Aufgabe 2: Stilvolle Päckchen

Teilnahme-ID: 71358

Bearbeiter/-in dieser Aufgabe:
Florian Werth

15. April 2024

Inhaltsverzeichnis

1 Lösungsidee **2**

2 Umsetzung **5**
2.1 Boxen erstellen . . . 6
2.2 Bron-Kerbosch Algorithmus . . . 7

3 Laufzeitanalyse **7**

4 Beispiele **9**
4.1 paeckchen0.txt . . . 9
4.2 paeckchen1.txt . . . 10
4.3 paeckchen2.txt . . . 11
4.4 paeckchen3.txt . . . 12
4.5 paeckchen4.txt . . . 13
4.6 paeckchen5.txt . . . 14
4.7 paeckchen6.txt . . . 14
4.8 paeckchen7.txt . . . 15

5 Quellcode **16**
5.1 Klasse Graph . . . 16
5.2 Klasse Clique . . . 16
5.3 Funktion add_first_constraints . . . 18
5.4 Funktion get_n_boxes . . . 18
5.5 main.py . . . 18

1 Lösungsidee

Die Stilrichtungen lassen sich als Knoten in einem Graphen darstellen. Die Kompatibilität zweier Stilrichtungen stellt eine Kante dar. Man versetzt sich in die Lage einer Stilrichtung und fragt sich: "Mit welchen Stilrichtungen kann ich zusammen in Outfitboxen vorkommen?" Die Antwort ist mit Stilrichtungen, die untereinander alle kompatibel sind. Das entspricht in der Graphentheorie einer Clique. Es kann mehrere Cliquen geben zu denen eine Stilrichtung gehört. Als Kleidungsstück einer Stilrichtung fragt man sich dann: "Zu welcher Clique möchte ich gehören?" Um dem armen Kleidungsstück die Wahl zu erleichtern, betrachten wir nur Cliquen, die mit keinen anderen Stilrichtungen mehr erweiterbar sind, sogenannte maximale Cliquen. Das kann man machen, weil jede Clique ein Subgraph einer maximalen Clique ist und somit keine Information verloren gehen darüber, wie man Stilrichtungen zusammensetzen kann.

Wie vorhin bereits erwähnt, ein Kleidungsstück fragt sich, in welche Clique es gehen soll. Anders ausgedrückt suchen wir eine Zuweisung von Kleidungsstücken zu Cliquen, sodass die Anzahl an zugewiesenen Kleidungsstücken maximiert wird. Sei s die Anzahl an Sorten und $(x_1, \ldots, x_s)$ die einer Clique zugewiesene Anzahl an Kleidungsstücken pro Sorte. Damit eine Zuweisung gültig ist, muss für jede Clique gelten: $\max(x_1 \ldots x_s) \leq 3 \cdot \min(x_1 \ldots x_s)$. Das entspricht der Regel, dass von jeder Sorte mindestens ein, jedoch nicht mehr als drei Kleidungsstücke eingepackt werden sollen. Das gilt, weil jede Box mindestens ein Kleidungsstück jeder Sorte benötigt. Daraus folgt, dass wir maximal $\min(x_1 \ldots x_s)$ Boxen erstellen können. Pro Box können insgesamt drei Kleidungsstücke gepackt werden. Mit $\min(x_1 \ldots x_s)$ Boxen können wir also maximal $3 \cdot \min(x_1 \ldots x_s)$ Kleidungsstücke von jeder Sorte packen und deshalb gilt die obige Ungleichung.

Die Zuweisung von Kleidungsstücken zu Cliquen lässt sich als ganzzahliges Optimierungsproblem formulieren. Jede Clique erhält für jede Art von Kleidungsstück (Sorte, Stilrichtung), das man ihr zuweisen könnte eine nicht negative Ganzzahlvariable, die beschreibt wie viele Stücken dieser Art man der Clique zuweist.

Die Summe aller Variablen gleicher Art (Sorte, Stilrichtung) muss kleiner gleich der gegebenen Anzahl sein, damit nicht mehr zugewiesen wird als verfügbar ist.

Innerhalb jeder Clique werden Variablen gleicher Sorte zusammenaddiert. Wir erhalten wieder $(x_1 \ldots x_s)$. Die Ungleichung $\max(x_1 \ldots x_s) \leq 3 \cdot \min(x_1 \ldots x_s)$ lässt sich auf folgende Weisen implementieren. Entweder man sagt für jedes x_i, dass es kleiner gleich dreimal allen anderen x_j, $(1 \leq i, j \leq s, i \neq j)$ sein soll. Bei z.B. drei Sorten (x_1, x_2, x_3) ergibt das die Ungleichungen:

$$x_1 \leq 3 \cdot x_2$$
$$x_1 \leq 3 \cdot x_3$$
$$x_2 \leq 3 \cdot x_1$$
$$x_2 \leq 3 \cdot x_3$$
$$x_3 \leq 3 \cdot x_1$$
$$x_3 \leq 3 \cdot x_2$$

Man erhält im Allgemeinen $s^2 - s$ Ungleichungen für jede Clique oder man führt eine Variable min für das Minimum und max für das Maximum von (x_1, x_2, x_3) ein. Für min sagt man, dass es kleiner gleich allen Sorten sein muss:

$$min \leq x_1$$
$$min \leq x_2$$
$$min \leq x_3$$

und für max macht man das Gegenteil davon:

$$max \geq x_1$$
$$max \geq x_2$$
$$max \geq x_3$$

zum Schluss sagt man:

$$max \leq 3 \cdot min$$

Hierfür benötigt man nur $2 \cdot s + 1$ Ungleichungen.

Durch Ausprobieren habe ich herausgefunden, dass wenn die Ganzzahligkeitsbedingung für die Zuweisungsvariablen entfernt wird und die min/max Variablen jedoch ganzzahlig sein müssen, die entstehende Lösung trotzdem ganzzahlig ist. Ich bin mir da zwar nicht sicher, doch ich vermute folgendes: Ein ganzzahliges lineares Optimierungsproblem kann man auch in Matrixform aufschreiben. Ist die Matrix A total unimodular und der Vektor b integral, erfüllt die Lösung der LP-Relaxation auch die Ganzzahligkeitsbedingung. Im vorhandenen Problem ist das nicht der Fall, weil die Lösung der LP-Relaxation nicht ganzzahlig wird. Man betrachte das Optimierungsproblem nur mit den Ungleichungen, dass die gegebene Anzahl nicht überschritten werden soll. Die Matrix A hat in jeder Spalte nur genau einen 1 Eintrag ansonsten 0, weil jede Variable genau einmal mit Koeffizent 1 in einer Ungleichung vorkommt. Diese Matrix ist wahrscheinlich total unimodular. Der Vektor b ist ganzzahlig, weil die gegebenen Anzahlen auch ganzzahlig sind. Ich denke daher, dass im originalen Problem die teilweise LP-Relaxation zu einer ganzzahligen Lösung führt, weil die Submatrix total unimodular und der Subvektor integral ist. Da ich das alles nicht bewiesen habe, werden alle im Programm generierten Lösungen auf Ganzzahligkeit hin überprüft. Falls das immer stimmen sollte, wird aus dem ganzzahligen Optimierungsproblem ein gemischt ganzzahliges Optimierungsproblem mit viel weniger Ganzzahlvariablen, was sich in der Laufzeitanalyse positiv bemerkbar machen wird.

Die formulierten Optimierungsprobleme werden durch einen externen Solver gelöst. Die dadurch erhaltene Lösung ist (hoffentlich) optimal.

Für das Auflisten aller maximalen Cliquen wird der Bron-Kerbosch Algorithmus verwendet.

Wie aus den zugewiesenen Kleidungsstücken letztendlich die Boxen erstellt werden steht in der Umsetzung. Für die Anzahl an Boxen die eine Clique mit $(x_1 \dots x_s)$ bereitstellen kann gilt:

$$\text{minimale Anzahl an Boxen} = \lceil \frac{\max(x_1 \dots x_s)}{3} \rceil \tag{1}$$

$$\text{maximale Anzahl an Boxen} = \min(x_1 \dots x_s) \tag{2}$$

Gleichung (1) funktioniert, weil von jeder Sorte können höchstens drei Kleidungsstücke in eine Box aufgenommen werden. Teilt man $\max(x_1 \dots x_s)$ durch 3 weiß man also wie viele Boxen man mindestens benötigt, um alles unterzubringen. Man rundet auf, weil es nur ganze Boxen gibt. Die Gleichung (2) wurde bereits kurz erklärt.

Daraus folgt, dass man aus denen einer Clique zugewiesenen Kleidungsstücken, k Boxen erstellen kann. (minimale Anzahl an Boxen $\leq k \leq$ maximale Anzahl an Boxen)

Mit diesen Informationen kann man eine Erweiterung implementieren, bei der neben der Maximierung der benutzen Kleidungsstücke auch eine bestimmte Anzahl z.B. n Boxen gefordert wird. Wir fügen die folgenden Ungleichungen hinzu:

$$n \geq \sum^{\text{Von allen Cliquen}} \text{minimale Anzahl an Boxen} \tag{3}$$

$$n \leq \sum^{\text{Von allen Cliquen}} \text{maximale Anzahl an Boxen} \tag{4}$$

Für die Bedingungen (3) und (4) benutzen wir die min/max Variablen der Cliquen. Die min/max Variablen müssen nicht unbedingt den echten Werten für das jeweilige Minimum oder Maximum entsprechen. Sie werden sich diesen jedoch so weit wie nötig annähern, um die Bedingungen zu erfüllen, sowie um eine

optimale Lösung zu erhalten. Die untere Grenze minimale Anzahl an Boxen wird durch max beschreiben und die obere Grenze maximale Anzahl an Boxen durch min.

Zum Schluss, falls eine Lösung gefunden wurde und die n Boxen bereitgestellt werden sollen, muss man wissen wie viele Boxen jede Clique bereitstellen soll. Man berechnet die Differenz d.

$$d = n - \sum^{\text{Von allen Cliquen}} \text{(minimale Anzahl an Boxen)} \tag{5}$$

Die Differenz ist die Anzahl an Boxen, die extra zur minimalen Anzahl erstellt werden soll. Eine Clique erstellt so viele Boxen wie möglich extra, sodass $d \geq 0$ bleibt und zieht die Anzahl an extra genommenen Boxen von d ab. Konkret gilt für eine Clique dann: $\textit{spielraum} = \textit{maximale Anzahl an Boxen} - \textit{minimale Anzahl an Boxen}$. Die Anzahl an extra zu nehmenden Boxen ist $\min(d, \textit{spielraum})$.

2 Umsetzung

Die Lösung wurde in Python3 geschrieben. Die Eingabedateien werden eingelesen. Die Kleidungsstücke mit deren Anzahl werden in einer 2-dimensionalen Liste Inventar nach Sorte und Stilrichtung gespeichert. Mit der Klasse Graph wird der Graph aus Stilrichtungen und Kompatibilitäten in einer Adjazenzliste gespeichert. Die Methode bron-kerbosch gibt die Einteilung von Stilrichtungen zu maximalen Cliquen zurück.

Graph
+adj_list: list[list[int]]
+Graph(amount_nodes:int) +add_edge(start:int,end:int) +bron_kerbosch(R:set,P:set,X:set): generator

Für ILP und MILP wird die Python Bibliothek PuLP 2.8 benutzt. Die Klasse Clique dient der Repräsentation einer maximalen Clique mit den ihr zugewiesenen Kleidungsstücken.

Clique
+amount_categories: int +amount_styles: int +useable_styles: list[int] +variables: list[list[LpVariable]] +id: int +min_variable: LpVariable +max_variable: LpVariable
+Clique(amount_categories:int,amount_styles:int, usable_styles:list[int],inventory:list[list[int]], variable_type:String,id:int,) +category_sum(type:int): LpAffineExpression +categories_total(): list[int] +add_second_constraint_quadratic(model:LpProblem) +add_second_constraint_linear(model:LpProblem) +get_boxes(amount:None): list[list[list[int]]] +is_integral(): bool

Die Variablen, die sagen wie viel Kleidungsstücke einer Art einer Gruppe zugewiesen werden, sind vom Typ LpVariable. Damit man gut auf diese Variablen zugreifen kann werden sie innerhalb einer 2-dimensionalen Liste gespeichert, sodass die Position innerhalb der Liste die Sorte und Stilrichtung angibt. In allen Positionen ohne wird einfach None gespeichert. Zur Initialisierung eines Clique Objekts bekommt es die eigenen Stilrichtungen übergeben. Für alle eigenen Stilrichtung wird für jede Sorte deren gegebene Anzahl im Inventar größer als 0 ist in der Liste eine zugehörige LpVariable erstellt. Der Parameter variable_type vom Konstruktor bestimmt, ob diese Variablen ganzzahlig oder kontinuierlich sind und was die min/max Variablen sind.

```
def add_first_constraint(model, groups, amount_categories, amount_styles, inventory):
```

Fügt die Ungleichungen (1) in einer dreifach verschachtelten for-Schleife hinzu. Für jede Art von Kleidungsstück (Sorte, Stilrichtung) wird die Summe aller LP-Variablen dieser Art gebildet über alle Cliquen gebildet. Als Ungleichung wird jeweils Summe $\leq$ Inventar[Sorte][Stilrichung] hinzugefügt.

```
def category_sum(self, category) -> LpAffineExpression:
```

Gibt die Summe aller Variablen gleicher Sorte innerhalb der Clique zurück.

```
def add_second_constraint_quadratic(self, model):
```

Fügt die Ungleichungen (2) mit $s^2 - s$ Ungleichungen für eine Clique hinzu. In einer zweifach verschachtelten for-Schleife wird für jede Sorte (beschränkte Sorte) für alle anderen Sorten die Ungleichung category_sum(beschränkte Sorte) $\leq 3 \cdot$ category_sum(sorte). Die Rückgabewerte von category_sum werden vorberechnet und wiederverwendet.

```
def add_second_constraint_linear(self, model):
```

Fügt die Ungleichungen (2) mit $2 \cdot s + 1$ Ungleichungen für eine Clique hinzu. In einer for-Schleife werden für alle Sorten die Ungleichungen min_variable $\leq$ category_sum(Sorte) und max_variable $\geq$ category_sum(Sorte) hinzugefügt. Zum Schluss wird die Ungleichung max_variable $\leq 3 \cdot$ min_variable hinzugefügt.

```
def add_n_boxes_constraint(model, n, cliques):
```

Fügt die Ungleichungen das n Boxen erstellt werden können hinzu. min_amount_boxes ist gleich der Summe aller max_variablen und max_amount_boxes ist gleich der Summe aller min_variablen. Die Ungleichungen $n \geq$ min_amount_boxes / 3 und $n \leq$ max_amount_boxes werden hinzugefügt

2.1 Boxen erstellen

```
def categories_total(self) -> list[int]:
```

Gibt eine Liste zurück, die für jede Sorte die Anzahl an Kleidungsstücken, die diese Clique besitzt, zurückgibt.

```
def get_boxes(self, amount=None) -> list[list[list[int]]]:
```

Stellt für eine Clique die Boxen zusammen und gibt sie zurück. Anfangs ist der Vektor Boxen leer und im Vektor categories_total ist die Anzahl an Kleidungsstücken pro Sorte zusammengefasst. Die Variable remaining_amount wird mit der Anzahl an zu erstellenden Boxen initialisiert. Die Anzahl ist min(categories_total) oder amount, wenn es nicht None ist. Solange remaining_amount größer als 0 ist, wird folgendes gemacht: Ein leerer Vektor Box wird erstellt, in dem die genommenen Kleidungsstücke als Vektor mit (Sorte, Stilrichtung, Anzahl) gespeichert werden. Für jede Sorte wird berechnet wie viel genommen werden soll und in der Variablen take gespeichert. Von jeder Sorte wird so viel wie möglich genommen, sodass danach noch remaining_amount - 1 Boxen erstellt werden können. Daher ist max(categories_total[Sorte] - (remaining_amount - 1), 3) die Anzahl an Kleidungsstücken, die je Sorte genommen wird. Für jede Stilrichtung wird dann folgendes gemacht solange take > 0: Wir nehmen so viel von self.variables[category][style] wie möglich und speichern es in took. took ist min(self.variables[category][style].varValue, take). Wir ziehen took von take, self.variables[Sorte][Stilrichtung] und categories_total[category] ab und fügen den Vektor (Sorte, Stilrichtung, took) dem Vektor Box hinzu. Nachdem über alle Sorten iteriert wurde, wird der Boxvektor dem Vektor an Boxen hinzugefügt. remaining_amount wird um eins verringert. Zum Schluss wird der Vektor Boxen zurückgeben.

```
def get_n_boxes(n, cliques):
```

Hat man mit der Funktion add_n_boxes_constraint sichergestellt, dass man n Boxen erstellen kann, stellt diese Funktion diese n Boxen zusammen und gibt sie zurück. Der Vektor Boxen ist leer. Zuerst wird für jede Clique die minimale und maximale Anzahl an Boxen berechnet und in jeweils einer Liste gespeichert. Danach wird d berechnet. Dann wird in einer for-Schleife durch die Cliquen iteriert. Für jede Clique wird Clique.get_boxes(amount=extra+minimale_anzahl_boxen) aufgerufen (So wie in der Lösungsidee beschrieben) und der Rückgabewert wird Boxen hinzugefügt. Von d wird extra abgezogen. Zum Schluss wird der Vektor Boxen zurückgegeben.

2.2 Bron-Kerbosch Algorithmus

Die Aufgabe alle maximalen Cliquen zu finden, könnte man versuchen mittels Brute-Force lösen. Sei r die Anzahl an Stilrichtungen. Man generiert alle $r^2 - 1$ Teilmengen des Graphen. Alle Teilmengen, die keine Clique sind, werden gelöscht und alle Cliquen, die ein Teil einer anderen Clique sind, werden gelöscht. Am Ende hat man dann alle maximalen Cliquen. Das geht besser mit dem Bron-Kerbosch Algorithmus, dessen Laufzeit jedoch trotzdem exponentiell ist. Der Algorithmus benutzt die Mengen R, P und X. Beim ersten Aufruf enthält P alle Knoten des Graphen und R und X sind leer. R repräsentiert die derzeitige Clique, die aufgebaut wird. P repräsentiert Knoten mit der die derzeitige Clique erweitert werden kann.

Der Pseudocode wurde der entsprechenden Wikipedia Seite [1] entnommen. Für die Wahl des Pivotelements aus $P \cup X$ wird der Knoten mit dem größten Grad gewählt.

```
algorithm BronKerbosch2(R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ⋃ X
    for each vertex v in P \ N(u) do
        BronKerbosch2(R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

3 Laufzeitanalyse

Sei s die Anzahl an Sorten, r die Anzahl an Stilrichtungen und e die Anzahl an Kompatibilitäten. Sagen wir der Einfachheit halber, dass wir m maximale Cliquen mit jeweils k Stilrichtung finden und pro Clique v Arten von zuweisbaren Kleidungsstücken haben ($k \leq v \leq k \cdot s$). Die Erweiterung mit n Boxen wird nicht betrachtet.

Im Folgenden sind die betrachteten Varianten:

ilp-quadratic	pro Clique $s^2 - s$ Ungleichungen
ilp-linear	pro Clique $2s + 1$ Ungleichungen
milp	(fast) wie ilp-linear, jedoch mit teilweiser LP-Relaxation

Den Graph der Stilrichtungen zu erstellen ist $\mathcal{O}(r + e)$.

Der Bron-Kerbosch Algorithmus besitzt eine worst-case Laufzeit von $\mathcal{O}(3^{r/3})$ [2].

Für die Laufzeitangaben konnte ich meistens einfach die Schleifen zählen.

Die Cliquen Objekte zu erstellen ist $\mathcal{O}(m \cdot (r \cdot s + k \cdot s))$.

Die Constraints (1) hinzuzufügen ist $\mathcal{O}(s \cdot r \cdot m)$

Die Constraints (2) hinzuzufügen ist bei ilp-quadratic $\mathcal{O}(m \cdot (s \cdot k + s^2))$

Die Constraints (2) hinzuzufügen ist bei ilp-linear und milp $\mathcal{O}(m \cdot (s \cdot k))$

Das Hinzufügen der Kostenfunktion ist $\mathcal{O}(m \cdot s \cdot r)$

[1] https://en.wikipedia.org/wiki/Bron-Kerbosch_algorithm
[2] siehe vorherige

bei (M)ILPs handelt es sich um NP-schwere Probleme. Die Laufzeit ist im schlechtesten Fall exponentiell. In der Praxis ist die Laufzeit jedoch für die Problemgröße (siehe Beispiele) gut. Grob gesagt ist die Laufzeit von der Struktur des Problems, der Anzahl und Art der Variablen und Ungleichungen abhängig. Ganzzahlvariablen haben einen höheren Einfluss als kontinuierliche Variablen. Dies sollte sich bei der praktischen Laufzeitanalyse zeigen.

	ganzzahlige Variablen	kontinuierliche Variablen	Ungleichungen
ilp-quadratic	$m \cdot v$	0	$m \cdot (s^2 - s)$
ilp-linear	$m \cdot v$	$2 \cdot m$	$m \cdot (2 \cdot s + 1)$
milp	$2 \cdot m$	$m \cdot v$	$m \cdot (2 \cdot s + 1)$

Sei p die geplante Anzahl an Boxen jeder Gruppe. Das erstellen der Boxen ist $\mathcal{O}(m \cdot (s \cdot k + p \cdot s \cdot k))$

Die Laufzeitanalyse habe ich mithilfe zufällig generierter Eingabedateien ergänzt, um die Auswirkung von der Anzahl an Sorten, Kleidungsstücken und Stilrichtungen beim Bron-Kerbosch Algorithmus und (M)ILP zu sehen. Die Chance das zwei Stilrichtungen kompatibel sind, ist $\frac{1}{3}$ und das ein Kleidungsstück einer Sorte und Stilrichtung existiert $\frac{9}{10}$. Die Anzahl an verfügbaren Kleidungsstücken pro Stilrichtung und Sorte liegt immer zufällig zwischem 1 und 1000, außer bei der Untersuchung der Auswirkung der Anzahl an Kleidungsstücken.

Für den Bron-Kerbosch Algorithmus habe ich Eingabedateien für $r = 10...350$ generiert und die Laufzeit gemessen (Diagramm 1). Die Laufzeit sieht exponentiell aus. Die Laufzeit des Solvers ist nach Anzahl an Stilrichtungen (Diagramm 2), Anzahl an Sorten (Diagramm 3), nach Anzahl an Stilrichtungen und Sorten (Diagramm 4) und nach Anzahl verfügbaren Kleidungsstücken pro Art zufällig zwischen 1 und x (Diagramm 5) in den nachfolgenden Diagrammen dargestellt. Da ilp-quadratic relativ langsam war, ist es nur teilweise in Diagramm 2 zu sehen und ansonsten ausgelassen.

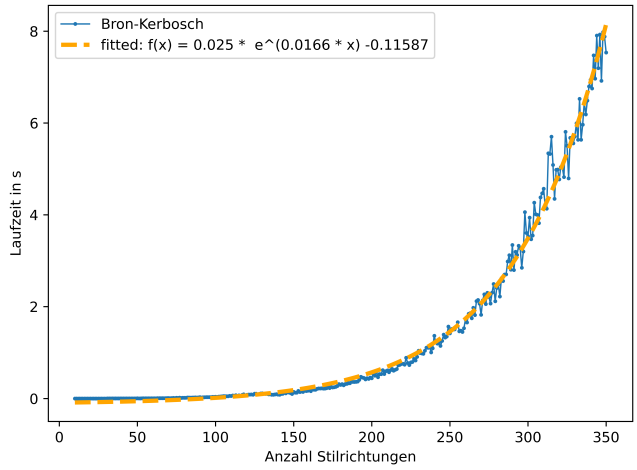

Abbildung 1: Bron-Kerbosch Laufzeit

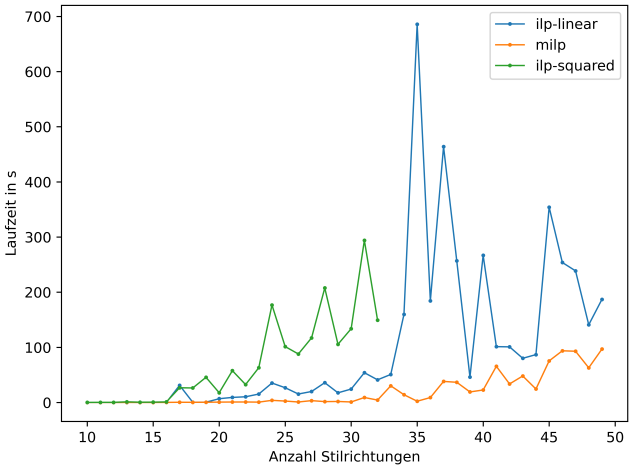

Abbildung 2: Anzahl an Sorten ist 30

Die Anzahl an Kleidungsstücken (Abbildung 4) scheint keinen großen Einfluss zu haben. Vielleicht liegt das aber auch an den Eingabedateien bzw. das der betrachtete x-Abschnitt zu klein ist.

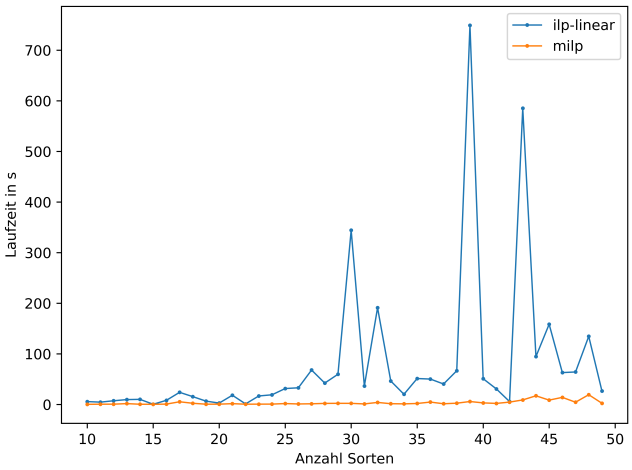

Abbildung 3: Anzahl an Stilrichtungen ist 30

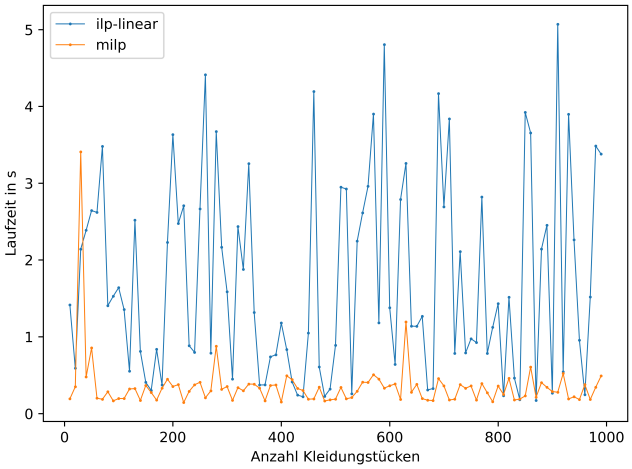

Abbildung 4: Anzahl an Sorten und Stilrichtungen ist 20

4 Beispiele

Die gegebenen Beispiele konnten alle in ca. 30ms gelöst werden. Die Tabelle zeigt die Anzahl an benutzen Kleidungsstücken. Bei allen Formulierungen kam dieselbe Lösung raus. Die Leistungsfähigkeit bei größeren Eingaben wurde bereits in der Laufzeitanalyse gezeigt, weshalb hier nur auf die Beispiele vom BWINF eingegangen wird.

Eingabedatei	Lösung
paeckchen0.txt	11/11
paeckchen1.txt	499/499
paeckchen2.txt	32/32
paeckchen3.txt	94/96
paeckchen4.txt	437/437
paeckchen5.txt	155/174
paeckchen6.txt	748/770
paeckchen7.txt	569/700

In der Ausgabe sind die Boxen durch leere Zeilen getrennt. Kleidungsstücke einer Box werden im Format <Sorte> <Stilrichtung> <Anzahl> ausgegeben. Längere Ausgaben sind durch ... gekürzt. Die vollständigen Ausgaben sind unter loesungen/ zu finden. Das Programm wird mit den Parametern <Modus> <Eingabepfad> <Ausgabepfad> gestartet. Die Modi sind:

- ilp-quadratic
- ilp-linear
- milp

Die Funktion n Boxen zu erstellen muss man im Quelltext einstellen. Neben der eigentlichen Ausgabe (bei den Beispielen) kommt immer noch die Ausgabe mit n Boxen, wobei n so gewählt wurde, dass es die maximale Anzahl an erstellbaren Boxen ist. Die obere Grenze wurde per Hand gefunden. Ein extra Programm validator.out wurde geschrieben, der einen gegebenen Input und Output auf Gültigkeit überprüft.

4.1 paeckchen0.txt

Die mit milp erhaltene Lösung ist falsch. Die erhaltene Lösung verletzt die Bedingung, nicht mehr Stücken als verfügbar sind von einem Kleidungsstück zuzuweisen. Das eigentliche Problem ist, dass in der Statusmeldung trotzdem Optimal angezeigt wird. Ich würde sehr sehr gerne wissen, woran das liegt. Bei ilp-linear und ilp-quadratic gab es insgesamt keine Probleme. Für die anderen Beispiele gibt milp gültige Lösungen aus.

```
# Lösung ist falsch Kleidungsstück (1,2) wurde 5-Mal zugewiesen
# und Kleidungsstück (1,1) kein einziges Mal
Modus:  milp
status: 1, Optimal
benutzte Kleidungsstuecke: 11.0
erstellte Boxen: 3
1 2 3
2 2 1
3 2 1

1 2 1
2 2 1
3 2 1

1 2 1
2 2 1
3 2 1

Dauer: 0.01396 s
```

```
Modus: ilp-linear
status: 1, Optimal
benutzte Kleidungsstücke: 11.0
erstellte Boxen: 3
1 1 3
2 2 1
3 2 1

1 2 1
2 2 1
3 2 1

1 2 1
2 2 1
3 2 1

Dauer: 0.00611 s
```

```
# Maximale Anzahl an Boxen
Modus: ilp-linear
status: 1, Optimal
benutzte Kleidungsstücke: 11.0
erstellte Boxen: 3
1 1 3
2 2 1
3 2 1

1 2 1
2 2 1
3 2 1

1 2 1
2 2 1
3 2 1

Dauer: 0.00506 s
```

4.2 paeckchen1.txt

```
Modus:  milp
status: 1, Optimal
benutzte Kleidungsstuecke: 499.0
erstellte Boxen: 132
1 1 3
```

```
2 1 1
3 1 3

...

1 4 1
2 4 1
3 4 1

Dauer: 0.01802 s
```

```
# Maximale Anzahl an Boxen
Modus: ilp-linear
status: 1, Optimal
benutzte Kleidungsstücke: 499.0
erstellte Boxen: 142
1 1 3
2 1 1
3 1 1

...

1 4 1
2 4 1
3 4 1

Dauer: 0.009 s
```

4.3 paeckchen2.txt

```
Modus:  milp
status: 1, Optimal
benutzte Kleidungsstuecke: 32.0
erstellte Boxen: 6
1 3 1
2 5 2
3 2 1
3 5 1

1 2 1
2 2 1
2 6 2
3 2 2
3 8 1

1 6 1
2 6 1
2 8 1
3 8 1

1 1 1
2 4 2
3 5 2

1 7 1
2 7 3
3 7 1
3 9 2

1 7 1
2 7 1
3 9 2

Dauer: 0.02815 s
```

```
# Maximale Anzahl an Boxen
Modus: ilp-linear
status: 1, Optimal
benutzte Kleidungsstücke: 32.0
erstellte Boxen: 6
1 3 1
2 5 2
3 5 1

1 2 1
2 2 1
2 6 2
3 2 3

1 6 1
2 6 1
2 8 1
3 8 2

1 1 1
2 4 2
3 5 2

1 7 1
2 7 3
3 9 3

1 7 1
2 7 1
3 9 1
3 7 1

Dauer: 0.02335 s
```

4.4 paeckchen3.txt

```
Modus:  milp
status: 1, Optimal
benutzte Kleidungsstuecke: 94.0
erstellte Boxen: 12
1 1 3
2 1 3
3 1 1
4 1 2
4 2 1
5 6 3

...

1 9 1
2 10 1
3 10 1
4 9 1
5 10 1

Dauer: 0.03054 s
```

```
# Maximale Anzahl an Boxen
Modus: ilp-linear
status: 1, Optimal
benutzte Kleidungsstücke: 94.0
erstellte Boxen: 13
1 1 3
2 1 3
3 1 1
4 1 2
4 2 1
```

```
5 6 3

...

1 7 1
2 10 1
3 10 1
4 9 1
5 10 1

Dauer: 0.01292 s
```

4.5 paeckchen4.txt

```
Modus:  milp
status: 1, Optimal
benutzte Kleidungsstuecke: 437.0
erstellte Boxen: 31
1 24 1
2 16 3
3 16 1
4 16 3
5 16 3
6 16 1
7 1 3

...

1 18 1
2 13 1
3 21 3
4 13 3
5 13 2
6 13 2
6 21 1
7 20 3

Dauer: 0.02153 s
```

```
# Maximale Anzahl an Boxen
Modus: ilp-linear
status: 1, Optimal
benutzte Kleidungsstücke: 437.0
erstellte Boxen: 35
1 24 1
2 16 3
3 16 2
4 16 3
5 16 1
6 16 2
7 1 3

...

1 18 1
2 13 1
3 21 2
4 13 3
5 13 2
6 13 2
7 20 1

Dauer: 0.01641 s
```

4.6 paeckchen5.txt

```
Modus:  milp
status: 1, Optimal
benutzte Kleidungsstuecke: 155.0
erstellte Boxen: 19
1 1 1
2 1 3
3 2 2
4 2 3
5 2 3

...

1 21 3
2 16 3
3 21 3
4 21 3
5 17 1

Dauer: 0.01211 s
```

```
# Maximale Anzahl an Boxen
Modus: ilp-linear
status: 1, Optimal
benutzte Kleidungsstücke: 155.0
erstellte Boxen: 19
1 1 1
2 1 3
3 2 2
4 2 3
5 2 3

...

1 21 3
2 16 3
3 21 3
4 21 3
5 17 1

Dauer: 0.01273 s
```

4.7 paeckchen6.txt

```
Modus:  milp
status: 1, Optimal
benutzte Kleidungsstuecke: 748.0
erstellte Boxen: 66
1 1 1
2 3 3
3 2 3
4 2 3
5 10 1

...

1 7 2
2 9 3
3 9 3
4 10 1
5 10 1

Dauer: 0.0206 s
```

```
# Maximale Anzahl an Boxen
Modus: ilp-linear
status: 1, Optimal
benutzte Kleidungsstücke: 748.0
erstellte Boxen: 100
1 1 1
2 3 3
3 2 3
4 2 3
5 10 1

...

1 7 2
2 9 1
2 7 2
3 9 2
3 7 1
4 10 1
5 10 1

Dauer: 0.0128 s
```

4.8 paeckchen7.txt

```
Modus:  milp
status: 1, Optimal
benutzte Kleidungsstuecke: 569.0
erstellte Boxen: 50
1 1 1
2 3 3
3 2 3
4 3 3
5 10 1

...

1 7 2
2 9 1
3 9 2
4 10 1
5 10 2

Dauer: 0.01918 s
```

```
# Maximale Anzahl an Boxen
Modus: ilp-linear
status: 1, Optimal
benutzte Kleidungsstücke: 569.0
erstellte Boxen: 50
1 1 1
2 3 3
3 2 3
4 3 3
5 10 1

...

1 7 2
2 9 1
3 9 2
4 10 1
5 10 3

Dauer: 0.01344 s
```

5 Quellcode

5.1 Klasse Graph

```
class Graph():
def __init__(self, amount_nodes):
    self.adj_list = [[] for _ in range(amount_nodes)]

def add_edge(self, start, end):
    self.adj_list[start].append(end)

    def bron_kerbosch(self, P, R=set(), X=set()):

      # Ist P und X leer haben wir eine maximale Clique gefunden
      if not P and not X:
          yield R


      # Pivot Element auswählen
      best = 0
      pivot = -1
      for v in P.union(X):
          if len(self.adj_list[v]) >= best:
              pivot = v
              best = len(self.adj_list)

      # Ueber alle Knoten in P,
      # die nicht Nachbarn vom Pivotelement sind iterieren
      for v in P.difference(self.adj_list[pivot]):
          # Rekursiver Aufruf
          yield from self.bron_kerbosch(
                  R=R.union([v]),
                  P=P.intersection(self.adj_list[v]),
                  X=X.intersection(self.adj_list[v]))

          P.remove(v)
          X.add(v)
```

5.2 Klasse Clique

```
class Clique():
def __init__(self, amount_categories, amount_styles, useable_styles, inventory, variable_type, id):
    self.amount_categories = amount_categories
    self.amount_styles = amount_styles

    self.variables = [[None
                    for _ in range(amount_styles)]
                    for _ in range(amount_categories)]

    # Als Praefix für die Namen der Variablen
    self.id = id

    self.useable_styles = useable_styles

    # LpVariablen erstellen
    for style in useable_styles:
        for category in range(amount_categories):
            if inventory[category][style] > 0:
                name = str(id) + "_" + str(category) + "_" + str(style)
                self.variables[category][style] = LpVariable(name=name, lowBound=0, cat=variable_type)

    min_max_type = "Integer"
    if variable_type == "Integer":
        min_max_type = "Continuous"

    # Werden zwar immer erstellt, jedoch bei ilp-squared nicht dem Modell hinzugefügt
    self.min_variable = LpVariable(name=str(id) + "_min", lowBound=0, cat=min_max_type)
```

```
        self.max_variable = LpVariable(name=str(id) + "_max", lowBound=0, cat=min_max_type)

    # Gibt zurück die Summe aller Variablen dieser Sorte
    def category_sum(self, category) -> LpAffineExpression:
        sum = 0
        for style in self.useable_styles:
            if self.variables[category][style] is not None:
                sum += self.variables[category][style]
        return sum

    # Gibt zurück für jede Sorte die Anzahl an Kleidungsstücken
    def categories_total(self) -> list[int]:
        total = [0 for i in range(self.amount_categories)]
        for category in range(self.amount_categories):
            for style in self.useable_styles:
                if self.variables[category][style] is not None:
                    total[category] += self.variables[category][style].varValue
        return total

    def add_second_constraint_quadratic(self, model):
        cache = [self.category_sum(category) for category in range(self.amount_categories)]
        for constraint_category in range(self.amount_categories):
            for category in range(self.amount_categories):
                if category == constraint_category:
                    continue
                model += cache[constraint_category] <= 3 * cache[category]

    def add_second_constraint_linear(self, model):
        for category in range(self.amount_categories):
            sum = self.category_sum(category)
            model += self.min_variable <= sum
            model += self.max_variable >= sum
        model += self.max_variable <= 3 * self.min_variable

    def get_boxes(self, amount=None) -> list[list[list[int]]]:
        categories_total = self.categories_total()

        remaining_amount = min(categories_total)
        if amount is not None:
            remaining_amount = amount

        boxes = []

        while remaining_amount > 0:
            box = []

            for category in range(self.amount_categories):

                # Soviel nehmen das wir danach noch remaing_amount-1 Boxen erstellen können
                take = min(categories_total[category] - (remaining_amount - 1), 3)
                for style in self.useable_styles:
                    if take == 0:
                        break
                    elif self.variables[category][style] is None:
                        continue
                    elif int(self.variables[category][style].varValue) == 0:
                        continue

                    took = int(min(self.variables[category][style].varValue, take))

                    take -= took
                    self.variables[category][style].varValue -= took
                    categories_total[category] -= took

                    box.append([category + 1, style + 1, took])

            boxes.append(box)
            remaining_amount -= 1
        return boxes

    ...
```

5.3 Funktion add_first_constraints

```
def add_first_constraint(model, groups, amount_types, amount_styles, inventory):
for type in range(amount_types):
    for style in range(amount_styles):
        sum = 0
        for gruppe in groups:
            if gruppe.variables[type][style] is not None:
                sum += gruppe.variables[type][style]
        model += sum <= inventory[type][style]
```

5.4 Funktion get_n_boxes

```
def get_n_boxes(n, cliques):
  boxes = []

  cliques_min_amount = []
  cliques_max_amount = []
  for i in range(len(cliques)):
      categories_total = cliques[i].categories_total()
      cliques_min_amount.append(math.ceil(max(categories_total) / 3))
      cliques_max_amount.append(min(categories_total))

  # Wie viel insgesamt zusätzlich zur minimalen Anzahl an Boxen
  # genommen werden soll
  d = n - sum(cliques_min_amount)

  for i in range(len(cliques)):
      spielraum = cliques_max_amount[i] - cliques_min_amount[i]
      extra_nehmen = min(d, spielraum)

      boxes = boxes + cliques[i].get_boxes(amount=cliques_min_amount[i] + extra_nehmen)
      d -= extra_nehmen

  return boxes
```

5.5 main.py

```
  ...

  amount_categories, amount_styles, edges, inventory = read_input(sys.argv[2])

  # Stilgraphen erstellen
  graph = Graph(amount_styles)
  for edge in edges:
      graph.add_edge(edge[0], edge[1])
      graph.add_edge(edge[1], edge[0])

  # maximale Cliquen finden
  maximal_cliques = list(graph.bron_kerbosch(P=set([node for node in range(amount_styles)])))


  model = LpProblem(name="Do-something", sense=LpMaximize)

  cliques = []
  for id, clique_styles in enumerate(maximal_cliques):
      cliques.append(Clique(amount_categories, amount_styles, clique_styles, inventory, variable_type, id))

  # Kostenfunktion hinzufügen
  cost_expression = 0
  for clique in cliques:
      for category in range(amount_categories):
          for style in range(amount_styles):
              if clique.variables[category][style] is not None:
```

```
                cost_expression += clique.variables[category][style]
model += cost_expression


# Constraint (1) hinzufügen
add_first_constraint(model, cliques, amount_categories, amount_styles, inventory)

# Constraint(2) hinzufügen
for clique in cliques:
    if mode == modes[0]:
        clique.add_second_constraint_quadratic(model)
    else:
        clique.add_second_constraint_linear(model)

# N Boxen Constraint hinzufügen
#n = 50
#add_n_boxes_constraint(model, n, cliques)

status = model.solve(PULP_CBC_CMD(msg=False))

if model.status == -1:
    print("konnte nicht gelöst werden")
    exit()

for clique in cliques:
    if clique.is_integral() == False:
        print("Alarmstufe Rot, ALARMSTUFE ROT")
        exit(1)

# (Falls N) genau N Boxen erstellen
#boxes =get_n_boxes(n, cliques)
boxes = []
for clique in cliques:
    boxes = boxes + clique.get_boxes()

write_output(sys.argv[3], boxes)
```