew}} = (E_{\text{rew}}, V_{\text{rew}})$, der aufgrund der Strukturänderung der Rewiring-Methode neue Kanten besitzt.

Sei i die Iterationsvariable, wobei i im Intervall $[1, \ldots, r]$ liegt und i eine natürliche Zahl ist. Sei H_i die Embedding-Node-Matrix für einen Schritt i. Die Zahl r ist ein Hyperparameter, der die maximale Anzahl an Optimierungsschritten der Zeichnung begrenzt. Zur Initialisierung wird $H_1 := H$ gesetzt. Dann wird auf dem Graphen G eine Graph-Convolution ausgeführt, d. h.

$$H_{i^*} \leftarrow \text{Conv}_{\text{E}}(E, H_i),$$

wobei H_{i^*} eine temporäre Embedding-Matrix darstellt. In Abbildung 2.4b ist der Vorgang dargestellt. Dieses Embedding wird durch den Decoder dec auf Positionen abgebildet

$$\Gamma_i \leftarrow \text{dec}(H_{i^*}).$$

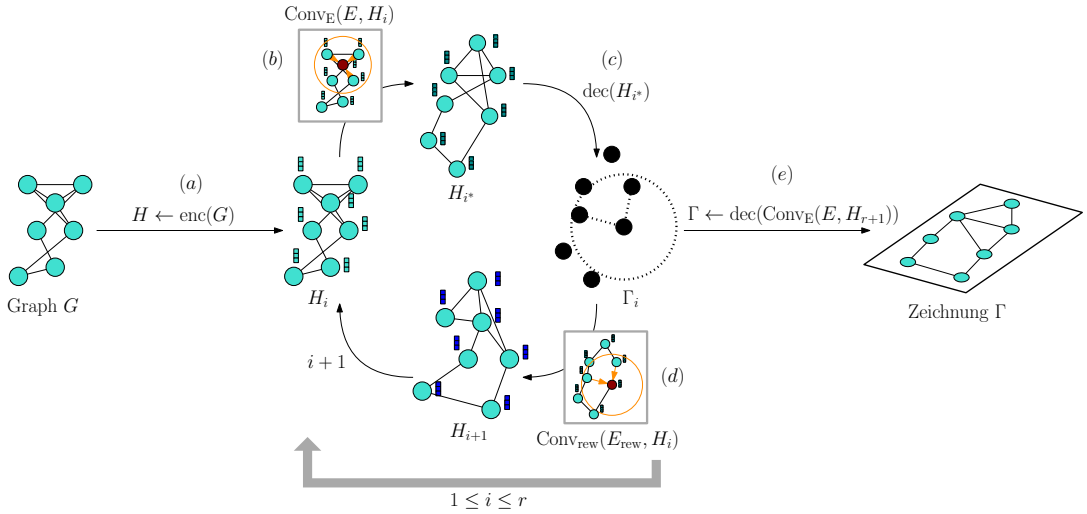


Abb. 2.4.: Abstrakte Darstellung eines Ablaufs von CoRe-GD ohne den Verfeinerungsmechanismus. Der Encoder von CoRe-GD ordnet dem Graph G ein Node-Embbedding H zu, wobei jeder Knoten hier drei Features enthält (siehe a). Durch eine Graph-Convolution auf dem originalen Graph werden die Vektorrepräsentationen der Knoten aktualisiert (siehe b). Dieser Node-Embedding mit aktualisierten Features der Knoten H_{i^*} wird durch den Decoder in eine Punktwolke Γ_i abgebildet (siehe c). Auf dieser wird der Rewiring-Prozess angewandt, wobei in der Abbildung dieser Vorgang durch einen Kreis dargestellt wird und in diesem werden die *nächsten* Nachbarknoten durch eine Kante verbunden (hier zwei Nachbarknoten). Durch die Graph-Convolution auf diesen „neuen“ Kanten werden die Vektorrepräsenationen neu aktualisiert. Nach einer Anzahl von r Iterationen, wird eine Graph-Convolution auf dem originalen Graphen und eine Dekodierung der Node-Embeddings zu Positionen ausgeführt (siehe e). Abbildung in Anlehnung an Grötschla et al. [GMVW24], S. 4

Abbildung 2.4c zeigt diesen Prozess. Die Punktwolke Γ_i wird der Rewiring-Methode f_{rew} übergeben und diese erzeugt eine neue Kantenmenge E_{rew} (siehe Abschnitt 2.3.6):

$$E_{\mathrm{rew}} \leftarrow f_{\mathrm{rew}}(\Gamma_i).$$

Auf dieser neuen Kantenmenge E_{rew} wird eine Graph-Convolution durchgeführt, damit sich der Node-Embedding von H_{i*}, nach den Initialisierungen der Knoten, neu positionieren kann:

$$H_{i+1} \leftarrow \mathrm{Conv_{rew}}(E_{\mathrm{rew}}, H_{i*}),$$

wobei in der Projektstruktur von CoRe-GD[4] für beide Graph-Convolutionen $\mathrm{Conv_E}$ und $\mathrm{Conv_{rew}}$ der gleiche Convolution-Operator benutzt wird (betrachte diesen Prozess in Abbildung 2.4d).

[4]https://github.com/floriangroetschla/CoRe-GD

Nach der Terminierung wird abschließend eine Graph-Convolution auf den Kanten E von G durchgeführt

$$H \leftarrow \mathrm{Conv_E}(E, H_{r+1}).$$

Durch den Decoder dec wird das Embedding H auf einer Zeichnung Γ abgebildet

$$\Gamma \leftarrow \mathrm{dec}(H).$$

In Abbildung 2.4e ist die Graph-Convolution auf den originalen Kanten und der Decoder-Schritt zu sehen. Der Stress der vorhergesagten Zeichnung wird mithilfe der Fehlerfunktion bestimmt und durch Backpropagation werden die Gewichte aktualisiert.

Fehlerfunktion CoRe-GD von Grötschla et al. optimiert den Stress einer Zeichnung Γ eines Graphen G. Aufgrund der Skalierungsmöglichkeit von CoRe-GD mithilfe des Verfeinerungsmechanismus, besitzt dieser eine *skalierbare* Stressfunktion mit einem Skalierungsfaktor α, der die möglichst beste Skalierung einer Zeichnung findet. Dessen ausführliche Erklärung ist in der Arbeit von CoRe-GD zu finden [GMVW24]. Die Definition des *mittleren skalierungsinvarianten* Stresses über alle Graph-Instanzen lautet:

$$\frac{1}{|D|} \sum_{G \in D} \mathrm{stress}(G, \alpha_{G,\Gamma_G} \cdot \Gamma_G),$$

wobei D den Datensatz mit dem Graphen G, Γ_G eine Zeichnung von G und α_{G,Γ_G} den dazugehörenden Skalierungsfaktor bezeichnet. Neben dem skalierbaren Stress besitzt CoRe-GD auch die klassische Stressfunktion und den mittleren skalierungs-invarianten Stress, der nach der Anzahl der Knotenpaaren normalisiert ist.

Komplexität Da wir den Verfeinerungsmechanismus nicht benutzen, wird die Laufzeit ohne Verfeinerung vorgestellt. Die Laufzeit von CoRe-GD ist *subquadratisch*.

Sei $G = (V, E)$ ein Graph und r der begrenzende Parameter. Dann ist die Laufzeit von CoRe-GD

$$T_{\mathrm{CoRe-GD}} \in \mathcal{O}\big(\underbrace{(|E|^{\frac{3}{2}} + |E| + |V|)}_{\text{Initialisierung}} + r \cdot \underbrace{(|V| \log |V| + |V|)}_{\text{Rewiring}} + E \big) = \mathcal{O}(|E|^{\frac{3}{2}}).$$

Die Initialisierung der Knoten besteht aus den Laufzeiten zur Berechnung der Laplacian-Positional-Encodings [DJL+22], Beacon-Nodes und des Rauschens. Die Laufzeit zur Berechnung der Laplace-Positional-Encodings ist $\mathcal{O}(|E|^{\frac{3}{2}})$ [DJL+22]. Da die Anzahl der Beacon-Nodes k konstant ist und die Entfernungen dieser bis zu den Knoten mit der Breitensuche gefunden werden, liegt die Laufzeit in $\mathcal{O}(k \cdot |E|) = \mathcal{O}(|E|)$. Den Knoten ein Rauschen zuzuweisen liegt, in $\mathcal{O}(|V|)$. Die Rewiring-Methode wird in CoRe-GD mit den K-Nearest-Neighbour-Graphen durchgeführt. Laut Grötschla et al. können die mit *k-d-Trees* in $\mathcal{O}(|V| \log |V|)$ durchgeführt werden. Es werden in der Rewiring Methode n Kanten hinzugefügt, wobei $n := |V|$. Demnach liegt die Laufzeit in $\mathcal{O}(|V|)$. Am Ende des Algorithmus werden alle Kanten neu gesetzt ausgegeben, was $\mathcal{O}(E)$ Zeit benötigt. Somit liegt die Laufzeit von CoRe-GD in $\mathcal{O}(|E|^{\frac{3}{2}})$ und ist subquadratisch.

3. Methodik

Wir verwenden CoRe-GD von Grötschla et al. [GMVW24], da dieser aktuell den besten Stresswert der neuronalen Netzwerke, welche Zeichnungen von Graphen generieren, erreicht. In der Arbeit von Grötschla et al. wurde CoRe-GD mit einigen klassischen Algorithmen und neuronalen Netzwerken zum Graphzeichnen bezüglich des Stresswerts verglichen. Wir verwenden hier CoRe-GD als eine Art *Blackbox*, die uns Positionen von Knoten liefert. Diese übertragen wir an vier klassische Algorithmen zum Graphzeichnen als Initialwerte. Eine Kombinationen CoRe-GD+„Variante“ bedeutet, dass eine Variante Knotenpositionen als initialen Werte von CoRe-GD erhalten hat, wobei „Variante“ einen klassischen Algorithmus der Graphenvisualisierung bezeichnet, die in dieser Arbeit näher vorgestellt werden. Zusätzlich wurde CoRe-GD so modifiziert, dass dieser auch Positionen der klassischen Algorithmen zur Initialisierung aufnehmen kann. Diese Kombinationen nennen wir „Variante“+CoRe-GD. Dieser Schritt wird erläutert, da dies in CoRe-GD [GMVW24] nicht vorgesehen ist. Durch diese Modifikation kann CoRe-GD die Zeichnungen der klassischen Algorithmen optimieren. Die Zeichnungen von CoRe-GD, von den klassischen Algorithmen und von den Kombinationen wurden nach verschiedenen Metriken untersucht.

3.1. Klassische Algorithmen zur Graphenvisualisierung

Zwei Algorithmen sind bereits vorgestellt worden: der Spring-Embedder von Fruchterman und Reingold [FR91], welcher auf der Grundlage von Kräften arbeitet, und die multidimensionale Skalierung von Kamada und Kawai [KK89]. Siehe dazu die Abschnitte 2.1.1 und 2.1.2.

Der Algorithmus ForceAtlas2 von Jacomy et al. [JVHB14][1] ist ein kräftebasierter Algorithmus zum Graphzeichnen. Im Gegensatz zu der Abstoßung im Spring-Algorithmus von Fruchterman und Reingold [FR91] (siehe Abschnitt 2.1.1) wird bei dem ForceAtlas2 eine Abstoßung abhängig vom Knotengrad durchgeführt. Die Idee von Jacomy et al. ist es die Abstoßungskraft zwischen starkverbundenen und schwachverbundenen Knoten abzuschwächen. Als Features besitzt ForceAtlas2 u. a. eine logarithmische Anziehungskraft [Noa07] oder eine modellierte Gravitationskraft um nicht-zusammenhängende Komponenten des Graphen ins Zentrum der Zeichenfläche zu ziehen.

Der Algorithmus $(\text{sgd})^2$ von Zheng et al. [ZGP17][2] optimiert den Stress einer Zeichnung mithilfe des *Stochastic-Gradient-Descent* (SGD). Der Gradienten-Abstieg ist eine

[1] Implementierung von ForceAtlas2 von Jacomy et al.: https://github.com/bhargavchippada/forceatlas2
[2] Implementierung von $(\text{sgd})^2$ von Zheng et al.: https://github.com/jxz12/s_gd2

iterative Methode, wobei in jedem Schritt eine gewählte Fehlerfunktion schrittweise optimiert wird. Im besten Fall konvergiert diese Methode und die Fehlerfunktion ist approximiert. Im Gegensatz zu den anderen hier benutzten kräftebasierten Algorithmen wird im (sgd)^2 nur eine Kante zu einer Zeit bewegt und optimal positioniert.

3.2. Positionsübergabe an CoRe-GD

Eine Übergabe von Knotenpositionen an CoRe-GD ist in der Arbeit von Grötschla et al. [GMVW24] nicht beabsichtigt. Wir modifizieren die Ausführungsmethode von CoRe-GD so, dass an diese Knotenpositionen übergeben werden kann, die diese Methode in der Rewiring-Methode verarbeitet. In Python [VRD09] können *None-Types* verwendet werden, welche einen Wert besitzen können. Falls nicht werden diese als *None* verwendet.

Sei $G = (V, E)$ ein Graph, und dieser wird an CoRe-GD übergeben. Zuerst wird nach der Architektur (siehe Abschnitt 2.3.6) eine standardmäßige Kodierung der Knoten zu dem Node-Embedding H durchgeführt. Nach unserer Implementierung wird, falls für den Parameter pos $= \Gamma_{\text{ALG}}$ gilt, wobei Γ_{ALG} die generierten Positionen der Zeichnung der klassischen Methoden als Punktwolke bezeichnen, Γ_{ALG} der internen Rewiring-Methode übergeben

$$E_{rew} \leftarrow f_{\text{rew}}(\Gamma_{\text{ALG}}).$$

Auf dem Graphen $G_{\text{rew}} := (V, E_{rew})$ wird eine Graph-Convolution durch geführt:

$$H^* \leftarrow \text{Conv}_{\text{rew}}(E_{\text{rew}}, H).$$

Das Embedding wird CoRe-GD übergeben und dieser führt einen gewöhnlichen Ablauf aus. Durch diesen Prozess kann CoRe-GD von Grötschla et al. [GMVW24] Positionen von Knoten aufnehmen.

3.3. Metriken und deren Implementierung

Wir haben, neben dem Stress, weitere Metriken der Zeichnungen von Graphen von CoRe-GD, den klassischen Methoden und Kombinationen untersucht. CoRe-GD ist in Python [VRD09] u. a. mithilfe den Bibliotheken PyTorch [PGM$^+$19] und PyGeometric (PyG) [FL19] implementiert[3]. In diesem Abschnitt werden die Metriken und deren Implementierung erläutert. Da die Zeichnungen der Algorithmen unterschiedlich skaliert worden waren, haben wir bestimmte Metriken nach der maximalen Seitenlänge der Bounding-Box normalisiert.

Für den folgenden Abschnitt sei ein Graph $G = (V, E)$ und eine Liste mit Koordinaten der Knoten $\Gamma(v)$ für alle Knoten v des Graphen G, die von den Algorithmen errechnet wurden, gegeben. Damit kann die Strecke einer Kante als $(\Gamma(v), \Gamma(u))$ für alle $\{u, v\} \in E$ berechnet werden.

[3] https://github.com/floriangroetschla/CoRe-GD

3.3.1. Stress

Gesucht ist der minimale Stress einer Zeichnung. Zur Berechnung des Stresswerts der Zeichnungen, benutzen wir die Implementierung der skalierten Stressfunktion von CoRe-GD [GMVW24][4].

3.3.2. Kantenkreuzung

Für die Experimenten werden Kantenkreuzungen als „Xing“ abgekürzt und gesucht ist die minimale Anzahl der Kantenkreuzungen einer Zeichnung.

Seien $(u, v), (r, s) \in E$ zwei nicht-adjazente mit $(u, v) \neq (r, s)$ und damit $s_1 := (\Gamma(u), \Gamma(v)), s_2 := (\Gamma(r), \Gamma(s))$ [Ren14]. Es wurde ausgeschlossen, dass s_1 und s_2 beide vertikal oder beide horizontal auf der Zeichenfläche liegen. Beide Strecken wurden in Parameterdarstellung gebracht, mit $t_1, t_2 \in [0, 1]$:

$$(1 - t_1) \cdot \Gamma(u) + t_1 \cdot \Gamma(v) \text{ für } s_1,$$
$$(1 - t_2) \cdot \Gamma(r) + t_2 \cdot \Gamma(s) \text{ für } s_2.$$

Dann gibt es drei Fälle: s_1 und s_2 liegen auf einer Gerade, s_1 und s_2 liegen auf unterschiedlichen parallelen Geraden oder die Geraden von s_1 und s_2 haben genau einen Schnittpunkt. Für alle Punkte $\Gamma(v)$ sei $\Gamma(v)_x$ die x-Koordinate und entsprechend $\Gamma(v)_y$ für die y-Koordinate. Für die Schnittpunktberechnung gilt dann

$$(1 - t_1) \cdot \Gamma(u)_x + t_1 \cdot \Gamma(v)_x = (1 - t_2) \cdot \Gamma(r)_x + t_2 \cdot \Gamma(s)_x \tag{3.1}$$
$$(1 - t_1) \cdot \Gamma(u)_y + t_1 \cdot \Gamma(v)_y = (1 - t_2) \cdot \Gamma(r)_y + t_2 \cdot \Gamma(s)_y. \tag{3.2}$$

Der Schnittpunkt erfüllt Gleichung (3.1) und Gleichung (3.2). Auflösung nach t_1 ergibt sich zu

$$t_1 = \frac{(\Gamma(u)_x - \Gamma(r)_x) \cdot (\Gamma(r)_y - \Gamma(s)_{y)} - (\Gamma(u)_y - \Gamma(r)_y) \cdot (\Gamma(r)_x - \Gamma(s)_x)}{(\Gamma(u)_x - \Gamma(v)_x) \cdot (\Gamma(r)_y - \Gamma(s)_y) - (\Gamma(u)_y - \Gamma(v)_y) \cdot (\Gamma(r)_x - \Gamma(s)_x)}. \tag{3.3}$$

Der Parameter t_2 ergibt sich analog durch Auflösung. Es gibt drei Fälle:

1. Falls der Nenner von Gleichung (3.3) gleich 0 ist, dann besitzen s_1 und s_2 die gleiche Steigung und schneiden sich nicht.

2. Falls der Nenner von Gleichung (3.3) ungleich 0 ist, dann gibt es zwei Fälle
   a. Es gilt $t_1 < 0$ oder $t_1 > 1$ oder $t_2 < 0$ oder $t_2 > 1$, dann schneiden sich die Strecken nicht, nur die Geraden auf denen die Strecken liegen.
   b. Es gilt $0 \leq t_1 \leq 1$ und $0 \leq t_1 \leq 1$, dann schneiden sich die Strecken. Der Schnittpunkt kann berechnen werden, indem t_1 bzw. t_2 in s_1 bzw. s_2 eingesetzt wird.

[4] https://github.com/floriangroetschla/CoRe-GD

3.3.3. Metriken bezüglich der Kantenlängen

Um Aussagen über eine einheitliche Kantenlänge einer Zeichnung treffen zu können, wurden zwei Metriken implementiert: die Standardabweichung bzgl. der Kantenlänge und das Kantenlängenverhältnis. Beides wurde mit der NumPy-Bibliothek [HMvdW$^+$20] von Python berechnet. Die Ergebnisse wurden nach der Bounding-Box normalisiert, da die Zeichnungen der Methoden unterschiedlich skaliert sind. Normalisiert wurden diese nach der längsten Seite der Bounding-Box. In den Experimenten wird die Standardabweichung bzgl. der Kantenlänge als „Edge“ abgekürzt. Hier ist es besser,wenn der Wert der Standardabweichung klein ist, denn die Kanten sollten möglichst einheitlich lang sein und nicht bzgl. ihrer Länge schwanken. Das Kantenlängenverhältnis beschreibt das Verhältnis der kleinsten zur größten Kante bezüglich der Länge. Das Kantenlängenverhältnis soll möglichst groß sein. Dieses wurde in den Experimenten als „Ratio“ abgekürzt.

3.3.4. Minimaler Abstand eines Knotens zu einem nicht-adjazenten Knoten

Der minimale Abstand eines Knoten zu einem nicht-adjazenten Knoten wird als „Dist-NN“ abgekürzt. Der Abstand wurde nach der längsten Seite der Bounding-Box normalisiert. Dieser Abstand soll möglichst groß sein, um eine Überlappung der Knoten zu vermeiden und diese unterscheiden zu können. Wenn $\Gamma(v)$ und $\Gamma(u)$ zwei Knotenpositionen sind, wobei $u, v \in G$ und $u \notin N(v)$. Dann lässt sich der Abstand durch die Abstandsberechnung $\text{dist}(u, v)$ im Raum mit der Python Bibliothek NumPy [HMvdW$^+$20] berechnen.

3.3.5. Minimaler Abstand eines Knotens zu einer nicht-inzidenten Kante

Der Abstand eines Knoten zu einer nicht-inzidente Kante sollte möglichst groß sein, damit Knoten nicht auf Kanten liegen. Diese Metrik wird in Kapitel 4 als „DistNE“ abgekürzt. Seien ein Punkt $\Gamma(v)$ und eine Kante $e := (\Gamma(r), \Gamma(s))$ gegeben. Es wird mit Vektoren und deren Richtungen gearbeitet [Gee24]. Seien die Vektoren $\overrightarrow{rv} := \overrightarrow{\Gamma(r)\Gamma(v)}$, $\overrightarrow{sr} := \overrightarrow{\Gamma(r)\Gamma(s)}$ und $\overrightarrow{sv} := \overrightarrow{\Gamma(s)\Gamma(v)}$. Dann gibt es drei Fälle

1. Der nächste Punkt auf der Strecke e von $\Gamma(v)$ ist der Endpunkt $\Gamma(s)$ von s, falls das Skalarprodukt $\overrightarrow{sv} \cdot \overrightarrow{rs}$ positiv ist.

2. Der nächste Punkt auf der Strecke e von $\Gamma(v)$ ist der Endpunkt $\Gamma(r)$ von s, falls das Skalarprodukt $\overrightarrow{rs} \cdot \overrightarrow{rv}$ negativ ist.

3. Falls die Skalarprodukte $\overrightarrow{sv} \cdot \overrightarrow{rs}$ und $\overrightarrow{rs} \cdot \overrightarrow{rv}$ unterschiedliche Vorzeichen besitzen, dann existiert ein Punkt w auf der Strecke e, wobei die Strecke $[v, w]$ der minimale Abstand der Distanz zwischen dem Punkt $\Gamma(v)$ und der Kante e berechenbar ist durch

$$|vw| = \frac{|\overrightarrow{rs} \times \overrightarrow{rv}|}{|\overrightarrow{rs}|},$$

wobei $\times$ das Kreuzprodukt und $|\cdot|$ den Betrag darstellen [Ser].

3.3.6. Minimaler Kreuzungswinkel

In der Implementierung wird der kleinste Kreuzungswinkel einer Zeichnung bestimmt. Je größer der kleinste Kreuzungswinkel der Zeichnungen ist desto besser, um die Kanten unterscheiden zu können. Dieser wird in den Experimenten als „XAngle“ abgekürzt. Seien zwei Kanten $s_1 := (\Gamma(u), \Gamma(v)), s_2 := (\Gamma(r), \Gamma(s))$ in der Zeichnung Γ, die einen Schnitt haben und zwei Geraden g_1, g_2, wobei s_1 auf g_1 und s_2 auf g_2 liegt. Da zwei Punkte auf den Geraden g_1 und g_2 gegeben sind, kann der Richtungsvektor berechnet werden. Seien somit $\overrightarrow{s_1}$ und $\overrightarrow{s_2}$ Richtungsvektoren. Dann ist der Winkel α zwischen beiden Vektoren mithilfe des Skalarprodukts $\cdot$ und der Arkkosinus-Funktion arccos berechenbar mit

$$\alpha = \arccos\Big(\frac{\overrightarrow{s_1} \cdot \overrightarrow{s_2}}{\|\overrightarrow{s_1}\|\|\overrightarrow{s_2}\|}\Big), \tag{3.4}$$

wobei $\|\cdot\|$ die Norm des Vektors bezeichnet. Die Python-Bibliothek NumPy [HMvdW+20] stellt die Funktionen bereit und damit ist der Winkel zwischen den Richtungsvektoren berechenbar, welcher gerade der Schnittwinkel ist.

3.3.7. Winkelauflösung

In der Implementierung wird der minimale Winkel zwischen zwei Kanten, die von einem Knoten ausgehen, bestimmt. Dieser Winkel soll möglichst groß sein, um zu zeigen, dass eine Unterscheidbarkeit der Kanten möglich ist. Die Winkelauflösung wird durch „AngRes“ abgekürzt. Die Kantenmenge wurde so sortiert, dass zu jedem Knoten u inzidente Kanten die Kanten die Form (u, v) besitzen. Dies bedeutet, dass der gemeinsame Knoten zweier adjazenten Kanten auf der linken Seite liegt. Der Winkel zwischen adjazenten Kanten $s_1 := (\Gamma(u), \Gamma(v))$, $s_2 := (\Gamma(r), \Gamma(w))$ wurde gemäß Gleichung (3.4) ermittelt.

3.3.8. Abweichung des minimalen Winkels vom optimalen Winkel

Wie die Winkelauflösung gewährleistet auch diese Metrik eine Unterscheidbarkeit von ausgehenden Knoten, falls diese Abweichung klein ist. Diese Abweichung wird durch „OptAng“ abgekürzt. Diese Metrik verwendet die Winkelauflösung, um den minimalen Winkel zwischen zwei adjazenten Kanten zu bestimmen. Wir haben die minimalen Winkel um jeden Knoten $\alpha_{\min}(v)$ errechnet. Für die durchschnittliche Abweichung der minimalen Winkel vom optimalen Winkel gilt dann mit $n := |V|$

$$\mathrm{dist}_{\mathrm{abw}}(v) = \frac{1}{n} \sum_{v \in N(v)} \frac{360}{\deg(v)} - \alpha_{\min}(v),$$

wobei $\deg(v)$ für den Grad des Knoten v steht und $\alpha_{\min}(v)$ den minimalen Winkel um den Knoten v darstellt. Die durchschnittliche Abweichung des minimalen Winkel vom optimalen Winkels ist die Differenz des optimalen Winkels, d. h. $360/\deg(v)$, vom minimalen Winkel aller Knoten v. Der Winkel wurde wie für die Winkelauflösung berechnet.

4. Experiment

In diesem Experiment werden die Zeichnungen der Methoden nach den Metriken, die in der Methodik vorgestellt wurden, untersucht.

4.1. Aufbau

Für unser Experiment verwenden wird einen Teil des Datensatzes der Rome-Graphen[1] mit 11534 ungerichteten Graphen, die bis zu 100 Knoten besitzen. Wir wenden unsere Analyse auf dem Datensatz an, auf dem CoRe-GD von Grötschla et al. [GMVW24] evaluiert wurde. Dieser Datensatz besteht aus 1000 Graphen. Da CoRe-GD auf den restlichen Graphen trainiert wurde, werden diese nicht verwendet. Wir verwenden den vortrainierten CoRe-GD, der sich in der Projektstruktur von CoRe-GD befindet[2]. Als klassische Methoden haben wir Algorithmen zum Graphzeichnen gewählt, denen initiale Positionen übergeben werden können: (sgd)^2 von Zheng et al. [ZGP17][3], ForceAtlas2 von Jacomy et al. [JVHB14][4], Spring von Fruchterman und Reingold [FR91] und die multidimensionale Skalierung von Kamada und Kawai [KK89]. Wir verwenden die gleichen Parameter von (sgd)^2 und ForceAtlas2 wie die Autoren. Der Spring- und der Kamada–Kawai-Algorithmus sind intern in der Python-Bibliothek [VRD09] NetworkX [HSC08] optimiert.

Bei der Kombination CoRe-GD+„Variante“ erhält die klassische Methode die Zeichnung von CoRe-GD als initiale Positionen. Bei der Kombination „Variante“+CoRe-GD wurde zuerst ein Ablauf eines klassischen Algorithmus durchgeführt und dessen Positionen wurden an CoRe-GD übergeben. Insgesamt haben wir damit 13 unabhängige Methoden: CoRe-GD, die vier klassischen Algorithmen, vier Kombinationen mit „Variante“+CoRe-GD und vier Kombinationen mit CoRe-GD+„Variante“. Die Abarbeitungsreihenfolge der Methoden wurde randomisiert, damit keine von dieser Reihenfolge bezüglich der Laufzeit profitiert. Wir wenden auf jedem Graph die 13 Methoden an, wobei gegebene Metriken der Graphenvisualisierung und die Laufzeit bestimmt werden.

[1] http://www.graphdrawing.org/data.html
[2] https://github.com/floriangroetschla/CoRe-GD
[3] https://github.com/jxz12/s_gd2
[4] https://github.com/bhargavchippada/forceatlas2

Wie in Abschnitt 3.3 erwähnt, werden diese neun Metriken bestimmt:

- Stress (Stress)
- Kantenkreuzungen (Xing)
- Standardabweichung bezüglich der Kantenlänge (Edge)
- Kantenlängenverhältnis (Ratio)
- Minimaler Abstand eines Knoten zu einem nicht-adjazenten Knoten (DistNN)
- Minimaler Abstand eines Knoten zu einer nicht-inzidenten Kanten (DistNE)
- Minimaler Kreuzungswinkel (XAngle)
- Winkelauflösung (AngRes)
- Durchschnittliche Abweichung des minimalen Winkels vom optimalen Winkel (OptAng).

Die Abkürzungen in Klammern beschreiben die Metriken in den nachfolgenden Tabellen. Die Laufzeit ist durch „Time“ gekennzeichnet. Demnach besteht eine Durchführung des Experiments aus insgesamt 13000 Ausführungen, wobei eine Ausführung eine Methode auf einem Graph mit zehn Werten bedeutet.

4.2. Evaluation

In diesem Abschnitt werden die Untersuchungsergebnisse vorgestellt. In den Abbildungen werden die durchschnittlichen Werte der Qualitätsmetriken bezüglich der Anzahl an Knoten der Graphen dargestellt. Zur Darstellung der Werte der Qualitätsmetriken haben wir die Python-Bibliothek Matplotlib von Hunter [Hun07] verwendet.

In der Tabelle 4.1 sind die durchschnittlichen Werte der Qualitätsmetriken der hier genannten Methoden dargestellt. In allen Tabellen sind in der ersten Spalte die hier benutzten Methoden aufgeführt und deren Namen sind u. U. abgekürzt. ForceAtlas2 erreicht den höchsten und CoRe-GD+Kamda–Kawai den niedrigsten Stresswert. CoRe-GD und Kombinationen mit „Variante“+CoRe-GD erzielen einen sehr guten Stresswert. Die klassischen Algorithmen profitieren von den schon vorgerechneten Positionen von CoRe-GD, außer der Spring, welcher einen schlechteren Stresswert errechnet. Die Kombination CoRe-GD+ForceAtlas2 erzielt die minimale Anzahl an Kantenkreuzungen mit 22,28, welche 23% besser als CoRe-GD alleine ist. Den höchsten Wert an Kantenkreuzungen erreicht die Kombination CoRe-GD+Spring. Die Längen- und Abstandsberechnungen werden nach der Bounding-Box normalisiert, aufgrund der unterschiedlichen Skalierung der Methoden. Die einheitliche Länge der Kanten ist von CoRe-GD und den Kombinationen „Variante“+CoRe-GD am besten gewährleistet. ForceAtlas2 erreicht den schlechtesten Wert der Kantenlängen. Die besten Werte für das Kantenlängenverhältnis erzielen der Kamada–Kawai-Algorithmus und CoRe-GD+Kamda–Kawai. Bei Spring, CoRe-GD+Spring, ForceAtlas2 und CoRe-GD+ForceAtlas2 ist das Kantenlängenverhältnis am

Tab. 4.1.: Die durchschnittlichen Werte der Methoden bezüglich der Qualitätsmetriken auf dem kompletten Datensatz. Die Zeilen beschreiben die Methoden und die Spalten die Qualitätsmetriken. Die Richtung der Pfeile hinter den Metriken kennzeichnet, ob größere oder kleine Werte für die Metrik besser sind. Farblich-hinterlegte Werte kennzeichnen den besten (Grün) bzw. schlechtesten (Rot) Wert innerhalb einer Spalte.

Methode	Stress↓	Xing↓	Edge↓	Ratio↑	DistNN↑	DistNE↑	XAngle↑	AngRes↑	OptAng↓	Time↓
CoRe	233,31	29,03	0,0209	0,0744	0,0591	0,0209	29,64	14,68	46,22	0,113
$(sgd)^2$	235,91	30,42	0,0215	0,0739	0,0520	0,0182	27,61	12,60	46,80	0,001
CoRe+$(sgd)^2$	235,15	30,46	0,0213	0,0736	0,0526	0,0184	27,61	12,92	46,79	0,113
$(sgd)^2$+CoRe	233,51	28,93	0,0209	0,0745	0,0591	0,0205	29,53	14,47	46,16	0,118
FA2	424,65	35,59	0,0552	0,0536	0,0560	0,0105	27,46	8,61	45,91	0,010
CoRe+FA2	302,19	22,28	0,0447	0,0495	0,0549	0,0135	29,20	10,19	40,26	0,125
FA2+CoRe	233,39	28,89	0,0209	0,0745	0,0589	0,0204	29,67	14,50	46,18	0,129
Spring	375,75	40,19	0,0330	0,0538	0,0441	0,0102	24,24	8,07	49,28	0,008
CoRe+Sp.	412,68	45,60	0,0375	0,0584	0,0467	0,0089	23,22	7,30	53,23	0,124
Sp.+CoRe	233,43	28,96	0,0208	0,0744	0,0588	0,0205	29,58	14,39	46,18	0,124
KK	239,11	29,96	0,0218	0,0754	0,0588	0,0194	29,68	14,33	46,73	0,025
CoRe+KK	232,30	29,08	0,0209	0,0754	0,0593	0,0209	29,78	14,99	46,08	0,136
KK+CoRE	233,44	28,99	0,0209	0,0742	0,0591	0,0207	29,42	14,51	46,16	0,142

schlechtesten. Der Abstand der Knoten zu einem nicht-adjazenten Knoten ist bei Spring und CoRe-GD+Spring am kleinsten, und somit erreichen diese Methoden den schlechtesten Wert. CoRe-GD, Kamada–Kawai, Kombinationen mit „Variante“+CoRe-GD und die CoRe-GD+Kamada–Kawai-Kombination erzielen sehr gute Werte. Das gleiche gilt für den minimalen Abstand eines Knotens zu einer nicht-inzidenten Kante. Den schlechtesten Wert erzielt hier CoRe-GD+Spring, wobei auch Spring und ForceAtlas2 eher schlechte Werte erreichen. Die Winkel werden hier in Grad angegeben. Einen sehr guten Wert des minimalen Kreuzungswinkel erzielt CoRe-GD+Kamda–Kawai, gefolgt von Kamada–Kawai, CoRe-GD und Kombinationen von „Variante“+CoRe-GD. Den schlechtesten Wert bezüglich des Kreuzungswinkels erreicht hier CoRe-GD+Spring. Bezüglich der Winkelauflösung ist der beste Wert der Winkelauflösung von CoRe-GD+Kamada–Kawai mehr als doppelt so groß wie die Winkelauflösung von CoRe-GD+Spring. Bei der Winkelauflösung erzielen auch CoRe-GD, die Kombinationen „Variante“+CoRe-GD und der Kamada–Kawai-Algorithmus sehr gute Werte. Die Methode CoRe-GD+ForceAtlas2 erzielt bezüglich der durchschnittlichen Abweichung des minimalen Winkels vom optimalen Winkel den kleinsten Wert. Den größten Wert liefert CoRe-GD+Spring. Die Laufzeit wurde in Sekunden berechnet. Mit Abstand erzielt $(sgd)^2$ den besten Wert bezüglich der Laufzeit, während die Methode Kamada–Kawai+CoRe-GD die höchste Laufzeit besitzt. Grundsätzlich lässt sich sagen, dass CoRe-GD mindestens eine Größenordnung langsamer als die verwendeten klassischen Graphzeichnenalgorithmen ist. Dementsprechend ist die Laufzeit einer Kombination mit CoRe-GD erwartungsgemäß höher.

In Abbildung 4.1 sind 13 Zeichnungen des Graphen mit der ID 2 aus dem randomisierten Testsatz des Rome-Graphen-Datensatzes von CoRe-GD zu sehen. In der Tabelle 4.2

Tab. 4.2.: Werte der Metriken zum Graphen mit ID 2 im Testdatensatz

Methode	Stress↓	Xing↓	Edge↓	Ratio↑	DistNN↑	DistNE↑	XAngle↑	AngRes↑	OptAng↓	Time↓
CoRe	49,92	2	0,0090	0,0774	0,0511	0,0308	54,01	28,47	27,72	0,099
$(sgd)^2$	50,79	2	0,0111	0,0734	0,0380	0,0066	48,84	5,18	28,41	0,001
CoRe+$(sgd)^2$	49,96	3	0,0088	0,0715	0,0454	0,0165	49,81	13,33	27,40	0,120
$(sgd)^2$+CoRe	50,02	2	0,0091	0,0825	0,0492	0,0273	55,12	25,75	28,06	0,107
FA2	243,73	10	0,0555	0,0510	0,0486	0,0054	56,49	6,80	36,52	0,013
CoRe+FA2	84,22	1	0,0367	0,0405	0,0615	0,0398	89,36	29,40	24,30	0,129
FA2+CoRe	50,00	2	0,0091	0,0823	0,0496	0,0280	54,92	25,93	28,04	0,135
Spring	123,59	6	0,0263	0,0387	0,0341	0,0037	48,06	3,63	36,89	0,005
CoRe+Spring	192,87	7	0,0358	0,0571	0,0514	0,0008	34,79	14,95	45,20	0,069
Spring+Core	50,03	2	0,0091	0,0826	0,0491	0,0272	55,13	25,66	28,07	0,096
KK	49,75	3	0,0096	0,0832	0,0601	0,0312	49,48	27,95	28,06	0,011
CoRe+KK	49,71	2	0,0092	0,0768	0,0594	0,0297	50,14	33,81	27,37	0,084
KK+CoRe	50,03	2	0,0091	0,0826	0,0492	0,0272	55,06	25,75	28,05	0,090

sind die Werte dieser Zeichnungen aufgeführt. Wie in Tabelle 4.1 zu beobachten ist, erzielt CoRe-GD+ForceAtlas2 weniger Kantenkreuzungen als CoRe-GD selbst. In der Zeichnung von CoRe-GD+ForceAtlas2 kann man gut erkennen, dass Knoten, die nur zu einem Knoten adjazent sind, nah bei dem Nachbarknoten liegen. Wir vermuten, dass dies in der internen Abstoßungskraft von ForceAtlas2, die eine Abstoßung zwischen stark-verbundenen und schwach-verbundenen Knoten abschwächt, begründet ist. Im Anhang (Kapitel A) sind weitere Zeichnungen von Graphen mit dazugehörenden Werten der Metriken dargestellt.

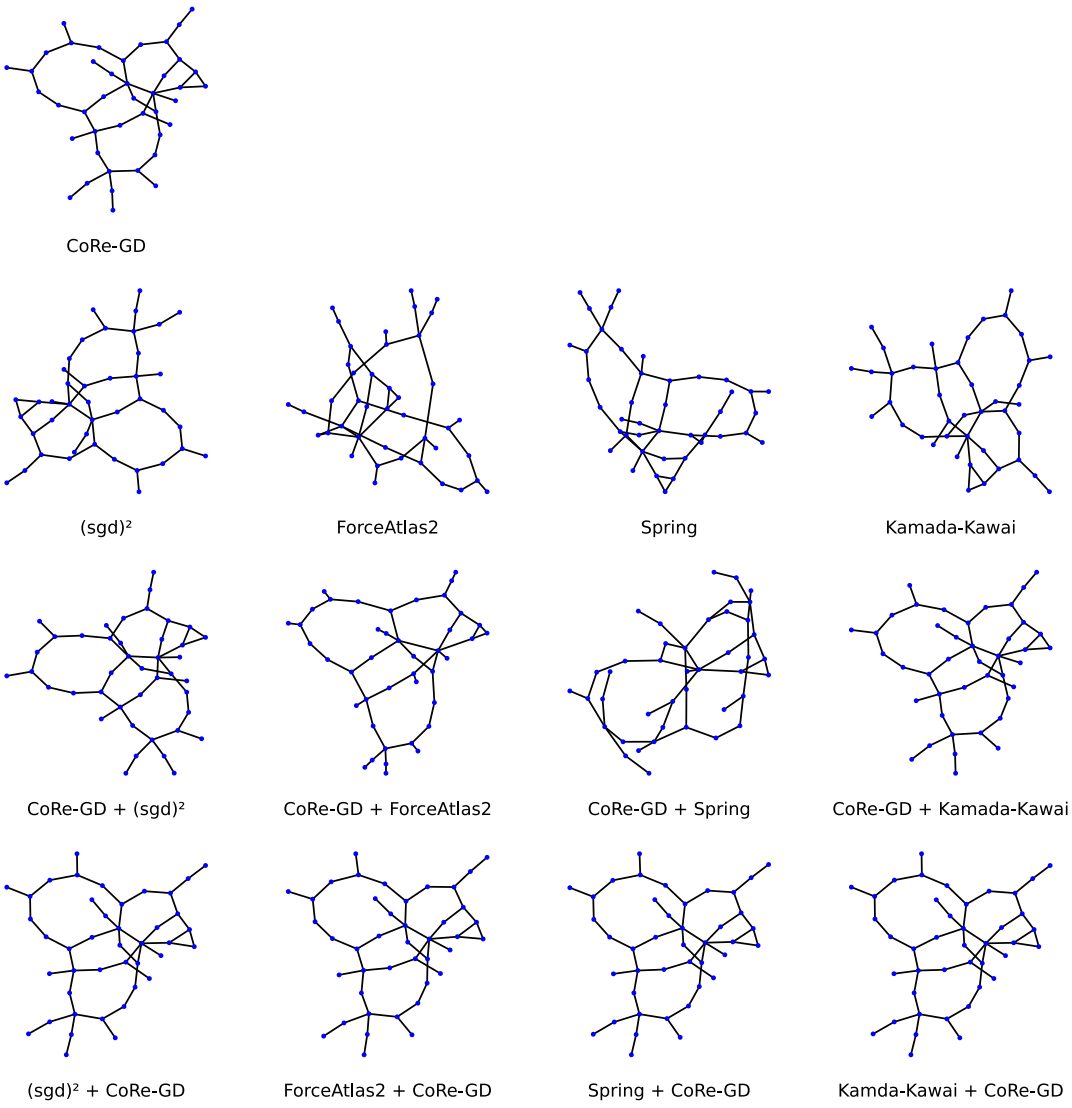
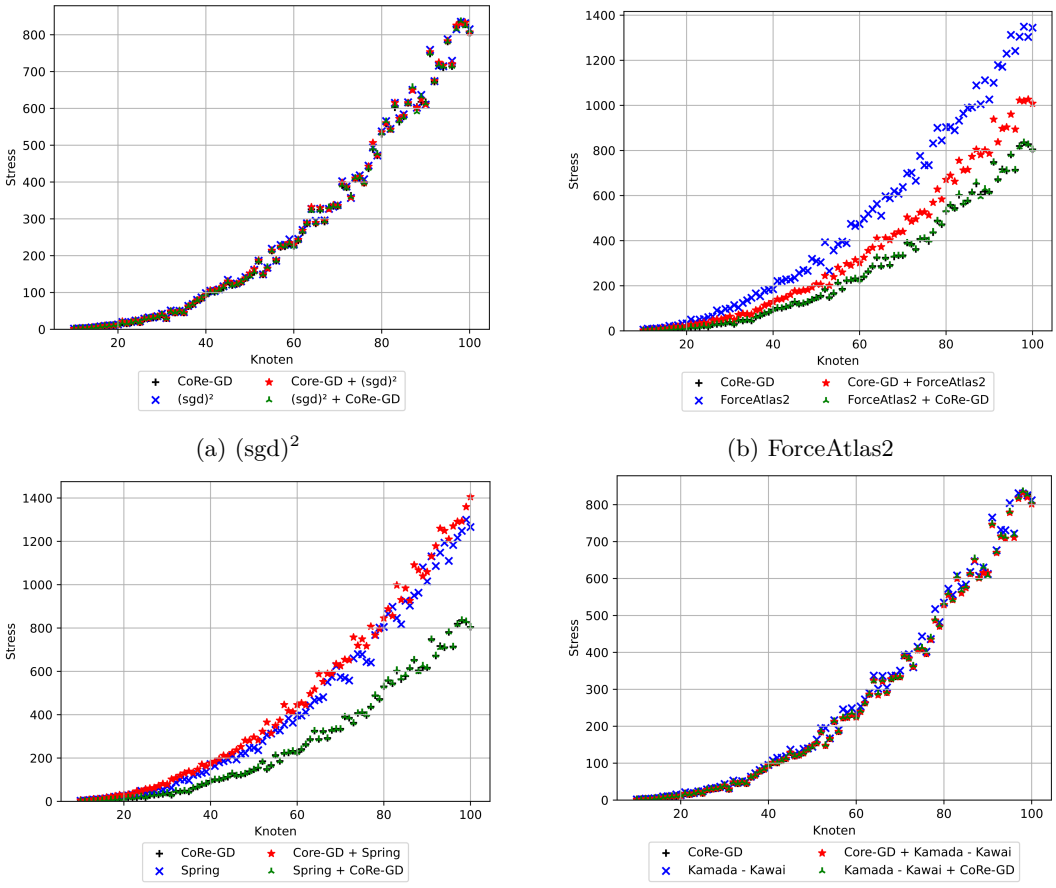


Abb. 4.1.: Graph mit ID 2 im Testdatensatz

4.2.1. Stress



(a) $(\text{sgd})^2$

(b) ForceAtlas2

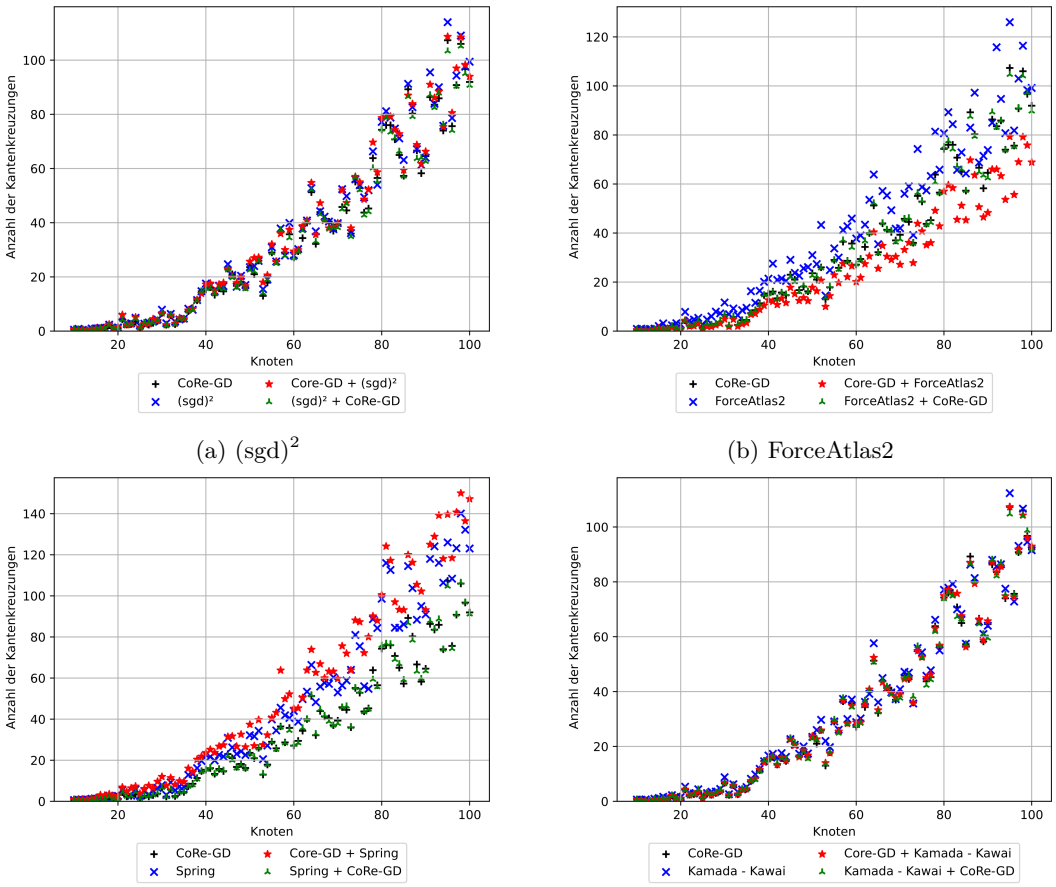
(c) Spring

(d) Kamada–Kawai

Abb. 4.2.: Stress

Im Allgemeinen steigt der Stress aller Zeichnungen, je größer der Graph wird. Die Methoden $(\text{sgd})^2$ (siehe Abbildung 4.2a), Kamada–Kawai (siehe Abbildung 4.2d), CoRe-GD und die Kombinationen weisen nahezu einen identischen Verlauf auf. In der Abbildung 4.2b ist ersichtlich, dass die Kombination ForceAtlas2+CoRe-GD und CoRe-GD die niedrigsten Werte erzielt. ForceAtlas2 profitiert von CoRe-GD als Preprocessing-Methode. Der klassische Spring-Embedder selbst profitiert nicht von den Positionen, die dieser von CoRe-GD erhält, sondern generiert einen schlechteren Stresswert (siehe Abbildung 4.2c). CoRe-GD, Spring+CoRe-GD und ForceAtlas2+CoRe-GD erzielen etwa gleiche Ergebnisse.

4.2.2. Kantenkreuzungen



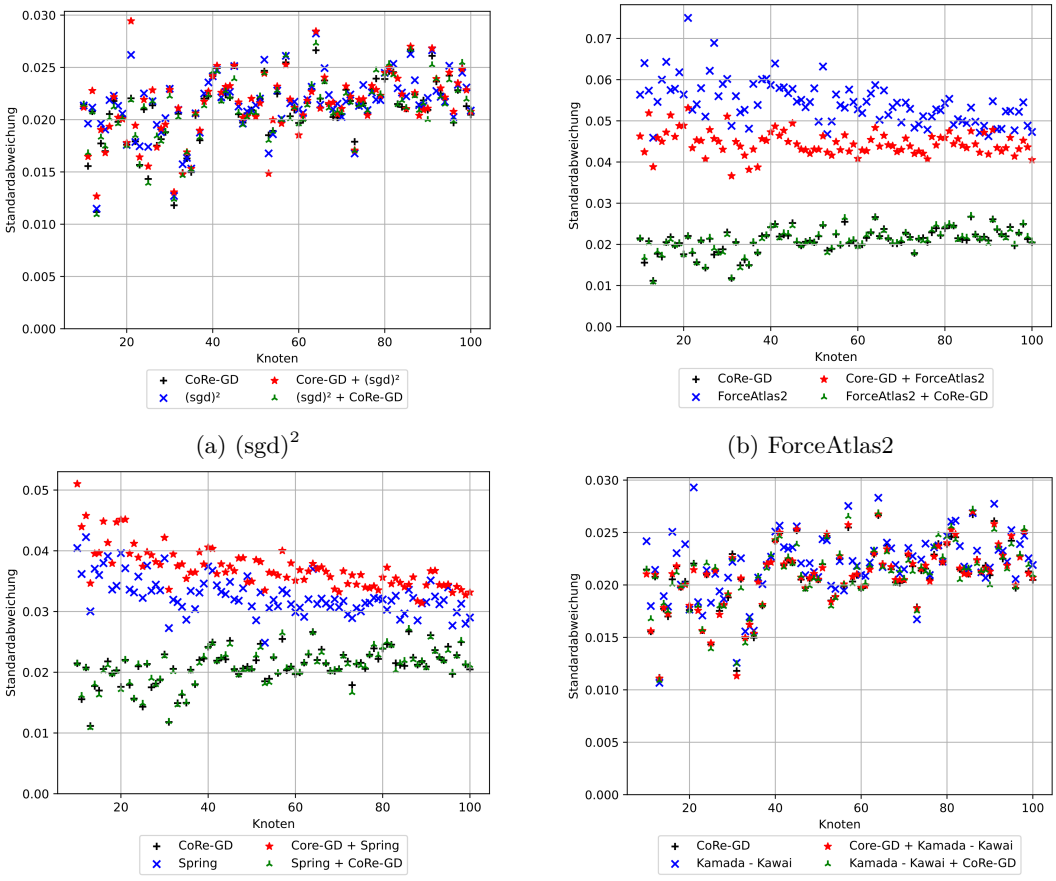
(a) $(\text{sgd})^2$ (b) ForceAtlas2

(c) Spring (d) Kamada–Kawai

Abb. 4.3.: Anzahl der Kantenkreuzungen

Generell steigt die Anzahl der Kantenkreuzungen, je größer die Graphen werden. In den Abbildungen 4.3a und 4.3d ist zu beobachten, dass $(\text{sgd})^2$, Kamada–Kawai, CoRe-GD und die Kombinationen nahezu gleiche Werte erreichen. In der Abbildung 4.3c ist zu sehen, dass CoRe-GD und Spring+CoRe-GD etwa gleich gute Werte erreichen. Die Werte von Spring+CoRe-GD liegen oberhalb von Spring und erreichen in der Kombination schlechtere Werte als die einzelnen Methoden. Wie in Tabelle 4.1 schon bemerkt, besitzt CoRe-GD+ForceAtlas2 eine deutlich niedrige Anzahl an Kantenkreuzungen als CoRe-GD und die Kombination ForceAtlas2+CoRe-GD. In Abbildung 4.3b ist zu sehen, dass ForceAtlas2 eine höhere Anzahl an Kantenkreuzungen aufweist als CoRe-GD.

4.2.3. Standardabweichung bezüglich der Kantenlänge



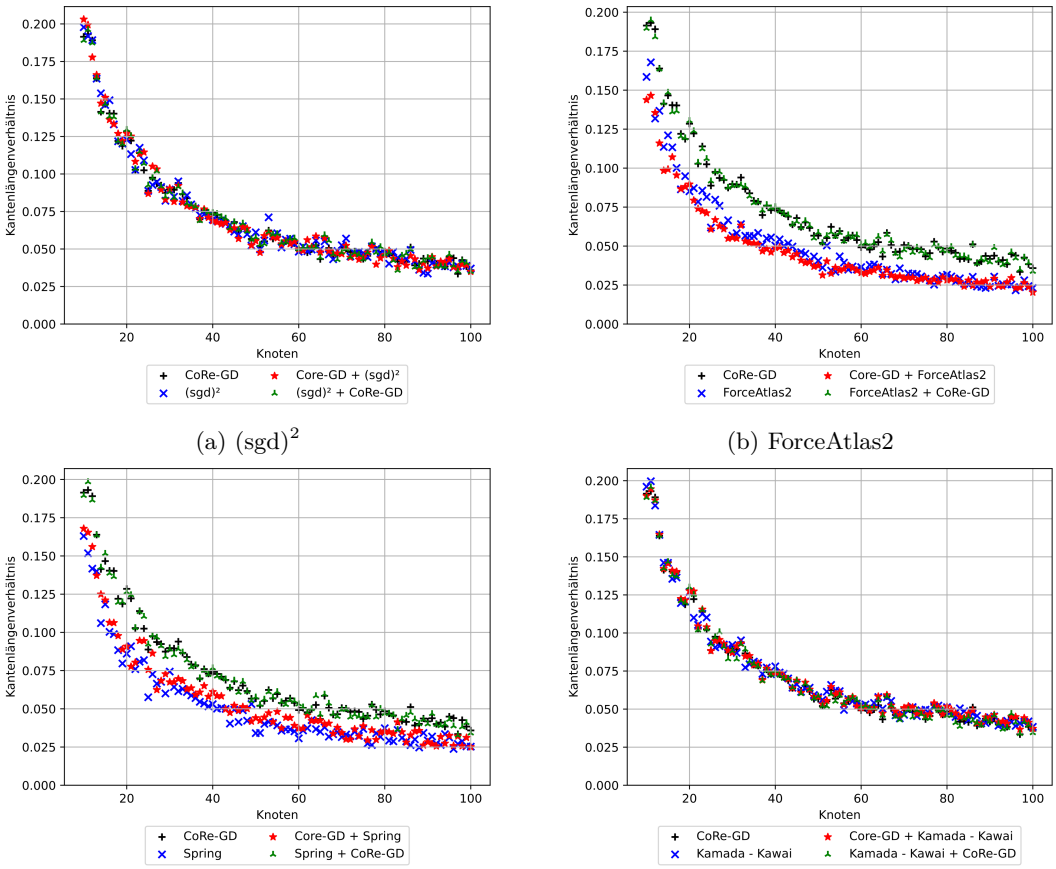
(a) $(\text{sgd})^2$ (b) ForceAtlas2

(c) Spring (d) Kamada–Kawai

Abb. 4.4.: Standardabweichung bezüglich der Kantenlänge

Es ist nicht erkennbar ist, ob die Metrik der Standardabweichung bezüglich der Kantenlänge von der Graphengröße abhängig ist. In Abbildung 4.4a ist ersichtlich, dass $(\text{sgd})^2$, das neuronale Netzwerk CoRe-GD und die Kombinationen fast identisch gleich lange Kanten aufweisen. Der klassische Algorithmus Kamada–Kawai verfügt über einen leicht höheren Wert der Standardabweichung bzgl. der Kantenlängen gegenüber CoRe-GD und den Kombinationen (siehe Abbildung 4.4d). Die Methode CoRe-GD+ForceAtlas2 weist durchgehend eine etwa doppelt so hohe Standardabweichung wie CoRe-GD und ForceAtlas2+CoRe-GD auf. Der klassische Algorithmus ForceAtlas2 besitzt eine höhere Standardabweichung bzgl. der Kantenlänge als die Kombination CoRe-GD+ForceAtlas2 (siehe Abbildung 4.4b). In Tabelle 4.1 ist dies ebenfalls gut zu beobachten. In Abbildung 4.4c ist zu sehen, dass CoRe-GD und Spring+CoRe-GD eine niedrigere Standardabweichung bzgl. der Kantenlängen als Spring und CoRe-GD+Spring aufweisen. Die Standardabweichnung von CoRe-GD+Spring ist größer als die von Spring.

4.2.4. Kantenlängenverhältnis



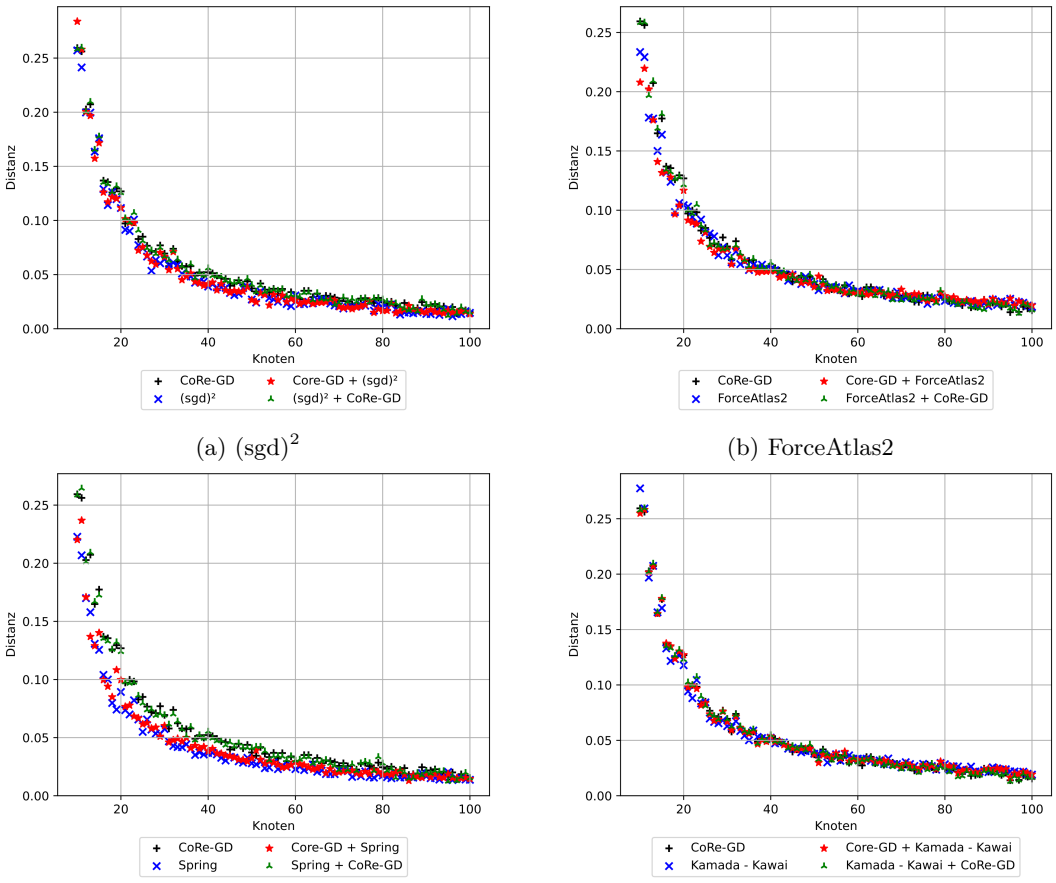
(a) $(\text{sgd})^2$ (b) ForceAtlas2

(c) Spring (d) Kamada–Kawai

Abb. 4.5.: Kantenlängenverhältnis

Das Kantenlängenverhältnis sinkt, je größer der Graph wird. Die Kurven von $(\text{sgd})^2$ und Kamada–Kawai weisen mit dem neuronalen Netzwerk CoRe-GD und deren Kombination nahezu identische Werte auf (siehe Abbildungen 4.5a und 4.5d). In Abbildung 4.5b ist zu sehen, dass der klassische Algorithmus ForceAtlas2 ein niedrigen Kantenlängenverhältnis als das neuronale Netzwerk CoRe-GD und die Kombination ForceAtlas2+CoRe-GD aufweist, aber leicht höher als CoRe-GD+ForceAtlas2 liegt. Im Allgemeinen erreicht die Methode CoRe-GD+Spring einen leicht höheren Wert als der klassische Spring-Embedder. In Abbildung 4.5c ist zu sehen, dass CoRe-GD und die Kombination Spring+CoRe-GD einen besseren Wert bezüglich des Kantenverhältnisses besitzen als CoRe-GD+Spring.

4.2.5. Minimale Distanz eines Knotens zu einem nicht-adjazenten Knoten



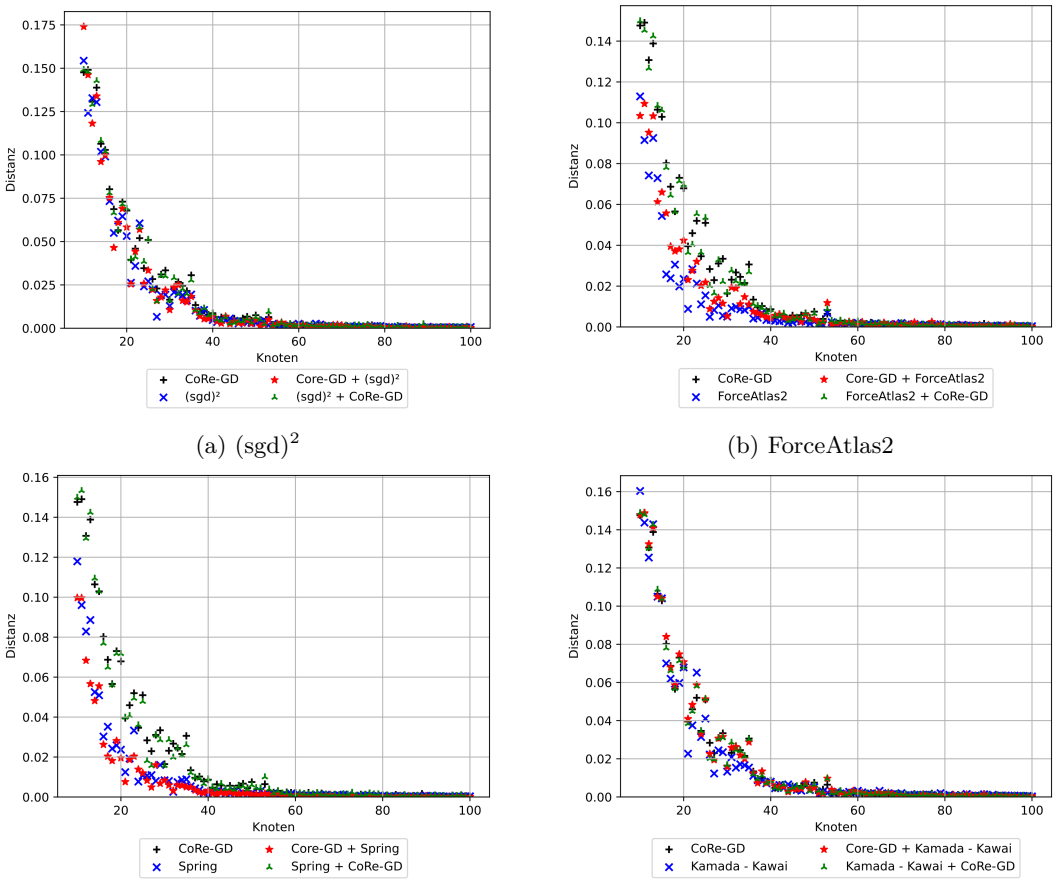
(a) $(\text{sgd})^2$ (b) ForceAtlas2

(c) Spring (d) Kamada–Kawai

Abb. 4.6.: Abstand von Knoten zu nicht-adjazenten Knoten

Im Allgemeinen sinken die Abstandswerte, je größer die Knotenanzahl ist. Die Distanzwerte von Kamada–Kawai, CoRe-GD und dessen Kombinationen mit Kamada–Kawai sind im Allgemeinen fast identisch (siehe Abbildung 4.6d). In der Abbildung 4.6a ist zu beobachten, dass der klassische $(\text{sgd})^2$ mit der Kombination $(\text{sgd})^2$+CoRe-GD nahezu gleiche Werte erzielt und minimal unter den Werten des neuronalen Netzwerks CoRe-GD und der Kombination CoRe-GD+$(\text{sgd})^2$ liegt. Die Werte von ForceAtlas2, CoRe-GD und den ForceAtlas2-Kombinationen liegen in unmittelbarer Nähe. In der Abbildung 4.6b ist ersichtlich, dass CoRe-GD+ForceAtlas2 bis ca. 70 Knoten minimal schlechtere Werte erzielt und dann leicht bessere Werte als die restlichen Methoden erreicht. Der klassische Spring-Embedder und die Kombination CoRe-GD+Spring erzielen schlechtere Werte als CoRe-GD und Spring-CoRe-GD (siehe Abbildung 4.6c).

4.2.6. Minimale Distanz eines Knotens zu einer nicht-inzidenten Kanten



(a) $(\text{sgd})^2$

(b) ForceAtlas2

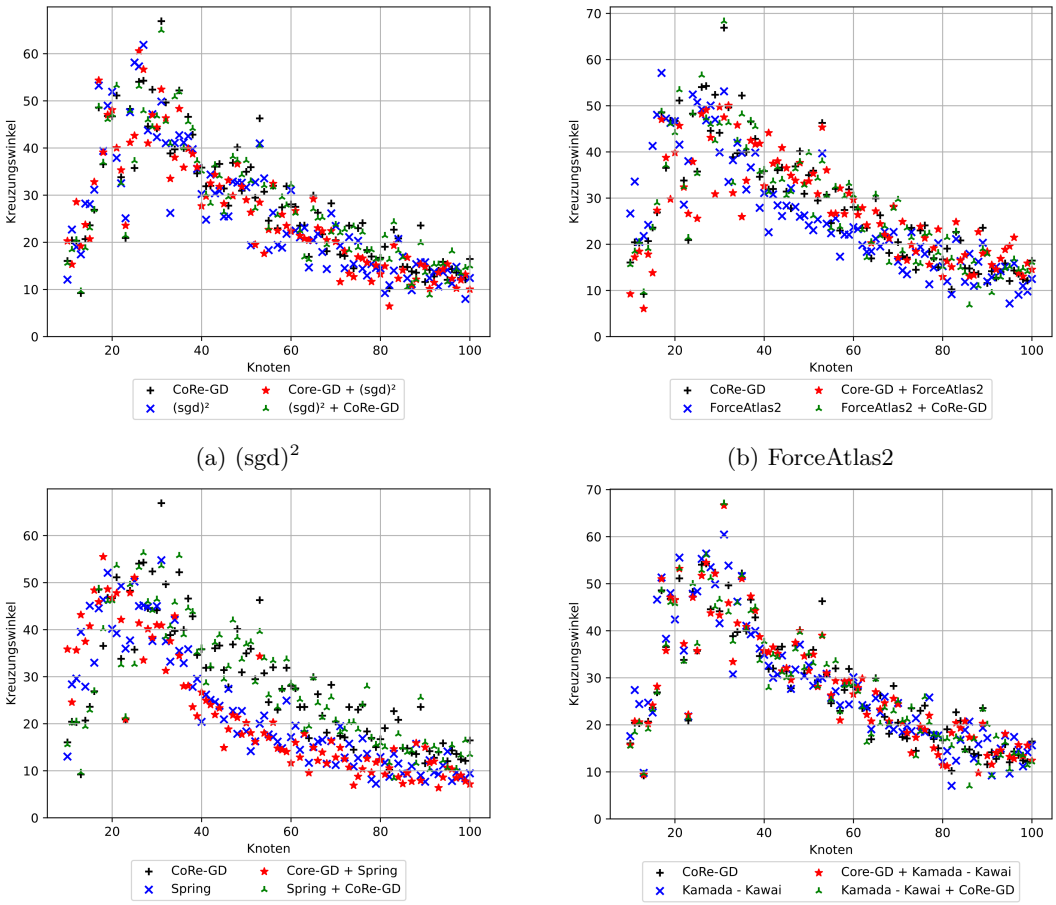
(c) Spring

(d) Kamada–Kawai

Abb. 4.7.: Abstand von Knoten zu nicht-inzidenten Kanten

Abhängig von der Knotenanzahl, fallen die Werte in allen Abbildungen bis zu einer Graphengröße von etwa 55 Knoten stark ab und nähern sich der 0 an. In allen Abbildungen erreichen CoRe-GD und die Kombinationen mit „Variante“+CoRe-GD die besseren Werte. In der Abbildung 4.7d ist zu sehen, dass alle Methoden fast gleiche Werte erzielen. Auch in Abbildung 4.7a ist zu beobachten, dass die Werte der Methoden sich fast gleichen. Die Abstandswerte von ForceAtlas2 und CoRe-GD+ForceAtlas2 sind niedriger als die des neuronalen Netzwerks CoRe-GD und ForceAtlas2+CoRe-GD, wobei CoRe-GD+ForceAtlas2 noch bessere Werte als der klassische ForceAtlas2 erreicht (siehe Abbildung 4.7b). In der Abbildung 4.7c ist ersichtlich, dass die Werte von Spring und CoRe-GD+Spring unterhalb der Werte von CoRe-GD und Spring+CoRe-GD liegen.

4.2.7. Minimaler Kreuzungswinkel



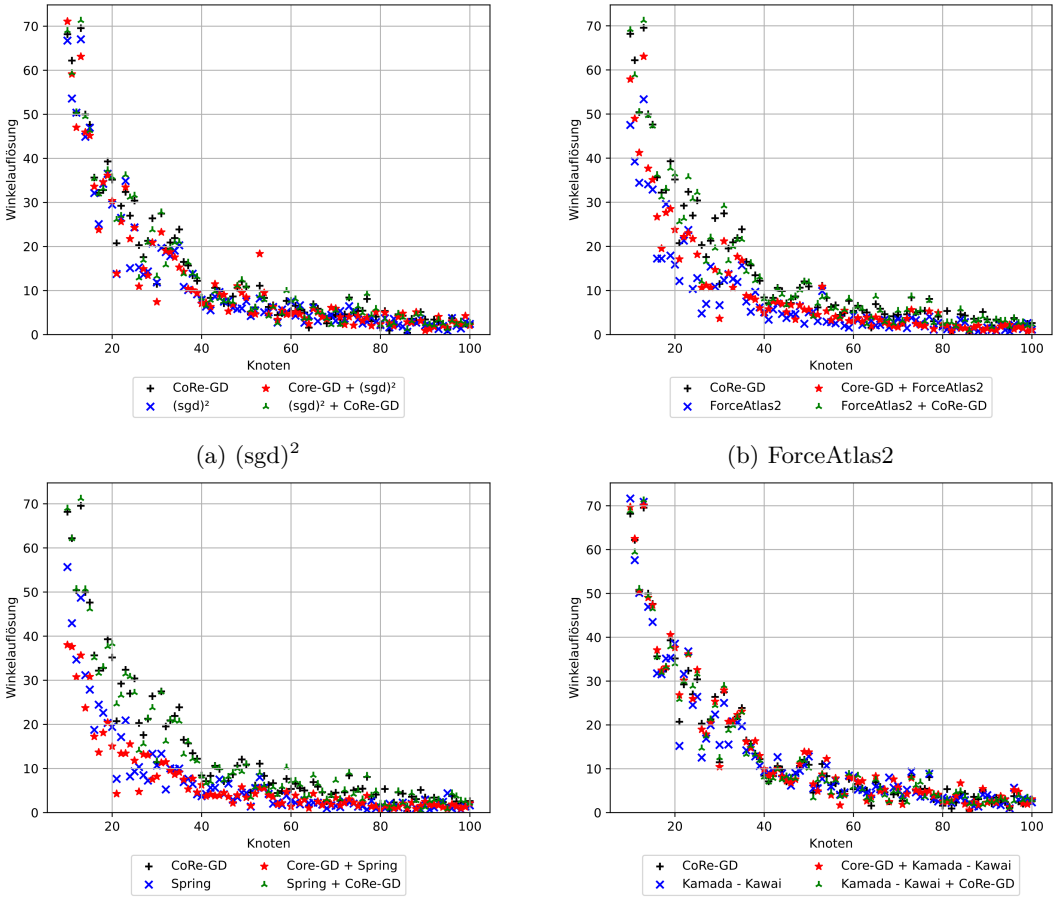
(a) $(\text{sgd})^2$ (b) ForceAtlas2

(c) Spring (d) Kamada–Kawai

Abb. 4.8.: Minimaler Kreuzungswinkel

Wir stellen fest, dass nahezu alle Werte der Methoden in Bezug auf den Kreuzungswinkel bis etwa 30 Knoten auf ungefähr 45–55 Grad ansteigen und anschließen allmählich abfallen. In den Abbildungen 4.8a und 4.8d ist ersichtlich, dass die Kurven von CoRe-GD, $(\text{sgd})^2$, Kamada–Kawai und deren Kombinationen fast identische Werte bezüglich der Kreuzungswinkel erzielen. Die Methode CoRe-GD+ForceAtlas2 besitzt bis 40 Knoten leicht kleiner Werte bzgl. des Kreuzungswinkels als CoRe-GD und ForceAtlas2+CoRe-GD, erreicht dann aber leicht höhere Werte. Der Verlauf von ForceAtlas2 ist nach links verschoben (siehe Abbildung 4.8b). Der Verlauf von Spring und CoRe-GD+Spring ist ebenfalls, wie schon bei klassischen ForceAtlas2, nach links verschoben. Diese erreichen kurz bis etwa 20 Knoten bessere Werte bzgl. des Kreuzungswinkels und fallen dann schneller ab als die Kurven von CoRe-GD und Spring+CoRe-GD (siehe Abbildung 4.8c).

4.2.8. Winkelauflösung



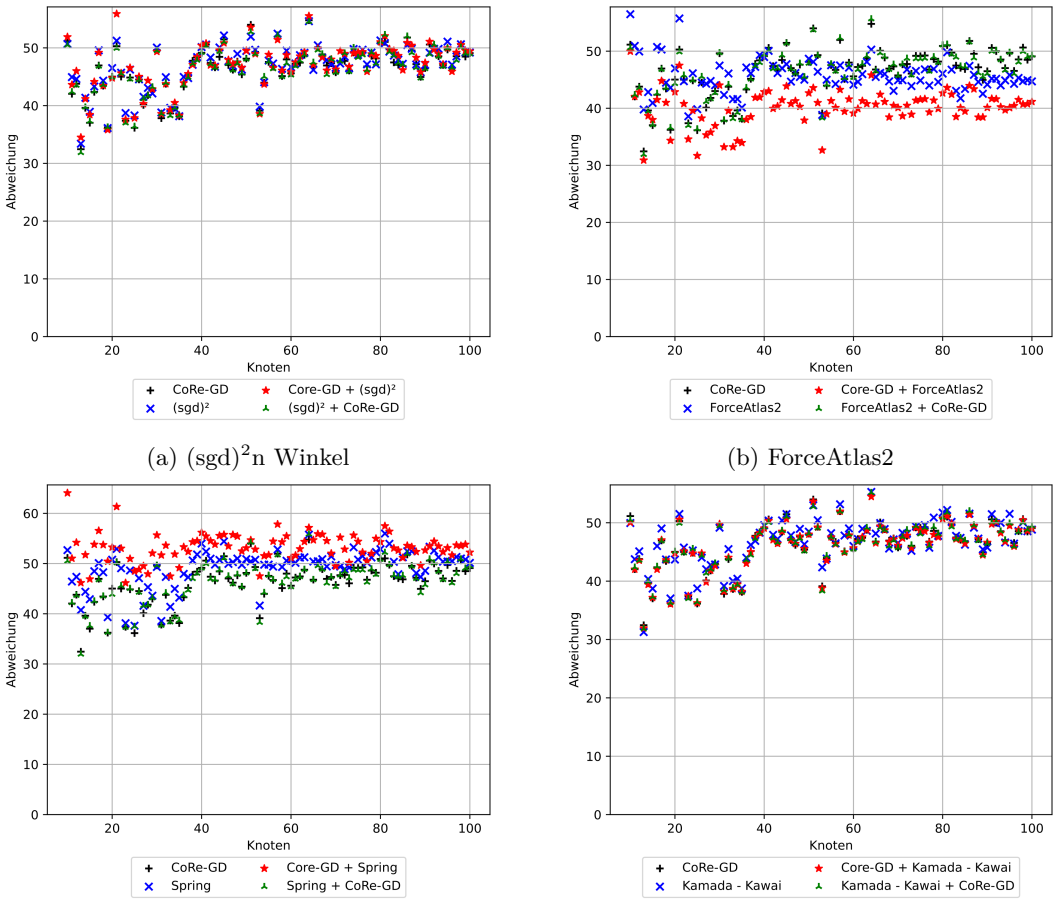
(a) $(sgd)^2$ (b) ForceAtlas2

(c) Spring (d) Kamada–Kawai

Abb. 4.9.: Minimaler Winkel an einem Knoten

Es ist zu beobachten, dass die Werte bezüglich der Winkelauflösung aller Methoden abhängig von der Knotenanzahl der Graphen abfallen. Die Werte von Kamada–Kawai, CoRe-GD und dessen Kombinationen mit Kamada–Kawai sind nahezu identisch (siehe Abbildung 4.9d). In der Abbildung 4.9a ist zu sehen, dass die Werte bzgl. der Winkelauflösung von $(sgd)^2$ und CoRe-GD+$(sgd)^2$ leicht niedriger sind als die Werte von CoRe-GD und $(sgd)^2$+CoRe-GD. Das neuronale Netzwerk CoRe-GD und ForceAtlas2+CoRe-GD erreichen bessere Werte bezüglich der Winkelauflösung als ForceAtlas2 und CoRe-GD+ForceAtlas2. Die Methode CoRe-GD+ForceAtlas2 erreicht leicht bessere Werte als ForceAtlas2 alleine (siehe Abbildung 4.9b). Der Abbildung 4.9c ist zu entnehmen, dass der Spring bessere Werte als die Kombination CoRe-GD+Spring erzielt. Allerdings erreichen CoRe-GD und Spring+CoRe-GD bessere Werte.

4.2.9. Abweichung des minimalen Winkels vom optimalen Winkel



(a) $(sgd)^2$n Winkel

(b) ForceAtlas2

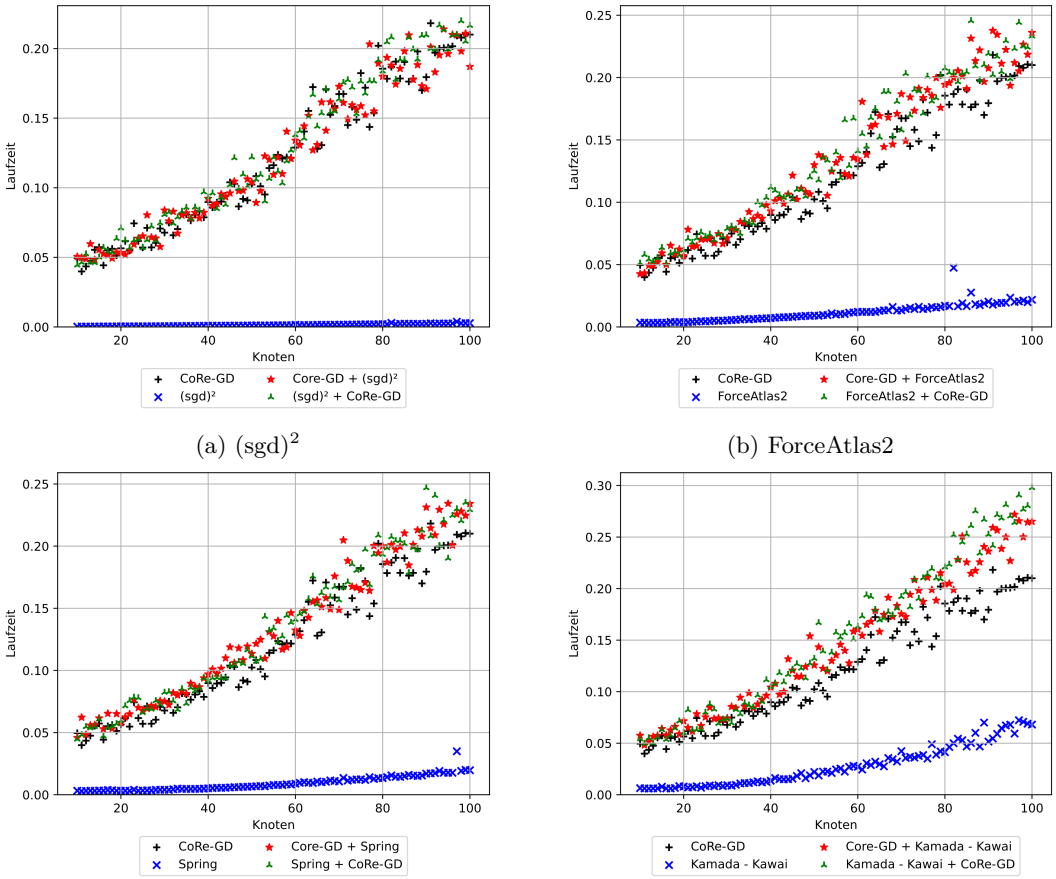
(c) Spring

(d) Kamada–Kawai

Abb. 4.10.: Durchschnittliche Abweichung der minimalen Winkel vom optimalen Winkel

Die durchschnittliche Abweichung des minimalen Winkels zum optimalen Winkel soll so klein wie möglich sein. Die Kurven der Methoden schwanken zwischen 30 und 60 Grad. Die Methoden Kamada–Kawai, $(sgd)^2$, CoRe-GD und die Kombination mit Kamada–Kawai und $(sgd)^2$ erzielen nahezu die gleichen Werte (siehe Abbildungen 4.10a und 4.10d). In Abbildung 4.10c sind die Kurven von Spring und der Kombination CoRe-GD+Spring oberhalb von CoRe-GD und Spring+CoRe-GD. Der klassische Spring erreicht bessere Werte als die Kombination CoRe-GD+Spring. Wie bereits Tabelle 4.1 bemerkt, weist die Kombination CoRe-GD+ForceAtlas2 eine geringere durchschnittliche Abweichung des minimalen Winkels vom optimalen Winkel als ForceAtlas2, CoRe-GD und CoRe-GD+ForceAtlas2 auf. Siehe dazu Abbildung 4.10b.

4.2.10. Laufzeit



(a) $(sgd)^2$

(b) ForceAtlas2

(c) Spring

(d) Kamada–Kawai

Abb. 4.11.: Laufzeit

In den Abbildungen 4.11a, 4.11b, 4.11c und 4.11d ist zu beobachten, dass die Laufzeiten der klassischen Algorithmen eine Größenordnung besser sind als die des neuronalen Netzwerks CoRe-GD und dessen Kombinationen, wobei $(sgd)^2$ die beste und Kamada–Kawai die schlechteste Laufzeit der klassischen Algorithmen aufweisen. Die Kombinationen verschlechtern die Laufzeit von CoRe-GD erwartungsgemäß etwas, wobei $(sgd)^2$ das neuronale Netzwerk CoRe-GD am wenigsten (siehe Abbildung 4.11a) und Kamada–Kawai dieses am meisten verschlechtert (siehe Abbildung 4.11d). In Abbildungen ist zu sehen, dass die Kombination „Variante“+CoRe-GD und die CoRe-GD+„Variante“ ähnliche Werte bezüglich der Laufzeit besitzen.

5. Diskussion

In diesem Abschnitt werden die Untersuchungsergebnisse und ein Einsatz der Kombinationen diskutiert. In den nachfolgenden Tabellen 5.1 und 5.2 sind die Werte der Methoden, d. h. CoRe-GD von Grötschla et al. [GMVW24], $(\text{sgd})^2$ von Zheng et al. [ZGP17], ForceAtlas2 von Jacomy et al. [JVHB14], Spring von Fruchterman und Reingold [FR91] und der multidimensionalen Skalierung von Kamada und Kawai [KK89], bezüglich ihrer Kombinationen aufgeführt. Diese Werte bezeichnen eine prozentuale Verbesserung oder Verschlechterung der Metriken im Hinblick auf die Kombinationen gegenüber CoRe-GD und den klassischen Algorithmen.

Im Folgenden, werden die Verbesserung bzw. Verschlechterung der Einzelmethoden gegenüber der Kombination CoRe-GD+„Variante“ dargestellt. Siehe dazu Tabelle 5.1.

CoRe-GD+$(\text{sgd})^2$ Die Zeichnung von CoRe-GD lässt sich durch eine Hintereinanderschaltung mit $(\text{sgd})^2$ nicht verbessern. Mit CoRe-GD als Preprocessing-Methode wird die Zeichnung des Algorithmus $(\text{sgd})^2$ kaum verändert. Der Unterschied der Laufzeit von $(\text{sgd})^2$ gegenüber CoRe-GD+$(\text{sgd})^2$ ist allerdings außerordentlich hoch. Eine Verbesserung der Zeichnung von CoRe-GD durch eine Optimierung durch $(\text{sgd})^2$ stellt sich nicht ein. Insgesamt kann diese Kombination gegenüber den Einzelmethoden nicht empfohlen werden.

CoRe-GD+ForceAtlas2 Eine Zeichnung von CoRe-GD lässt sich durch ForceAtlas2 nur bezüglich der Anzahl der Kantenkreuzungen und der Abweichung des minimalen Winkels vom optimalen Winkel erreichen – dies dafür relativ deutlich (23% bzw. 13%). ForceAtlas2 profitiert stark von den übergebenen Positionen von CoRe-GD. Die größ-

Tab. 5.1.: Die prozentuale Verbesserung (positiv) und Verschlechterung (negativ) der Kombination CoRe-GD+„Variante“ gegenüber den einzelnen Methoden . Zeilen- und Spaltenbedeutungen wie in Tabelle 4.1.

Methode	Stress	Xing	Edge	Ratio	DistNN	DistNE	XAngle	AngRes	OptAng	Time
CoRe	-0,8%	-4,9%	-2,2%	-1,1%	-11,0%	-11,8%	-6,8%	-12,0%	-1,2%	-0,2%
$(\text{sgd})^2$	0,3%	-0,1%	0,7%	-0,3%	1,1%	1,6%	0,0%	2,6%	0,0%	-9233,5%
CoRe	-29,8%	23,3%	-114,0%	-33,5%	-1,5%	-30,6%	-1,0%	-29,7%	13,1%	-11,9%
FA2	28,7%	37,4%	19,0%	-7,7%	-1,9%	28,7%	6,3%	18,3%	12,3%	-1122.3%
CoRe	-76,9%	-57,1%	-79,4%	-21,5%	-20,9%	-57,3%	-21,7%	-50,3%	-15,2%	-9,4%
Spring	-9,8%	-13,5%	-13,7%	8,6%	6,0%	-12,2%	-4,2%	-9,5%	-8,0%	-1384,7%
CoRe	0,4%	-0,2%	-0,3%	1,3%	0,5%	-0,1%	0,4%	2,1%	0,3%	-19,8%
KK	2,8%	2,9%	4,1%	0,0%	0,9%	8,0%	0,3%	4,6%	1,4%	-443,3%

Tab. 5.2.: Die prozentuale Verbesserung und Verschlechterung der Kombination „Variante“+CoRe-GD gegenüber den einzelnen Methoden . Zeilen- und Spaltenbedeutungen wie in Tabelle 4.1

Methode	Stress	Xing	Edge	Ratio	DistNN	DistNE	XAngle	AngRes	OptAng	Time
CoRe	-0,1%	0,3%	-0,1%	0.1%	0.1%	-1,9%	-0,4%	-1,4%	0,1%	-4,5%
$(sgd)^2$	1,0%	4,9%	2,7%	0,8%	13,6%	13,0%	6,9%	14,9%	1,4%	-9637,8%
CoRe	0,0%	0,5%	-0,1%	0,1%	-0,3%	-2,4%	0,1%	-1,2%	0,1%	-13,7%
FA2	45,0%	18,8%	62,1%	39,0%	5,2%	94,2%	8,0%	68,4%	-0,6%	-1153,5%
CoRe	-0,1%	0,2%	0,2%	-0,1%	-0,5%	-1,9%	-0,2%	-2,0%	0,1%	-9,1%
Spring	37,9%	28,0%	36,8%	38,1%	33,4%	101,6%	22,0%	78,4%	6,3%	-1380,7%
CoRe	-0,1%	0,1%	-0,3%	-0,3%	0,1%	-0,8%	-0,8%	-1,1%	0,1%	-25,7%
KK	2,4%	3,2%	4,1%	-1,6%	0,5%	7,1%	-0,9%	1,3%	1,2%	-470,2%

te prozentuale Verbesserung zeigt sich in den Kantenkreuzungen, Stress und Abstand der Knoten zu nicht-inzidenten Kanten. Allerdings verschlechtert sich die Laufzeit von ForceAtlas2 durch die Kombination mit CoRe-GD erheblich. Je nach Priorisierung der Metriken und der zur Verfügung stehenden Laufzeit, kann diese Kombination daher bedingt empfohlen werden.

CoRe-GD+Spring Es ist zu beobachten, dass Spring die Zeichnung von CoRe-GD als Initialwert deutlich verschlechtert. Nach der Tabelle 5.1 profitiert Spring von der guten Zeichnung, die CoRe-GD erstellt hat, kaum. Nur im Hinblick der Standardabweichung bzgl. der Kantenlänge und der Distanz eines Knoten zu nicht-adjazenten Knoten ist eine leichte Verbesserung festzustellen. Die restlichen Werte der anderen Metriken werden allerdings negativ beeinträchtigt und der Zuwachs der Laufzeit ist enorm. Von dieser Kombination kann abgeraten werden.

CoRe-GD+Kamada–Kawai Mit dem Kamada–Kawai-Algorithmus lässt sich CoRe-GD im Hinblick auf die Metriken Stress, Kantenlängenverhältnis, kleinste Distanz zweier nicht-benachbarter Knoten, Kreuzungswinkel, Winkelauflösung und Abweichung des kleinsten Winkels vom optimalen Winkel leicht verbessern. Kamada–Kawai profitiert leicht von den Positionen von CoRe-GD als initiale Positionen für die Ausführung. Nicht zu vernachlässigen ist zudem der Laufzeitzuwachs der Kombination im Vergleich zu den Einzelmethoden, insbesondere Kamada–Kawai. Insgesamt kann die Kombination gegenüber den Einzelmethoden daher kaum empfohlen werden.

Im Folgenden, werden die Verbesserung bzw. Verschlechterung der Einzelmethoden gegenüber der Kombination „Variante“+CoRe-GD dargestellt. Siehe dazu Tabelle 5.2.

$(sgd)^2$+CoRe-GD Das neuronale Netzwerk CoRe-GD profitiert nicht von einer vorverarbeiteten Zeichnung von $(sgd)^2$. Im Hinblick auf $(sgd)^2$ werden diese Werte der Kombination geringfügig verbessert. Allerdings besitzt $(sgd)^2$+CoRe-GD eine wesentlich

schlechtere Laufzeit als der klassische $(sgd)^2$. Gegenüber CoRe-GD kann diese Kombination nicht empfohlen werden.

ForceAtlas2+CoRe-GD Eine Verbesserung der Werte von CoRe-GD durch Übergabe einer Zeichnung von ForceAtlas2 stellt sich nicht ein. Die Zeichnung von ForceAtlas2 lässt sich aber durch das neuronale Netzwerk CoRe-GD wesentlich optimieren. Allerdings wird die Laufzeit durch Hinzunahme von CoRe-GD auch hier schlecht. Diese Kombination kann gegenüber CoRe-GD nicht empfohlen werden.

Spring+CoRe-GD Eine Übergabe einer Zeichnung von Spring verbessert eine Zeichnung von CoRe-GD nicht. Wiederum lässt sich eine Zeichnung von Spring allerdings durch CoRe-GD wesentlich verbessern. Die Laufzeit von Spring+CoRe-GD ist allerdings hoch gegenüber dem klassischen Spring. Gegenüber CoRe-GD kann diese Kombination nicht empfohlen werden.

Kamada–Kawai + CoRe-GD CoRe-GD profitiert nicht von einer Übergabe einer Zeichnung, die der Kamada–Kawai-Algorithmus erstellt hat. Eine Zeichnung von Kamda–Kawai lässt sich durch das neuronale Netzwerk CoRe-GD nur geringfügig verbessern, während die Laufzeit stark steigt. Insgesamt kann diese Kombination insbesondere gegenüber CoRe-GD nicht empfohlen werden.

Fazit Es ist zu beobachten, dass der Algorithmus, der als Postprocessing-Methode benutzt wird, das Ergebnis der Zeichnung maßgeblich bestimmt. Bis auf minimale Unterschiede sind die Ergebnisse von CoRe-GD und der Kombinationen mit „Variante“+CoRe-GD fast gleich. Daher ist eine Optimierung von CoRe-GD durch eine Übergabe von Positionen, die durch die klassischen Algorithmen vorberechnet werden, nicht sinnvoll. Diese Kombination ist nicht zu empfehlen, da das neuronale Netzwerk CoRe-GD allein i. A. bessere Werte liefert.

Das Modell CoRe-GD als Preprocessing-Methode zu wählen, ist nur dann sinnvoll, falls eine Metrik optimiert werden soll und die Laufzeit zweitrangig ist. ForceAtlas2 und, in weitaus geringerem Maße, Spring profitieren von einer vorverarbeiteten Zeichnung von CoRe-GD. ForceAtlas2 optimiert eine Zeichnung von CoRe-GD bezüglich der Anzahl der Kantenkreuzungen und der durchschnittlichen Abweichung des minimalen Winkels vom optimalen Winkel (um 23% bzw. 13%). Daher ist diese Kombination bedingt zu empfehlen, falls diese Metriken priorisiert sind. Die Kombination CoRe-GD+Spring ist nicht zu empfehlen. Die Methoden $(sgd)^2$ und Kamada–Kawai profitieren kaum von der Positionen von CoRe-GD und sind aufgrund des Laufzeitzuwachses insgesamt nicht zu empfehlen.

Eine Verbesserung des neuronalen Netzwerks CoRe-GD von Grötschla et al. [GMVW24] bezüglich dessen Zeichnung durch die hier benutzten Kombinationen stellte sich kaum ein. Eine Optimierung einer Zeichnung von klassischen Algorithmen durch CoRe-GD ist möglich, allerdings stellt sich die Frage, ob man CoRe-GD nicht von Anfang an nutzen

sollte oder aufgrund der wesentlich höheren Laufzeit nur einen klassischen Graphzeichnenalgorithmus nutzen sollte.

Eine hauptsächliche Limitation der Ergebnisse ist es, dass die Methoden Positionen der Knoten in unterschiedlicher numerischer Präzision liefern. Die Zeichenflächen der Algorithmen sind ebenfalls unterschiedlich skaliert und geformt. Aus diesem Grund haben wir die Koordinaten der Knotenpositionen nach der längsten Seite der Bounding-Box normalisiert. Die Bounding-Box der jeweiligen Methode ist allerdings unterschiedlich skaliert und nicht quadratisch. Falls der Graph auf der Zeichenfläche anders positioniert oder geformt ist, besitzt auch die Bounding-Box andere Werte, z. B. eine Scherung der Graphzeichnung. Somit lässt sich ein Vergleich der Zeichnungen nur bedingt durchführen. Außerdem sind interne Rundungsfehler von Python [VRD09] bei den Berechnungen der Metriken aufgetreten, da die Methode Werte unterschiedlicher Präzision lieferten. Aus diesem Grund können die Werte weiter verfälscht werden. Eine weitere Limitation ergibt sich daraus, dass die Methoden unterschiedliche Datenstrukturen mit den Knotenpositionen aufnehmen und liefern. Aus diesem Grund müssen intern Zwischenberechnungen ausgeführt werden, damit die Ausgabedatenstruktur der Eingabedatenstruktur entspricht. Durch diese Zwischenberechnungen wird die Laufzeit erhöht. Intern hat Python [VRD09] mehrere Zeitmesser, um die Laufzeit zu bestimmen. Aufgrund mangelnder Präzision eines Zeitmessers, der ausschließlich die CPU-Zeit berechnet, haben wir uns für einen Zeitmesser entschieden, der die Zeit des Prozesses berechnet auch wenn dieser z. B. inaktiv ist. Somit ist die Laufzeitberechnung nicht exakt. Als Knotenintialisierung verwendet CoRe-GD u. a. randomisierte Werte, um eine Unterscheidbarkeit des Message-Passing-Algorithmus gewährleisten zu können. Wir vermuten, dass aus diesem Grund die Zeichnungen von CoRe-GD nach jeder Ausführung leicht verändert oder gespiegelt wurden und vermutlich daher liefert jede Durchführung des Experiments leichte Unterschiede bezüglich der Metriken. Für diese Arbeit haben wir uns nur auf einen Teil der klassischen Graphzeichnenalgorithmen beschränkt. Wir können die Methoden in zwei Gruppen unterteilen: kräftebasierte Methoden (Spring, ForceAtlas2) und stressbasierte Methoden (Kamda–Kawai, $(\text{sgd})^2$, CoRe-GD). Beide Gruppen liefern jeweils ähnliche Ergebnisse. Hier wurden die beiden Gruppen gegenübergestellt. Trotzdem befolgen die beiden Algorithmen-Typen andere Strategien. Das Ziel der stressbasierten Algorithmen ist es, den Stress zu minimieren und damit eine einheitliche Kantenlänge zu schaffen, damit die graphentheoretische der euklidischen Distanz möglichst genau entsprechen. Im Gegensatz dazu versuchen kräftebasierte Algorithmen eine Anordnung der Knoten im Raum zu schaffen, wobei adjazente Knotenpaare nah beieinander und nicht-adjazente Knotenpaare weit voneinander liegen. D. h. kräftebasierte Algorithmen „versuchen“ nicht den Stress oder einheitliche Kantenlängen zu minimieren bzw. maximieren. Somit ist eine Gegenüberstellung der stressbasierten und kräftebasierten Algorithmen nur bedingt sinnvoll. Es ist damit zu erwarten, dass stressbasierte Algorithmen in einigen Metriken besser abschneiden als kräftebasierte Algorithmen.

Wir haben vor, die weiteren der Modelle DeepGD, $(\text{DNN})^2$ und SmartGD von Wang et al. [WYHS23] und derer Kombinationen nach den Metriken zu untersuchen. Eine Anwendung von DeepGD wäre interessant, da dieses Modell eine Fehlerfunktion besitzt, die auf

mehrere Metriken gleichzeitig trainiert werden kann. Zusätzliche klassische Graphzeichnenalgorithmen, die ebenfalls in das Experiment aufgenommen werden könnten, wären z. B. der Neato-Algorithmus von Gansner et al. [GKN04] und der sfdp-Algorithmus von Yifan [Hu06].

Als weiterer Ausblick könnte eine andere Fehlerfunktion für CoRe-GD definiert werden. Beispielsweise eine Fehlerfunktion, die mehrere Metriken gleichzeitig optimieren kann, wie DeepGD von Wang et al. [WYHS21]. Da eine Verbesserung durch eine Kombination nicht erzielt werden kann, könnten neue Aspekte auf interne Verbesserungen an CoRe-GD abzielen. Grötschla et al. [GMVW24] haben in CoRe-GD drei Convolution-Layer benutzt. Andere Convolution-Layer, die ebenfalls auf die Struktur eines Graphen abzielen, sind vielversprechend, wie z. B. das *Topological-Adaptive*-Convolution-Layer von Du et al. [DZW+17]. Eine weitere Verbesserungsmöglichkeit liegt in der Laufzeit von CoRe-GD und es könnten weitere Features definiert werden, die die strukturelle Wahrnehmung der Knoten innerhalb des Graphen weiter erhöhen.

6. Zusammenfassung und offene Fragen

In dieser Arbeit wurde analysiert, ob es machbar ist, das neuronale Netzwerk CoRe-GD von Grötschla et al. [GMVW24] mithilfe von klassischen Algorithmen $(sgd)^2$ von Zheng et al. [ZGP17], ForceAtlas2 Jacomy et al. [JVHB14], Spring von Fruchterman und Reingold [FR91] und die multidimensionale Skalierung von Kamada und Kawai [KK89] zu optimieren. Anfangs wurde diese Arbeit mit den Grundlagen der Graphenvisualisierung und der klassischen neuronalen Netzwerke eingeführt. Neuronale Netzwerke, die Graphen zeichnen können, wurden vorgestellt und es wurde ausführlich die Funktionsweise von CoRe-GD erläutert. In der Methodik haben wir unsere Untersuchungsmethode vorgestellt, um zu überprüfen, ob CoRe-GD durch eine Kombination mit klassischen Algorithmen optimierbar ist und, ob die klassischen Algorithmen sich durch CoRe-GD optimieren lassen. Im nächsten Kapitel haben wir unsere Untersuchungsergebnisse vorgestellt und diese in Kapitel 5 diskutiert.

Offenbar lässt sich die Zeichnung von CoRe-GD bezüglich der gegebenen Metriken durch einen klassischen Algorithmus nicht optimieren. CoRe-GD erzielt allein sehr gute Werte bezüglich der hier benutzten Metriken. Vereinzelt können bestimmte Metriken durch eine zusätzliche Kombination weiter verstärkt werden. Probleme ergeben sich im Hinblick auf Laufzeit. Anzumerken ist, dass es einerseits interessant ist, dass stressbasierte Algorithmen nahezu gleich sehr gute Werte wie ein neuronales Netzwerk erzielen, andererseits ist es interessant, dass es einem Lernalgorithmus möglich ist mit einer anderen Strategie und Implementierung nahezu ähnlich gute oder sogar bessere Werte als klassische stressbasierte Methoden zu erreichen.

Als offene Frage wäre es interessant zu wissen, aus welchem Grund der Spring-Embedder von den sehr guten Koordinatenbelegungen von CoRe-GD nicht profitiert. Als Methode in Python [VRD09] erhält der Spring *randomisierte* initiale Positionen und erreicht bessere Werte, als bei Verwendung der guten initialen Positionen von CoRe-GD. Es stellt sich außerdem die Frage, wieso die Kurven der Werte der Methoden bezüglich des Kreuzungswinkels bis 40–50 Knoten steigen und dann abfallen (siehe dazu 4.8).

Literaturverzeichnis

[Ant01] Martin Anthony: *Discrete Mathematics of Neural Networks*, Band 8 der Reihe *SIAM monographs on discrete mathematics and applications*. Society for Industrial and Applied Mathematics, 2001, 10.1137/1.9780898718539.

[ASN+13] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski und Alexander J. Smola: Distributed large-scale natural graph factorization. In: *22nd international conference on World Wide Web (WWW 2013)*, WWW, Seiten 37–48. International World Wide Web Conferences Steering Committee / Association for Computing Machinery, 2013, 10.1145/2488388.2488393.

[BMRW98] Therese Biedl, Joe Marks, Kathy Ryall und Sue Widesides: Graph Multidrawing: Finding Nice Drawings Without Defining Nice. In: *6th International Symposium on Graph Drawing (GD 1998)*, Band 1547 der Reihe *Lecture Notes in Computer Science*, Seiten 347–355. Springer, 1998, 10.1007/3-540-37623-2_26.

[BN03] Mikhail Belkin und Partha Niyogi: Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Computation*, 15(6):1373–1396, 2003, 10.1162/089976603321780317.

[BOS+18] Christian Bauckhage, César Ojeda, Jannis Schücker, Rafet Sifa und Stefan Wrobel: Informed Machine Learning Through Functional Composition. In: *Conference "Lernen, Wissen, Daten, Analysen"(LWDA 2018)*, Band 2191 der Reihe *CEUR Workshop Proceedings*, Seiten 33–37. CEUR-WS.org, 2018. `https://ceur-ws.org/Vol-2191/paper4.pdf`.

[CP96] Michael K. Coleman und D. Stott Parker: Aesthetics-based Graph Layout for Human Consumption. *Software: Practice and Experience*, 26(12):1415–1438, 1996, 10.1002/(SICI)1097-024X(199612)26:12%3C1415::AID-SPE69%3E3.0.CO;2-P.

[Del34] B. N. Delaunay: Sur la sphère vide. *Bull. Acad. Sci. URSS*, 1934(6):793–800, 1934. `https://www.mathnet.ru/eng/im4937`.

[DFPP90] H. De Fraysseix, J. Pach und R. Pollack: How to draw a planar graph on a grind. *Combinatorica*, 10(1):41–51, 1990, 10.1007/BF02122694.

[DH96] Ron Davidson und David Harel: Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics (TOG)*, 15(4):301–331, 1996, 10.1145/234535.234538.

[DJL+22] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio und Xavier Bresson: Benchmarking Graph Neural Networks. *Journal of Machine Learning Research*, 24(43):1–48, 2022. https://jmlr.org/papers/v24/22-0567.html.

[DZW+17] Jian Du, Shanghang Zhang, Guanhang Wu, José M. F. Moura und Soummya Kar: Topology adaptive graph convolutional networks. arXiv preprint, 2017. https://doi.org/10.48550/arXiv.1710.10370.

[Ead84] P. Eades: A heuristic for graph drawing. *Congressus Numerantium*, 42(11):149–160, 1984. https://www.cs.ubc.ca/~will/536E/papers/Eades1984.pdf.

[FCL+23] Jiarui Feng, Yixin Chen, Fuhai Li, Anindya Sarkar und Muhan Zhang: How Powerful are K-hop Message Passing Graph Neural Networks. In: *Advances in Neural Information Processing Systems 35 (NeuIPS 2023)*, Seiten 4776–4790. Curran Associates, Inc., 2023. https://papers.nips.cc/paper_files/paper/2022/hash/1ece70d2259b8e9510e2d4ca8754cecf-Abstract-Conference.html.

[FL19] Matthias Fey und Jan Eric Lenssen: Fast Graph Representation Learning with PyTorch Geometric. arXiv preprint, 2019. http://arxiv.org/abs/1903.02428, 7th International Conference on Learning Representations (Workshop Paper); https://github.com/pyg-team/pytorch_geometric; Zugegriffen: 18.07.2024 (Python-Bibliothek).

[FR91] Thomas M. J. Fruchterman und Edward M. Reingold: Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991, 10.1002/spe.4380211102.

[Gee24] GeekForGeeks: Minimum distance from a point to the line segment using Vectors. https://www.geeksforgeeks.org/minimum-distance-from-a-point-to-the-line-segment-using-\vectors/?ref=gcse_ind, 2024. Zugegriffen: 18.07.2024; Zuletzt bearbeitet: 01.03.2024.

[GHN13] Emden R. Gansner, Yifan Hu und Stephen North: A Maxent-Stress Model for Graph Layout. *IEEE Transactions on Visualization and Computer Graphics*, 19(6):927–940, 2013, 10.1109/TVCG.2012.299.

[GKN04] Emden Gansner, Yehuda Koren und Stephen North: Graph Drawing by Stress Majorization. In: *12th International Symposium on Graph Drawing (GD 2004)*, Band 3383 der Reihe *Lecture Notes in Computer Science*, Seiten 239–250. Springer, 2004, 10.1007/978-3-540-31843-9_25.

[GLA+22] Loann Giovannangeli, Frédéric Lalanne, David Auber, Romain Giot und Romain Bourqui: Deep Neural Network for DrawiNg Networks, (DNN)^2. In: *Lecture Notes in Computer Science 12868*, Seiten 375–390. Springer, 2022, 10.1007/978-3-030-92931-2_27.

[GMVW24] Florian Grötschla, Joël Mathys, Robert Veres und Roger Wattenhofer: CoRe-GD: A Hierarchical Framework for Scalable Graph Visualization with GNNs. arXiv preprint, 2024. 10.48550/ARXIV.2402.06706. 12th International Conference on Learning Representations (Poster Presentation).

[GSC00] Felix A. Gers, Jürgen Schmidhuber und Fred A. Cummins: Learning to Forget: Continual Prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000, 10.1162/089976600300015015.

[GSR+17] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals und George E. Dahl: Neural Message Passing for Quantum Chemistry. In: *34th International Conference on Machine Learning (ICML 2017)*, Band 70 der Reihe *Proceedings of Machine Learning Research*, Seiten 1263–1272. PMLR, 2017. `https://proceedings.mlr.press/v70/gilmer17a.html`.

[Ham20] William L. Hamilton: Graph Representation Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159, 2020, 10.1007/978-3-031-01588-5.

[HC19] Binxuan Huang und Kathleen M. Carley: Residual or Gate? Towards Deeper Graph Neural Networks for Inductive Graph Representation Learning. arXiv preprint, 2019. `http://arxiv.org/abs/1904.08035`.

[HMvdW+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke und Travis E. Oliphant: Array programming with NumPy. *Nature*, 585(7825):357–362, 2020, 10.1038/s41586-020-2649-2. (Python-Bibliotek).

[HS97] Sepp Hochreiter und Jürgen Schmidhuber: Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997, 10.1162/neco.1997.9.8.1735.

[HSC08] Aric Hagberg, Pieter Swart und Daniel Chult: Exploring Network Structure, Dynamics, and Function Using NetworkX. In: *7th Python in Science Conference*, SCIPY, Seiten 11–15, 2008. `https://www.researchgate.net/publication/236407765_Exploring_`

Network_Structure_Dynamics_and_Function_Using_NetworkX, Zugegriffen: 18.07.2024; (Python-Bibliothek).

[Hu06] Yifan Hu: Efficient and high quality force-directed graph drawing. *Mathematica Journal*, 10:31–71, 2006. http://yifanhu.net/PUB/graph_draw_small.pdf.

[Hun07] J. D. Hunter: Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007, 10.1109/MCSE.2007.55.

[HYL17] William L. Hamilton, Rex Ying und Jure Leskovec: Representation Learning on Graphs: Methods and Applications. *IEEE Data Eng. Bull.*, 40(3):52–74, 2017. http://sites.computer.org/debull/A17sept/p52.pdf.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren und Jian Sun: Deep Residual Learning for Image Recognition. In: *2016 EEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)*, Seiten 770–778. IEEE, 2016, 10.1109/CVPR.2016.90.

[JJ20] Yu Jin und Joseph F. JáJá: Network Summarization with Preserved Spectral Properties. arXiv preprint, 2020. http://arxiv.org/abs/1802.04447.

[Jol18] Alexia Jolicoeur-Martineau: The relativistic discriminator: a key element missing from standard GAN. arXiv preprint, 2018. http://arxiv.org/abs/1807.00734, 7th International Conference on Learning Representations (Poster Presentation).

[JVHB14] Mathieu Jacomy, Tommaso Venturini, Sebastien Heymann und Mathieu Bastian: ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software. *PLoS One*, 9(6), 2014, 10.1371/journal.pone.0098679.

[KB13] Mirza Klimenta und Ulrik Brandes: Graph Drawing by Classical Multidimensional Scaling: New Perspectives. In: Walter Didimo und Maurizio Patrignani (Herausgeber): *20th International Symposium on Graph Drawing (GD 2012)*, Band 7704 der Reihe *Lecture Notes in Computer Science*, Seiten 55–66. Springer, 2013, 10.1007/978-3-642-36763-2_6.

[KK89] Tomihisa Kamada und Satoru Kawai: An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989, 10.1016/0020-0190(89)90102-6.

[KM20] Oh-Hyun Kwon und Kwan-Liu Ma: A Deep Generative Model for Graph Layout. *IEEE Transactions on Visualisation & Computer Graphics*, 26(01):665–675, 2020, 10.1109/TVCG.2019.2934396.

[Kob14] S. G. Kobourov: *Handbook of Graphdrawing and Visuaization (Chapter 12: Force-directed drawing algorithms)*. Taylor & Francis Group; Chapman & Hall/CRC, 2014, 10.1201/b15385. Seiten 383–408; Editor Roberto Tamassia.

[Kru64] Joseph B. Kruskal: Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964, 10.1007/BF02289565.

[KW16] Thomas N. Kipf und Max Welling: Semi-Supervised Classification with Graph Convolutional Networks. arXiv eprint, 2016. `http://arxiv.org/abs/1609.02907`, 5th International Conference on Learning Representations (Poster Presentation).

[Mis17] Sanatan Mishra: Unsupervised Learning and Data Clustering. `https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a`, 2017. Zugegriffen: 29.07.2024; Zuletzt bearbeitet: 19.05.2017.

[Noa07] Andreas Noack: *Unified Quality Measures for Clusterings, Layouts, and Orderings of Graphs, and Their Application as Software Design Criteria*. PhD-Thesis, Brandenburgische Technische Universität Cottbus, 2007. `https://opus4.kobv.de/opus4-btu/frontdoor/index/index/docId/377`.

[ON15] Keiron O'shea und Ryan Nash: An introduction to convolutional neural networks. arXiv preprint, 2015. `http://arxiv.org/abs/1511.08458`.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai und Soumith Chintala: PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: *Advances in Neural Information Processing Systems (NeuIPS 2019)*. Curran Associates Inc., 2019. `https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html`.

[Pur97] Helen Purchase: Which aesthetic has the greatest effect on human understanding? In: Giuseppe DiBattista (Herausgeber): *5th International Symposium on Graph Drawing (GD 1997)*, Band 1353 der Reihe *Lecture Notes in Computer Science*, Seiten 248–261. Springer, 1997, 10.1007/3-540-63938-1_67.

[Ren14] Matthias Renz: Algorithmen der Computer Geometrie WS 2014/2015. `https://www.dbs.ifi.lmu.de/Lehre/GIS/WS1415/Skript/GIS_WS14_`

06.pdf, 2014. Zugegriffen: 22.07.2024; Ludwig-Maximilian-Universität; Skript zur Vorlesung Geo-Informationssysteme: Kapitel 6; von Matthias Renz (2014), Peter Kröger (2011) basierend auf dem Skript von Christian Böhm aus dem Sommersemester 2009.

[RGD+22] Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf und Dominique Beaini: Recipe for a General, Powerful, Scalable Graph Transformer. In: Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho und A. Oh (Herausgeber): *Advances in Neural Information Processing Systems 35 (NeuIPS 2022)*, NeuIPS, 2022. http://papers.nips.cc/paper_files/paper/2022/hash/5d4834a159f1547b267a05a4e2b7cf5e-Abstract-Conference.html.

[Ser] Serlo. https://de.serlo.org/mathe/2137/abstand-eines-punktes-zu-einer-geraden-berechnen-analytische-geometrie. Zugegriffen: 19.7.2024; (Analytische Geometrie).

[SGT+09] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner und Gabriele Monfardini: The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009, 10.1109/TNN.2008.2005605.

[She62] Roger N. Shepard: The analysis of proximities: Multidimensional scaling with an unknown distance function. I. *Psychometrika*, 27(2):125–140, 1962, 10.1007/BF02289630.

[Sme21] Bart M.N. Smets: Mathematics of Neural Networks (Lecture Notes Graduate Course). arXiv preprint, 2021. 10.48550/arXiv.2403.04807. Lecture Notes des Kurses 2MMA80 Mathematics of Neuronal Networks (Eindhoven University of Technology); von 2021 bis 2023.

[SOK12] Hartmut Noltemeier Sven Oliver Krumke: *Graphentheoretische Konzepte und Algorithmen*. Vieweg+Teubner Verlag Wiesbaden, 2012, 10.1007/978-3-8348-2264-2.

[STT81] Kozo Sugiyama, Shojiro Tagawa und Mitsuhiko Toda: Methods for Visual Understanding of Hierarchical System Structures. *IEEE Trans. Syst. Man Cybern.*, 11(2):109–125, 1981, 10.1109/TSMC.1981.4308636.

[SYT22] Yasunori Shiono, Toshihiro Yoshizumi und Kensei Tsuchida: Improvement of Fuzzy Graph Drawing Using Partition Tree. *J. Adv. Comput. Intell. Intell. Informatics*, 26(1):17–22, 2022, 10.20965/JACIII.2022.P0017.

[TR05] Martyn Taylor und Peter J. Rodgers: Applying graphical design techniques to graph visualisation. In: *9th International Conference on Information Visualisation (IV 2005)*, Seiten 651–656. IEEE, 2005, 10.1109/IV.2005.19.

[Tut63] W. T. Tutte: How to Draw a Graph. *Proceedings of the London Mathematical Society (3)*, s3-13(1):743–767, 1963, 10.1112/plms/s3-13.1.743.

[VCC+18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò und Yoshua Bengio: Graph Attention Networks. arXiv preprint, 2018. `http://arxiv.org/abs/1710.10903`, 6th International Conference on Learning Representations (Poster Presentation).

[VRD09] Guido Van Rossum und Fred L. Drake: *Python 3 Reference Manual*. CreateSpace, 2009. `https://dl.acm.org/doi/book/10.5555/1593511`.

[Wal01] C. Walshaw: A Multilevel Algorithm for Force-Directed Graph Drawing. In: *8th International Symposium on Graph Drawing (GD 2000)*, Band 1984 der Reihe *Lecture Notes in Computer Science*, Seiten 171–182. Springer, 2001, 10.1007/3-540-44541-2_17.

[WCPZ22] Lingfei Wu, Peng Cui, Jian Pei und Liang Zhao: *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer, 2022, 10.1007/978-981-16-6054-2.

[WJW+20] Yong Wang, Zhihua Jin, Qianwen Wang, Weiwei Cui, Tengfei Ma und Huamin Qu: DeepDrawing: A Deep Learning Approach to Graph Drawing. *IEEE Transactions on Visualization & Computer Graphics*, 26(01):676–686, 2020, 10.1109/TVCG.2019.2934798.

[WPCM02] Colin Ware, Helen Purchase, Linda Colpoys und Matthew McGill: Cognitive Measurements of Graph Aesthetics. *Information Visualization*, 1(2):103–110, 2002, 10.1057/palgrave.ivs.9500013.

[WYHS21] Xiaoqi Wang, Kevin Yen, Yifan Hu und Han-Wei Shen: DeepGD: A Deep Learning Framework for Graph Drawing Using GNN. *IEEE Computer Graphics and Applications*, 41(5):32–44, 2021, 10.1109/MCG.2021.3093908.

[WYHS23] Xiaoqi Wang, Kevin Yen, Yifan Hu und Han Wei Shen: SmartGD: A GAN-Based Graph Drawing Framework for Diverse Aesthetic Goals. *IEEE Transaction on Visualization and Computer Graphics*, 30:5666–5478, 2023, 10.1109/TVCG.2023.3306356.

[XHLJ19] Keyulu Xu, Weihua Hu, Jure Leskovec und Stefanie Jegelka: How Powerful are Graph Neural Networks? In: *7Th Internationsal Conference on Learning Representations (ICLR 2019)*, Band 162 der Reihe *Proceedings of Machine Learning Research*, Seiten 23341–23362. PMLR, 2019. `https://proceedings.mlr.press/v162/wang22am.html`.

[XZ19] Zhiang Xu und Pengzhou Zhang: An Improved Force-directed Algorithm Based on PageRank. In: *4th Advanced Information Technology, Electronic*

and Automation Control Conference (IAEAC 2019), Band 1, Seiten 2507–2510. IEEE, 2019, 10.1109/IAEAC47372.2019.8997830.

[YFK+14] Ömer Yaveroğlu, Sean Fitzhugh, Maciej Kurant, Athina Markopoulou, Carter Butts und Natasa Przulj: Ergm.graphlets: A Package for ERG Modeling Based on Graphlet Statistics. *Journal of Statistical Software*, 65(12), 2014, 10.18637/jss.v065.i12.

[ZCZ+20] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu und Maosong Sun: Graph Neural Networks: A Review of Methods and Applications. *AI Open*, 1:57–81, 2020, 10.1016/j.aiopen.2021.01.001.

[ZGP17] Jonathan X. Zheng, Dan F. M. Goodman und Samraat Pawar: Graph Drawing by Weighted Constraint Relaxation. arXiv preprint, 2017. `http://arxiv.org/abs/1710.04626`.

[ZJAS22] Lingxiao Zhao, Wei Jin, Leman Akoglu und Neil Shah: From Stars to Subgraphs: Uplifting Any GNN with Local Structure Awareness. arXiv preprint, 2022. `https://arxiv.org/abs/2110.03753`, 10th International Conference on Learning Representations (Poster Presentation).

[ZLWH24] Bohang Zhang, Shengjie Luo, Liwei Wang und Di He: Rethinking the Expressive Power of GNNs via Graph Biconnectivity. arXiv preprint, 2024. 10.48550/ARXIV.2301.09505. 11th International Conference on Learning Representations (Paper).

[ZWBW22] Johannes Zink, Julian Walter, Joachim Baumeister und Alexander Wolff: Layered drawing of undirected graphs with generalized port constraints. *Computational Geometry*, Seiten 105–106, 2022, 10.1016/j.comgeo.2022.101886.

A. Anhang

In diesem Abschnitt sind ausgewählte Graphen mit deren Zeichnungen aller Methoden mit den dazugehörenden Werten dargestellt. Die Abbildungen wurden mithilfe der Python-Bibliothek NetworkX [HSC08] erstellt. Wie schon in der Arbeit benutzt ist der Stress durch die Spalte “Stress‘", die Anzahl der Kantenkreuzungen durch die Spalte “Xing‘", die Standardabweichung bezüglich der Kantenlänge durch die Spalte “Edge‘", das Kantenverhältnis durch die Spalte “Ratio‘", Abstand eines Knoten von nicht-adjazenten Knoten durch die Spalte “DistNN‘", Abstand eines Knoten von nicht-inzidenten Kanten durch die Spalte “DistNE‘", der Kreuzungswinkel durch die Spalte “Xangle‘", die Winkelauflösung durch die Spalte “AngRes‘", die Abweichung des minimalen Winkels vom optimalen Winkel durch die Spalte “OptAng‘ünd die Laufzeit durch die Spalte “Time‘"gekennzeichnet. “Edge‘", “Ratio‘", “DistNN‘ünd “DistNE‘"wurden nach der maximalen Seite der Bounding-Box normalisiert. Die Winkelberechnungen sind “Gradmaße‘ünd die Laufzeit ist in Sekunden berechnet.

Die Abbildung A.1 zeigt die Zeichnungen des Graphen mit der ID 1 aus dem Testsatz von CoRe-GD und die Tabelle A.1 zeigt dessen Werte bezüglich der Metriken.

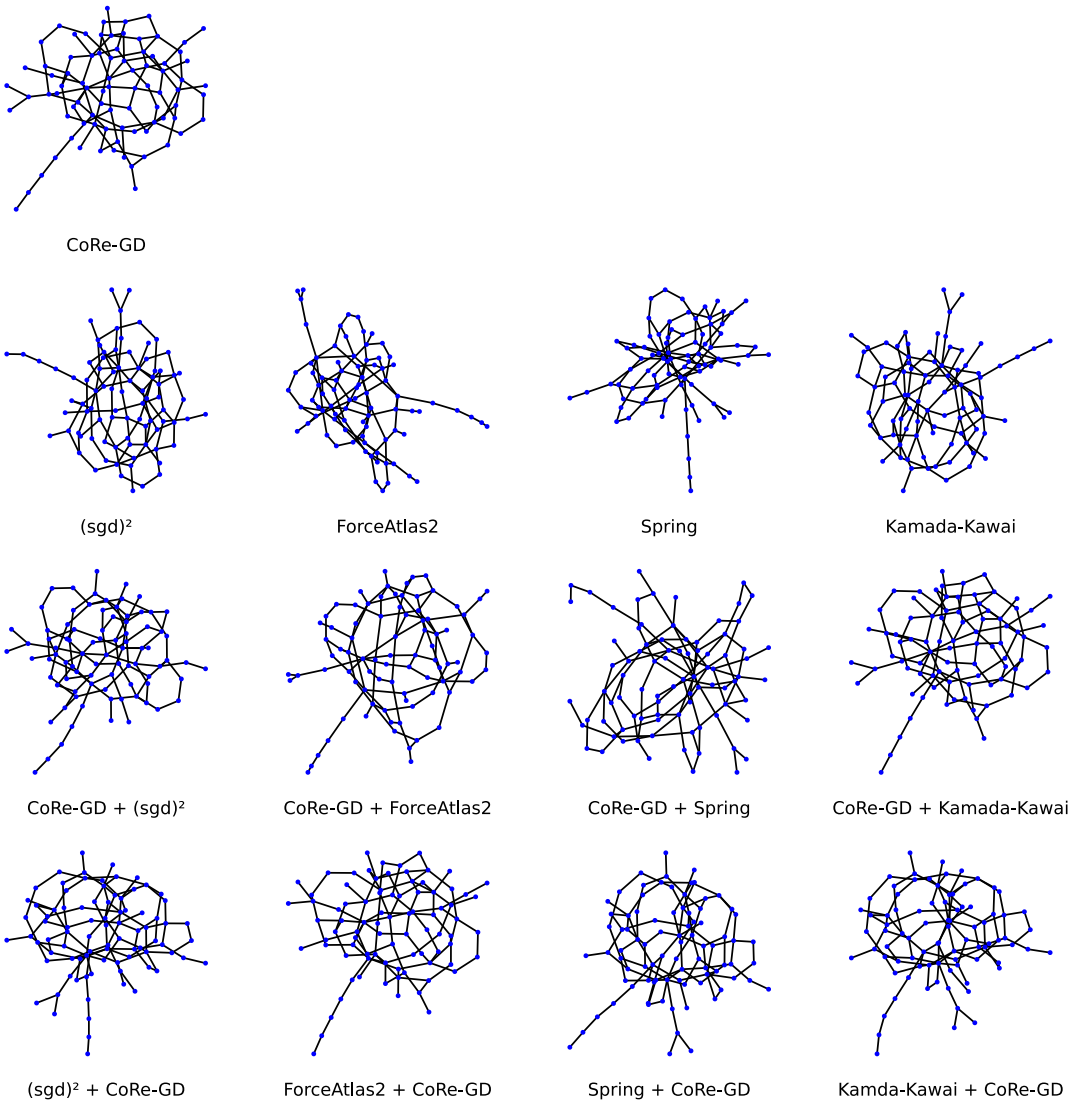


Abb. A.1.: Graph 1

Tab. A.1.: Werte der Metriken zu Graph 1

Methode	Stress↓	Xing↓	Edge↓	Ratio↑	DistNN↑	DistNE↑	XAngle↑	AngRes↑	OptAng↓	Time↓
CoRe	720,08	75	0,0262	0,0490	0,0315	0,0005	21,08	3,58	44,96	0,123
$(sgd)^2$	729,80	72	0,0208	0,0241	0,0169	0,0002	16,65	3,72	43,55	0,002
CoRe+$(sgd)^2$	738,45	70	0,0263	0,0444	0,0138	0,0009	15,53	0,74	49,04	0,115
$(sgd)^2$+CoRe	722,02	76	0,0230	0,0325	0,0184	0,0015	8,50	2,64	46,97	0,165
FA2	1149,59	77	0,0540	0,0335	0,0187	0,0002	4,30	2,74	47,53	0,019
CoRe+FA2	840,97	54	0,0443	0,0303	0,0255	0,0001	11,26	1,63	35,41	0,220
FA2+CoRe	723,91	67	0,0255	0,0395	0,0141	0,0001	13,22	2,99	46,63	0,234
Spring	1017,16	93	0,0297	0,0224	0,0125	0,0004	14,14	0,85	54,60	0,014
CoRe+Spring	992,29	90	0,0365	0,0538	0,0212	0,0006	5,79	1,68	47,26	0,133
Spring+Core	715,89	78	0,0256	0,0501	0,0065	0,0000	18,45	1,96	48,66	0,231
KK	704,49	65	0,0234	0,0527	0,0152	0,0003	10,11	2,87	49,16	0,087
CoRe+KK	717,74	66	0,0254	0,0514	0,0344	0,0015	18,77	2,29	45,73	0,180
KK+CoRe	716,78	79	0,0254	0,0243	0,0050	0,0001	4,66	2,63	49,47	0,234

Die Abbildung A.2 zeigt die Zeichnungen des Graphen mit der ID 4 aus dem Testsatz von CoRe-GD und die Tabelle A.2 zeigt dessen Werte bezüglich der Metriken.

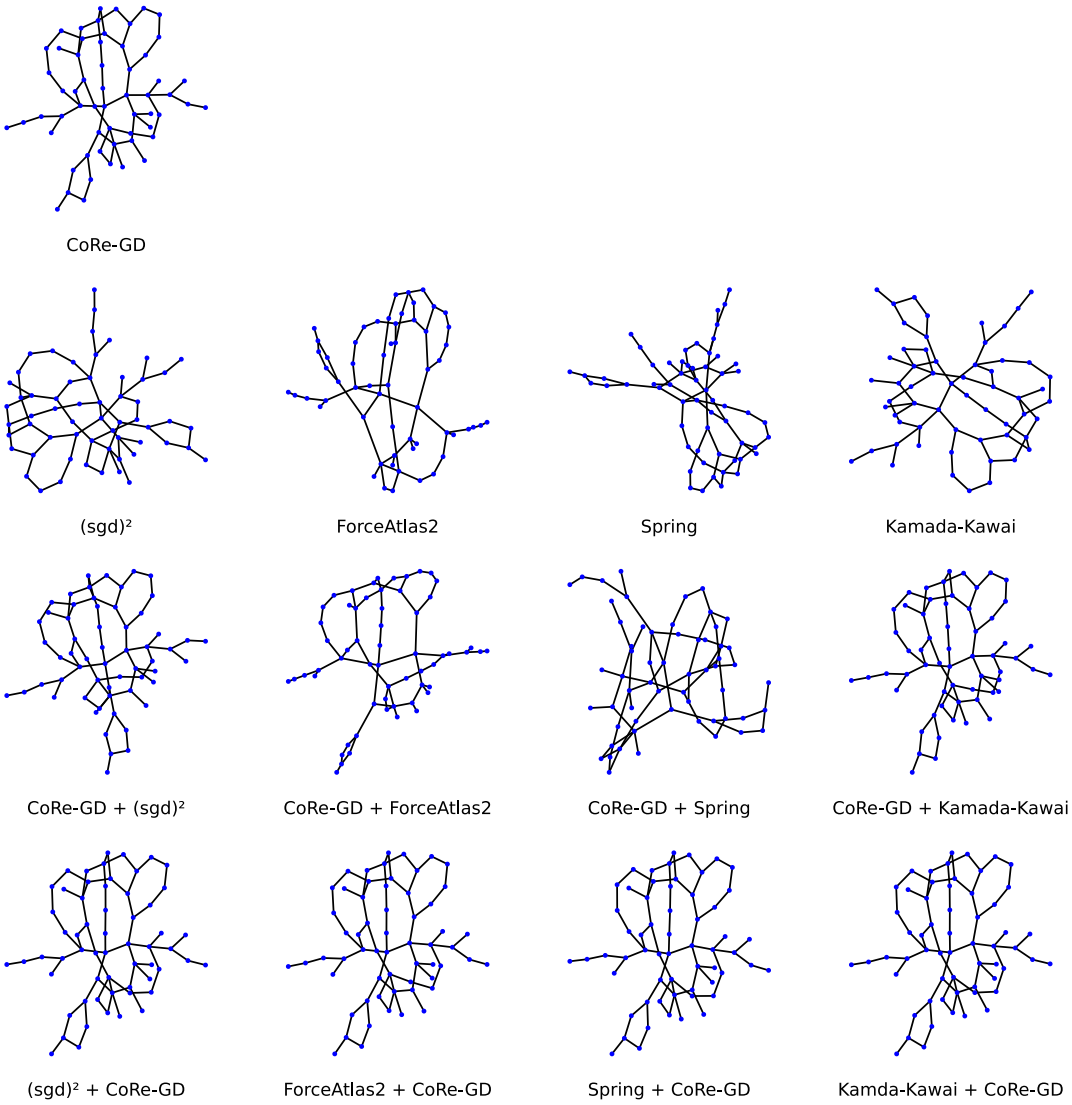


Abb. A.2.: Graph 4

Tab. A.2.: Werte der Metriken zu Graph 4

Methode	Stress↓	Xing↓	Edge↓	Ratio↑	DistNN↑	DistNE↑	XAngle↑	AngRes↑	OptAng↓	Time↓
CoRe	199,57	7	0,0139	0,0511	0,0307	0,0014	54,53	10,07	44,06	0,124
$(sgd)^2$	261,40	11	0,0190	0,0524	0,0403	0,0027	34,34	6,89	44,56	0,001
$CoRe(sgd)^2$	201,70	11	0,0151	0,0392	0,0254	0,0003	8,83	0,56	46,06	0,113
$(sgd)^2$+Core	199,74	8	0,0144	0,0481	0,0230	0,0063	44,69	12,93	44,34	0,081
FA2	374,01	6	0,0368	0,0231	0,0226	0,0013	52,68	3,06	37,30	0,011
CoRe+FA2	294,29	6	0,0395	0,0285	0,0250	0,0029	47,57	7,96	34,61	0,128
FA2+CoRe	199,50	7	0,0137	0,0484	0,0336	0,0006	49,90	13,47	44,05	0,167
Spring	398,92	17	0,0298	0,0478	0,0268	0,0013	35,06	0,99	48,86	0,008
CoRe+Spring	486,29	26	0,0323	0,0614	0,0190	0,0005	21,57	9,71	47,39	0,127
Spring+Core	199,74	8	0,0144	0,0482	0,0242	0,0072	45,14	13,14	44,34	0,132
KK	198,34	9	0,0151	0,0624	0,0329	0,0054	49,81	21,10	43,54	0,021
CoRe+KK	198,64	9	0,0131	0,0544	0,0290	0,0076	49,54	21,24	43,49	0,142
KK+CoRe	199,75	8	0,0144	0,0479	0,0232	0,0069	44,72	13,01	44,37	0,122

Die Abbildung A.3 zeigt die Zeichnungen des Sterngraphen mit 50 Kanten aus der Python-Bibliothek NetworkX [HSC08] und die Tabelle A.3 zeigt dessen Werte bezüglich der Metriken. Dieser und der folgende Graph wurden für CoRe-GD und die Kombinationen vorverarbeitet. Interessant ist, dass CoRe-GD den Stress der Sterngraphen nicht gut berechnen kann. Der Grund ist vermutlich, dass CoRe-GD auf den Rome-Graphen trainiert wurde und dort kein Graph mit einem hohen Knotengrad existiert hat.

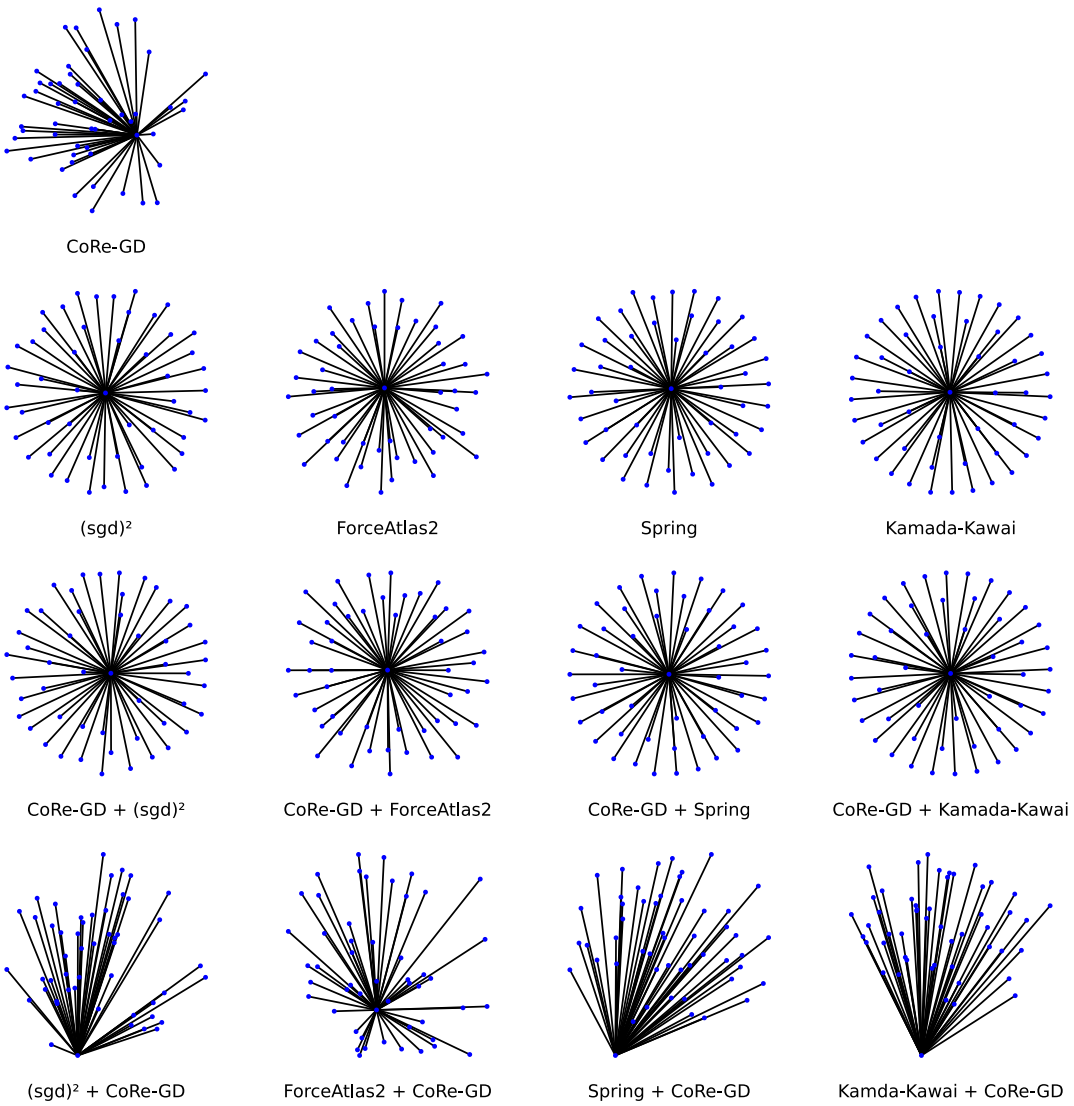


Abb. A.3.: Sterngraph

Tab. A.3.: Werte der Metriken zum Sterngraph

Methode	Stress↓	Xing↓	Edge↓	Ratio↑	DistNN↑	DistNE↑	XAngle↑	AngRes↑	OptAng↓	Time↓
CoRe	514,47	0	0,1563	0,0734	0,0173	0,0001	0	0,03	7,17	0,162
$(sgd)^2$	414,20	0	0,0970	0,1428	0,0788	0,0013	0	0,30	6,90	0,001
CoRe+$(sgd)^2$	414,23	0	0,0951	0,1362	0,0698	0,0016	0	0,30	6,90	0,070
$(sgd)^2$+Core	609,24	0	0,1826	0,1404	0,0101	0,0003	0	0,02	7,18	0,088
FA2	420,83	0	0,0787	0,2576	0,0714	0,0004	0	0,07	7,13	0,009
CoRe+FA2	418,44	0	0,0788	0,2715	0,0690	0,0000	0	0,01	7,19	0,099
FA2+CoRe	510,69	0	0,2064	0,0726	0,0198	0,0006	0	0,17	7,03	0,126
Spring	413,66	0	0,0913	0,2299	0,1018	0,0004	0	0,06	7,14	0,007
CoRe+Spring	413,55	0	0,0920	0,2221	0,1041	0,0007	0	0,12	7,08	0,096
Spring+Core	702,20	0	0,2021	0,1845	0,0239	0,0011	0	0,12	7,08	0,097
KK	413,23	0	0,0971	0,2280	0,1041	0,0001	0	0,03	7,17	0,038
CoRe+KK	413,24	0	0,0973	0,2260	0,1083	0,0001	0	0,01	7,19	0,118
KK+CoRe	740,61	0	0,1856	0,2981	0,0059	0,0008	0	0,09	7,11	0,110

Die Abbildung A.4 zeigt die Zeichnungen eines Gittergraphen. Dieser Graph wurde in der Python-Bibliothek NetworkX [HSC08] bereitgestellt und die Tabelle A.4 zeigt dessen Werte bezüglich der Metriken.

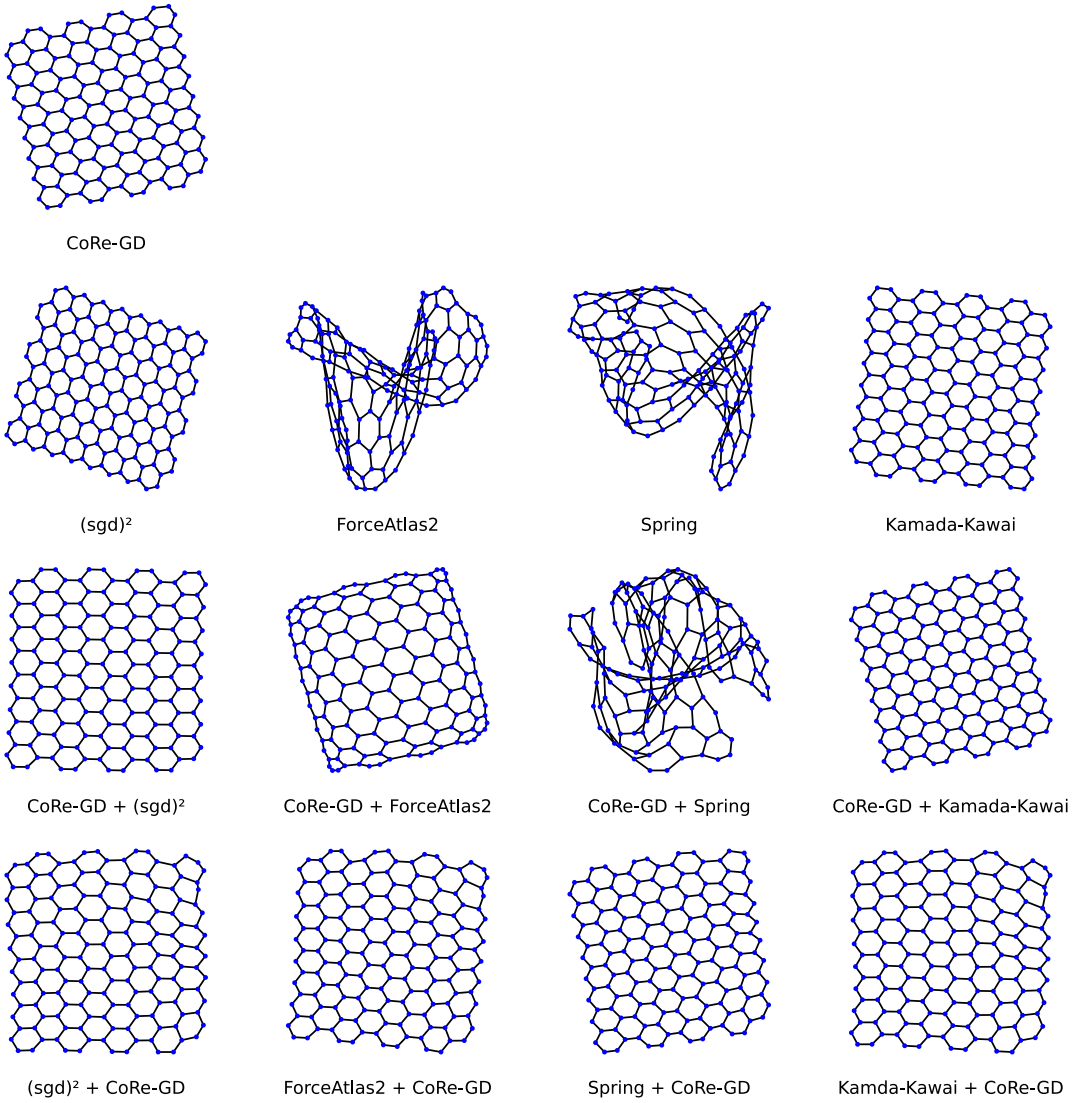


Abb. A.4.: Gittergraphen im Wabenmuster

Tab. A.4.: Werte der Metriken zum Gittergraphen mit Wabenmuster

Methode	Stress↓	Xing↓	Edge↓	Ratio↑	DistNN↑	DistNE↑	XAngle↑	AngRes↑	OptAng↓	Time↓
CoRe	137,37	0	0,0027	0,0498	0,0932	0,0498	0	104,31	16,01	0,250
$(sgd)^2$	129,12	0	0,0014	0,0536	0,0917	0,0536	0	109,72	13,91	0,009
CoRe+$(sgd)^2$	129,11	0	0,0016	0,0626	0,1080	0,0626	0	108,47	14,00	0,297
$(sgd)^2$+Core	153,02	0	0,0048	0,0354	0,0960	0,0354	0	81,48	17,50	0,284
FA2	3170,71	68	0,0204	0,0163	0,0123	0,0001	2,62	5,72	43,41	0,040
CoRe+FA2	398,22	0	0,0141	0,0242	0,0433	0,0242	0	58,88	17,00	0,287
FA2+CoRe	148,14	0	0,0043	0,0403	0,0930	0,0403	0	88,24	16,35	0,371
Spring	3356,78	78	0,0204	0,0202	0,0109	0,0001	17,37	1,08	47,02	0,044
CoRe+Spring	4051,72	96	0,0216	0,0213	0,0128	0,0000	2,79	0,95	54,17	0,321
Spring+Core	142,00	0	0,0034	0,0431	0,0915	0,0431	0	102,87	16,58	0,304
KK	128,80	0	0,0012	0,0599	0,1028	0,0599	0	111,01	13,72	0,186
CoRe+KK	128,80	0	0,0011	0,0567	0,0973	0,0567	0	111,01	13,72	0,282
KK+CoRe	157,84	0	0,0052	0,0337	0,0907	0,0337	0	76,03	17,28	0,450