Simultane Pfadzeichnungen auf dem Gitter

Timur Sultanov
Universität Trier, 21. Mai 2025

Motivation

- **Simultaneous Embedding** = gleichzeitige Darstellung mehrerer Graphen mit gleicher Knotenmenge
- Besonders relevant bei nur geringfügigen Unterschieden

Motivation

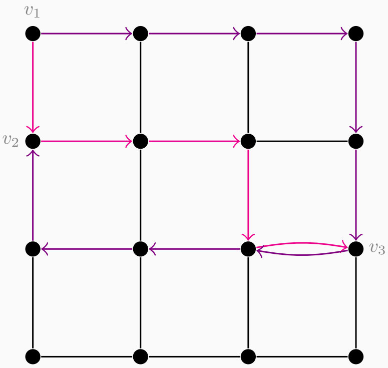
- **Simultaneous Embedding** = gleichzeitige Darstellung mehrerer Graphen mit gleicher Knotenmenge
- Besonders relevant bei nur geringfügigen Unterschieden
- **Ziel:**
  - Gemeinsame und differenzierende Strukturen visuell erfassbar machen
  - Kompakte, konfliktfreie und gut lesbare Einbettung
- **Herausforderungen:**
  - Begrenzter Platz auf dem Gitter (Flächenminimierung)
  - Minimierung der Biegungen zur Verbesserung der Lesbarkeit
  - Vermeidung von Kreuzungen und Selbstüberschneidungen

Ziel der Arbeit

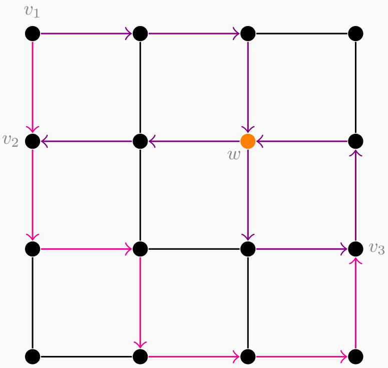
- Einbettung von zwei Pfaden auf einem Gitter
- Möglichst kompakt
- Pfade dürfen sich überlappen, aber nicht selbst schneiden

Ziel der Arbeit

- Einbettung von zwei Pfaden auf einem Gitter
- Möglichst kompakt
- Pfade dürfen sich überlappen, aber nicht selbst schneiden

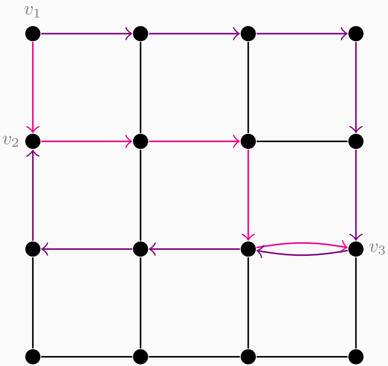


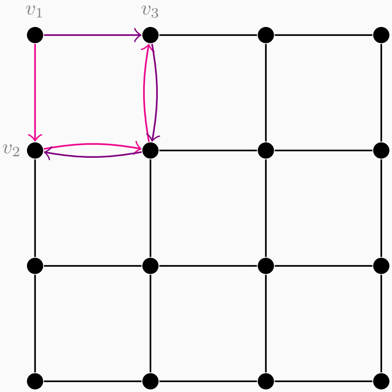
(a) Erlaubte Zeichnung



(b) Verbotene Zeichnung

Möglichst kompakt/Flächenminimierend





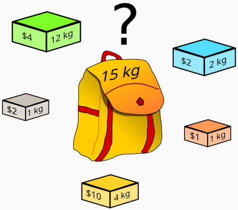
Lineare & ganzzahlige Optimierung

Lineare Optimierung(LP): Optimierung einer linearen Zielfunktion unter linearen Nebenbedingungen
Ganzzahlige lineare Programmierung (ILP): Variablen dürfen **nur ganzzahlige Werte** annehmen

Lineare Optimierung(LP): Optimierung einer linearen Zielfunktion unter linearen Nebenbedingungen
Ganzzahlige lineare Programmierung (ILP): Variablen dürfen **nur ganzzahlige Werte** annehmen

Beispiele für ILP-Anwendungen:

- Travelling Salesman Problem
- quadratische Zuordnungsproblem
- Knapsack Problem



Implementierung in Python mit gurobipy

- Große Lesbarkeit und Wartbarkeit des Codes
- Nahtlose Einbettung von Gurobi via gurobipy API
- Obwohl Python langsamer als C++ ist, hat das keine negativen Auswirkungen, da die eigentliche Lösung im Gurobi-Core (C++) abläuft

Implementierung in Python mit gurobipy

- Große Lesbarkeit und Wartbarkeit des Codes
- Nahtlose Einbettung von Gurobi via gurobipy API
- Obwohl Python langsamer als C++ ist, hat das keine negativen Auswirkungen, da die eigentliche Lösung im Gurobi-Core (C++) abläuft

Gurobi als leistungsstarker ILP-Solver:

- Kein eigener Algorithmus notwendig
- nutzt *hybride Verfahren* zur Lösung von ILPs

Implementierung in Python mit gurobipy

- Große Lesbarkeit und Wartbarkeit des Codes
- Nahtlose Einbettung von Gurobi via gurobipy API
- Obwohl Python langsamer als C++ ist, hat das keine negativen Auswirkungen, da die eigentliche Lösung im Gurobi-Core (C++) abläuft

Gurobi als leistungsstarker ILP-Solver:

- Kein eigener Algorithmus notwendig
- nutzt *hybride Verfahren* zur Lösung von ILPs

In einer Vergleichstudie(2023) wurden fünf kommerzielle und freie ILP-Solver gegenübergestellt: Gurobi zählt zu den **schnellsten und zuverlässigsten** Solvern

Variablen des ILP-Modells

Zuweisung von Knoten zu Gitterpunkten:

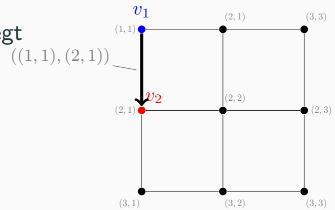
$$\sigma(v, p) = \begin{cases} 1, & \text{falls } v \text{ auf Gitterpunkt } p \text{ liegt} \\ 0, & \text{sonst} \end{cases}$$

Belegung von Gitterkanten durch Pfadkanten:

$$\mu(e, p, q) = \begin{cases} 1, & \text{falls } e \text{ die Gitterkante } (p, q) \text{ nutzt} \\ 0, & \text{sonst} \end{cases}$$

Zuweisung von Knoten zu Gitterpunkten:

$$\sigma(v, p) = \begin{cases} 1, & \text{falls } v \text{ auf Gitterpunkt } p \text{ liegt} \\ 0, & \text{sonst} \end{cases}$$

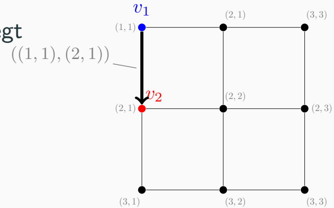


Belegung von Gitterkanten durch Pfadkanten:

$$\mu(e, p, q) = \begin{cases} 1, & \text{falls } e \text{ die Gitterkante } (p, q) \text{ nutzt} \\ 0, & \text{sonst} \end{cases}$$

Zuweisung von Knoten zu Gitterpunkten:

$$\sigma(v, p) = \begin{cases} 1, & \text{falls } v \text{ auf Gitterpunkt } p \text{ liegt} \\ 0, & \text{sonst} \end{cases}$$



Belegung von Gitterkanten durch Pfadkanten:

$$\mu(e, p, q) = \begin{cases} 1, & \text{falls } e \text{ die Gitterkante } (p, q) \text{ nutzt} \\ 0, & \text{sonst} \end{cases}$$

Zielfunktion: $Z = \sum_{e \in E} \sum_{(p,q) \in F} \mu(e, p, q)$

Nebenbedingungen

(1) Jeder Knoten muss auf genau einem Gitterpunkt liegen

(2) Auf einem Gitterpunkt darf höchstens ein Knoten liegen

Nebenbedingungen

(1) Jeder Knoten muss auf genau einem Gitterpunkt liegen

(2) Auf einem Gitterpunkt darf höchstens ein Knoten liegen

(3) Kanten müssen kontinuierlich gezeichnet werden

(4) Kanten dürfen keine Gitterpunkte durchlaufen, an dem ein Knoten liegt, der nicht zu der jeweiligen Kante inzident ist

(5) Es darf keine Überschneidungen innerhalb eines Pfades geben

Nebenbedingungen

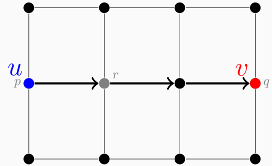
(1) Jeder Knoten muss auf genau einem Gitterpunkt liegen:

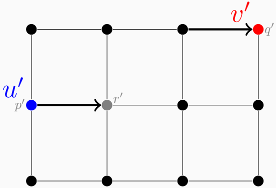
$$\forall v \in V \sum_{p \in P} \sigma(v, p) = 1$$

(2) Auf einem Gitterpunkt darf höchstens ein Knoten liegen:

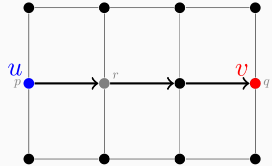
$$\forall p \in P \sum_{v \in V} \sigma(v, p) \leq 1$$

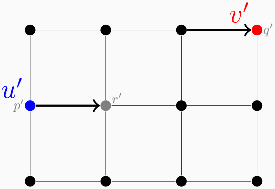
(3) Kanten müssen kontinuierlich gezeichnet werden:





(3) Kanten müssen kontinuierlich gezeichnet werden:





$$\forall e = (u, v) \in E, u, v \in V, \forall p \in D:$$

$$\sum_{(p,q)\in F} \mu(e, p, q) - \sum_{(q,p)\in F} \mu(e, q, p) = \sigma(u, p) - \sigma(v, p)$$

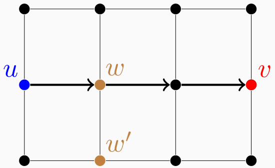
Nebenbedingungen

(4) Kanten dürfen keine Gitterpunkte durchlaufen, an dem ein Knoten liegt, der nicht zu der jeweiligen Kante inzident ist:
Summe aller eingehenden und ausgehenden genutzten Gitterkanten am Gitterpunkt p von einer Kante e:

$$flow\_sum = \sum_{(p,q) \in F} \mu(e,p,q) + \sum_{(q,p) \in F} \mu(e,q,p)$$

(4) Kanten dürfen keine Gitterpunkte durchlaufen, an dem ein Knoten liegt, der nicht zu der jeweiligen Kante inzident ist:
Summe aller eingehenden und ausgehenden genutzten Gitterkanten am Gitterpunkt p von einer Kante e:

$$\mathit{flow\_sum} = \sum_{(p,q)\in F} \mu(e,p,q) + \sum_{(q,p)\in F} \mu(e,q,p)$$



$$\mathit{flow\_sum} \leq 2 \cdot (1 - \sigma(w,p))$$

Nebenbedingungen

(5) Es darf keine Überschneidungen innerhalb eines Pfades geben:

Sei nun W die Menge an Pfaden. Für alle $P_i = (V_i, E_i) \in W$ und für jeden Gitterpunkt $p \in D$ definieren wir die aggregierte Anzahl an korrespondierenden Gitterkanten an p, die von den Kanten $e_i \in E_i$ genutzt werden:

$$aggregated\_flow = \sum_{e \in p_i} \Big(\sum_{(p,q) \in F} \mu(e, p, q) + \sum_{(q,p) \in F} \mu(e, q, p) \Big)$$

Nebenbedingungen

(5) Es darf keine Überschneidungen innerhalb eines Pfades geben:

Sei nun W die Menge an Pfaden. Für alle $P_i = (V_i, E_i) \in W$ und für jeden Gitterpunkt $p \in D$ definieren wir die aggregierte Anzahl an korrespondierenden Gitterkanten an p, die von den Kanten $e_i \in E_i$ genutzt werden:

$$aggregated\_flow = \sum_{e \in p_i} \Big(\sum_{(p,q) \in F} \mu(e, p, q) + \sum_{(q,p) \in F} \mu(e, q, p) \Big)$$

Nun gibt es zwei Szenarien für jeden Pfad:

- Gitterpunkt p wird nicht passiert $\rightarrow$ $aggregated\_flow = 0$
- Gitterpunkt p wird passiert $\rightarrow$ $aggregated\_flow = 1 \vee 2$

$$aggregated\_flow \leq 2$$

Testinstanzen und Gitterwahl

1. Testreihe:

- 10 zufällige Testfälle für $5 - 11$ Knoten
- Wahl des Gitters: $(\lceil n/2 \rceil + 1 \times \lceil n/2 \rceil + 1)$

Testinstanzen und Gitterwahl

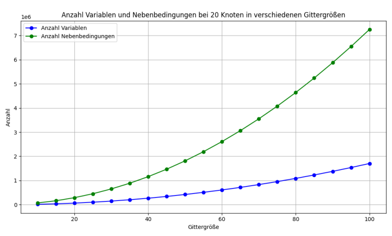
1. Testreihe:

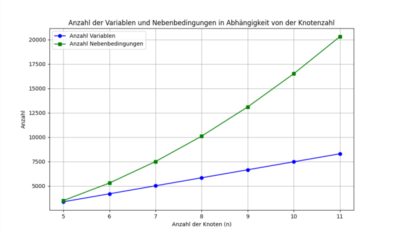
- 10 zufällige Testfälle für $5 - 11$ Knoten
- Wahl des Gitters: $(\lceil n/2 \rceil + 1 \times \lceil n/2 \rceil + 1)$

2. Testreihe:

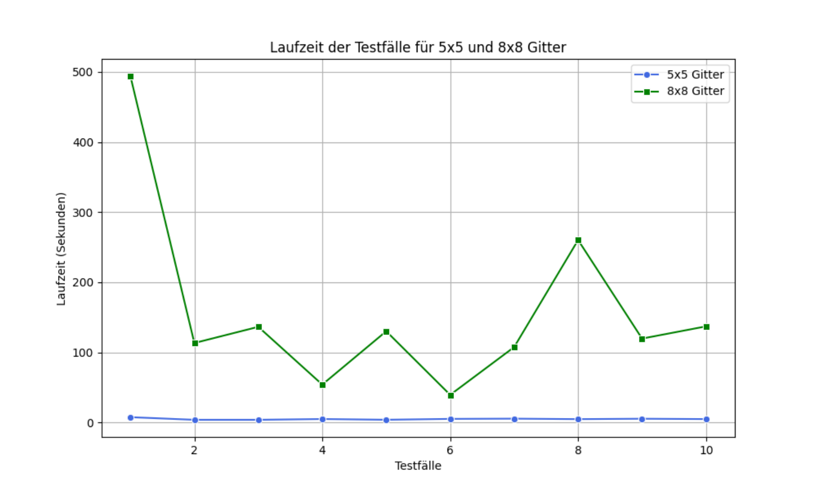
- Anzahl gleicher adjazenter Knoten für Pfade mit 10 Knoten
- jeweils 10 Testfälle

Modellkomplexität

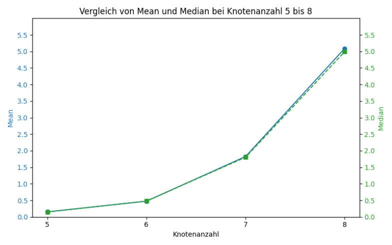


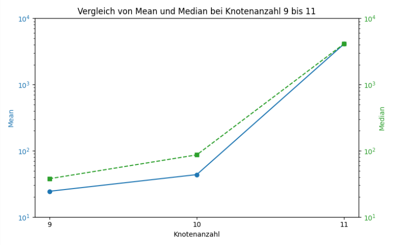


Laufzeiten bei fester Knotenzahl: 5×5 vs. 8×8 Gitter



Laufzeiten bei wachsender Knotenzahl (5–11)



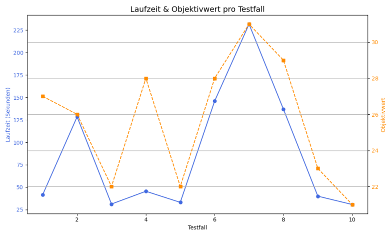


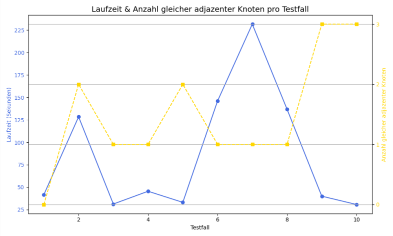
Ursachen für Laufzeitschwankungen bei 10 Knoten

Mögliche Ursachen:

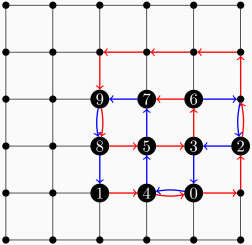
- Wert der Zielfunktion
- Anzahl gleicher adjazenter Knoten
- Komplexität der resultierenden Einbettung

Ursachen für Laufzeitschwankungen bei 10 Knoten

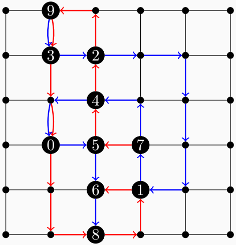




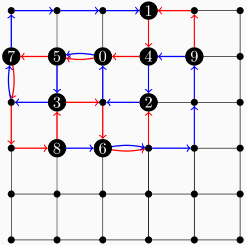
Graphische Darstellung der Testfälle



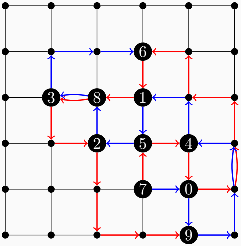
Testfall 3: 31,22s



Testfall 4: 45,44s



Testfall 6: 145,99s



Testfall 7: 231,68s

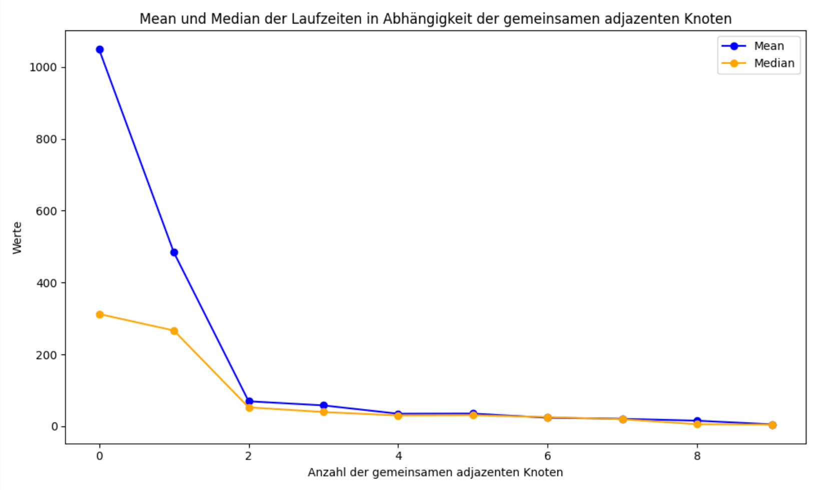
Laufzeiten und Pfadlängenverteilung bei 10 Knoten

Testfall	Laufzeit (s)	Kanten (L:A)
1	41,64	2:4, 3:1, 6:1
2	128,42	2:4, 3:1, 7:1
3	31,22	2:2, 6:1
4	45,44	2:5, 3:1, 6:1
5	33,06	2:3, 5:1
6	146,00	2:5, 3:2, 4:2
7	231,68	2:2, 3:1, 4:2, 6:1
8	136,91	2:2, 3:1, 4:3
9	39,98	2:4, 3:2
10	30,67	2:3, 3:2

Beobachtungen:

- Höhere Laufzeiten bei komplexerer Struktur
- Viele Kanten mit Länge > 1 erhöhen die Rechenzeit
- Kürzere, kompakte Pfade führen zu schnelleren Lösungen

Laufzeitverhalten bei steigender Anzahl identischer adjazenter Knoten



Fazit: Bewertung des Modells

- Für kleine Instanzen ($\leq$ 8 Knoten) sehr zuverlässig und schnell
- Rechenzeiten stabil und gut prognostizierbar
- Ab 12 Knoten: drastischer Anstieg der Laufzeit (bis Stunden/Tage)
- Gründe:
  - Große Gitter $\Rightarrow$ viele Variablen & Nebenbedingungen
  - Komplexere Pfade mit Überlappungen und langen Kanten
- Modell ist konzeptionell korrekt, aber skaliert nicht gut

Optimierungsmöglichkeiten

Ziel: Verbesserung der Skalierbarkeit und Laufzeit

- **Heuristische Verfahren:**
  - Greedy, Simulated Annealing, etc.
  - Schnell gute Lösungen für große Instanzen
  - Können als obere Schranke dienen
- **Vorverarbeitung der Eingabe:**
  - Graphvereinfachung, Symmetrieerkennung
  - Reduktion unnötiger Redundanz im Modell