INSTITUT FÜR INFORMATIK
Algorithmische Bioinformatik

Universitätsstr. 1 D–40225 Düsseldorf

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Cluster Editing mit ganzzahliger linearer Programmierung: eine Open-Source-Lösung

Martin Klümpen

Bachelorarbeit

Beginn der Arbeit:	08. Mai 2018
Abgabe der Arbeit:	07. August 2018
Gutachter:	Prof. Dr. Gunnar W. Klau
	Prof. Dr. Michael Leuschel

Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 07. August 2018

Martin Klümpen

Zusammenfassung

Integer Linear Programming (deutsch: ganzzahliges lineares Programm) (ILP) ist eine Methode zur Bestimmung von optimalen Lösungen für Optimierungsprobleme. Diese wird in dem Programm Yoshiko genutzt um das Weighted Cluster Editing(WCE) Problem zu lösen. Dabei wird zunächst aus dem zugrunde liegenden vollständigen Graphen ein ganzzahliges lineares Programm erstellt und von einem Solver berechnet. Der bis jetzt verwendete Solver ist CPLEX, eine kommerzialisierte Bibliothek entwickelt von IBM. Dessen Funktion soll von der Open Source Bibliothek SYMPHONY, einem Projekt der COIN-OR Foundation, übernommen werden. Weighted Cluster Editing ist ein schweres Problem und kann nicht effizient gelöst werden. Es stellt sich die Frage wie gut die Algorithmen in der Praxis arbeiten oder bis zu welchem Limit sie noch praktisch effizient sind. Dafür wurden eine Reihe von Tests auf einem Hochleistungs-Rechencluster ausgeführt. Diese decken neben dem normalen WCE auch einige Spezialisierungen von dem Problem ab. Gemessen wurden die Laufzeiten der Programmaufrufe unter verschiedenen Voraussetzungen. Mit einem Programm konnten die daraus gewonnenen Daten aufbereitet und grafisch veranschaulicht werden. Aus den Ergebnissen lässt sich deuten, dass die CPLEX Bibliothek erkennbar schneller arbeitet als SYMPHONY. Trotzdem ist die Open-Source-Variante in einem gewissen Rahmen ein guter Ersatz. Weil die Laufzeit aber auch von der Instanz-Art, WCE-Art und den Parameterwerten abhängt, muss dieser Rahmen für jedes Problem separat bestimmt werden.

Inhaltsverzeichnis

1 Einleitung

Yoshiko ist ein Programm, das Instanzen des Weighted Cluster Editing Problemes löst. Dazu verwendet es entweder eine Heuristik oder einen exakten Algorithmus mit einem ganzzahligen linearen Programm (ILP), welches von dem ILP-Solver CPLEX gelöst wird. Für die Arbeit mit von Graphen wird die LEMON Bibliothek verwendet. Das Programm ist in C++ implementiert und wurde von Gunnar Klau, Emanuel Laude und Philipp Sphor [5] entwickelt. Die Funktion von CPLEX wird in dieser Arbeit durch COIN-OR SYMPHONY ersetzt. Das COIN-OR (Computational Infrastructure for Operations Research) Projekt wurde im Jahr 2000 mit dem Zweck gegründet, mathematische Programme open-Source anzubieten. Dies geschah ganz im Sinne des wissenschaftlichen Fortschritts, da jeder in der Lage war diese Programme weiterzuentwickeln, zu optimieren oder sogar ganz neue Algorithmen zu entwerfen [3]. Aus dieser Initiative ging auch die SYMPHONY Bibliothek hervor. Mit ihr ist es möglich ein ILP darzustellen und durch zahlreiche Optionen den Lösungsprozess zu steuern. SYMPHONY besitzt keinen eigenen Solver, sondern benutzt einen von mehreren zur Auswahl stehenden Solvern. In diesem Fall wird der CLP-Solver verwendet (COIN-OR LP-Solver).

Für den Vergleich der beiden Yoshikoversionen, mit CPLEX oder SYMPHONY, werden eine Reihe von Tests auf dem Hochleistungs-Rechencluster des Zentrums für Informations- und Medientechnologie (ZIM) der Heinrich Heine Universität (HHU) Düsseldorf durchgeführt. Es werden zwei verschiedene Arten von Zufallsgraphen erstellt und dem Programm übergeben. Diese Tests decken neben normalen Instanzen des Weighted-Cluster-Editing auch Spezialisierungen des Problems ab.

Es wird vermutet, dass CPLEX schneller ist als SYMPHONY. In dieser Arbeit stellt sich die Frage, welche Einschränkungen durch die Nutzung von SYMPHONY entstehen und in welchem Rahmen sie noch vertretbar sind. Zum Abschluss soll eine Aussage darüber getroffen werden wo die aktuellen Grenzen liegen und, ob die Open-Source-Variante gegenüber dem kommerzialisierten Solver eine Alternative darstellt.

2 Methoden

2.1 Graphendefinitionen

1. **Definition: Ungerichteter Graph**
   Sei V eine endliche Menge und $E \subseteq \binom{V}{2}$, dann ist $G(V, E)$ ein ungerichteter Graph mit Knotenmenge V und Kantenmenge E.

2. **Definition: Inzidente Kante**n
   Sei $G(V, E)$ ein ungerichteter Graph. Für zwei Knoten $u, v \in V$, ist eine Kante $e \in E$ inzident genau dann, wenn $e = \{u, v\}$.

3. **Definition: Kantengewichte**
   Sei $G(V, E)$ ein ungerichteter Graph und $f : E \mapsto \mathbb{Z}$ die Kostenfunktion von G. Das Gewicht einer Kante $e \in E$ ist $f(e)$.

4. **Definition: Knotengrad**
   Sei $G(V, E)$ ein ungerichteter Graph, $n = |V|$ und $f : E \mapsto [0, .., n-1]$ eine Funktion mit $f(v) = |\{\{u, v\} \in E \mid u \in V\}|$. Der Knotengrad von einem Knoten $v \in V$ ist $f(v)$. Damit ist der Knotengrad die Anzahl aller inzidenten Kanten.

5. **Definition: Vollständiger ungerichteter Graph**
   Sei $G(V, E)$ ein ungerichteter Graph. G ist vollständig genau dann, wenn für jedes $u, v \in V$, $u \neq v$, $gilt : \{u, v\} \in E$.

6. **Definition: Gilbert-Graph**(vgl. Gilbert [4])
   Sei $n \in \mathbb{N}$ und $p \in \mathbb{R}$, $0 \leq p \leq 1$. Das Tupel $G(n, p)$ beschreibt einen Graphen $G(V, E)$ mit $|V| = n$ Knoten und für jede Kante $\{u, v\}$ wird mit Wahrscheinlichkeit p entschieden, ob sie in E liegt oder nicht.

2.2 Ganzzahlige lineare Programmierung

2.2.1 Definitionen

1. **Definition: Optimierungsproblem**
   Sei E eine endliche Menge von Lösungen, I eine Menge mit Teilmengen von E mit gültigen Lösungen und $c : I \mapsto \mathbb{R}$ eine Kostenfunktion. Gesucht ist eine gültige Lösung $i \in I$, die c minimiert oder maximiert:

   $$OPT(Instanz) = min\{c(i)|i \in I\}$$

   oder

   $$OPT(Instanz) = max\{c(i)|i \in I\}$$

2.2.2 Darstellungen

Allgemein kann man ein ILP mit n Variablen und m Nebenbedingungen wie folgt darstellen:

Gegeben sei eine lineare Funktion, die Objektfunktion

$$\sum_{i=1}^{n} a_i x_i \quad , a_i \in \mathbb{Z},$$

welche unter den gegebenen Nebenbedingungen

$$\begin{aligned} \sum_{i=1}^{n} b_{1i} x_i \leq S_1 \quad & , b_{1i} \in \mathbb{Z} \\ \vdots & \\ \sum_{i=1}^{n} b_{mi} x_i \leq S_m \quad & , b_{mi} \in \mathbb{Z} \end{aligned}$$

durch die Belegung der Variablen x_1 bis x_n minimiert oder maximiert wird. Es dürfen beliebig viele Nebenbedingungen aufgestellt werden, die jeweils ihre eigene Liste von Faktoren b_1 bis b_n und eine obere Schranke S für die jeweilige Summe besitzen. Fasst man all diese Listen von Faktoren, Variablen und Schranken in Vektoren und Matrizen zusammen, erhält man eine noch kompaktere Darstellung des Problems.

Mathematische Darstellung:

$$min\{c^T x | Ax \leq b, x \in \mathbb{Z} \geq 0\}$$

mit
$c \in \mathbb{R}^n$, ein Vektor mit den Faktoren der Objektfunktion
x, ein n-dimensionaler Vektor mit den Variablen
$b \in \mathbb{R}^m$, ein Vektor mit den Schranken der Nebenbedingungen
$A \in \mathbb{R}^{n \times m}$, eine Matrix mit Faktoren, wobei die Zeile a_{i1} bis a_{in} und b_i zusammen eine Nebenbedingung bilden[7].

2.3 Weighted Cluster Editing

2.3.1 Definitionen

1. **Definition: Clique**
   Sei $G(V, E)$ ein ungerichteter Graph und $A \subseteq V$. A ist eine Clique genau dann, wenn gilt: $\forall u, v \in V, u \neq v \iff \{u, v\} \in E$.

2. **Definition: Cliquengraph**
   Sei $G(V, E)$ ein ungerichteter Graph. G ist eine Cliquengraph genau dann, wenn er eine Vereinigung von disjunkten Cliquen ist.

3. **Definition: Konflikttriple** (vgl. Laude [6])
   Sei $G(V, E)$ ein ungerichteter Graph und $A = \{u, v, w\} \subseteq V$. A ist ein Konflikttriple genau dann, wenn zwischen uvw genau zwei der drei möglichen Kanten existieren.

4. **Satz: G ist ein Cliquengraph $\iff$ G enthält keine Konflikttriple** (vgl. Böcker, Briesemeister, Bui & Truss[1])
   Sei $G(V, E)$ ein ungerichteter Graph. G ist ein Cliquengraph genau dann, wenn $\forall A \subseteq \binom{V}{3}$ gilt: A ist kein Konflikttriple.

5. **Definition: Weighted-Cluster-Editing (WCE)** (vgl. Böcker, Briesemeister & Klau [2])
   Gegeben: Ein ungerichteter Graph $G(V, E)$ und Kostenfunktion c.
   Gesucht: Eine Menge von Kantenmodifikationen mit minimalen Kosten so, dass der modifizierte Graph G' ein Cliquengraph ist. Wobei die Kosten die folgende Bedeutung haben:
   $c(e)_{e \in E} > 0$: Kante e kann für Kosten $c(e)$ aus dem Originalgraph entfernt werden.
   $c(e)_{e \in E} < 0$: Kante e kann für Kosten $-c(e)$ dem Originalgraph hinzugefügt werden.

6. **Definition: K-Weighted-Cluster-Editing (K-WCE)**
   Gegeben: Ein ungerichteter Graph $G(V, E)$, eine Kostenfunktion c und ein Parameter K.
   Gesucht: Eine Menge von Kantenmodifikationen mit minimalen Kosten so, dass der modifizierte Graph G' ein Cliquengraph mit genau K Cliquen ist.

7. **Definition: MinMax-Weighted-Cluster-Editing (MM-WCE)**
   Gegeben: Ein ungerichteter Graph $G(V, E)$, eine Kostenfunktion c und zwei Parameter min und max.
   Gesucht: Eine Menge von Kantenmodifikationen mit minimalen Kosten so, dass der modifizierte Graph G' ein Cliquengraph ist, in dem jedes Cluster mindestens min und maximal max Knoten enthält.

2.3.2 Weighted-Cluster-Editing als ganzzahliges lineares Programm

Im Folgenden wird ein allgemeines ILP für Weighted-Cluster-Editing nach dem Ansatz von Laude[6] formuliert.

Zunächst sei $G(V, E)$ der Originalgraph, $G'(V', E')$ der Lösungsgraph, c die Kostenfunktion und X eine Menge von Variablen mit $x_{uv} = 0$ wenn die Kante $\{u, v\} \in E'$ und $x_{uv} = 1$, wenn die Kante $\{u, v\} \in E'$. Es ergeben sich vier mögliche Szenarien für jede Kante:

$\{u,v\} \in E \wedge \{u,v\} \in E' \implies$ Modifikationskosten: 0
$\{u,v\} \in E \wedge \{u,v\} \notin E' \implies$ Modifikationskosten: $c(\{u,v\})$
$\{u,v\} \notin E \wedge \{u,v\} \in E' \implies$ Modifikationskosten: $-c(\{u,v\})$
$\{u,v\} \notin E \wedge \{u,v\} \notin E' \implies$ Modifikationskosten: 0

Die Summe der Modifikationskosten soll minimiert werden. Daraus ergibt sich die Objektfunktion

$$\sum_{e \in E} c(e) \quad - \sum_{u,v \in V} x_{uv} c(\{u,v\})$$

Außerdem wird durch die Nebenbedingungen sichergestellt, dass der Lösungsgraph ein Cliquengraph ist. Dazu wird der aus Satz 2.3 verwendet, nach dem ein Cliquengraph vorhanden ist, sobald die Lösung keine Konflikttriple enthält. Daher werden für jedes Knotentriple folgende drei Nebenbedingungen erstellt:

$$\begin{aligned} x_{uv} + x_{uw} - x_{vw} &\leq 1 \\ x_{uv} - x_{uw} + x_{vw} &\leq 1 \\ -x_{uv} + x_{uw} + x_{vw} &\leq 1 \\ \forall \{u,v,w\} &\in \binom{V}{3} \end{aligned}$$

Angenommen es existieren genau zwei der drei Kanten, dann ist es nicht möglich, dass alle drei Ungleichungen erfüllt sind. Für jede andere Belegung der Kanten gelten diese Bedingungen. Das ILP zu einer WCE Instanz besteht aus $n = \binom{|V|}{2}$ Variablen und $m = 3\binom{|V|}{3}$ Nebenbedingungen.

2.3.3 Spezialisierung von Weighted-Cluster-Editing

1. **K-Clustering:** Beim K-Clustering handelt es sich um eine Spezialisierung des normalen WCE Problems. Für bestimmte Instanzen möchte man in der Lage sein die Form der Lösung vorzugeben, indem man angibt wie viele Cluster der Lösungsgraph enthalten soll. Um dieses Ergebnis zu erhalten, werden in den Nebenbedingungen des ILP zusätzliche Einschränkungen getroffen. Dafür simuliert man K weitere Knoten, die mit jedem anderen Knoten verbunden werden, aber nicht untereinander. Sie werden in das ILP eingebaut ohne in dem Original- oder Lösungsgraphen vorhanden zu sein. Auf diese Weise kann man Einschränkungen der Lösung vorgeben, ohne die Instanz direkt zu verändern. Das K darf nicht beliebig gewählt werden, nur bei $1 \leq K \leq |V|$ besitzt eine Instanz eine gültige Lösung.

   Für eine K-WCE Instanz wird zusätzlich zu der Menge X eine Menge Y mit $K \cdot |V|$ Variablen definiert. Dabei ist $y_{ui} = 0$, wenn Knoten u nicht zu Cluster i gehört, und $y_{ui} = 1$, wenn Knoten u zu Cluster i gehört. Wenn zwei Knoten $u, v \in V$ mit einer Kante verbunden sind, dann liegen sie in demselben Cluster: $x_{uv} = 1 \implies \forall i : y_{ui} + y_{vi} = 2 \vee y_{ui} + y_{vi} = 0$. Wenn jedoch zwei Knoten $u, v \in V$ keine Kante besitzen, liegen sie in zwei verschiedenen Clustern: $x_{uv} = 0 \implies \forall i : y_{ui} + y_{vi} \leq 1$. Daraus

ergibt sich, ähnlich zu den Nebenbedingungen aus 2.3.2, für alle Knotenpaare $u, v \in V$ und alle Cluster i:

$$x_{uv} + y_{ui} - y_{vi} \leq 1$$
$$x_{uv} - y_{ui} + y_{vi} \leq 1$$
$$-x_{uv} + y_{ui} + y_{vi} \leq 1$$

Dennoch kann es vorkommen, dass entweder ein Cluster leer bleibt, dann hätte der Lösungsgraph weniger als K Cluster, oder, dass ein Knoten alleine steht, ohne einem Cluster anzugehören, dann besäße der Lösungsgraph mehr als K Cluster. Daher werden weiter Nebenbedingungen benötigt, die jedem Cluster mindestens einen Knoten zuordnen:

$$\sum_{u \in V} y_{ui} \geq 1 \quad , 1 \leq i \leq K$$

Und Nebenbedingungen, die jedem Knoten genau einem Cluster zuordnen:

$$\sum_{i=1}^{K} y_{ui} = 1 \quad , \forall u \in V$$

Zu den n Variablen und m Nebenbedingungen aus dem allgemeinen WCE kommen weitere $K \cdot n$ Variablen und $3K\binom{|V|}{2} + K + |V|$ Nebenbedingungen. Es ergibt sich:

$$n_K = \binom{|V|}{2} + Kn$$
$$m_K = 3(K\binom{|V|}{2} + \binom{|V|}{3}) + K + |V|$$

2. **MinMax-Clustering**: Ähnlich wie das K-Clustering ist das MinMax-Clustering eine Spezialisierung des allgemeinen Clustering Problems. Hierbei soll der Lösungsgraph nur Cliquen einer bestimmten Größe enthalten. Auch dafür werden weiter Nebenbedingungen benötigt, die den Lösungsgraphen einschränken. Wenn eine Clique n Knoten besitzt, hat jeder Knoten genau $n-1$ inzidente Kanten. Also muss für jedes $u \in V$ die Summe der inzidenten Kanten beschränkt werden. Für jeden Knoten ergeben sich daher, in einer MM-WCE Instanz, die folgenden zwei Nebenbedingungen:

$$\sum_{v \in V} x_{uv} \geq min - 1 \quad ; u \in V$$
$$\sum_{v \in V} x_{uv} \leq max - 1 \quad ; u \in V$$

So entstehen keine neuen Variablen und $2|V|$ weitere Nebenbedingungen, so dass insgesamt $m_{MM} = 3\binom{|V|}{3} + 2|V|$. Die min und max Werte dürfen nicht beliebig gewählt werden. Zum Beispiel besitzt eine Instanz mit 8 Knoten und $min = 5$, $max = 7$ Werten keine gültige Lösung, da nicht jedes Cluster die Bedingungen erfüllen kann.

2.4 Yoshiko

2.4.1 Ein- und Ausgabeformate

Das Yoshiko Programm erhält als Eingabe immer einen vollständigen Graphen mit einer Gewichtsfunktion. Ein Graph der Größe n und Kostenfunktion c wird dabei wie folgt dargestellt:

Zeile 1:	Anzahl der Knoten
Zeile 2 bis n+1:	Namen der n Knoten
n+2 bis 2n+1:	Obere Dreiecksmatrix mit Eintrag $a_{ij} = c(\{i, j\})$ mit $i > j$

Des Weiteren kann man Yoshiko mit einer Vielzahl von Parametern starten, die zum Beispiel die maximale Laufzeit, Anzahl der genutzten Prozessorkerne oder das Eingabe- und Ausgabeformat bestimmen. Dabei kann man auch festlegen, ob eine WCE Instanz gelöst werden soll oder eine Spezialisierung wie K-WCE.
Die Ausgabe erfolgt in Listen, wobei jede Liste die Namen der Knoten eines Clusters enthält. Außerdem kann man unter Verwendung von Cytoscape[8] und dem Yoshiko Wrapper[9] auch eine grafische Ausgabe erhalten.

2.4.2 Algorithmen

Zunächst wird versucht Teile der Instanz mit Hilfe von sechs Reduktionsregeln zusammenzufassen. Diese Regeln prüfen, ob die Instanz unter Beibehalt aller optimalen Lösungen verkleinerbar ist. Zum Beispiel können so Kanten identifiziert werden, die entweder sicher oder nie Teil einer Lösung sind. Dadurch könnten einige Knoten bei entsprechender Änderung der Kantengewichte zu einem Knoten zusammengefasst werden. So entstehen reduzierte Instanzen, die schneller zu lösen sind, aber die gleiche optimale Lösung enthalten.

Anschließend kann der Benutzer wählen, ob die Lösung durch eine Heuristik oder durch ein ILP bestimmt wird.

Heuristik: Eine Heuristik ist ein effizienter Algorithmus, der keine Garantie über die Qualität der Lösung gibt. Trotzdem überzeugen Heuristiken meistens mit schnellen Laufzeiten und guten Lösungen, wenn keine optimale Lösung benötigt wird. Das Yoshiko Programm verwendet den Ansatz einer Heuristik von Böcker et al (2011[2]) verändert von Laude[6].

ILP: Aus dem Originalgraphen wird nach dem Verfahren aus 2.3.2 ein ILP aufgestellt. Dieses ILP wird dann an einen ILP-Solver weitergereicht und gelöst. Auf diese Art erhält man immer eine optimale Lösung, sofern die Instanz eine gültige Lösung besitzt. Bei Benutzung der ILP-Methode kann der Nutzer außerdem auch K-Clustering und MinMax-Clustering betreiben. Dies ist mit der Heuristik nicht möglich.

3 Symphony

3.1 Weighted Cluster Editing in SYMPHONY

In CPLEX ist man in der Lage eine Nebenbedingung für ein ILP mit einer simplen Schreibweise darzustellen:

$$a_1 \cdot x_1 + a_3 \cdot x_3 - a_6 \cdot x_6 \leq 1$$

Die Faktoren der Variablen, die in dieser Zeile nicht explizit genannt werden, haben automatisch den Wert 0.

In SYMPHONY dagegen werden die Nebenbedingungen in der mathematischen Matrixschreibweise angegeben. Eine Nebenbedingung ist als 5-Tupel definiert:

$$(num, Col, Fac, lb, ub)$$

$num \in [1..n]$,	die Anzahl der definierten Spalten in dieser Zeile
$Col \subseteq [1..n], \lvert Col\rvert = n$,	eine Liste mit den Indizes der definierten Spalten in dieser Zeile
$Fac \subset \mathbb{R}, \lvert Fac\rvert = n$,	eine Liste der Einträge die in den definierten Spalten stehen
$lb \in \mathbb{Z}$,	der Eintrag in dem Vektor der die unteren Schranken enthält
$ub \in \mathbb{Z}$,	der Eintrag in dem Vektor der die oberen Schranken enthält

Alle nicht definierten Spalten in dieser Zeile werden als 0 interpretiert. Zum Beispiel definieren die drei Tupel $(3, (1, 3, 4), (1, 1, -1), -\infty, 1)$, $(3, (1, 3, 4), (1, -1, 1), -\infty, 1)$ und $(3, (1, 3, 4), (-1, 1, 1), -\infty, 1)$ die folgende Matrix:

$$A = \begin{pmatrix} 1 & 0 & 1 & -1 \\ 1 & 0 & -1 & 1 \\ -1 & 0 & 1 & 1 \end{pmatrix} B_{lb} = \begin{pmatrix} -\infty \\ -\infty \\ -\infty \end{pmatrix} B_{ub} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Multipliziert man die Matrix A zeilenweise mit dem Variablenvektor $x = \binom{x_1}{x_2}{x_3}{x_4}$ erhält man die drei Nebenbedingungen:

$$\begin{aligned} 1 \cdot x_1 + 0 \cdot x_2 + 1 \cdot x_3 - 1 \cdot x_4 &\leq 1 \\ 1 \cdot x_1 + 0 \cdot x_2 - 1 \cdot x_3 + 1 \cdot x_4 &\leq 1 \\ -1 \cdot x_1 + 0 \cdot x_2 + 1 \cdot x_3 + 1 \cdot x_4 &\leq 1 \end{aligned}$$

Diese drei Ungleichungen definieren außerdem ein Knotentriple. Jede WCE Instanz lässt sich so in die Matrixschreibweise und damit in ein ILP übersetzen:
Sei $G(V, E)$ der Graph und $costs$ die Kostenfunktion einer WCE Instanz, $n = \lvert\binom{V}{2}\rvert$ und $f : E \mapsto [1, .., n]$ eine bijektive Funktion, die jeder Kante eindeutig einer Zahl zuordnet. Die folgenden Tupel definieren die Matrix:

$$\begin{gathered} (3, (f(\{u, v\}), f(\{u, w\}), f(\{w, v\})), (1, 1, -1), -\infty, 1) \\ (3, (f(\{u, v\}), f(\{u, w\}), f(\{w, v\})), (1, -1, 1), -\infty, 1) \\ (3, (f(\{u, v\}), f(\{u, w\}), f(\{w, v\})), (-1, 1, 1), -\infty, 1) \\ \forall \{u, v, w\} \in \binom{V}{3} \end{gathered}$$

Nach 2.3.2 setzt sich die Objektfunktion zusammen aus der Summe der Kanten die in dem Originalgraph vorhanden sind minus der Summe der Kanten aus dem Lösungsgraph. Diese Funktion muss durch die Multiplikation $c^T x$ ausgedrückt werden. Hierbei ist x der Variablenvektor und c der Objektvektor, der die Faktoren der Objektfunktion enthält. Für die negative Summe wird jeder Eintrag c_i aus c definiert als $c_i = -costs(f^{-1}(i))$. Jedes x_i wird so der entsprechenden Kante und somit den Kosten zugeordnet. Es fehlt die positive Summe. Sei x_{n+1} eine neue Variable die x hinzugefügt wird. Diese Variable hat in allen Nebenbedingungen den Faktor 0 und ist immer gleich 1. Der entsprechende Eintrag c_{n+1} aus c entspricht der Summe aller Kanten mit positiven Kosten. Der Vektor mit den Faktoren der Objektfunktion:

$$c = \begin{pmatrix} -costs(f^{-1}(1)) \\ -costs(f^{-1}(2)) \\ \vdots \\ -costs(f^{-1}(n)) \\ \sum_{costs(\{u,v\})>0} costs(\{u,v\}) \end{pmatrix}$$

Sobald die gesamte Instanz in dieser Form durch ein ILP repräsentiert ist, kann dieser Ausdruck an den Solver übergeben werden. Nachdem das ILP gelöst wurde, wird aus den Variablen die jetzt eindeutig mit 1 oder 0 belegt sind ein Graph erzeugt:

$$\forall 1 \leq j \leq n : x_j = 1 \iff f^{-1}(j) \in E'$$

3.2 K-Clustering

Für eine K-WCE Instanz werden die gleichen Vektoren und Tupel angegeben wie bei einer WCE Instanz. Der Variablenvektor wird um die Variablen x_{n+2} bis $x_{n+K\cdot|V|+1}$ erweitert. Die entsprechenden Faktoren aus dem Objektvektor werden auf 0 gesetzt. Weiterhin sei $g : V \times [1,..,K] \mapsto [n+2,..,n+K\cdot|V|+1]$ eine bijektive Funktion, die jedem Paar aus einem Knoten und einem Cluster eindeutig eine Zahl zuweist. Daraus ergeben sich die folgenden Tupel:

$$(3, (f(\{u,v\}), g((u,i)), g((v,i))), (1,1,-1), -\infty, 1)$$
$$(3, (f(\{u,v\}), g((u,i)), g((v,i))), (1,-1,1), -\infty, 1)$$
$$(3, (f(\{u,v\}), g((u,i)), g((v,i))), (-1,1,1), -\infty, 1)$$
$$\forall \{u,v,w\} \in \binom{V}{2}, 1 \leq i \leq K$$

Damit ist sichergestellt, dass je zwei Knoten entweder demselben Cluster angehören oder verschiedenen. Das reicht aber noch nicht um in der Lösung genau K Cluster zu erhalten. Zusätzlich darf kein Cluster leer sein

$$\forall 1 \leq i \leq K : (|V|, (g((v_1,i)), g((v_2,i)), .., g((v_{|V|},i)), (1_1, 1_2, .., 1_{|V|}), 1, |V|)$$

und jeder Knoten muss genau einem Cluster angehören

$$\forall v \in V : (K, (g((v,1)), g((v,2)), .., g((v,K))), (1_1, 1_2, .., 1_K), 1, 1)$$

Nach diesen Veränderungen ist Yoshiko in der Lage mit SYMPHONY jede gültige K-WCE Instanz zu lösen. Jede gültige Lösung enthält dabei genau K disjunkte Cluster.

3.3 Minimale und Maximale Clustergröße

Auch hier müssen weitere Einschränkungen in dem ILP vorgenommen werden. Dabei wird jeder Knoten betrachtet und folgende Schranken erstellt:

$$(|V|-1, (f(\{v,u_1\}), f(\{v,u_2\}), .., f(\{v,u_{|V|-1}\}), (1_1, 1_2, .., 1_{|V|-1}), min-1, max-1)$$
$$\forall v \in V, \quad u_1, u_2, .., u_{|V|-1} \in V \setminus \{v\}$$

Jeder Knoten in einem Cluster der Größe n hat genau $n-1$ Nachbarn und somit $n-1$ inzidente Kanten. Daher müssen hier die min und max Werte minus eins betrachtet werden. Yoshiko ist jetzt in der Lage MM-WCE Instanzen mit SYMPHONY zu lösen. Diese Funktion ermöglicht es, in einem gegeben Graphen Cluster einer bestimmten Größe zu finden. Auch in der Kombination mit K-Clustering funktioniert dieser Algorithmus. So können auch genau K Cluster der Größe min bis max gefunden werden.

4 Tests

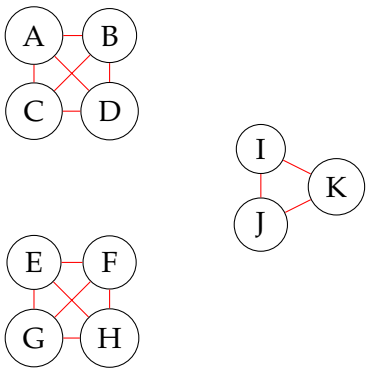
4.1 Testziele

Yoshiko ist jetzt in der Lage eine WCE Instanz entweder mit CPLEX oder SYMPHONY zu lösen. Idealerweise soll deren Funktion komplett von der Open Source Varianten übernommen werden. Dafür muss jedoch entschieden werden, ob sie überhaupt eine gute Alternative darstellt. Die aktuelle Annahme ist, dass CPLEX einen deutlichen Zeitvorteil hat. Es stellt sich die Frage, welche Voraussetzungen erfüllt sein müssen, damit man SYMPHONY als gute Alternative zu CPLEX bezeichnen kann. In den Tests wird folgender Wert betrachtet: Die Zeit die zum Lösen von Instanzen benötigt wird, im Vergleich zu der Größe der Instanz. Am Ende kann dann eine Aussage der Form "SYMPHONY ist bis zu einer Instanzgröße von n eine erfolgreiche Alternative zum Lösen von WCE/K-WCE/MM-WCE Problemen." getroffen werden.

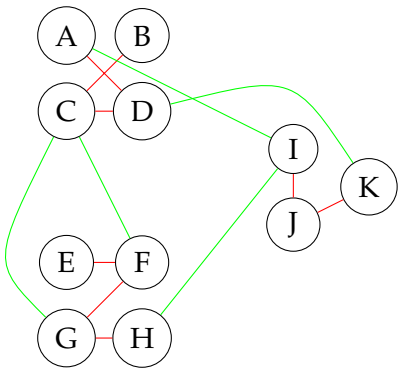
4.2 Erstellen von Zufallsgraphen

4.2.1 Gute Zufallsinstanzen

Für gute Zufallsinstanzen werden zunächst Cliquengraphen erstellt. Kanten, die existieren, sind mit hohen Kosten belegt. Kanten, die nicht existieren, erhalten hohe negativen Kosten. Dieser Graph wird durch zufälliges Hinzufügen und Herausnehmen von Kanten verändert. Entfernte Kanten erhalten niedrige negative Kosten und hinzugefügte erhalten niedrige Kosten. Je nach dem mit welcher Wahrscheinlichkeit dies gemacht wird, haben die Graphen am Ende immer noch eine erkennbare Struktur. Außerdem sollten durch die Kosten eher die alten Cliquen wiederhergestellt werden. Der ursprüngliche Graph ist keine optimale Lösung. Durch diese Methode wird aber trotzdem eine relativ leicht lösbare Instanz für das Programm erzeugt.



(a) Cliquengraph



(b) Guter Zufallsgraph

Abbildung 1: Erzeugen von guten Instanzen

4.2.2 Schlechte Zufallsinstanzen

Schlechte Zufallsinstanzen sind Graphen, die keinerlei Struktur aufweisen. Hier werden daher Gilbert-Graphen $G(n, p)$ mit $p = \frac{1}{2}$ verwendet. Zusätzlich werden die Kosten der Kanten zufällig gewählt. So erhält man einen gleichverteilten Graphen der im Durchschnitt keine Strukturen aufweist.

4.3 Tests

Für die Tests wurde ein Bash-Skript geschrieben und auf dem Hochleistungs-Rechencluster des Zentrum für Informations- und Medientechnologie (ZIM) an der HHU ausgeführt. Die zwei Arten von Zufallsinstanzen werden als Eingabe für das Programm verwendet und die Laufzeit mit dem Bash "time"-Befehl gemessen. Die resultierenden Daten enthalten für jeden Aufruf sowohl die Anzahl der Knoten einer Instanz, als auch die gemessene Zeit in Minuten und Sekunden. Diese Daten werden mit einem Python Programm unter Verwendung der Matplotlib Bibliothek aufbereitet und in einem Graphen veranschaulicht.
Jede generierte Instanz wird von beiden Yoshiko-Versionen einmal berechnet. Es werden mehrere Instanzen derselben Größe getestet und der Durchschnitt der Laufzeiten wird berechnet. So verzerren einzelne Messungen nicht das Gesamtbild. Die maximale Rechenzeit wird für jeden Test auf 300 Sekunden gesetzt. Des Weiteren wurden die Reduktionsregeln für die Tests ausgeschaltet. Eine reduzierte Instanz wird deutlich schneller berechnet als eine nicht reduzierte. Da nicht jeder Graph reduziert werden kann, würde es so zu vermehrten Ausreißern in den Daten kommen. Es können keine Werte angeben werden, welche Instanzgrößen sich genau auf welche Knotenanzahl reduzieren lassen. Daher ein Beispiel: Die Testeingabe "fusarium.txt" aus dem "data/tests" Ordner in den Yoshikodateien (siehe Anhang A) kann von 223 Knoten auf 63 Knoten reduziert werden. Außerdem erhält man so ein klares Bild von der Laufzeit der Bibliotheken im Vergleich zu der Anzahl der Nebenbedingungen, die für eine Instanz aufgestellt werden. Diese Anzahl ist auch eine indirekte Darstellung der Größe des aufgestellten ILP.
So werden zunächst gute und schlechte WCE Instanzen getestet, gefolgt von guten und schlechten Instanzen für K-WCE. Da das Lösen von MM-WCE Instanzen mit CPLEX nicht implementiert ist wird dafür nur SYMPHONY getestet.

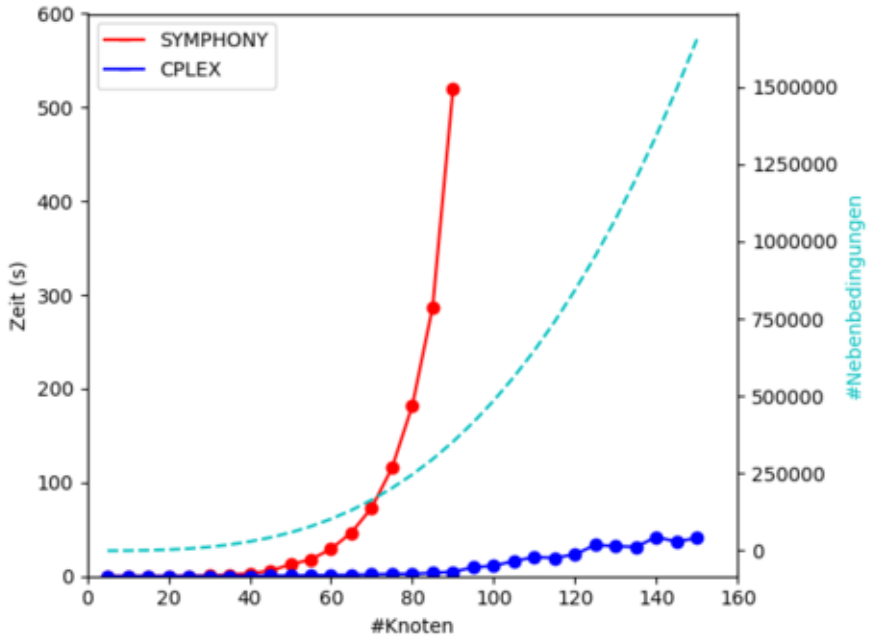


Abbildung 2: Gute WCE Instanzen

In Abbildung 2 ist eindeutig zu erkennen, dass SYMPHONY zum Lösen jeder Instanz länger braucht. Des Weiteren kann man bei steigender Knotenanzahl ein weitaus stärkeres Wachstum beobachten als bei CPLEX. Hier wäre die Grenze für Instanzen die man gut mit SYMPHONY lösen kann bei circa 90 Knoten.

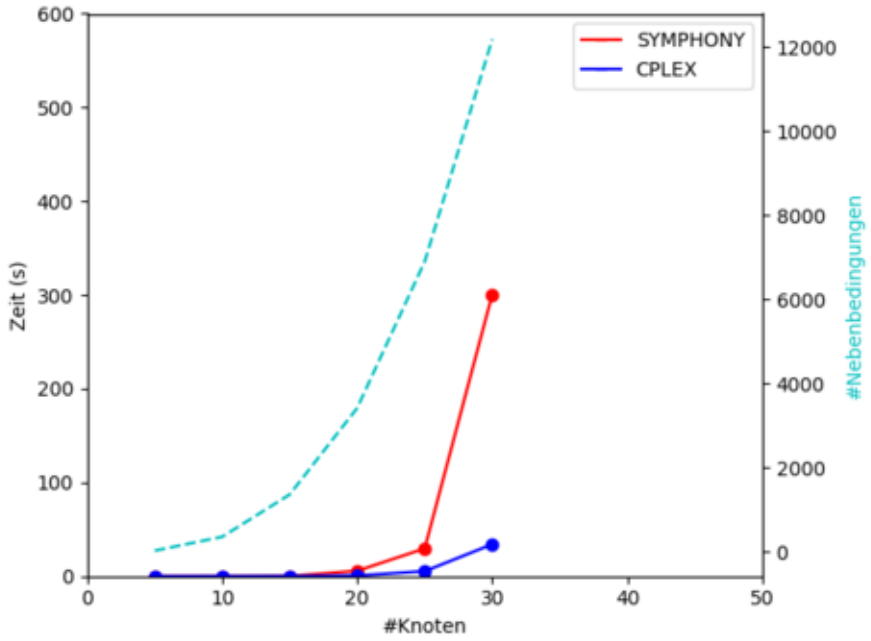


Abbildung 3: Schlechte WCE Instanzen

Aus Abbildung 3 lässt sich die Selbe Behauptung ableiten wie aus 2, was diese weiter unterstützt. Jedoch erkennt man diese Entwicklung bei den schlechten Instanzen schon mit sehr viel kleineren Instanzen. Sowohl SYMPHONY als auch CPLEX, weisen bei einer Größe von 30 Knoten schon Laufzeiten auf, die bei guten Instanzen erst bei 90 Knoten zu sehen sind. Daher ist der Rahmen für schlechte Instanzen nur bei 30 Knoten.

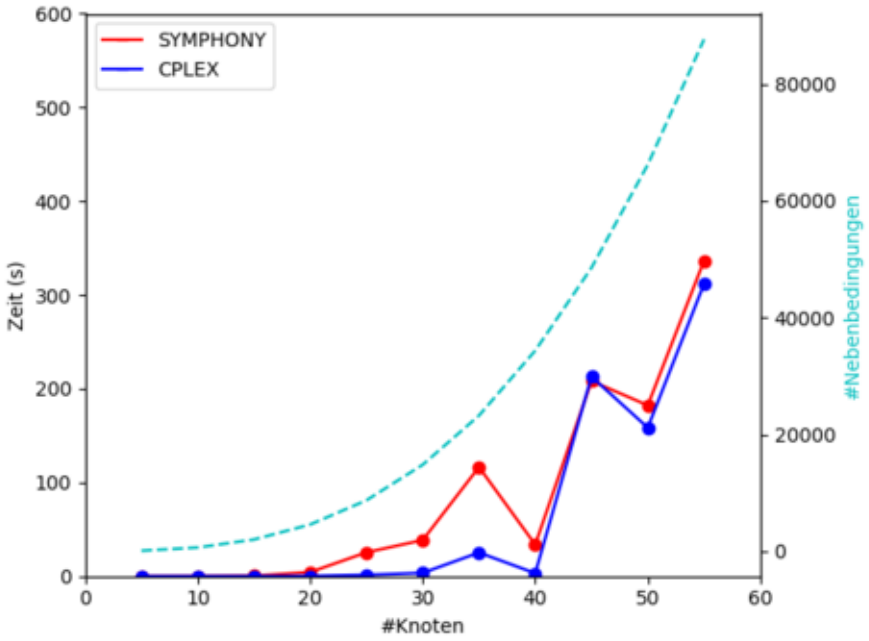


Abbildung 4: Gute 2-WCE Instanzen

In Abildung 4 und 5 kann man neben dem üblichen Anstieg der Laufzeiten auch ein Zick-Zack-Muster erkennen. Bei verschiedenen K Werten scheint die Laufzeit nicht nur von der Knotenanzahl, sondern auch von dem Parameter K abzuhängen. Der Rahmen für Gute K-WCE Instanzen liegt hier bei circa 60 Knoten.

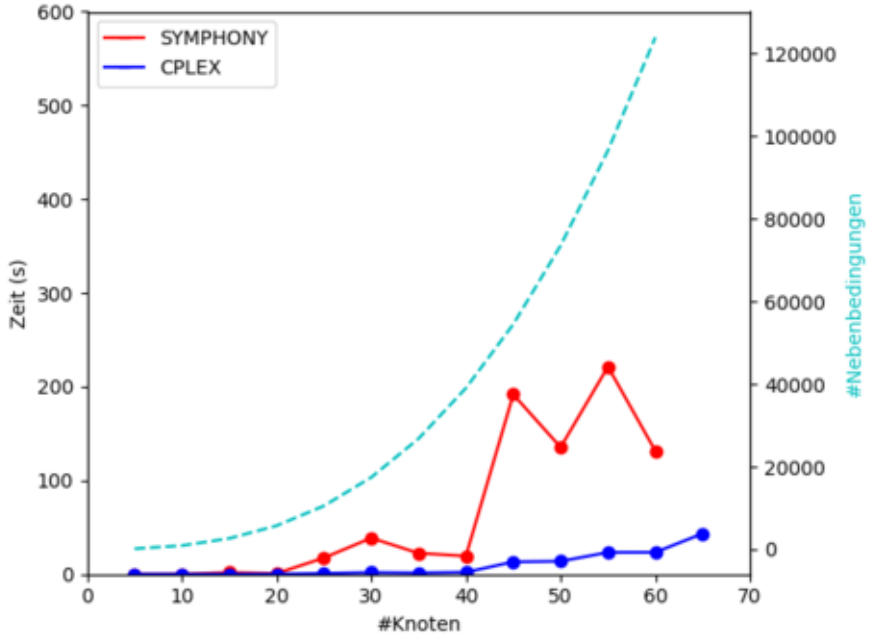


Abbildung 5: Gute 4-WCE Instanzen

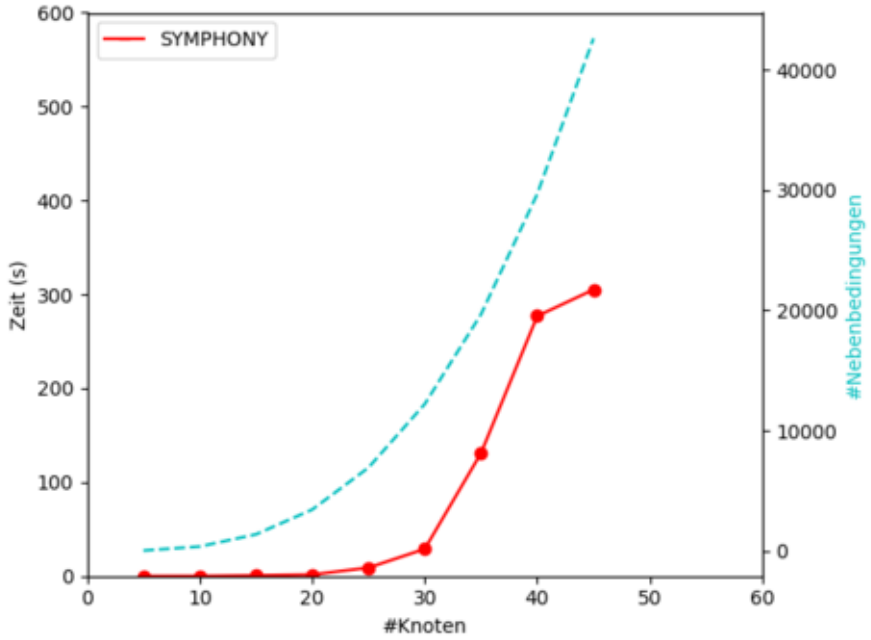


Abbildung 6: Gute M5M5-WCE Instanzen

Die Laufzeiten für MM-Instanzen aus Abbildung 6 sind im Vergleich zu K-WCE Instanzen schlechter. Es ist anzunehmen, dass eine ähnliche Implementierung mit CPLEX auch zu besseren Laufzeiten führen würde. Der Rahmen lässt sich auch circa 45 Knoten setzen.

5 Fazit

5.1 Ergebnisse

Die Tests aus 4.3 zeigen eindeutig, dass CPLEX unter den selben Voraussetzungen schneller arbeitet als SYMPHONY. Dabei spielt es keine Rolle um welche WCE-Art es sich handelt oder mit welchen Parametern ein Aufruf gestartet wird. Eine Ähnliche Beobachtung lässt für MM-WCE Instanzen vermuten. Jedoch ist es interessant zu beobachten, wie stark die Laufzeit von der Qualität der Eingabe abhängt. Schlechte Instanzen brauchen dabei circa so lange wie gute Instanzen dreifacher Größe, obwohl die Anzahl der Nebenbedingungen polynomiell zu der Anzahl der Knoten steigt. Trotzdem lässt sich aus den Tests eine relative Aussage über SYMPHONY machen. Für alle guten Instanzen, mit circa 90 Knoten oder Instanzen die sich auf unter circa 90 Knoten reduzieren lassen, ist SYMPHONY eine praktische Alternative zu CPLEX. Da die Reduktionsregeln für die Tests ausgeschaltet wurden, ist es durchaus möglich deutlich größere Eingaben mit SYMPHONY zu berechnen, solange sich diese reduzieren lassen.

5.2 Ausblick

Es gibt einige Funktionen die in Yoshiko mit CPLEX implementiert wurden, die noch nicht in die Version mit SYMPHONY übernommen wurden. So kann sich der Benutzer unter Verwendung von CPLEX mehr als nur eine optimale Lösung ausgeben lassen. Dies ist vor allem für den praktischen Gebrauch sehr nützlich. Es kann passieren, dass es für eine WCE Instanz entweder mehrere optimale Lösungen oder eine Vielzahl von Lösungen gibt die ähnlich gut sind. In diesen Fällen ist es dann sehr wichtig, sich diese Lösungen einzeln anzuschauen und zu entscheiden, welche in dem Kontext wirklich die beste Lösung darstellt.
Des Weiteren gibt es Verfahren in der ganzzahligen linearen Optimierung mit denen man in der Lage ist während des Lösungsprozesses zusätzliche Einschränkungen vorzunehmen. So lassen sich auf mit SYMPHONY benutzerdefinierte Änderungen am Lösungsprozess vornehmen. Diese wurden in bis jetzt nicht implementiert.
Andere Verbesserungen können in beiden Versionen vorgenommen werden. Zum einen gibt es noch keine Implementierung für MM-WCE Instanzen mit CPLEX. In Verbindung damit kann man sich auch Gedanken über zusätzliche Fehlermeldungen machen. Da die min und max Werte für MM-WCE Instanzen nicht beliebig gewählt werden dürfen, könnte man zusätzliche Abfragen einbauen. So wird sichergestellt, dass eine Berechnung erst dann stattfindet, wenn vorher geprüft wurde, ob das ILP überhaupt eine Lösung besitzen kann. Dafür muss bestimmt werden, ob sich die Instanzgröße n zerlegen lässt in eine Summe der möglichen Clustergrößen: $\sum_{i=min}^{max} a_i * i = n \quad , a_i \in \mathbb{N}$.

6 Danksagung

Abschließend möchte ich mich bei meinem Betreuer Prof. Dr. Gunnar W. Klau bedanken. Durch Ihn habe ich ein Thema erhalten, an dem ich durch diese Arbeit großes Interesse

entwickelt habe. Die regelmäßigen Treffen haben eine gute Absprache ermöglicht, welche mir zu einer zielgerichteten Arbeitsweise verholfen haben.
Zugleich gilt ein großer Dank Philipp Spohr. Er konnte mir stets bei technischen Problemen rund um das Yoshiko Programm und das Rechencluster helfen.
Computational support and infrastructure was provided by the “Centre for Information and Media Technology” (ZIM) at the University of Düsseldorf (Germany).

A Quellcode und Testdaten

GitHub-Repository mit dem Quellcode und den Testdaten: `https://github.com/maklu112/yoshiko/tree/george`
Commit vom 6. August 2018: bfaa12cbb20df95804f932024e4c91137bde9c6e

B Zusätzlich ausgewertete Daten

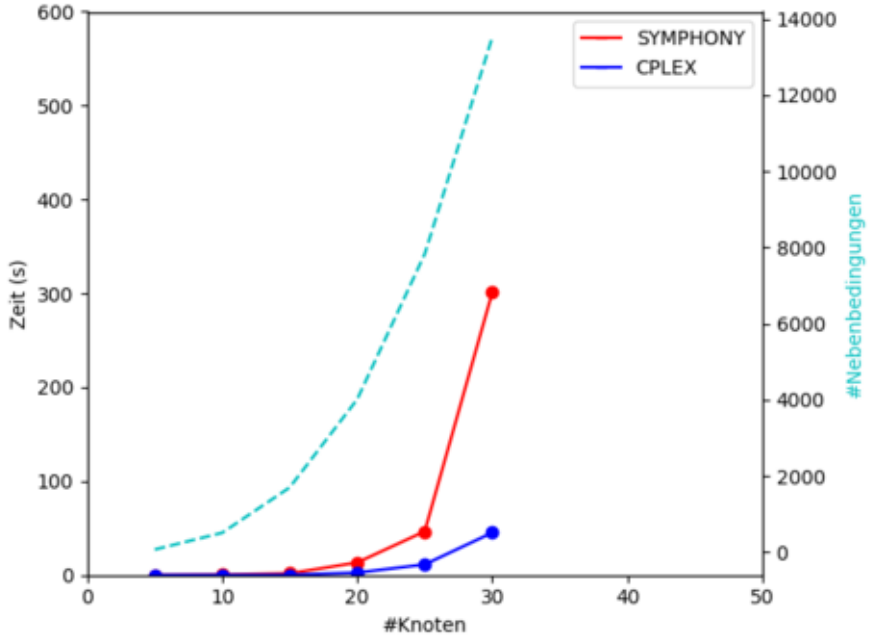


Abbildung 7: Schlechte 4-WCE Instanzen

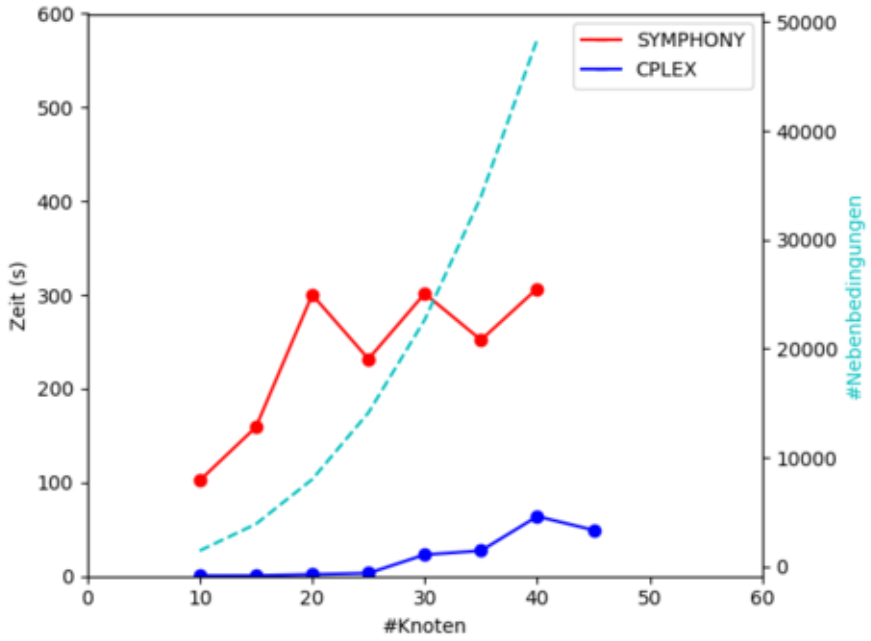


Abbildung 8: Gute 8-WCE Instanzen

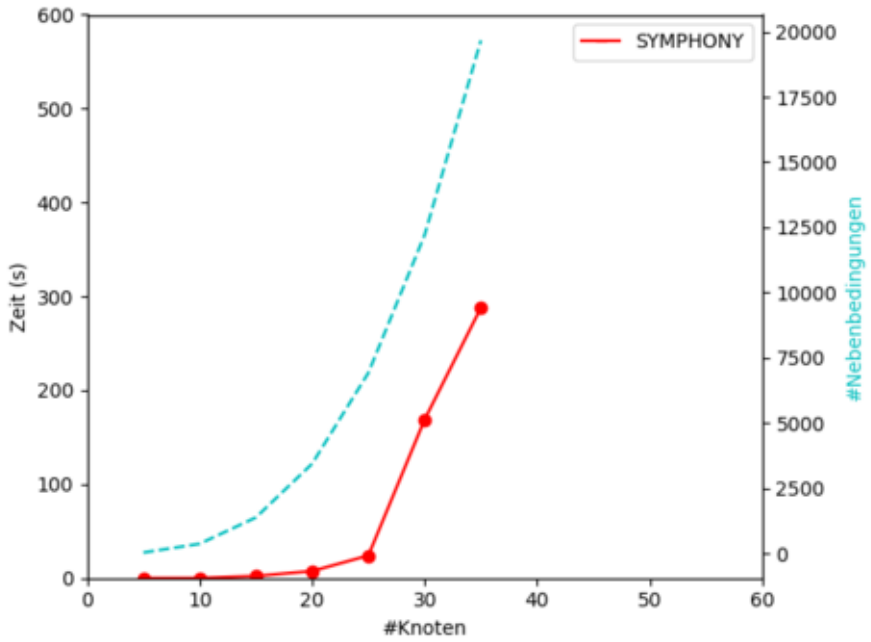


Abbildung 9: Schlechte M5M5-WCE Instanzen

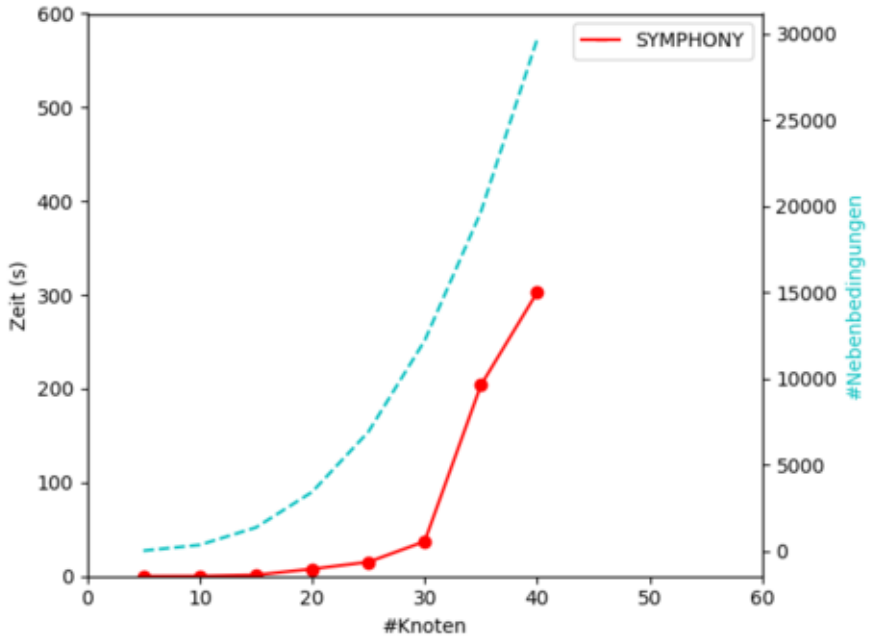


Abbildung 10: Gute M4M6-WCE Instanzen

Abbildungsverzeichnis

Abkürzungsverzeichnis

ILP Integer Linear Programming (deutsch: ganzzahliges lineares Programm)

HHU Heinrich Heine Universität

ZIM Zentrum für Informations- und Medientechnologie

Bash Bourne-again shell

Literatur

[1] BÖCKER, S., BRIESEMEISTER, S., BUI, Q., AND TRUSS, A. Going weighted, parameterized algorithms for cluster editing. In *Combinatorial Optimization and Applications* (August 2008), Springer. DOI:10.1007/978-3-540-85097-7.

[2] BÖCKER, S., BRIESEMEISTER, S., AND KLAU, G. W. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica 60*, 2 (July 2011), 316–334. DOI: 10.1007/s00453-009-9339-7.

[3] COIN-OR-FOUNDATION. About the foundation, April 2018. URL=`https://www.coin-or.org/about-the-foundation`. Stand: 4. August 2018.

[4] GILBERT, E. N. Random graphs. *The Annals of Mathematical Statistics 30* (Dezember 1959). Institute of Mathematical Statistics.

[5] KLAU, G., LAUDE, E., AND SPOHR, P. Yoshiko (version 2.2.0)[software], 2018. URL=`https://github.com/ls-cwi/yoshiko`. Stand: 4. August 2018.

[6] LAUDE, E. *Identifikation von Arten durch gewichtetes Cluster-Editing*. Julias-Maximilians-Universität Würzburg, Februar 2013.

[7] PAPADIMITRIOU, C. H., AND STEIGLITZ, K. *Combinatorial Otimization: Algorithms and Complexity*. Prentic-Hall, Januar 1982. DOI: 10.1109/TASSP.1984.1164450.

[8] SHANNON, P., MARKIEL, A., OZIER, O., BALGIA, N. S., WANG, J. T., RAMAGE, D., AMIN, N., SCHWIKOWSKI, B., AND IDEKER, T. Cytoscape: A software environment for integrated model of biomolecular ineraction networks. *Genome Research* (2003).

[9] SPOHR, P. *Developing and Evaluating a Cytoscape App for Graph-Based Clustering*. Heinrich Heine Universität Düsseldorf, November 2017.