Aufgabe 2: Stilvolle Päckchen

Teilnahme-ID: 69082

Bearbeiter dieser Aufgabe:
Vincent Xigu Liu

5. März 2024

Inhaltsverzeichnis

Das vorliegende Problem, das im Folgenden Päckchen-Pack-Problem (kurz PPP) genannt werden soll, erinnert an eine Variante des Cliquenproblems und hat zum Ziel, zueinander passende Kleidungsstücke zu finden, welche in Boxen gepackt werden. Wir modellieren das Problem als ungerichteten Graphen $G = (V, E)$, wobei jeder Knoten $V \in G$ ein Stil repräsentiert und einem Tupel $t_V \in T, t_V = (x_1, x_2, \dots, x_s)$ zugeordnet wird, wobei $s \in S$ die zugehörige Sorte ist und x_s die Anzahl des jeweiligen Kleidungsstückes einer Sorte repräsentiert. Zwischen zwei Knoten sei jetzt nun genau dann eine Kante, wenn deren Stile zusammenpassen. Da innerhalb einer gepackten Box alle Stile zueinander passen müssen, bilden alle ausgewählten Stile, die in einer Box sind, einen zugehörigen kompletten Teilgraphen. Zum Packen einer Box wählen wir also Stile aus, dessen Knoten in G mit einem kompletten Teilgraphen korrespondieren und „entnehmen“ aus den zugehörigen Tupeln t_v von jedem Knoten $V \in G$ Kleidungsstücke so, dass in der Box mindestens 1 Kleidungsstück jeder Sorte und höchstens 3 Kleidungsstücke jeder Sorte sind.

Das Ziel ist es nun, durch Packoperationen die gesamte Kleidungsstückanzahl im Graphen, oder formal ausgedrückt folgenden Ausdruck zu minimieren:

$$\sum_{t \in T} \sum_{x \in t} x$$

1. Lösungsidee

Wir vermuten zunächst, dass das Problem NP-vollständig ist. Ein konkreter Beweis wird weiter unten angegeben. Wir betrachten daher im Folgenden insbesondere Ansätze mit exponentieller Laufzeit, um das Problem zu lösen und bedienen uns Methoden linearer Optimierung, da sich aus einer gegebene Probleminstanz gut ein ganzzahliges lineares Programm herstellen lässt. Wichtig ist zu beachten, dass der folgende Lösungsansatz exakte und optimale Ergebnisse liefert und daher ineffizienter als heuristische Algorithmen sein kann. Da der Algorithmus jedoch alle Beispielseingaben in sehr kurzer Zeit löst, ist dieser zumindest für Beispieleingaben gegebener Größenordnung effizient. Die Grenzen dieses Ansatzes werden weiter unten diskutiert.

1.1. Bemerkungen zur Problemmodellierung

Da die Zeitkomplexität gängiger Methoden zum Lösen ganzzahliger linearer Programme mindestens proportional mit der Anzahl von Variablen steigt, minimieren wir zunächst mithilfe einer Hilfsumformung des Problems die benötigten Variablen, indem wir statt der Einzelpäckchen alle Kleidungsstücke in maximalen Cliquen betrachten, die in Boxen gepackt werden sollen. Diese Betrachtung ist möglich, weil folgendes Lemma gilt:

Lemma 1.1: Wenn in einem kompletten, ungerichteten Graphen G und der zugehörigen Kleidungsstücktabelle T die größte Summe einer Sorte nicht größer ist als das Dreifache der kleinsten Summe einer Sorte, gibt es ein Programm mit polynomieller Laufzeit, welche alle Kleidungsstücke in Päckchen packen kann.

Beweis. Da zunächst G komplett ist, lassen sich beliebige zwei Stile aus dem Graphen sich miteinander kombinieren. Wir betrachten nun die Summen der jeweiligen Sorten und ermitteln die größte Summe S_{max} und die kleinste Summe S_{min}. Anschließend erstellen wir S_{min} Päckchen und verteilen alle Kleidungsstücke, die zu der Sorte von S_{min} gehören, so auf die Päckchen, dass in jedem Päckchen genau ein Kleidungsstück landet. Nun verteilen wir all die andere Kleidung der anderen Sorten so in die S_{min} Päckchen, sodass in jedem Päckchen dann von jeder Sorte mindestens ein Kleidungsstück und maximal drei Kleidungsstücke vorhanden sind. Die Sorte der Kleidungsstücke ist dabei egal, da aufgrund der Komplettheit des Graphen garantiert ist, dass jeder Stil zu jedem anderen Stil passt Die untere Schranke ist möglich, weil es keine Sorte gibt, die insgesamt weniger als S_{min} Kleidungsstücke gibt, und die obere

Schranke ist möglich, weil es keine Sorte gibt, die mehr als S_{max} Kleidungsstücke hat und nach Voraussetzung $S_{max} \leq 3 * S_{min}$ gilt.

Betrachten wir die Laufzeit eines solchen Programms, ist da Summieren der Kleidungsstücke sowie das Finden der maximalen und minimalen Summe sicher in $O((n * m)^2)$ möglich, wenn n die Anzahl der Sorten und m die Anzahl der Stile ist. Da das Verteilen jedes Kleidungsstückes in die Boxen anschließend auch in Polynomialzeit lösbar ist, ist die Gültigkeit des Lemmas gezeigt. ■

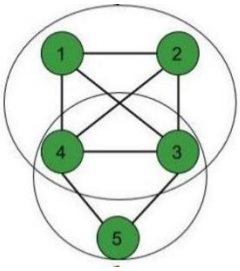

Abb.1 : Maximale Cliquen eines Graphen. Die Knoten 3 und 4 sind dabei Teil von zwei Cliquen

Da wegen Lemma 1.1 somit Lösungen, die von den Sortensummen kompletter Graphen abhängen schnell in Päckchen umwandeln kann, wollen wir im Folgenden nur alle maximalen komplette Teilgraphen, auch genannt Cliquen, im Graphen betrachten und mit deren Summen der jeweiligen Sorten arbeiten. Es ist dabei zu beachten, dass es in einem Graphen Knoten gibt, welche Teil mehrerer maximalen Cliquen sind (siehe Abb.1), sodass es unsere Aufgabe es ist, festzulegen, welcher Anteil der Kleidung dieses Knotens jeweils in welche Clique fließt. Die Ausgabe, die wir also brauchen, würde daraus bestehen, wie viele Kleidungsstücke aus jeder maximalen Clique zu Packungen umgewandelt werden sollen und aus welchen Knoten diese stammt, wenn die maximale Clique Knoten mit anderen maximalen Cliquen gemeinsam hat.

1.2. Ganzzahlige Lineare Optimierung mit Enumeration maximaler Cliquen

Der folgende Lösungsansatz nutzt Lemma 1.1 und betrachtet die Summen der Kleidungsstücke in den maximalen Cliquen in dem Graphen. Dafür nutzen wir drei Schritte:

- Enumeration aller maximalen Cliquen im Graph G und Finden von „doppelten Knoten“
- Definieren eines Linearen Programmes und Ganzahlige Lineare Optimierung
- Umwandlung der Lösung des linearen Programms zu Päckchen

1.2.1. Enumeration maximaler Cliquen und Finden „doppelter Knoten“

Um alle maximalen Cliquen in unserem Graphen zu enumerieren, bedienen wir uns hierbei am Bron-Kerbosch-Algorithmus.[1] Dieses stellt einen rekursiven Backtracking-Ansatz dar, mit welchem die maximalen Cliquen in einem ungerichteten Graph

[1] https://en.wikipedia.org/wiki/Bron%E2%80%93Kerbosch_algorithm

gefunden werden können. Anschließend iterieren wir durch die Knoten und ermitteln, ob ein Knoten in mehr als einer maximalen Clique vorkommt. Diese markieren wir dann als „doppelte Knoten“ und müssen besonders berücksichtigt werden.

1.2.2. Formulierung des Ganzzahligen Linearen Programms

Zunächst ermitteln wir die Summe der Kleidungsstücke in den Cliquen ohne „doppelten Knoten“ und halten diese als $S_{c,s}$ fest, wobei $c \in C$ eine in 1.2.1 ermittelte maximale Clique ist und s ∈ S eine Sorte ist. Wir formulieren nun das Problem als ganzzahliges lineares Programm so, dass die Variablen jeweils die Sortensummen der jeweiligen Cliquen beziehungsweise bei „doppelten Knoten“ diese der Anteil jeder Sorte ist, die dem benachbarten Knoten zugeordnet werden soll. Dabei heißt die Anzahl der Sorte s in der Clique c ohne doppelte Knoten $a_{c,s}$ und die Anzahl der Kleidung der Sorte s im Knoten V ∈ G, welche der Clique c zugeordnet wird, $v_{c,V,s}$. Alle Variablen sind dabei positive ganze Zahlen oder 0. Das lineare Programm sieht dann so aus:

$maximiere$:

$$\sum_{c\in C}\sum_{s\in S} t_{c,s} + \sum_{c\in C}\sum_{s\in S}\sum_{V\in G} v_{c,V,s}$$

sodass:

$$a_{c,s} \le S_{c,s}$$ für alle $c \in C$ und $s \in S$

$$\sum_{c\in C} v_{c,V,s} \le t_{v,s}$$ für alle $V \in G$, die „doppelt“ sind und $s \in S$

$$a_{c,s_1} + \sum_{V\in c} v_{c,V,s_1} - 3 * a_{c,s_2} - 3 * \sum_{V\in c} v_{c,V,s_2} \le 0$$ für alle $c \in C$ und alle paarweise verschiedene $s_1, s_2 \in S$.

Mit der Notation $V \in c$ ist hierbei gemeint, dass alle Knoten V, die in der Clique c sind, betrachtet werden. In der Zielfunktion werden alle betrachteten Variablen, deren Summe die Anzahl packbarer Kleidungsstücke repräsentiert, dargestellt. Die ersten zwei Einschränkungen, welche aus $|C| * |S| + \left|G_{doppelt}\right| * |S|$ vielen Ungleichungen bestehen, wobei $G_{doppelt}$ die Anzahl der doppelten Knoten im Graphen darstellt, schränken die Anzahl möglicher Kleidungsstücke nach der Eingabe in die Stile-Tabelle nach oben ein. Die dritte Einschränkung beschreibt für jede Clique mit $|S|^2$, also insgesamt mit $|C| * |S|^2$ Ungleichungen dann die in Lemma 1.1 geforderten Verhältnisse der Summen, damit dieses angewandt werden kann.

Lemma 1.2.2.1: Die Gesamtanzahl der Lösungen des linearen Programms übersteigt nicht die Gesamtzahl der verfügbaren Kleidungsstücke von jedem Knoten.

Beweis. Folgt sofort aus den durch die Einschränkungen 1 und 2 aufgestellten Ungleichungen, die besagen, dass die Summen der entsprechenden Variablen kleiner gleich die gegebene Anzahl der Kleidungsstücke sein müssen.

Lemma 1.2.2.2: Die Summen der Variablen für jede Clique sind so, dass keine Summe mehr als dreimal größer ist als die kleinste Summe.

Beweis. Da keine Summe größer als drei Mal so groß sein darf wie die kleinste Summe, ist sie auch kleiner als drei Mal so groß wie jede andere Summe. Seien $S_1, S_2, \dots, S_n$ die Summen der *n* Sorten einer Clique. Wenn jede Summe nun kleiner gleich das Dreifache einer anderen Summe ist, also $S_i \leq 3 * S_j \Leftrightarrow S_i - 3 * S_j \leq 0, i \neq j$, ist der Satz erfüllt. Da dies in der dritten Einschränkung für jede Clique gesichert ist, ist die Behauptung gezeigt.

Satz 1.2.2: Die optimale Lösung des oben vorgestellten linearen Programms repräsentiert die optimale legitime Verteilung der Kleidungsstücke nach maximalen Cliquen.

Beweis. Folgt unmittelbar aus Lemma 1.2.2.1 und Lemma 1.2.2.2.■

1.2.3 Packen der Lösung in Päckchen

Im Lemma 1.1 wurde ein Algorithmus aufgezeigt, welcher die gegebenen Kleidungsstücke anhand von Cliquen in Päckchen zusammenpacken kann. Da die Lösungen des linearen Programms genau solche Cliquenverteilungen beschreiben, können wir nun genau diesen Algorithmus benutzen, um entsprechende Päckchen zu packen.

Genaueres zur Implementation ist hierbei im Quellcode auskommentiert nachzulesen.

1.2.3 Laufzeitbetrachtung

Wir erkennen, dass die Enumeration aller maximalen Cliquen in $O\left(3^{n/3}\right)$ durchführbar ist[2], wobei *n* die Anzahl der Stile im Graphen ist. Die Laufzeit des ILP-Optimierungsprozesses ist dabei asymptotisch nicht höher $O(2^{N*R})$, wobei R die Anzahl der Terme und N die Anzahl der Variablen sind. Der Packprozess an sich gliedert sich durch die Anzahl der Kleider und die Anzahl der erstellten Boxen. Wir müssen jedes Kleidungsstück betrachten und diese in eine Box packen. Wenn also r die Anzahl der Kleidungsstücke ist, ist die asymptotische Laufzeit des Programms nicht höher als $O(r^2)$. Zusammenfassend ist also die asymptotische Laufzeit des Programms $O\left(3^{n/3} + 2^{N*R} + r^2\right)$, was einer exponentiellen Laufzeit von $O(2^{N*R})$ bei sehr großen Angaben entspricht, weil diese Komponente der Laufzeit am schnellsten wächst. Da hierbei das Produkt $N * R$ bei einer Zunahme an Sorten deutlich schneller als porpotional wächst (da mit mehr stilen und somit kombinationen die cliquen auch größer werden, was) und

[2] https://en.wikipedia.org/wiki/Bron%E2%80%93Kerbosch_algorithm

zudem ein exponentieller Faktor $3^{n/3}$ in der Laufzeit enthalten ist, vermuten wir hierbei, dass eine Erhöhung der Anzahl von Stilen zu einem deutlichen Anstieg in der Laufzeit führt.

2. Komplexitätsbetrachtung

Wie schon im oberen Text angedeutet, gehen wir von einer NP-schwere, genauer von einer NP-Vollständigkeit des Problems aus. Im Folgenden soll ein Beweis hierfür erbracht werden, indem wir zeigen, dass das Problem in der Klasse NP liegt und anschließend einen Beweis für die NP-schwere durch eine Polynomialzeitreduktion von 3-CNF-SAT [3] auf dieses Problem durchführen. Dabei sei die formale Sprache des korrespondierenden Entscheidungsproblems:

PPP = {⟨G, T⟩ : G = (V, E) ist ein ungerichteter Graph,
r ∈ ℕ ist die Anzahl der Sorten,
k ∈ ℕ ist eine Anzahl an Päckchen,
P ist die Menge der Packungen,
T ist die Menge der r-Tupel t_v für jeden Knoten $V \in$ G
und alle Elemente aus t_v sind positive ganze Zahlen, und
es existieren mindestens k *r* – Tupel $p \in P$, welche die Kleidungsstücke aus T eindeutig abbilden und für diese die entsprechenden Regeln zum Päckchenpacken erfüllt sind.
}

Lemma 2.1.1: Das Päckchen-Pack-Problem ist Komplexitätsklasse NP zugehörig.
Beweis. Die zu zeigende Behauptung ist äquivalent zu einer Annahme, dass ein Polynomialzeitalgorithmus zur Verifikation eines Zertifikats einer entsprechenden Instanz von PPP vorliegt. [4] Es sei dafür das Zertifikat die Menge P, welche alle Packungen zu einem Graphen beinhaltet. Der Algorithmus prüft nun, ob für jedes Kleidungsstück aus den Tupeln $p \in$ P höchstens ein Kleidungsstück bzw. Eintrag in $t_v \in T$ existiert und überprüft für jedes p, ob diese auch die gegebenen Regeln (minimale und maximale Anzahl der Kleidungsstücke, zusammenpassen der Stile, gekennzeichnet durch die Kanten im Graphen G) erfüllen. Da ein solcher Algorithmus in Polynomialzeit läuft, ist die Behauptung gezeigt.

Lemma 2.1.2: Das Päckchen-Pack-Problem ist NP-schwer.
Beweis. Um die NP-schwere des PPP zu zeigen, führen wir eine Polynomialzeitreduktion von 3-SAT auf PPP durch und zeigen, dass ein Päckchen derart gepackt werden kann, dass diese eine Lösung für die gegebene Instanz von 3-CNF-SAT darstellt. Dazu sei eine Instanz von 3-CNF-SAT mit s Klauseln $C_1, C_2, \dots, C_s$ und t Variablen $x_1, x_2, \dots, x_t$ gegeben. Ohne Beschränkung der Allgemeinheit nehmen wir an, dass in jeder Klausel nicht eine Variable und ihre Negation gleichzeitig vorzufinden ist, weil diese Klausel ansonsten in jedem Fall wahr ist und wir diese nicht

[3] https://de.wikipedia.org/wiki/3-SAT
[4] „Introduction to algorithms“, second edition, S.981

beachten müssen. Nun konstruieren wir für jede der r Variablen genau zwei Knoten in G und entsprechende Tupel in T, welche jeweils zwei Stile repräsentieren und benennen diese nach den Variablen. Für eine Variable x_1 konstruieren wir also beispielsweise zwei Stile x_1 und $\overline{x_1}$. Für jede der s Klausel erstellen wir eine Sorte und legen damit fest, dass die Tupel in T s-Tupel sind. Zudem erstellen wir pro Klausel drei Kleidungsstücke. Die Kleidungsstücke verteilen wir nun den Variablen in den jeweiligen Klauseln entsprechend in den korrespondierenden Stilen von T, wobei die Klausel an sich die Sorte des Kleidungsstücks bestimmt. Ist beispielsweise die Klausel $C_1 = (x_1 \vee \overline{x_2} \vee x_3)$, so addieren wir Kleidungsstücke der Sorte 1 mit den Stilen x_1, $\overline{x_2}$ und x_3 zu T. Zuletzt legen wir fest, dass jeder Stil mit allen anderen Stilen außer der eigenen Komplementärvariable zusammenpasst. Der Stil x_1 würde also mit allen Variablen außer $\overline{x_1}$ zusammenpassen.

Nun zeigen wir, dass eine Instanz von 3-CNF-SAT dann wahr ist, wenn eine derart konstruierte Instanz von PPP mit k = 1 wahr ist, und dann falsch ist, wenn die PPP-Instanz falsch ist. Wenn PPP zur gegebenen Instanz mindestens ein Päckchen packen kann, gilt dann nach den Regeln zu den Päckchen:

1. Mindestens ein Stil und höchstens drei Stile pro Klausel kann gepackt werden, dies korrespondiert damit, dass in der entsprechenden 3-CNF-SAT-Instanz pro Klausel mindestens eine Variable und höchstens drei Variablen Wahr sind.
2. In jeder Packung ist von den paarweise konstruierten Stilen höchstens eins der zwei Stile vorhanden, da diese nicht miteinander kombinierbar sind. Es kann also beispielsweise nur x_1 oder $\overline{x_1}$ in der Packung enthalten sein. Dies korrespondiert damit, dass in der entsprechenden 3-CNF-SAT-Instanz jede Variable nur wahr oder falsch sein kann. Taucht ein Stil und sein Komplementärstil in der Packung garnicht auf, gilt für die 3-CNF-SAT-Instanz, dass diese Variable beliebig wahr oder falsch sein kann.

Da die zwei Bedingungen hinreichend sind für die Erfüllbarkeit der korrespondierenden 3-CNF-SAT-Instanz, ist gezeigt, dass diese dann wahr ist, wenn PPP wahr ist. Eine analoge Begründung sagt zudem aus, dass die 3-CNF-SAT entsprechend dann falsch ist, wenn PPP falsch ist. Damit ist die oben genannte Behauptung gezeigt.

Da der Reduktionsalgorithmus zunächst in Polynomialzeit G (inklusive seiner Kanten) und T erstellt und anschließend in Polynomialzeit anhand der 3-CNF-SAT-Instanz die t_v-Tupel aus T füllt, ist die asymptotische Laufzeit polynomial. Daraus ergibt sich, dass 3-SAT in Polynomialzeit auf PPP reduzierbar ist und somit aufgrund der NP-schwere von 3-CNF-SAT auch NP-schwer ist.

Satz 2.1: Das Päckchen-Pack-Problem (PPP) ist NP-vollständig.
Beweis. Folgt direkt aus den Lemmata 2.1.1 und 2.1.2. ■

3. Boxengenerator

Da alle Beispielseingaben relativ schnell gelöst werden können, ist es sinnvoll, anhand großer Eingaben die Grenzen des Programms auszutesten und zu ermitteln, wann eine Errechnung der benötigten Kleidungsstücke zu lange dauert. Dann ist es möglich, sich im nächsten Schritt Gedanken über mögliche Heuristiken für große Beispielseingaben

zu machen, wenn die Größenordnung bekannt ist. Auch wollen wir hiermit untersuchen, wann eher wenige Kleidungsstücke gepackt werden können und wann eher viele Kleidungsstücke gepackt werden können. Dazu erstellen wir einen Algorithmus, welcher anhand von Parametern eine Eingabetabelle erstellt und ermitteln daran die benötigte Laufzeit unseres ILP-Programms. Dabei berücksichtigen wir folgende Parameter:

1. Anzahl der Sorten
2. Anzahl der Stile
3. Dichte des Graphen (wieviel Prozent der Kanten werden tataächlcih errichtet bzw. wie oft passen Stile zueinander?)
4. Wahrscheinlichkeit, dass von einer Sorte Restkleidung übrigbleibt
5. Minimale und maximale Anzahl der gepackten Kleidungsstücke, vorausgesetzt diese sind vorhanden

Mithilfe eines Pseudo-Zufallszahlengenerators werden diese Parameter nun in einem Programm eingestellt, passende Beispiele generiert und in eine Textdatei geschrieben, welche dann vom Programm ausgeführt werden kann. Die konkreten Ergebnisse sind dann in der Rubrik „Beispiele“ einzusehen. Das Generatorprogramm ist mit in der Abgabe enthalten.

4. Implementierung

Die Lösungsidee implementieren wir in Python mit der PuLP-Bibliothek, welche einen ILP-solver mit Optimierungen bereitstellt, um das lineare Programm zu lösen. Um das eingereichte Programm auszuführen, ist daher die Installation der PuLP-Bibliothek notwendig. Für die Darstellung des Graphen und der entsprechenden Tupeln nutzen wir dabei ein Dictionary, das für jeden Knoten mit Nummer als Schlüssel die Liste `[[liste_an_benachbarten_knoten],[tupeln_der_knoten]]` enthält. Die Programme sind nach ihren Aufgaben benannt und enthalten am Anfang eine Kurzbeschreibung ihrer Funktionalität sowie in den Absätzen nähere Erläuterung ihrer Funktion.

5. Beispiele

Im Folgenden fassen wir alle aufgezeigten Beispiele und ihre Laufzeit sowie die Anzahl nicht gepackter Kleidungsstücke zusammen. Die Laufzeiten messen wir dabei im Programm direkt mithilfe der Time-Bibliothek von Python. Die genauen Packungen und die Programmausgaben sind im Wesentlichen weiter unten zusammengefasst.

Eingabedatei	**Laufzeit (in Sekunden)**	**Anzahl übriggebliebener Kleidungsstücke**
Paeckchen0.txt	0.060964822769165	0
Paeckchen1.txt	0.133914470672607	0

Paeckchen2.txt	0.124993562698364	0
Paeckchen3.txt	0.246849060058593	2
Paeckchen4.txt	1.309187889099121	0
Paeckchen5.txt	0.234364748001098	19
Paeckchen6.txt	0.168896198272705	22
Paeckchen7.txt	0.153902053833007	131

Die Programme liefen dabei auf einem Kern eines Intel i5-2430M.

Nun zeigen wir den Ausschnitt aus den Ausgaben, welche die Packungen darstellen, auf. Die Päckchen sind dazu ihrer zugehörigen Clique nach gegliedert, und jede Zeile stellt ein Päckchen dar. Die Kleidung wird durch eine Liste `[stil, sorte]` repräsentiert, während Kleidung gleicher Sorte auch nochmal in Eckigen Klammern zusammengefasst wurden. Aus Platzgründen werden die Ausgaben mit Schriftgröße 9 abgedruckt.

Paeckchen0

```
gepackte Kleidungsstücke:     11.0
tatsächliche Kleidungsstücke:  11
Übrige Kleidungsstücke:  0.0
Clique:  {1, 2}
clique: 0
[[1, 0], [2, 0]] [[1, 1]] [[1, 2]]
[[1, 0], [2, 0]] [[1, 1]] [[1, 2]]
[[1, 0]] [[1, 1]] [[1, 2]]
```

Paeckchen1

```
gepackte Kleidungsstücke:     499.0
tatsächliche Kleidungsstücke:  499
Übrige Kleidungsstücke:  0.0
Clique:  {1, 2}
Clique:  {2, 3}
Clique:  {3, 4}
clique: 0
[[2, 0], [1, 0]] [[2, 1], [1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1], [1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1], [1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1], [1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1], [1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1], [1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1], [1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1], [1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1], [1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1], [1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1], [1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1], [1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1]] [[1, 2]]
```

[[2, 0], [1, 0]] [[2, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[2, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[1, 1]] [[1, 2]]
[[2, 0], [1, 0]] [[1, 1]] [[1, 2]]
[[2, 0]] [[1, 1]] [[1, 2]]
[[2, 0]] [[1, 1]] [[1, 2]]
[[2, 0]] [[1, 1]] [[1, 2]]
[[2, 0]] [[1, 1]] [[1, 2]]
[[2, 0]] [[1, 1]] [[1, 2]]
[[2, 0]] [[1, 1]] [[1, 2]]
[[2, 0]] [[1, 1]] [[1, 2]]
[[2, 0]] [[1, 1]] [[1, 2]]
[[2, 0]] [[1, 1]] [[1, 2]]
[[1, 0]] [[1, 1]] [[1, 2]]
[[1, 0]] [[1, 1]] [[1, 2]]
clique: 1
[[3, 0], [3, 0], [3, 0]] [[2, 1]] [[2, 2], [2, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[2, 1]] [[2, 2], [2, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[2, 1]] [[2, 2], [2, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[2, 1]] [[2, 2], [2, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[2, 1]] [[2, 2], [2, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[2, 1]] [[2, 2], [2, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[2, 1]] [[2, 2], [2, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[2, 1]] [[2, 2], [2, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[2, 1]] [[2, 2], [2, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[2, 1]] [[2, 2], [2, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[2, 1]] [[2, 2], [2, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[2, 1]] [[2, 2], [2, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[2, 1]] [[2, 2], [2, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[2, 1]] [[2, 2], [3, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[2, 1]] [[2, 2], [3, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[2, 1]] [[2, 2], [3, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[3, 1]] [[2, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[3, 1]] [[2, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[3, 1]] [[2, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[3, 1]] [[2, 2], [3, 2]]

```
[[3, 0], [3, 0], [3, 0]] [[3, 1]] [[2, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[3, 1]] [[2, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[3, 1]] [[2, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[3, 1]] [[2, 2], [3, 2]]
[[3, 0], [3, 0], [3, 0]] [[3, 1]] [[2, 2], [3, 2]]
clique: 2
[[3, 0], [4, 0]] [[3, 1], [3, 1], [4, 1], [4, 1]] [[4, 2]]
[[4, 0], [4, 0]] [[3, 1], [4, 1], [4, 1]] [[4, 2]]
[[4, 0], [4, 0]] [[3, 1], [4, 1], [4, 1]] [[4, 2]]
[[4, 0], [4, 0]] [[3, 1], [4, 1], [4, 1]] [[4, 2]]
[[4, 0], [4, 0]] [[3, 1], [4, 1], [4, 1]] [[4, 2]]
[[4, 0], [4, 0]] [[3, 1], [4, 1], [4, 1]] [[4, 2]]
[[4, 0], [4, 0]] [[3, 1], [4, 1], [4, 1]] [[4, 2]]
[[4, 0], [4, 0]] [[3, 1], [4, 1], [4, 1]] [[4, 2]]
[[4, 0], [4, 0]] [[3, 1], [4, 1], [4, 1]] [[4, 2]]
[[4, 0]] [[3, 1], [4, 1], [4, 1]] [[4, 2]]
[[4, 0]] [[3, 1], [4, 1], [4, 1]] [[4, 2]]
[[4, 0]] [[3, 1], [4, 1], [4, 1]] [[4, 2]]
[[4, 0]] [[3, 1], [4, 1], [4, 1]] [[4, 2]]
[[4, 0]] [[3, 1], [4, 1], [4, 1]] [[4, 2]]
[[4, 0]] [[3, 1], [4, 1], [4, 1]] [[4, 2]]
[[4, 0]] [[3, 1], [4, 1], [4, 1]] [[4, 2]]
[[4, 0]] [[3, 1], [4, 1], [4, 1]] [[4, 2]]
[[4, 0]] [[3, 1], [4, 1], [4, 1]] [[4, 2]]
[[4, 0]] [[3, 1], [4, 1]] [[4, 2]]
```

Paeckchen2

```
gepackte Kleidungsstücke:      32.0
tatsächliche Kleidungsstücke:  32
Übrige Kleidungsstücke:  0.0
Clique:  {1, 2, 3, 9}
Clique:  {1, 2, 3, 5}
Clique:  {8, 1, 2, 6}
Clique:  {8, 1, 2, 9}
Clique:  {1, 4, 5}
Clique:  {9, 3, 7}
clique: 0
clique: 1
[[3, 0]] [[2, 1], [5, 1]] [[2, 2], [2, 2], [2, 2]]
clique: 2
[[2, 0]] [[8, 1], [6, 1]] [[8, 2]]
[[6, 0]] [[6, 1], [6, 1]] [[8, 2]]
clique: 3
clique: 4
[[1, 0]] [[5, 1], [4, 1], [4, 1]] [[5, 2], [5, 2], [5, 2]]
clique: 5
[[7, 0]] [[7, 1], [7, 1]] [[9, 2], [9, 2], [7, 2]]
[[7, 0]] [[7, 1], [7, 1]] [[9, 2], [9, 2]]
```

Paeckchen3

```
gepackte Kleidungsstücke:      94.0
tatsächliche Kleidungsstücke:  96
Übrige Kleidungsstücke:  2.0
Clique:  {8, 1, 2, 6}
Clique:  {8, 2, 6, 7}
Clique:  {2, 3, 7}
Clique:  {9, 3, 7}
Clique:  {4, 5}
Clique:  {6, 7, 8, 9, 10}
clique: 0
[[8, 0], [8, 0], [1, 0], [1, 0]] [[2, 1], [1, 1], [1, 1]] [[2, 2]] [[2, 3], [2, 3], [1, 3]] [[8, 4], [8, 4]]
```

[[8, 0], [1, 0]] [[2, 1], [1, 1]] [[1, 2]] [[2, 3], [1, 3]] [[8, 4], [6, 4]]
[[8, 0], [1, 0]] [[2, 1], [1, 1]] [[1, 2]] [[2, 3]] [[8, 4], [6, 4]]
clique: 1
[[7, 0]] [[6, 1], [7, 1]] [[6, 2], [6, 2]] [[6, 3]] [[6, 4]]
clique: 2
[[2, 0], [2, 0]] [[2, 1]] [[3, 2]] [[2, 3], [2, 3], [3, 3]] [[2, 4]]
clique: 3
[[7, 0], [9, 0], [9, 0]] [[9, 1]] [[3, 2], [9, 2], [9, 2]] [[9, 3]] [[3, 4]]
clique: 4
[[4, 0]] [[4, 1], [5, 1], [5, 1]] [[4, 2], [4, 2]] [[4, 3], [5, 3]] [[4, 4], [4, 4]]
[[4, 0]] [[4, 1], [5, 1], [5, 1]] [[4, 2], [4, 2]] [[4, 3], [5, 3]] [[4, 4], [4, 4]]
[[4, 0]] [[4, 1], [5, 1], [5, 1]] [[4, 2], [4, 2]] [[4, 3], [5, 3]] [[4, 4]]
clique: 5
[[7, 0]] [[8, 1], [8, 1], [10, 1]] [[10, 2]] [[9, 3], [9, 3], [9, 3]] [[9, 4], [9, 4], [10, 4]]

Paeckchen4

vepackte Kleidungsstücke: 437.0
tatsächliche Kleidungsstücke: 437
Übrige Kleidungsstücke: 0.0
Clique: {16, 1, 24, 23}
Clique: {17, 2, 10, 4}
Clique: {16, 9, 3, 23}
Clique: {10, 4, 6, 14}
Clique: {24, 7, 5, 23}
Clique: {8, 10, 12, 22}
Clique: {10, 12, 21}
Clique: {10, 21, 23}
Clique: {18, 11, 21, 15}
Clique: {21, 19, 12, 20}
Clique: {18, 21, 20, 13}
clique: 0
[[24, 0]] [[16, 1], [16, 1]] [[16, 2]] [[16, 3], [16, 3]] [[16, 4], [23, 4], [23, 4]] [[16, 5]] [[1, 6], [1, 6], [1, 6]]
[[24, 0]] [[16, 1], [16, 1]] [[16, 2]] [[16, 3], [16, 3]] [[16, 4], [23, 4], [23, 4]] [[16, 5]] [[1, 6], [1, 6], [1, 6]]
[[24, 0]] [[16, 1]] [[16, 2]] [[16, 3]] [[23, 4], [23, 4], [23, 4]] [[16, 5]] [[1, 6], [1, 6], [1, 6]]
[[24, 0]] [[16, 1]] [[16, 2]] [[16, 3]] [[23, 4], [23, 4], [23, 4]] [[16, 5]] [[1, 6], [1, 6], [1, 6]]
clique: 1
[[17, 0], [17, 0], [17, 0]] [[10, 1], [4, 1], [4, 1]] [[4, 2], [4, 2], [4, 2]] [[4, 3], [4, 3]] [[4, 4], [4, 4], [4, 4]] [[4, 5], [4, 5], [4, 5]] [[17, 6]]
[[17, 0], [17, 0], [17, 0]] [[4, 1], [4, 1], [4, 1]] [[4, 2], [4, 2], [4, 2]] [[4, 3], [4, 3]] [[4, 4], [4, 4]] [[4, 5], [4, 5], [4, 5]] [[17, 6]]
[[17, 0], [17, 0], [17, 0]] [[4, 1], [4, 1]] [[4, 2], [4, 2], [4, 2]] [[4, 3], [4, 3]] [[4, 4], [4, 4]] [[4, 5], [4, 5], [4, 5]] [[17, 6]]
clique: 2
[[9, 0]] [[16, 1]] [[16, 2], [16, 2]] [[16, 3], [16, 3], [16, 3]] [[16, 4], [16, 4], [16, 4]] [[16, 5], [16, 5]] [[9, 6], [9, 6], [9, 6]]
clique: 3
[[6, 0], [6, 0], [6, 0]] [[4, 1], [4, 1], [4, 1]] [[4, 2], [4, 2]] [[4, 3], [4, 3], [4, 3]] [[4, 4]] [[4, 5]] [[6, 6]]
clique: 4
[[24, 0]] [[5, 1], [5, 1]] [[5, 2], [5, 2]] [[5, 3], [5, 3]] [[5, 4], [5, 4]] [[5, 5], [5, 5]] [[5, 6], [7, 6], [7, 6]]
[[24, 0]] [[5, 1]] [[5, 2], [5, 2]] [[5, 3], [5, 3]] [[5, 4]] [[5, 5]] [[5, 6], [7, 6], [7, 6]]
[[24, 0]] [[5, 1]] [[5, 2]] [[5, 3]] [[5, 4]] [[5, 5]] [[5, 6], [7, 6], [7, 6]]
[[24, 0]] [[5, 1]] [[5, 2]] [[5, 3]] [[5, 4]] [[5, 5]] [[5, 6], [7, 6], [7, 6]]
[[24, 0]] [[5, 1]] [[5, 2]] [[5, 3]] [[5, 4]] [[5, 5]] [[5, 6], [7, 6], [7, 6]]
[[24, 0]] [[5, 1]] [[5, 2]] [[5, 3]] [[5, 4]] [[5, 5]] [[5, 6], [7, 6], [7, 6]]
clique: 5
[[12, 0], [12, 0], [12, 0]] [[10, 1], [22, 1], [22, 1]] [[12, 2], [22, 2], [22, 2], [22, 2]] [[8, 3], [22, 3], [22, 3]] [[8, 4], [22, 4], [22, 4]] [[8, 5], [22, 5]] [[8, 6]]
[[12, 0], [12, 0], [12, 0]] [[10, 1], [22, 1], [22, 1]] [[8, 2], [22, 2], [22, 2]] [[8, 3], [22, 3], [22, 3]] [[8, 4], [22, 4], [22, 4]] [[8, 5], [22, 5]] [[8, 6]]
clique: 6
clique: 7
[[10, 0], [10, 0], [10, 0]] [[10, 1]] [[21, 2], [21, 2]] [[21, 3], [21, 3]] [[23, 4], [23, 4]] [[21, 5], [21, 5]] [[23, 6], [23, 6],

[23, 6]]
[[10, 0], [10, 0], [10, 0]] [[10, 1]] [[21, 2], [21, 2]] [[21, 3], [21, 3]] [[23, 4], [23, 4]] [[21, 5], [21, 5]] [[23, 6], [23, 6], [23, 6]]
[[10, 0], [10, 0], [10, 0]] [[10, 1]] [[21, 2], [21, 2]] [[21, 3], [21, 3]] [[23, 4]] [[21, 5], [21, 5]] [[23, 6], [23, 6], [23, 6]]
[[10, 0], [10, 0], [10, 0]] [[10, 1]] [[21, 2], [21, 2]] [[21, 3]] [[23, 4]] [[21, 5], [21, 5]] [[23, 6], [23, 6], [23, 6]]
[[10, 0], [10, 0], [10, 0]] [[10, 1]] [[21, 2]] [[21, 3]] [[23, 4]] [[21, 5]] [[23, 6], [23, 6]]
[[10, 0], [10, 0], [10, 0]] [[10, 1]] [[21, 2]] [[21, 3]] [[23, 4]] [[21, 5]] [[23, 6], [23, 6]]
[[10, 0], [10, 0], [10, 0]] [[10, 1]] [[21, 2]] [[21, 3]] [[23, 4]] [[21, 5]] [[23, 6], [23, 6]]
[[10, 0], [10, 0]] [[10, 1]] [[21, 2]] [[21, 3]] [[23, 4]] [[21, 5]] [[23, 6], [23, 6]]
clique: 8
[[11, 0]] [[11, 1], [11, 1]] [[11, 2], [11, 2], [11, 2]] [[11, 3], [11, 3], [11, 3]] [[11, 4], [11, 4]] [[11, 5], [11, 5], [11, 5]] [[18, 6], [18, 6]]
clique: 9
[[19, 0]] [[12, 1], [12, 1], [12, 1]] [[12, 2], [12, 2]] [[12, 3], [12, 3], [12, 3]] [[12, 4]] [[12, 5], [12, 5], [21, 5]] [[20, 6]]
[[19, 0]] [[12, 1], [12, 1], [12, 1]] [[12, 2], [12, 2]] [[12, 3], [12, 3], [12, 3]] [[12, 4]] [[12, 5], [21, 5], [21, 5]] [[20, 6]]
clique: 10
[[18, 0]] [[13, 1], [13, 1], [13, 1]] [[21, 2], [13, 2], [13, 2]] [[13, 3], [13, 3], [13, 3]] [[13, 4], [13, 4], [13, 4]] [[21, 5], [13, 5], [13, 5]] [[20, 6], [20, 6], [20, 6]]
[[18, 0]] [[13, 1], [13, 1]] [[21, 2], [13, 2], [13, 2]] [[13, 3], [13, 3], [13, 3]] [[13, 4], [13, 4], [13, 4]] [[13, 5], [13, 5], [13, 5]] [[20, 6], [20, 6], [20, 6]]
[[18, 0]] [[13, 1], [13, 1]] [[21, 2], [13, 2], [13, 2]] [[13, 3], [13, 3], [13, 3]] [[13, 4], [13, 4], [13, 4]] [[13, 5], [13, 5], [13, 5]] [[20, 6], [20, 6], [20, 6]]
[[18, 0]] [[13, 1], [13, 1]] [[13, 2], [13, 2], [13, 2]] [[13, 3], [13, 3], [13, 3]] [[13, 4], [13, 4]] [[13, 5], [13, 5], [13, 5]] [[20, 6], [20, 6], [20, 6]]

Paeckchen5

gepackte Kleidungsstücke: 155.0
tatsächliche Kleidungsstücke: 174
Übrige Kleidungsstücke: 19.0
Clique: {1, 2, 3}
Clique: {3, 4, 7, 15}
Clique: {9, 10, 19, 4}
Clique: {18, 12, 5, 6}
Clique: {18, 5, 6, 7}
Clique: {8, 11, 12, 15}
Clique: {8, 20, 13, 14}
Clique: {16, 17, 11, 21}
clique: 0
[[1, 0]] [[1, 1], [1, 1], [1, 1]] [[1, 2], [2, 2]] [[1, 3], [2, 3], [2, 3]] [[1, 4], [2, 4], [2, 4]]
[[1, 0]] [[1, 1], [1, 1], [1, 1]] [[1, 2]] [[1, 3], [2, 3]] [[1, 4], [2, 4], [2, 4]]
clique: 1
[[15, 0], [15, 0]] [[3, 1], [3, 1]] [[4, 2], [4, 2], [15, 2]] [[15, 3]] [[7, 4]]
[[15, 0]] [[3, 1], [3, 1]] [[4, 2], [4, 2], [15, 2]] [[15, 3]] [[7, 4]]
[[15, 0]] [[3, 1]] [[4, 2], [4, 2], [15, 2]] [[15, 3]] [[7, 4]]
clique: 2
[[9, 0], [19, 0]] [[9, 1]] [[9, 2], [10, 2], [10, 2]] [[9, 3], [19, 3]] [[9, 4], [19, 4]]
[[9, 0], [19, 0]] [[9, 1]] [[9, 2], [10, 2], [10, 2]] [[9, 3], [19, 3]] [[9, 4], [19, 4]]
[[9, 0]] [[9, 1]] [[9, 2], [10, 2], [10, 2]] [[9, 3], [19, 3]] [[9, 4], [19, 4]]
clique: 3
[[18, 0], [18, 0]] [[18, 1], [18, 1]] [[5, 2]] [[18, 3], [18, 3]] [[6, 4], [6, 4], [6, 4]]
[[18, 0], [18, 0]] [[18, 1], [18, 1]] [[5, 2]] [[18, 3], [18, 3]] [[6, 4], [6, 4], [6, 4]]
clique: 4
clique: 5
[[8, 0], [8, 0], [8, 0]] [[15, 1], [15, 1]] [[15, 2]] [[11, 3], [11, 3], [11, 3]] [[12, 4], [12, 4]]
[[8, 0], [8, 0]] [[15, 1], [15, 1]] [[15, 2]] [[11, 3], [11, 3]] [[12, 4], [12, 4]]
[[8, 0], [8, 0]] [[15, 1], [15, 1]] [[15, 2]] [[11, 3], [11, 3]] [[12, 4], [12, 4]]
[[8, 0], [8, 0]] [[15, 1], [15, 1]] [[15, 2]] [[11, 3], [11, 3]] [[12, 4], [12, 4]]
clique: 6
[[20, 0], [13, 0], [13, 0]] [[20, 1], [20, 1]] [[20, 2]] [[20, 3]] [[20, 4], [20, 4]]
clique: 7
[[16, 0], [21, 0], [21, 0]] [[16, 1], [16, 1], [16, 1]] [[16, 2], [21, 2], [21, 2]] [[16, 3], [21, 3], [21, 3]] [[16, 4]]

Paeckchen6

```
gepackte Kleidungsstücke:      748.0
tatsächliche Kleidungsstücke:  770
Übrige Kleidungsstücke:  22.0
Clique:  {1, 2, 3, 4, 10}
Clique:  {1, 3, 5, 6, 10}
Clique:  {8, 1, 10, 4}
Clique:  {9, 10, 7}
clique: 0
[[1, 0]] [[4, 1]] [[2, 2], [2, 2], [2, 2]] [[1, 3], [1, 3], [4, 3], [2, 3]] [[10, 4], [10, 4], [10, 4]]
[[1, 0]] [[4, 1]] [[2, 2], [2, 2], [2, 2]] [[1, 3], [1, 3], [4, 3], [2, 3]] [[10, 4], [10, 4], [10, 4]]
[[1, 0]] [[4, 1]] [[2, 2], [2, 2], [2, 2]] [[1, 3], [1, 3], [4, 3], [2, 3]] [[10, 4], [10, 4], [10, 4]]
[[1, 0]] [[4, 1]] [[2, 2], [2, 2], [2, 2]] [[1, 3], [1, 3], [4, 3], [2, 3]] [[10, 4], [10, 4], [10, 4]]
[[1, 0]] [[4, 1]] [[2, 2], [2, 2], [2, 2]] [[1, 3], [1, 3], [4, 3], [2, 3]] [[10, 4], [10, 4], [10, 4]]
[[1, 0]] [[4, 1]] [[2, 2], [2, 2], [2, 2]] [[1, 3], [1, 3], [2, 3]] [[10, 4], [10, 4], [10, 4]]
[[1, 0]] [[4, 1]] [[2, 2], [2, 2], [2, 2]] [[1, 3], [1, 3], [2, 3]] [[10, 4], [10, 4], [10, 4]]
[[1, 0]] [[4, 1]] [[2, 2], [2, 2], [2, 2]] [[1, 3], [1, 3], [2, 3]] [[10, 4], [10, 4]]
[[1, 0]] [[4, 1]] [[2, 2], [2, 2], [2, 2]] [[1, 3], [1, 3], [2, 3]] [[10, 4], [10, 4]]
[[1, 0]] [[4, 1]] [[2, 2], [2, 2], [2, 2]] [[1, 3], [1, 3], [2, 3]] [[10, 4], [10, 4]]
[[1, 0]] [[4, 1]] [[2, 2], [2, 2], [2, 2]] [[1, 3], [1, 3], [2, 3]] [[10, 4], [10, 4]]
[[1, 0]] [[4, 1]] [[2, 2], [2, 2], [2, 2]] [[1, 3], [1, 3], [2, 3]] [[10, 4], [10, 4]]
[[1, 0]] [[4, 1]] [[2, 2], [2, 2], [2, 2]] [[1, 3], [1, 3]] [[10, 4], [10, 4]]
[[1, 0]] [[4, 1]] [[2, 2], [2, 2], [2, 2]] [[1, 3], [1, 3]] [[10, 4], [10, 4]]
[[1, 0]] [[4, 1]] [[2, 2], [2, 2], [2, 2]] [[1, 3], [1, 3]] [[10, 4], [10, 4]]
[[1, 0]] [[4, 1]] [[2, 2], [2, 2], [2, 2]] [[1, 3], [1, 3]] [[10, 4], [10, 4]]
[[1, 0]] [[4, 1]] [[2, 2], [2, 2], [2, 2]] [[1, 3], [4, 3]] [[10, 4], [10, 4]]
clique: 1
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1], [5, 1]] [[1, 2], [1, 2], [1, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1], [5, 1]] [[1, 2], [1, 2], [1, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1], [5, 1]] [[1, 2], [1, 2], [1, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1], [5, 1]] [[1, 2], [1, 2], [1, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1], [5, 1]] [[1, 2], [1, 2], [1, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1], [5, 1]] [[1, 2], [1, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1], [5, 1]] [[1, 2], [1, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1], [5, 1]] [[1, 2], [1, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1], [5, 1]] [[1, 2], [1, 2]] [[3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1], [5, 1]] [[1, 2], [1, 2]] [[3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1], [5, 1]] [[1, 2], [1, 2]] [[3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1], [5, 1]] [[1, 2], [1, 2]] [[3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1], [5, 1]] [[1, 2], [1, 2]] [[3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1], [5, 1]] [[1, 2], [1, 2]] [[3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1], [5, 1]] [[1, 2], [1, 2]] [[3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1], [5, 1]] [[1, 2], [1, 2]] [[3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1], [5, 1]] [[1, 2], [1, 2]] [[3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1], [5, 1]] [[1, 2], [1, 2]] [[3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1], [5, 1]] [[1, 2], [1, 2]] [[3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1], [5, 1]] [[1, 2], [1, 2]] [[3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1]] [[1, 2], [1, 2]] [[3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1]] [[1, 2], [1, 2]] [[3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1]] [[1, 2], [1, 2]] [[3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0], [6, 0]] [[3, 1], [3, 1]] [[1, 2], [1, 2]] [[3, 3], [3, 3]] [[10, 4]]
[[5, 0], [6, 0]] [[3, 1]] [[1, 2], [1, 2]] [[3, 3], [3, 3]] [[10, 4]]
clique: 2
[[8, 0], [8, 0], [8, 0]] [[4, 1]] [[1, 2]] [[4, 3], [4, 3], [4, 3]] [[10, 4]]
[[8, 0], [8, 0], [8, 0]] [[4, 1]] [[1, 2]] [[4, 3], [4, 3], [4, 3]] [[10, 4]]
[[8, 0], [8, 0], [8, 0]] [[4, 1]] [[1, 2]] [[4, 3], [4, 3], [4, 3]] [[10, 4]]
[[8, 0], [8, 0]] [[4, 1]] [[1, 2]] [[4, 3], [4, 3], [4, 3]] [[10, 4]]
[[8, 0], [8, 0]] [[4, 1]] [[8, 2]] [[4, 3], [4, 3], [4, 3]] [[10, 4]]
[[8, 0], [8, 0]] [[4, 1]] [[8, 2]] [[4, 3], [4, 3], [4, 3]] [[10, 4]]
[[8, 0], [8, 0]] [[4, 1]] [[8, 2]] [[4, 3], [4, 3], [4, 3]] [[10, 4]]
```

```
[[8, 0], [8, 0]] [[4, 1]] [[8, 2]] [[4, 3], [4, 3], [4, 3]] [[10, 4]]
[[8, 0], [8, 0]] [[4, 1]] [[8, 2]] [[4, 3], [4, 3], [4, 3]] [[10, 4]]
[[8, 0], [8, 0]] [[4, 1]] [[8, 2]] [[4, 3], [4, 3], [4, 3]] [[10, 4]]
[[8, 0], [8, 0]] [[4, 1]] [[8, 2]] [[4, 3], [4, 3], [4, 3]] [[10, 4]]
[[8, 0], [8, 0]] [[4, 1]] [[8, 2]] [[4, 3], [4, 3], [4, 3]] [[10, 4]]
[[8, 0], [8, 0]] [[4, 1]] [[8, 2]] [[4, 3], [4, 3], [4, 3]] [[10, 4]]
[[8, 0], [8, 0]] [[4, 1]] [[8, 2]] [[4, 3], [4, 3], [4, 3]] [[10, 4]]
[[8, 0], [8, 0]] [[4, 1]] [[8, 2]] [[4, 3], [4, 3]] [[10, 4]]
clique: 3
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [7, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4], [10, 4]]
[[9, 0], [7, 0]] [[9, 1], [9, 1], [7, 1]] [[9, 2], [9, 2], [7, 2]] [[10, 3]] [[10, 4], [10, 4], [10, 4]]
```

Paeckchen7

```
gepackte Kleidungsstücke:      569.0
tatsächliche Kleidungsstücke:  700
Übrige Kleidungsstücke:  131.0
Clique:  {8, 1, 2, 10}
Clique:  {1, 2, 3, 4, 10}
Clique:  {1, 10, 5, 6}
Clique:  {9, 10, 7}
clique: 0
clique: 1
[[1, 0]] [[3, 1], [3, 1], [3, 1]] [[2, 2], [2, 2], [2, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[1, 0]] [[3, 1], [3, 1], [3, 1]] [[2, 2], [2, 2], [2, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[1, 0]] [[3, 1], [3, 1], [3, 1]] [[2, 2], [2, 2], [2, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[1, 0]] [[3, 1], [3, 1], [3, 1]] [[2, 2], [2, 2], [2, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[1, 0]] [[3, 1], [3, 1], [3, 1]] [[2, 2], [2, 2], [2, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[1, 0]] [[3, 1], [3, 1], [3, 1]] [[2, 2], [2, 2], [2, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[1, 0]] [[3, 1], [3, 1], [3, 1]] [[2, 2], [2, 2], [2, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[1, 0]] [[3, 1], [3, 1], [3, 1]] [[2, 2], [2, 2], [2, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[1, 0]] [[3, 1], [3, 1], [3, 1]] [[2, 2], [2, 2], [2, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[1, 0]] [[3, 1], [3, 1], [3, 1]] [[2, 2], [2, 2], [2, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[1, 0]] [[3, 1], [3, 1], [3, 1]] [[2, 2], [2, 2], [2, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[1, 0]] [[3, 1], [3, 1], [3, 1]] [[2, 2], [2, 2], [2, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[1, 0]] [[3, 1], [3, 1], [3, 1]] [[2, 2], [2, 2], [2, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[1, 0]] [[3, 1], [3, 1], [3, 1]] [[2, 2], [2, 2], [2, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[1, 0]] [[3, 1], [3, 1], [3, 1]] [[2, 2], [2, 2], [2, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[1, 0]] [[3, 1], [3, 1], [4, 1]] [[2, 2], [2, 2], [2, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
[[1, 0]] [[3, 1], [3, 1], [4, 1]] [[2, 2], [2, 2], [2, 2]] [[3, 3], [3, 3], [3, 3]] [[10, 4]]
clique: 2
[[5, 0], [6, 0], [6, 0]] [[5, 1]] [[1, 2], [1, 2], [1, 2]] [[1, 3], [1, 3]] [[10, 4], [10, 4], [10, 4]]
[[5, 0], [6, 0], [6, 0]] [[5, 1]] [[1, 2], [1, 2], [1, 2]] [[1, 3], [1, 3]] [[10, 4], [10, 4], [10, 4]]
[[5, 0], [6, 0], [6, 0]] [[5, 1]] [[1, 2], [1, 2], [1, 2]] [[1, 3], [1, 3]] [[10, 4], [10, 4], [10, 4]]
[[5, 0], [6, 0], [6, 0]] [[5, 1]] [[1, 2], [1, 2], [1, 2]] [[1, 3], [1, 3]] [[10, 4], [10, 4], [10, 4]]
[[5, 0], [6, 0], [6, 0]] [[5, 1]] [[1, 2], [1, 2], [1, 2]] [[1, 3], [1, 3]] [[10, 4], [10, 4], [10, 4]]
[[5, 0], [6, 0], [6, 0]] [[5, 1]] [[1, 2], [1, 2], [1, 2]] [[1, 3], [1, 3]] [[10, 4], [10, 4], [10, 4]]
[[5, 0], [6, 0], [6, 0]] [[5, 1]] [[1, 2], [1, 2], [1, 2]] [[1, 3], [1, 3]] [[10, 4], [10, 4], [10, 4]]
[[5, 0], [6, 0], [6, 0]] [[5, 1]] [[1, 2], [1, 2], [1, 2]] [[1, 3], [1, 3]] [[10, 4], [10, 4], [10, 4]]
```

```
[[5, 0], [6, 0], [6, 0]] [[5, 1]] [[1, 2], [1, 2], [1, 2]] [[1, 3], [1, 3]] [[10, 4], [10, 4], [10, 4]]
[[5, 0], [6, 0], [6, 0]] [[5, 1]] [[1, 2], [1, 2], [1, 2]] [[1, 3], [1, 3]] [[10, 4], [10, 4], [10, 4]]
[[5, 0], [6, 0], [6, 0]] [[5, 1]] [[1, 2], [1, 2], [1, 2]] [[1, 3], [1, 3]] [[10, 4], [10, 4], [10, 4]]
[[5, 0], [6, 0], [6, 0]] [[5, 1]] [[1, 2], [1, 2], [1, 2]] [[1, 3], [1, 3]] [[10, 4], [10, 4], [10, 4]]
[[5, 0], [6, 0], [6, 0]] [[5, 1]] [[1, 2], [1, 2], [1, 2]] [[1, 3], [1, 3]] [[10, 4], [10, 4], [10, 4]]
[[5, 0], [6, 0], [6, 0]] [[5, 1]] [[1, 2], [1, 2], [1, 2]] [[1, 3]] [[10, 4], [10, 4], [10, 4]]
[[5, 0], [6, 0], [6, 0]] [[5, 1]] [[1, 2], [1, 2], [1, 2]] [[1, 3]] [[10, 4], [10, 4], [10, 4]]
[[5, 0], [6, 0], [6, 0]] [[5, 1]] [[1, 2], [1, 2], [1, 2]] [[1, 3]] [[10, 4], [10, 4], [10, 4]]
[[5, 0], [6, 0], [6, 0]] [[5, 1]] [[1, 2], [1, 2], [1, 2]] [[1, 3]] [[10, 4], [10, 4], [10, 4]]
[[5, 0], [6, 0], [6, 0]] [[5, 1]] [[1, 2], [1, 2], [1, 2]] [[1, 3]] [[10, 4], [10, 4], [10, 4]]
[[5, 0], [6, 0], [6, 0]] [[5, 1]] [[1, 2], [1, 2], [1, 2]] [[1, 3]] [[10, 4], [10, 4], [10, 4]]
[[5, 0], [6, 0]] [[5, 1]] [[1, 2], [1, 2]] [[1, 3]] [[10, 4], [10, 4], [10, 4]]
clique: 3
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4]]
[[9, 0], [7, 0], [7, 0]] [[9, 1], [9, 1]] [[9, 2], [9, 2], [9, 2]] [[10, 3]] [[10, 4], [10, 4]]
[[9, 0], [7, 0]] [[9, 1], [9, 1]] [[9, 2], [9, 2]] [[10, 3]] [[10, 4]]
```

6. Weitere Beispiele

Im folgenden Absatz werden weitere Beispiele aufgezeigt, welche mithilfe des Generators erstellt wurden. Da die Wirksamkeit des Programms bereits oben aufgezeigt wurde, interessieren uns nun hauptsächlich die Laufzeiten in Abhängigkeit zu den eingestellten Parametern. Mit dem Generatorprogramm wird dabei immer zunächst eine .txt-Datei erzeugt, welche anschließend in dem Solver eingelesen wird. Die Beispiele sind im Folgenden mit ihrer Laufzeit und den Parametergrößen in einer Tabelle aufgeführt. Dabei bedeuten die Graphen- und Kleidungsdichten jeweils, in wievielen möglichen Kanten- und Kleidungsvariablen tatsächlich einer genutzt wird. Eine Graphendichte von 2 bedeutet also, dass jede zweite mögliche Kante auch wirklich eine Kante darstellt. Man erkennt, dass ein starker Anstieg in der Laufzeit zur Berechnung von Kleidungstabellen mit vor allem sehr vielen Stilen stattfindet, womit unsere Vermutung aus Abschnitt 1.2.3 bestätigt werden kann. Gleichzeitig ist die Laufzeit schwächer abhängig von der Anzahl der Sorten sowie der Anzahl der Kleidungsstücke pro Sorte, was sich dadurch erklären lässt, dass die Anzahl der Sorten porpotional zu der Anzahl der Variablen ist, während die Laufzeit gar nicht mit der Anzahl der Kleidungsstücke zusammenhängt. Die leicht steigende Laufzeit bei sehr vielen Kleidungsstücken lässt sich daher in erster Linie durch die Zeit, die nötig ist, um die Päclchen auszugeben, erklären, welche jedoch keinen signifikanten Einfluss auf die Programmlaufzeit besitzt.

Laufende Nummer	Sorten	Stile	Graphen dichte	Min-kleider	Max-kleider	Kleider-dichte	Nicht gepackte Kleidung	Laufzeit (sekunden)
1	10	10	2	10	50	2	0	0.30964
2	50	10	2	10	50	2	0	3.66771
3	90	10	2	10	50	2	0	17.01055
4	130	10	2	10	50	2	0	26.54349
5	10	50	2	10	50	2	0	13.37247
6	10	90	2	10	50	2	0	296.80123
7	10	130	2	10	50	2	0	1547.69819
8	10	10	2	450	500	2	0	0.41574
9	10	10	2	1950	2000	2	0	1.867796
10	10	10	2	1450	1500	2	0	3.487898

7. Quellcode

Im Folgenden drucken wir sowohl den Hauptalgorithmus als auch das Generationsprogramm ab. Diese sind auch im Abgabeordner enthalten und können daher dort entsprechend eingesehen werden.

7.1 Hauptcode

```
#linear programming ansatz.

import pulp as pl
import time#zum Messen der Laufzeit

str = input("bitte eingeben, welche .txt Datei im selben Ordner geöffnet
werden soll: ")

file = open(str, "r")#wir öffnen die .txt datei uns lesen diese aus
content = file.readlines()
file.close()

time_start = time.time()

s,r = [eval(i) for i in content[0].split()]

passend = []
stopp_zeile = 0
for i in content:
    if i == "\n":
        stopp_zeile = content.index(i)
for i in range(1, stopp_zeile):
    passend.append([eval(j) for j in content[i].split()])
```

```
##wir erstellen ein dictionary.

stile_graph = {}

#datensruktur: name_des_knotens: [anzahl der kleidungsstücke]
for i in range(r):
    stile_graph[i+1] =[[],[0 for _ in range(s)]]

# einfügen der benachbarte knoten
for a, b in passend:
    stile_graph[a][0].append(b)
    stile_graph[b][0].append(a)#knoten sind immer gegenseitig

#einfügen der Kleidungsstücke
for i in content[stopp_zeile+1:]:
    sorte_i, stil_i, anzahl_i = [eval(j) for j in i.split()]
    stile_graph[stil_i][1][sorte_i-1] = anzahl_i

#wir geben den Graphen aus
print(stile_graph)

###################################
##TEIL FINDEN MAXIMALER CLIQUEN##
###################################

#Funktion: Bron-Kerbosch.
#Parameter: Graph in Dictionary-Form
#Rückgabe: Liste von maximalen Cliquen
#der Bron-Kerbosch-Algorithmus ist ein rekursiver Algorithmus,
#der alle maximalen Cliquen in einem Graphen findet.
#wir benutzen hierbei eine Version ohne pivoting, welche sich trotzdem als
sehr effizient erweist.
def bron_kerbosch(graph):
    def bron_kerbosch_rekursiv(R, P, X):#dies ist die eigentliche
Rekursionfunktion
        if not P and not X:
            cliques.append(R)
            return
        for v in list(P):
            bron_kerbosch_rekursiv(R.union({v}),
P.intersection(graph[v][0]), X.intersection(graph[v][0]))
            P.remove(v)
            X.add(v)

    cliques = []
```

```
    knoten = set(graph.keys())
    bron_kerbosch_rekursiv(set(), knoten, set())
    return cliques

#Funktion: Finde einfache Knoten
#Parameter: Liste von Cliquen
#Rückgabe: Liste von Knoten, die in genau einer Clique vorkommen
#Diese Funktion benötigen wir, um Variablen für die Lineare optimierung
erstellen zu können.
#Dadurch gehen wir jeden Knoten/stil einmal durch und tragen Knoten,
welche nur teil einer
#einzigen Clique sind, in eine Liste ein.
def finde_einfache_knoten(cliquen):
    knoten = []
    doppelte_knoten = []
    for clique in cliquen:
        for k in clique:
            if k not in knoten:
                knoten.append(k)
            elif k not in doppelte_knoten:
                doppelte_knoten.append(k)
            else:
                pass
    return list(set(knoten).difference(set(doppelte_knoten))),
doppelte_knoten

#Ausführen des Bron-Kerbosch-Algorithmus
cliquen = bron_kerbosch(stile_graph)
print("Cliquen: ", cliquen)#ausgeben der Cliquen

#Ausführen der Funktion finde_einfache_knoten und erstellen einer
zugehörigen liste
einfache_knoten, doppelte_knoten = finde_einfache_knoten(cliquen)
print("Einfache Knoten in Cliquen:", einfache_knoten)#ausgeben der
einfachen Knoten

############################
##TEIL LINEARE OPTIMIERUNG##
############################

#Funktion: Summe der sorten
#Argumente: Clique, doppelt
#Rückgabe: Liste von Summen der Kleidungsstücke
#Diese Funktion checkt ob eine Kombination funktionieren kann oder nicht.
```

```
#Eine Kombination funktioneirt dann NICHT, wenn max(Summe der knoten, die
einem Clique angehören)
#3 mal größer ist als min (summer aller knoten der cliquen für jeden
knoten.)
#Dadurch ist die Funktion sehr wichtig, wenn wir die Legitimität
bestimmter Cliquensummen überprüfen wollen
def summe_der_sorten(clique, doppelt = True):
    summe = [0 for _ in range(s)]
    if doppelt:
        for k in clique:#k steht für knoten
            summe = [sum(x) for x in zip(stile_graph[k][1], summe)]##hmm
gucken ob das optimale geschwindigkeit ist?
        return summe
    else:
        for k in clique:
            if k in einfache_knoten:
                summe = [sum(x) for x in zip(stile_graph[k][1],
summe)]##hmm gucken ob das optimale geschwindigkeit ist?
        return summe

#wir ermitteln zunächst die maximale Summe der Kleidungsstücke in Cliquen
ohne Doppelknoten für die obere Schranke
summen_ohne_doppelknoten = []
for clique in cliquen:
    summen_ohne_doppelknoten.append(summe_der_sorten(clique, doppelt =
False))
print("summen ohne Doppelknoten:")
print(summen_ohne_doppelknoten)

#erstelle Problem
problem = pl.LpProblem("StilvollePackungen", pl.LpMaximize)

#erstelle Entscheidungsvariablen
cliquenvariablen = []
for i in range(len(cliquen)):
    stile = []
    for j in range(s):
        stile.append(pl.LpVariable(f"clique{i}_sorte_{j}", lowBound = 0,
cat = 'Integer'))
    cliquenvariablen.append(stile)
print("Cliquenvariablen: ", cliquenvariablen)

doppelknotenvariablen = []
for i, clique in enumerate(cliquen):
    c = []#vorläufige liste aller variablen der Doppelknoten der Clique
```

```
    for count, k in enumerate(doppelte_knoten):
        if k in clique:
            stile = []
            for j in range(s):
                stile.append(pl.LpVariable(f"doppelknoten_clique{i}_knoten
{k}_sorte_{j}", lowBound = 0, cat = 'Integer'))
            c.append(stile)
        else:
            c.append(None)
    doppelknotenvariablen.append(c)
print("Doppelknotenvariablen: ", doppelknotenvariablen)

#maximierungsgleichung hinzufügen:
gleichung = None
for clique in doppelknotenvariablen:
    for knoten in clique:
        if knoten:
            for var in knoten:
                gleichung += var
for clique in cliquenvariablen:
    for var in clique:
        gleichung += var
problem += gleichung

#ungleichungen hinzufügen
for count, clique in enumerate(cliquen):
    #die summe darf die summe ohne doppelte knoten nicht übersteigen.
    for sorte in range(s):
        problem += cliquenvariablen[count][sorte] <=
summen_ohne_doppelknoten[count][sorte]

for count, k in enumerate(doppelte_knoten):
    #die summe der doppelten knoten muss immer zueinander passen.
    for sorte in range(s):
        linke_seite = None
        for clique, cliquenknoten in
enumerate(doppelknotenvariablen):#fügt alle doppelknoten innerhalb einer
Clique
            if cliquenknoten[count]:
                linke_seite += cliquenknoten[count][sorte]
        problem += linke_seite <= stile_graph[k][1][sorte]

for count, clique in enumerate(cliquen):#die summe der gepackten variablen
dürfen nicht größer als die dreifache Summe der korrespondierenden Summen
der anderen Variablen sein
```

```
    for stila in range(s):
        for stilb in range(s):
            if stila == stilb:
                continue#wir vermeiden selbstschleifen, dort ist die zu
zeigende Ungleichugn trivial
            linke_seite = None
            linke_seite += cliquenvariablen[count][stila]
            linke_seite -= 3*cliquenvariablen[count][stilb]
            for k in clique:
                if k in doppelte_knoten and
doppelknotenvariablen[count][doppelte_knoten.index(k)]:
                    linke_seite +=
doppelknotenvariablen[count][doppelte_knoten.index(k)][stila]
                    linke_seite -=
3*doppelknotenvariablen[count][doppelte_knoten.index(k)][stilb]
            problem += linke_seite <= 0

#löse das Problem
problem.solve()

summe_der_kleidung = 0

#ausgabe aller relevanten Variablen
print("Status:", pl.LpStatus[problem.status])
print("Doppelknotenvariablen:")
for clique in doppelknotenvariablen:
    for knoten in clique:
        if knoten:
            for var in knoten:
                print(f"{var.name}: {var.value()}")
                summe_der_kleidung += var.value()

print("cliquenvariablen:")
for clique in cliquenvariablen:
    for var in clique:
        print(f"{var.name}: {var.value()}")
        summe_der_kleidung += var.value()


#Wir summieren die gepackten und tatsächlichen Kleidungsstücke zusammen
und gucken,
#wieviele Kleidungsstücke noch übrigbleiben
print("gepackte Kleidungsstücke:     ", summe_der_kleidung)
summe_total = 0
for i in stile_graph:
```

```
    for j in stile_graph[i][1]:
        summe_total += j
print("tatsächliche Kleidungsstücke: ", summe_total)
print("Übrige Kleidungsstücke: ", summe_total-summe_der_kleidung)

#wir fangen an die Boxen zu erstellen und erstellen dazu erst die
Boxenverteilungen.
#die Bosenverteilung gibt dabei an, wieviele Kleidungsstücke welcher Sorte
in einer Clique
#gepackt werden muss.
boxenverteilung = []
for count in range(len(cliquen)):
    boxenverteilung.append([int(cliquenvariablen[count][i].value()+sum([do
ppelknotenvariablen[count][j][i].value() for j in
range(len(doppelte_knoten)) if doppelknotenvariablen[count][j]])) for i in
range(s)])

##############################################
##TEIL UMWANDLUNG DER LÖSUNGEN IN PÄCKCHEN##
##############################################

#FUNKTION: Printboxen
#Parameter: Keine
#Rückgabe: Keine
#Gibt die zu packenden Boxen anhand der Boxenverteilungen aus.
def printboxen():
    #tatsächliche Illustration der Boxen:
    packungen = []
    for count, clique in enumerate(cliquen):#wir erstellen eine
Packungsliste für jede Clique einzeln und geben diese auch Cliquenweise
aus
        packung_clique = []

        boxen_anzahl = min(boxenverteilung[count])
        min_index = boxenverteilung[count].index(boxen_anzahl)

        for i in range(boxen_anzahl):
            packung_clique.append([[] for _ in range(s)])#erstellt leere
Boxen boxen die gepackt werden müssen

        #nimmt aus den ganzen doppelten knoten die zugeteilten Kleider und
packt diese in die Boxen
        for sorte in range(s):
            box_counter = 0
            zu_packende_kleidung = 0
```

```
            for k in range(len(doppelte_knoten)):#für alle doppelten
knoten
                if doppelknotenvariablen[count][k]:
                    zu_packende_kleidung +=
doppelknotenvariablen[count][k][sorte].value()
            for k, knoten in enumerate(doppelte_knoten):#für alle
doppelten knoten
                if zu_packende_kleidung == 0:
                    break
                if doppelknotenvariablen[count][k]:
                    for i in
range(int(doppelknotenvariablen[count][k][sorte].value())):#für jede zu
packende kleidung:
                        zu_packende_kleidung -= 1
                        stile_graph[knoten][1][sorte] -= 1
                        packung_clique[box_counter%boxen_anzahl][sorte].ap
pend([knoten, sorte])
                        box_counter += 1

        #füllt in jede box 1 im minimum index (jetzt für auf jede sorte
erweitern und
        #wir sind fertig)
        for sorte in range(s):
            for k in
list(set(clique).difference(set(doppelte_knoten))):#füllt erst mal ein
Kleidungsstück pro clique auf.
                for i in range(boxen_anzahl):
                    if packung_clique[i][sorte] == []:#die sorte ist noch
leer! packe ein Kleidungsstück rein.
                        stile_graph[k][1][sorte] -= 1
                        packung_clique[i][sorte].append([k, sorte])#packt
das kleidungsstück in eine richtige box
                        boxenverteilung[count][sorte] -= 1#nimmt eins weg
von den zu packenden kleidungsstücken
                    else:
                        pass#fertig, wir müssen nichts machen

            box_counter = 0
            for knoten in
list(set(clique).difference(set(doppelte_knoten))):#füllt alle
kleidungsstücke auf
                if boxenverteilung[count][sorte] == 0:#die sorte ist
fertig!
                    break
```

```
                while stile_graph[knoten][1][sorte] > 0:#die sorte ist
fertig!
                    if boxenverteilung[count][sorte] == 0:
                        break
                    stile_graph[knoten][1][sorte] -= 1
                    packung_clique[box_counter%boxen_anzahl][sorte].append
([knoten, sorte])#packt das kleidungsstück in eine richtige box
                    boxenverteilung[count][sorte] -= 1#nimmt eins weg von
den zu packenden kleidungsstücken
                    box_counter += 1
        packungen.append(packung_clique)

    #ausgabe aller Packungen, die gepackt wurden
    o=0
    for item in packungen:
        print("clique:", o)
        o += 1
        for i in item:
            print(*i)

#ausführen der printboxen funktion
printboxen()

#Zeit stoppen, laufzeit berechnen und ausgeben
time_end = time.time()
print("Time: ", time_end-time_start)#Ausgeben der Laufzeit
```

7.2 Boxen-Generator

```
#Aufgabengenerator für Aufgabe 2.
import random

#Eingeben aller relevanter Parameter
sorten = int(input("bitte anzahl Sorten eingeben: "))
stile = int(input("bitte anzahl Stile eingeben: "))
graphendichte = int(input("bitte Graphendichte eingeben (jede nte Zahl hat
eine Verbindung): "))

maxkleider = int(input("bitte maxkleider eingeben: "))
minkleider = int(input("bitte minkleider eingeben: "))
kleidungsdichte = int(input("bitte Kleidungsdichte eingeben (JEde Nte
zelle hat Kleidungsstücke): "))

#wir öffnen das beigefügte .txt Dokument (am Anfang noch leer)
file = open("paeckchengen.txt", "w")
```

```
file.write(f"{sorten} {stile}\n")

#erstelle zufällige Stile, die zusammenpassen
for i in range(stile):
    for j in range(stile):
        if i>=j:
            continue
        if not random.randint(0,graphendichte):
            file.write(f"{i+1} {j+1}\n")
file.write("\n")

#erstelle zufällige Kleidungsstücke
for i in range(sorten):
    for j in range(stile):
        if not random.randint(0,graphendichte):
            file.write(f"{i+1} {j+1} {random.randint(minkleider,
maxkleider)}\n")

#fertig!
file.close()
print("die Ausgabe wurde unter 'paeckchengen.txt' gespeichert.")
```