Eine Optimierung des Stilvolle-Päckchen-Problems durch die Modellierung als ILP

Eine Fortführung der Wettbewerbsaufgabe „Stilvolle Päckchen“
im Rahmen des 42. Bundeswettbewerbs Informatik

Eine besondere Lernleistung von

Felix Ying Xin Du

Abgabedatum:	19. Dezember 2024
Interne Betreuung:	Ralf Böttcher
Institution:	Sächsisches Landesgymnasium Sankt Afra zu Meißen

Zusammenfassung

Als Fortführung einer Aufgabe des 42. Bundeswettbewerbs Informatik wird das Problem „Stilvolle Päckchen“ weiter behandelt und ein neuer Lösungsansatz formuliert. Nachdem eine theoretische Analyse zur Einordnung in die Komplexitätsklasse des Problems durchgeführt wurde, werden durch weitere Problemabstrahierungen Nebenbedingungen und eine Zielfunktion formuliert, die zu einer Lösung als Integer Linear Program führen. Ergebnisse verschiedener Lösungsansätze werden in den Aspekten Lösungsqualität und Laufzeit verglichen und daraus Vor- und Nachteile für die jeweiligen Ansätze abgeleitet. Des weiteren wird die Bewertung der Aufgabe durch den Wettbewerb reflektiert und die Lösung im Rahmen des Wettbewerbs diskutiert.

Inhaltsverzeichnis

1 Einleitung

Das Integer Linear Programming ist eine Methodik der mathematischen Optimierung, die in vielen Aspekten der echten Welt Anwendung findet. Vor allem in der Logistik, Planung und im Ressourcenmanagement, die Optimierungsaspekte über ganze Zahlen als lineare Zielfunktion darstellen können, können ILP's optimale, oder zumindest heuristisch fast optimale Lösungen erzeugen, um Probleme der echten Welt zu lösen. Im Rahmen des 42. Bundeswettbewerbs Informatik wurde eine Lösung zur 2. Runde eingereicht, die unter anderem die Aufgabe 2: „Stilvolle Päckchen“ behandelte und dafür eine Lösungsidee formulierte und umsetzte – in dieser Arbeit soll diese Aufgabe weitergeführt werden. Zunächst soll der Arbeitsstand der Einreichung aufgegriffen werden und innerhalb der theoretischen Analyse die NP-Schwere formal durch eine Reduktion nachgewiesen werden. Anschließend sollen verschiedene Lösungsideen, unter anderem die bereits aufgestellte, formuliert, algorithmisch betrachtet und umgesetzt werden, um deren Ergebnisse zu vergleichen und daraus die vorbetrachteten Lösungsansätze einordnen zu können. Problemmodellierungen und Lösungsansätze sollen ausführlich beschrieben werden, da diese notwendigerweise präzise ausgeführt werden müssen, um stimmige Lösungen zu erzeugen und Argumentationsgrundlage für die durch die Ergebnisanalyse erforderte Diskussion zu bieten. Des Weiteren richten sich alle neu entwickelten Lösungsansätze an der im Wettbewerb bearbeitete Idee; im Vergleich zu dieser soll das Ziel sein, Verfahren und Laufzeit zu optimieren und Redundanzen und Ineffizienzen zu reduzieren. Die Ergebnisse werden für die durch den Bundeswettbewerb vorgegeben Aufgaben erstellt und verglichen, um Vor- und Nachteile der verwendeten Lösungsansätze herauszuarbeiten und Rückbezug zum Wettbewerb zu ziehen. Schlussendlich soll der Bundeswettbewerb Informatik rückblickend reflektiert werden, indem die Bewertung der Einreichung herangezogen wird, und Punktabzüge und -zusprüche analysiert und eingeordnet werden, um anschließend Eigenschaften einer BwInf-Aufgabe herzuleiten und die eingereichte Lösung unter diesen Aspekten neu zu bewerten.

2 Theoretische Analyse

2.1 Problemdefinition

Das behandelte Stilvolle-Päckchen-Problem soll folgend mit SPP bezeichnet werden. Zudem soll es als folgendes Entscheidungsproblem formuliert werden: Können die Boxen mit gegebenen Kleidungstücken N so gepackt werden, dass mindestens x Kleidungsstücke verwendet werden? Jene Formulierung, die entsprechende Zielfunktion nach sich zieht, steht zwar diametral zum ursprünglich formulierten Ziel der Aufgabe, nämlich dass so wenig Kleidungsstücke wie möglich nicht verpackt werden, fordert allerdings dasselbe Ergebnis, da die Menge an Kleidungsstücken konstant ist. Eine Eingabe E wird mit $E = (T, S, N, U, x)$ angegeben, wobei T die Sortenmenge, S die Stilmenge und N die Kleidungsmenge darstellt und ein Kleidungsstück $n \in N$ mit $n = (t, s, q)$, wenn $t \in T$, $s \in S$ und $q \in \mathbb{N}$ gilt und q dabei die Anzahl des jeweiligen Kleidungsstück darstellt. Die Kombinationsmenge U gibt die Kombinationsmöglichkeiten zwischen zwei verschiedenen Stilen s_1 und s_2 an. Demnach gilt für U: $\{s_1, s_2\} \in U$ mit $s_1, s_2 \in S$. Des weiteren gelten folgende Bedingungen für alle gepackten Boxen $B = (N_B)$ mit $N_B \subset N$:

1. Für alle Elemente $n_i$undn_j, n in B muss für deren Stile s_i ein Element in U existieren. $(s_i, s_j) \in U$.

2. Für ein Element n_i gilt $q_i \leq 3$.

2.2 Beweis von NP-Schwere

Die NP-Schwere soll anhand einer Reduktion vom Cliquen-Problem gezeigt werden.

Cliquen-Problem 2.2.1. *Es sei ein Graph $G = (V, E)$ mit der Menge V an Knoten und der Menge E an Kanten, sowie eine Zahl $n, n \in \mathbb{N}$, die angibt, wie groß die Clique in G mindestens sein muss, gegeben.*

Lemma 2.2.2. *Eine Eingabe des Cliquen-Problems lässt sich in polynomieller Zeit in eine des SPPs umwandeln.*

Beweis. Wir verwenden die Menge V an Knoten des Graphen G, um die Sortenmenge T und die Stilmenge S zu erzeugen, zusammen mit einer Kontrollsorte τ und einem Hilfsstil σ. Deren genauere Bedeutung wird im Verlauf des Beweises klarer. Aus der Erzeugung der Stile durch die Knotenmenge V geht auch die Kombinationsmöglichkeiten jener durch die Kantenmenge E hervor. Für die zu verpackende Menge x soll folgende Definition gelten, die durch den Beweis ebenfalls hervorgeht. So gilt also Folgendes:

1. Sortenmenge T mit $T = V \cup \{\tau\}$; es gilt $O(|V| + 1)$.

2. Stilmenge S mit $S = V \cup \{\sigma\}$; es gilt ebenfalls $O(|V| + 1)$.

3. Kombinationsmenge U mit $U = E \cup \{(\tau, \sigma), (s, \sigma)\}$, wenn $s \in S$ gilt; es gilt $O(|E| + 2)$.

4. $x = |V| + n + 1$; es gilt $O(1)$.

Somit lässt sich eine Eingabe des Cliquen-Problems in polynomieller Zeit in eine Eingabe des SPPs umwandeln. □

Lemma 2.2.3. *Für $T = V \cup \{\tau\}$ existiert für eine Lösung maximal packbare Box.*

Beweis. Für die soeben definierten Mengen S und T sollen folgende Kombinationen für jeweils ein Kleidungsstück $n = (t, s)$ mit $t \in T$ und $s \in S$ die einzig Möglichen sein:

1. $n = (\tau, \sigma)$ mit.

2. $n = (t, \sigma)$ mit $t \in T \setminus \{\tau\}$.

3. $n = (t, s)$ mit $t \in T \setminus \{\tau\}$, $s \in S \setminus \{\sigma\}$, und $r = s$, da $V \subset T, S$ bzw. $T \cap S = V$ gilt.

Für die Kombination mit der Hilfssorte τ nehmen wir $q = 1$ an. Daraus geht hervor, dass es nur eine mögliche, regelkonforme Box geben kann, da es nur ein Kleidungsstück $n = (\tau, \sigma)$ mit einer Quantität $q = 1$ gibt und jede Box jede Sorte $s \in T$ enthalten muss. Somit hat eine potenzielle Lösung maximal eine packbare Box. □

Lemma 2.2.4. *Wenn eine potenzielle Lösung für SSP mit $x = |V| + n + 1$ existiert, enthält der Graph G zugleich eine n-Clique.*

Beweis. Aus Lemma 2.2 geht hervor, dass alle Stile der Kleidungsstücke (x, die verpackt werden sollen) eine Kante in U besitzen müssen. Es sollen $|V| + 1$ Kleidungsstücke mit dem Stil σ existieren. Demnach bleiben n Kleidungsstücke übrig, die $|S| - 1$ Stile abdecken müssen - es gilt $|S| - (|V| + 1) = n$. Wenn eine derartige Verteilung mit x Kleidung und besagten Bedingungen möglich sein soll, müssen alle Stile der n Kleidungsstücke miteinander kombinierbar sein. Das heißt, dass ihre Knoten in G alle eine Kante in E miteinander besitzen, sodass sie der Definition einer Clique entsprechen. Somit besitzt der Graph G eine Clique mit mindestens n Knoten. □

Lemma 2.2.5. *Eine Lösung des Cliquen-Problems lässt sich in polynomieller Zeit in eine Lösung des SPPs überführen.*

Beweis. Für einen Graphen G, der eine n-Clique besitzt, gilt, dass auch die zugehörigen Stile der n-Clique zwingend eine Kombination in U haben müssen. So bleiben $|V| + 1$ Kleidungsstücke übrig. Aus Lemma 2.2 geht hervor, dass diese alle dem Stil σ gehören; da σ ein Hilfsstil darstellt und mit jedem anderen Stil kombinierbar ist, besitzt der zugehörige Knoten in G auch Kanten mit jedem anderen Knoten in E. So gibt es auch eine dementsprechende Kombination in U. Die Zeitkomplexität entspricht den Vorgehensweisen in Lemma 2.2. □

Daraus geht hervor, dass SPP mindestens so schwer sein muss wie das Cliquen-Problem, da nach Lemma 2.2 eine Lösung des Cliquen-Problems in polynomieller Zeit in eine Lösung des SPPs umsetzbar ist. Da das Cliquen-Problem nach Karp's 21 NP-vollständigen Problemen auch eines dieser ist, ist das SPP wie soeben nachgewiesen auch NP-schwer.

Q.E.D.

3 Lösungsansätze

Angesichts der bewiesenen Komplexität des Problems, werden Lösungsansätze gesucht, die entweder gute Heuristiken erzeugen, oder für kleine Eingaben eine ideale Lösung, sprich ein maximales x, suchen bzw. den Einfluss der wachsenden Haupteingabegröße zu relativieren. Im Anschluss werden verschiedene Lösungsansätze präsentiert die alle für sich stehen sollen. Dabei ist in Kapitel 3.1 jener Lösungsansatz inkludiert, der innerhalb des Bundeswettbewerbs Informatik zur Einreichung verwendet wurde.

3.1 Modellierung als Graph

Die Formulierung als Problem der Graphentheorie liegt angesichts der Verwendung des Cliquen-Problems, ein Problem der Graphentheorie, nahe. Die in der Menge S gegebenen Stile, die unter der Berücksichtigung der Kombinationsmenge U in Verbindung zueinander stehen, lassen sich, wie auch aus Lemma 2.2 zu entnehmen ist, als Graph darstellen. Die Kleidungsmenge N lässt sich ebenso als Graph darstellen, bei dem jeder Knoten ein Kleidungsstück n mit den Eigenschaften (t, s) mit $t \in T$ und $s \in S$ abbildet. Zwei Knoten n_1, n_2 sind genau dann miteinander kombinierbar und besitzen eine Kante, wenn $\{t_1, t_2\} \in U$ gilt. So konstruieren wir einen Graphen $G = (V, E)$ mit $V = N$ und $E = U$, wenn für ein Kleidungsstück $n_1 = (t_1, s_1, q_1) \widehat{=} q_1 \cdot \bar{n}_1$ mit $\bar{n}_1 = (t, s)$ gilt. Jede Box die nun packbar ist, muss jede Sorte aus T enthalten, und vollständig miteinander kompatibel sein; dabei können die Sorten als Farben dargestellt werden. Dies entspricht, ähnlich wie in der Reduktion angewandt, einer Clique, die gleichzeitig jede Farbe des Graphen enthält. So kann folgendes Optimierungsproblem formuliert werden: Gegeben sei der Graph G, wie sollten gültige[1] Cliquen ausgewählt werden, sodass die verwendete Zahl an Kleidungsstücken x maximal wird. Diese Formulierung vereint Elemente aus mehreren, bereits bekannten, Problemen der Graphentheorie, so z.B. des Cliquen-Problems nach Carraghan und Pardalos [2] oder der Graphenfärbung nach Cormen u. a. [4, S. 425]. Ein bereits ähnliches oder gleiches, behandeltes Problem konnte im Rahmen der Recherche des Bundeswettbewerbs nicht gefunden werden.

3.2 Depth-First-Search Brute-Force-Algorithmus

Auch wenn die NP-Schwere dieses Problems im Kapitel 2.2 schon bewiesen wurde und demnach offensichtlich ist, dass eine gesichert ideale Lösung für alle Eingabegrößen nicht in polynomieller Zeit zu finden ist, kann ein Brute-Force-Ansatz für relativ kleine Angaben einen validen Lösungsansatz darstellen. So kann sichergestellt werden, dass die gegebene Lösung garantiert die Beste oder in der Menge der Besten liegt. Durch die Formulierung als Problem der Graphentheorie lässt sich ein Depth-First-Search Ansatz formulieren, der systematisch alle Möglichkeiten der Zusammenstellung von Cliquen in einem

[1] Nach gegebenen Regeln

Suchbaum erstellt. Hauptsächlich werden für alle Knoten ihre möglichen, minimalen Cliquen generiert und anschließend alle Kombinationen zwischen diesen Cliquen iteriert. Der BF-Algorithmus wird als Tiefensuche klassifiziert, weil jede Lösung mit ihren Boxen bzw. ihren Cliquen vollständig konfiguriert wird, bevor die nächste Lösung generiert wird. Für eine Lösung, deren Cliquen noch minimal sind, werden, nach der Iteration über die Cliquen, die restlichen Entitäten den Boxen der Lösung zugeordnet, ohne dabei die gegebenen Bedingungen zu verletzen. Nach dieser Zuweisung ist die Lösung vollständig. Falls diese ohne Rest ist, sprich keine Kleidungsstücke übrig bleiben, terminiert der Algorithmus. Falls nicht, wird die nächste Lösung generiert. Insgesamt entsteht eine Funktion, die alle möglichen Kombinationen aller minimalen Cliquen erstellt:

Algorithm 1 Kombinationen aller minimalen Cliquen in G

```
1: global globaler_speicher ← [ ]
2: function ALL_CLIQUE_COMBINATIONS(G = (N, E), box : list)
3:     for Knoten n ∈ N do
4:         min_cliques ← GET_MIN_CLIQUES(n)          ▷ Liste als return
5:         if LENGTH(min_cliques) = 0 then
6:             globaler_speicher.append(paeckchen)    ▷ Abbruchbedingung
7:         end if
8:         for Clique c ∈ min_cliques do
9:             G_copy ← G.copy()
10:            G_copy.remove(c)
11:            neue_box ← box.copy()
12:            neue_box.append(c)
13:            All_Clique_Combinations(G_copy, neue_box)    ▷ Rekursiver
    Aufruf
14:        end for
15:    end for
16:    return globaler_speicher
17: end function
```

Wie bereits erwähnt, wird hierbei durch die Funktion $Get\_Min\_Cliques(n)$ eine Liste an allen minimalen Cliquen generiert, die alle den Knoten n enthalten. „Der Algorithmus iteriert über ein Dictionary, welches als Schlüssel alle Sorten hat und als Werte alle benachbarten Knoten mit entsprechender Sorte speichert. Nun werden daraus alle möglichen Kombinationen gebildet; die Anzahl der Knoten einer Clique entspricht der Anzahl der Sorten.“[2] „Innerhalb von *get_min_cliques*() findet die Funktion *complete*() große Verwendung. Sie kontrolliert für jede Kombination an Knoten, ob diese Kombination einen vollständigen Subgraphen abbildet und somit eine Clique ist.“[3]

[2] siehe Anhang 7.4, S. 4
[3] siehe Anhang 7.4, S. 4

Algorithm 2 Alle minimalen Cliquen mit n

```
1: function GET_MIN_CLIQUES(G = (N, E), nachbar_sorten : dict, iterator : int, sub_ergebnis : list, ergebnis : list, blacklist : list, n)
2:     if iterator = len(nachbar_sorten) then
3:         if COMPLETE(G, sub_ergebnis) then                ▷ Kompletter Graph/Clique?
4:             pass
5:         else if Length(blacklist) = 0 then
6:             ergebnis.append(sub_ergebnis)
7:         else
8:             bool_copy ← False
9:             for x in blacklist do
10:                if sub_ergebnis[0] ∈ x and SORTED(sub_ergebnis) = SORTED(x) then
11:                    bool_copy ← True
12:                    break
13:                end if
14:            end for
15:            if bool_copy = False then
16:                res.append(sub_res)
17:            end if
18:        end if
19:    else
20:        keys ← list(neighbors_sorted)
21:        for x in nachbar_sorten[keys[iterator]] do
22:            lst ← sub_ergebnis.copy()
23:            lst.append(x)
24:            GET_MIN_CLIQUES(G, nachbar_sorten, iterator + 1, lst, ergebnis, blacklist,n)
25:        end for
26:    end if
27: end function
```

Output: ergebnis: list ⟶ Liste aller Kombinationen mit n
Zeitkomplexität für k : $AnzahlanSorten$ $O(k^n)$

Die Funktion *complete*() überprüft in jedem rekursiven Aufruf, ob der Subgraph G, angegeben als Parameter „complete“, sprich eine Clique ist.

Algorithm 3 sind die Knoten H eine Clique in G?

```
1: function COMPLETE(G = (N, E), H := H ⊆ N)
2:     SG ← G.subgraph(H) ← SG = (N, E)
3:     WG : SG
4:     for n in H do
5:         if len(WG(n) ∈ E von WG < len(H) − 1 then
6:             return False
7:         end if
8:     end for
9: end function
```

Output: boolean ⟵ bilden die Knoten H von G eine Clique in G?

3.2.1 Zeit- und Platzkomplexität

Für Algorithmus 1, der die Elternfunktion der Lösungsidee darstellt, gilt für die Knotenmenge N, ihre Kardinalität $k = |N|$, die Cliquenmenge C für jeden Knoten $v \in N$ und die jeweilige Kardinalität in Abhängigkeit von v, $c(v) = |C(v)|$, wenn n die maximale Tiefe des Suchbaums darstellt:

$$k \cdot O(c(v)^n) = O(c(v)^n) \tag{1}$$

Die in Algorithmus 1 inherenten Subalgorithmen, wie Algorithmus 2, der ebenfalls rekursiv und nicht polynomiell ist, sollen ignoriert werden, da diese durch ihre Elternfunktion, unabhängig von ihrer eigenen Zeitkomplexität, irrelevant für die gesamte Laufzeitapproximation sind. Es geht also hervor, dass dieser BF-Algorithmus in nicht polynomieller Zeit terminiert. Auch für geringere Eingabegrößen ist diese Zeitkomplexität für die Laufzeit generell nicht zumutbar. Aus den berechneten Beispielen, die in Kapitel 4 näher beleuchtet werden, geht hervor, dass sich dies auch tatsächlich in den Laufzeiten für vergleichsweise kleine Eingaben widerspiegelt. Aus Zeile 4 in Algorithmus 1 wird deutlich, dass für jeden rekursiven Aufruf eine neue Liste zur Speicherung erstellt wird. Nun ist also aus der nicht polynomiellen Zeitkomplexität zu schließen, dass der Algorithmus ebenfalls eine nicht polynomielle Platzkomplexität hat. So lässt sich erkennen, wieso dieser BF-Ansatz nur für kleine Eingaben geeignet ist; dies ist für sich betrachtet nicht unbedingt eine schlechte Eigenschaft, da für kleine Eingaben eine ideale Lösung garantiert werden kann. Allerdings ist für wachsende Eingabegrößen dies mit Sicherheit ein unberechenbarer Lösungsansatz. Diese Eignung des gegebenen Lösungsansatzes wird in Kapitel 5.2 fortführend kommentiert.

3.3 Integer Linear Program (ILP)

Ein Integer Linear Program (ILP) ist ein mathematisches Optimierungsproblem, eine Erweiterung des normalen Linear Program (LP), bei der die Entscheidungsvariablen nur ganze Zahlen (Integer) annehmen dürfen. Das Ziel eines ILP's ist es, eine lineare Zielfunktion über ein dazugehöriges Lineares

Gleichungssystem LGS, das Nebenbedingungen abbilden kann, zu optimieren, genauer gesagt zu minimieren. Für das SPP soll eine Problemmodellierung gefunden werden, die das Aufstellen eines ILP ermöglicht, um einen ILP-Solver verwenden zu können.

3.3.1 Lösungsansatz

Wie auch aus Kapitel 2.1 hervorgeht, ist eine Abstrahierung mit Hilfe von Graphen naheliegend. Gegeben sei die Stilmenge S mit der Kombinationsmenge E. Es lässt sich ein Stilgraph $SG = (S, E)$ generieren, aus dem sich ergibt, dass die Stile einer Box zwingend einer maximalen Clique in SG angehören. Wenn also aus SG alle maximalen Cliquen generiert werden, stellen diese und alle ihre jeweiligen Kombinationen ihrer Knoten die Möglichkeiten der Boxzusammenstellung dar, betreffend der Stile in den entsprechenden Boxen. Nun soll auf diese Cliquen die gesamte Kleidung möglichst effizient aufgeteilt werden, sodass so wenig wie möglich übrig bleibt bzw. so viel wie möglich verpackt wird. Dieses neue, nun formulierte Problem kann anschließend auf ein ILP reduziert werden, um schlussendlich den genannten ILP-Solver formulieren zu können.

3.3.2 Problemmodellierung zum ILP

Zunächst einmal müssen im Graphen SG alle maximalen Cliquen generiert werden. Dafür wurde der Bron-Kerbosch Algorithmus, der nach Bron und Kerbosch [1] durch eine Branch & Bound Methodik alle nicht verwendbaren Wege „abschneidet“, verwendet. Ursprünglich als Lösungsmethode für das Traveling-Salesman-Problem von Little u. a. [7] entwickelt, hilft es uns, im Gegensatz zu Algorithmus 1 aus Kapitel 3.2, die Laufzeit drastisch zu senken. Nach Bron und Kerbosch [1, S. 2] geht hervor, dass für eine steigende Anzahl an Cliquen, für zufällig generierte Graphen, die Laufzeit nicht statistisch signifikant steigt; tatsächlich ist ein, jedoch ebenfalls statistisch nicht signifikanter, Abfall der Laufzeit für die größte Testgröße bei 12856 detektierten maximalen Cliquen zu erkennen. Da es sich bei den Stilgraphen um keine besonders dicht oder dünn vernetzten Graphen handelt, da diese auch erst ab einer großen Anzahl an Knoten statistisch relevant für die Berechnung werden, ist, auch aufgrund der Einfachheit in der Implementierung, der Bron-Kerbosch-Algorithmus gewählt worden. Die Implementation und der nun gezeigte Pseudocode direkt zitiert nach Eppstein, Löffler und Strash [5] sowie die folgende Erklärung paraphrasiert. Der Bron-Kerbosch Algorithmus verlangt 3 disjunkte Mengen R, P, X an Knoten, wobei die Knoten in R eine nicht zwingend maximale Clique darstellen und $P \cup X = \Gamma(R)$ gilt, sprich jeder Knoten in der Vereinigungsmenge $P \cup X$ ist benachbart zu jedem anderen Knoten in R. Während die Knoten in P zur Clique R hinzugefügt werden, werden die Knoten in X ausgeschlossen. Weitere Ausführungen zum Algorithmus erübrigen sich, da sie aus dem Pseudocode hervorgehen.

Algorithm 4 Bron-Kerbosch Algorithmus

```
1: procedure BronKerbosch(P, R, X)
2:     if P ∪ X = ∅ then
3:         Report R as a maximal clique
4:     end if
5:     for each vertex v ∈ P do
6:         BronKerbosch(P ∩ Γ(v), R ∪ {v}, X ∩ Γ(v))
7:         P ← P \ {v}
8:         X ← X ∪ {v}
9:     end for
10: end procedure
```

3.3.3 Kommentar zur Optimierung durch Pivoting

Eppstein, Löffler und Strash [5] führen zusätzlich eine Optimierung des Bron-Kerbosch Algorithmus an, der durch eine Entartung die rekursiven Aufrufe des Algorithmus ordnet, um verbesserte Zeitkomplexität zu erlangen [5, S. 4–6]. In diesem Kontext soll zweitrangig sein, welcher Algorithmus zur Berechnung aller maximaler Cliquen in einem Graphen verwendet wird, zumal sich der pivotisierte Bron-Kerbosch Algorithmus nach Eppstein, Löffler und Strash [5] nur für sehr große Eingaben des behandelten Graphen G rentiert. Für die erlangten Ergebnisse in Kapitel 4 ist der klassische BK-Algorithmus verwendet worden. Für die berechnete Cliquenmenge C muss nun die Zuweisung der Kleidungsstücke, auf die Cliquen oder ihre Subgraphen geschehen. Somit resultiert daraus ein neues Problem, welches sich folgend formulieren lässt und der Art eines ILP wesentlich näher kommt:
Gegeben sei eine Menge N von $a = |N|$ Variablen $n_1, n_2, \ldots, n_n$; es gilt $N \in \mathbb{Z}_0^+$. Es wird absichtlich die Notation $\mathbb{Z}_0^+$ anstelle von $\mathbb{N}$ gewählt, da dies der Definition eines ILP entspricht [4]. Auch wenn, je nach Problemdefinition, der Zahlenbereich angepasst werden kann, wird im Sinne der Fachlichkeit diese Schreibweise gewählt. Außerdem soll die Cliquenmenge C mit $c = |C|$ aus c disjunkten Mengen $C_1, C_2, \ldots, C_c$ bestehen für die gilt $C_c \subset C$. Die Vereinigung aller disjunkten mengen C_c entspricht offensichtlich C und soll auch der Menge N entsprechen. Des Weiteren soll es weitere t disjunkte Mengen $T_1, T_2, \ldots, T_t$ geben deren Vereinigung T auch der Menge N entsprechen sollen, sowie k disjunkte Mengen $K_1, K_2, \ldots, K_k$ deren Vereinigung K ebenfalls N entspricht. Für die Formulierung der Ungleichungen werden nun auch $|K|$ Zahlen $z_1, z_2, \ldots, z_k$ eingeführt, für deren Vereinigung Z, $Z \in \mathbb{Z}_0^+$ gilt, eingeführt sowie eine Zahl x mit $x \in \mathbb{N}$ für die Umformulierung als Entscheidungsproblem. Zusammengefasst wurden folgende Mengen bestimmt, die alle N entsprechen:

1. Menge N mit $a = |N|$ Variablen $n_1, n_2, \ldots, n_n$, für die gilt $N \in \mathbb{Z}_0^+$
2. Cliquenmenge C, die die maximalen Cliquen des Stilgraphen SG darstellen,
3. die Sortenmenge T

4. Menge K, für die Sorte-Stil-Kombinationen
5. Menge Z an nicht negativen ganzen Zahlen
6. sowie z_k, die die Menge der Kleidung mit der Kombination K_k entspricht, und
7. Zahl x mit $x \in \mathbb{N}$

Nun kann ein erster Ansatz für ein ILP formuliert werden. Für alle Objekte in der Menge N sollen nun Zahlen innerhalb ihres Definitionsbereichs eingesetzt werden, um eine potenzielle Lösung generieren zu können.

1. Für jede Teilmenge in K, sprich K_i, ist ihre Summe der Elemente kleiner gleich das jeweilige r_i.

$$\sum_{n \in K_i} = n \leq z_i \leftarrow \forall i, 1 \leq i \leq k, |K| \tag{2}$$

2. Für eine Clique C_i mit zwei Schnittmengen mit $T_j$und T_k gilt, dass die Summe der Elemente der einen Schnittmenge durch die Bedingung, dass ein Paket maximal 3 gleiche Kleidungsstücke einer Sorte T_i in einer Box besitzen darf, maximal dreimal so groß sein darf, wie die Summe der anderen Schnittmenge. Es gilt:

$$\sum_{n \in C_i \cap T_j} n \leq 3 \cdot \sum_{n \in C_i \cap T_l} n \tag{3}$$

Wenn wir diese Problemabstraktion auf unsere ursprüngliche Entscheidungsproblemformulierung beziehen, dann soll die Summe aller Elemente von N $\leq x$ sein. Anschließend soll diese Problemformulierung weiter formalisiert und auf Kapitel 2.1, der Reduktion auf NP-Schwere bezogen werden.
Wenn aus der Kombinationmenge U und der Stilmenge S ein Stilgraph $G = (S, U)$ erstellt wird und der Bron-Kerbosch Algorithmus über G die maximalen Cliquen generiert, dann entspricht die Cliquenmenge C der generierten maximalen Cliquenmenge über G. Die Kardinalität der Menge T des hier formulierten Problems, entspricht der Sortenmenge T aus der Ursprungsreduktion. Für die Mengen S und T aus der Reduktion lassen sich durch die Eingabe Sorte-Stil Paare bilden. Die Kardinalität dieser Menge entspricht in der Neuformulierung k, sowie die Kardinalität der Menge K. Es gibt demnach genau k sich unterscheidende Kleidungsstücke. Ein Kleidungsstück ist in dieser genau denn existent, wenn die Kardinalität von $C_x \cap T_y \cap K_z$ genau 1 ist. In diesem Fall gibt es ein Element in der Schnittmenge, die dadurch entsteht, dass T_y ein Teilelement der Kombination K_z darstellt, und der Stil in K_z in der Clique C_x liegt. Alle Schnittmengen dieser drei definierten Mengen, die die Kardinalität 1 aufweisen, sind disjunkt und weisen die Vereinigungsmenge N auf. Somit entspricht die Kardinalität von N für alle Kombinationen der Summe aller disjunkter Mengen, erzeugt durch $C_x \cap T_y \cap K_z$. An dieser Stelle könnte, in Äquivalenz zu Kapitel 2, ein formaler Beweis geführt werden, der sich allerdings erspart, da sich, wie eben beschrieben, die Eingabe des neu formulierten

Problems auf das originale SPP reduzieren lässt. Das formale Beweisverfahren ist dem in Kapitel 2 identisch.[4]

3.4 Formulierung des ILP

Wie aus Kapitel 3.3.2 hervorgeht, ist das SPP als ILP darzustellen. Nach Cormen u. a. [4, S. 853] sind ILP's und somit auch ihre Solver NP-schwer und sind in Worst-Case Betrachtungen stark asymptotisch. Allerdings werden diese Grenzwerte selten überschritten, sodass ein ILP für viele Probleme Lösungen mit moderatem Zeitaufwand erzeugen kann. Bei richtiger, nun geschehener Formulierung lässt sich das SPP sehr schlüssig mit seinen Bedingungen in ein lineares Ungleichungssystem umwandeln. Auch ohne den Ansatz eines ILP's sind diese klar ersichtlich und leicht formulierbar. Anschließend an das Kapitel 3.3.2 kann das ILP folgendermaßen aufgestellt werden:

$$\text{Minimiere (Zielfunktion) } \sum_{n \in N} n \tag{4}$$

$$\text{sodass (Nebenbedingung) } \sum_{n \in K_i} n \leq z_i \qquad \forall i \quad 0 \leq i \leq k \tag{5}$$

Aus der informellen Reduktion des neu abstrahierten Problems auf das SPP geht hervor, dass N durch die Vereinigung der Iterationen über Teilmengen von C, T, K generiert wird. Weil die Kardinalität der Menge Z äquivalent zur Menge K ist, ergibt sich, dass N für folgende zu minimierende lineare Zielfunktion (1) maximal wird, weil z_i für $i \leq k$ die Anzahl aller Kombinationsmöglichkeiten von Stil und Sorte darstellt (2).

$$\text{Nebenbedingung } \sum_{n \in C_i \cap T_j} n \leq 3 \cdot \sum_{n \in C_i \cap T_l} n \tag{6}$$

$$j \neq k \quad \forall i \quad 0 \leq i \leq k \quad 1 \leq j, l \leq k$$

$$0 \leq n \quad | \quad n \in \mathbb{Z} \tag{7}$$

Die Nebenbedingung lässt sich unverändert übernehmen. Für eine Box, dessen Stile als eine Clique C_x dargestellt werden, können in selber von einer Sorte T_j nicht mehr als dreimal so viel existieren wie von einer anderen Sorte T_k. Aus dieser Zielfunktion bzw. der Nebenbedingungen werden je nach Einschränkungen verschieden viele Ungleichungen für das lineare Ungleichungssystem generiert. So gibt es a Variablen mit k Ungleichungen aus (2) und $ct^2 - ct$ Ungleichungen aus (3) und n bzw. $|\mathbb{N}|$ aus Ungleichung (4).

[4] Die Herleitung des ILP's auf diese Weise, entspringt einem Gespräch mit Rasmus Zenker am 22. November 2024 von 17:13 - 17:32. Dessen Arbeit wird beim Forschungsprozess dieser Arbeit nicht verwendet, ist ohnehin nicht öffentlich einsehbar und befindet sich nicht im Besitz des Autors, weshalb diese schlussendlich nicht im Literaturverzeichnis aufgeführt ist.

3.4.1 Rückführen des ILP's auf ein Ergebnis des SPP

Algorithm 5 ILP Rückführung auf SPP Algorithmus

```
1: Alle_Boxen ← []
2: for all 1 ≤ i ≤ c do
3:     Box ← []
4:     M ← C_i ∩ T_j für j ∈ ℤ, 1 ≤ j ≤ t mit kleinster Summe
5:     for all 1 ≤ j ≤ t do
6:         Sorte ← []
7:         while Σ_{n∈C_i∩T_j} n > 3 · Σ_{b∈M} b do
8:             n_l, a_x ∈ C_i ∩ T_j
9:             n_x → Sorte C_i ∩ T_j \ {n_x}
10:        end while
11:        Sorte → Box
12:    end for
13:    Box → Alle_Boxen
14: end for
15: return Alle_Boxen
```

Für alle Untermengen in C wird über die je enthaltenen Sorten iteriert. Dafür wird die Sorte in einer maximalen Clique bestimmt, die am wenigsten vertreten ist (Z. 4), um dann über die restlichen Sorten zu iterieren und für jeden Durchlauf für den gilt, dass eine Sorte mehr vertreten ist, als die Kleidungsstücke Marginalsorte M, ein Kleidungsstück n_i gespeichert wird, bis die Abbruchbedingung erfüllt ist. So werden Boxen in Listen gespeichert mit *list:* n, $n \in C_i \cap T_j$.

3.4.2 Kommentar zur Linear-Program-Relaxation

Es besteht die Möglichkeit ein ILP zu relaxieren, indem es zu einem LP (Linear-Program) umgewandelt wird und nicht ganzzahlige Werte zugelassen werden. Im Gegensatz zum ILP ist ein LP nicht NP-schwer. Es gibt geläufige Algorithmen wie Seidel's Algorithmus oder der Simplex Algorithmus, die im Worst-Case exponentielle Laufzeiten haben, allerdings gute Average-Case Laufzeiten aufweisen [3]. Eine derartige LP-Relaxation würde nach Ergebnisgewinnung ein Rundungsverfahren erfordern, um auf ein für das SPP aussagekräftiges Ergebnis zu kommen. Eine derartige LP-Relaxation wurde nicht implementiert.

3.5 Zeitkomplexität

3.5.1 Bron-Kerbosch Algorithmus

Nach Eppstein, Löffler und Strash [5] hat der BK-Algorithmus eine Worst-Case Laufzeit von $O(3^{n/3})$, da in einem ungerichteten Graph mit n-Knoten maximal $3^{n/3}$ Cliquen existieren; die Worst-Case Laufzeit ist somit exponentiell. Der Average-Case des Algorithmus ist typischerweise deutlich besser; vor allem für

unsere Kontextualisierung der Stile mit der entsprechenden Eingabegröße n ist auch schon die Worst-Case Laufzeit akzeptabel.

$$O(3^{\frac{n}{3}}) \tag{8}$$

3.5.2 ILP-Solver

Wie gesagt, sind ILP's nach Cormen u. a. [4] NP-schwer. Dementsprechend gibt es nur heuristische oder approximative Algorithmen, die in polynomieller Zeit ILP's lösen. Welche genaue Zeitkomplexität der ILP-Algorithmus hat, hängt vom ILP-Solver ab.

Kommentar zur Umsetzung Der ILP wurde in der Programmiersprache Python 3.11.9, mit der Bibliothek PuLP erstellt und berechnet. „Linear programing is done via the Revised Simplex Method [...]." [8] Der Simplexalgorithmus hat nach Goldfarb [6] eine exponentielle Laufzeit im Worst-Case.

$$O(2^n) \tag{9}$$

3.5.3 Rückführungsalgorithmus

Der Algorithmus besteht hauptsächlich aus drei verschachtelten Schleifen, abhängig von c, t, k. Wenn c die Mächtigkeit der Cliquenmenge, t die Anzahl der Sorten und k die Mächtigkeit der Schnittmenge $C_i \cap T_j$, dann gilt:

$$O(c \cdot t \cdot k) \tag{10}$$

4 Ergebnisse

Nachdem zwei Lösungsansätze für das SPP vorgestellt und diskutiert wurden, werden im Anschluss ihre Lösungen und Ergebnisse in den Aspekten der Lösungsqualität und der Laufzeit analysiert und verglichen, um anschließend eine Argumentationsgrundlage für eine Auseinandersetzung mit den sich auftuenden Vor- und Nachteilen zu bieten.

4.1 Beispiele

Alle Beispiele sind durch den Bundeswettbewerb erzeugt und verbreitet worden. Die Bearbeitung und Dokumentation dieser ist fester Bestandteil der Dokumentation im Rahmen der Einreichung[5]. In der Wettbewerbsaufgabe gab es keine ausführliche Auseinandersetzung mit den Ergebnissen; tatsächlich waren die Laufzeiten für die unterschiedlichen Beispielaufgaben nicht angegeben. Des Weiteren wurde aufgrund der Laufzeit, die für die größeren Beispiele vermutlich Tage, bis Wochen betragen hätte, ein „Overflow“ eingeführt[6]. Dabei handelt sich um einen Parameter, der eine Toleranz für den Algorithmus einführt, frühzeitig zu terminieren, wenn $100\% - Overflow$ verpackt wurden. Dies ist vor allem beim Vergleich der großen Eingaben erkenntlich, da dort ein signifikanter Unterschied in den Ergebissen zu sehen ist.

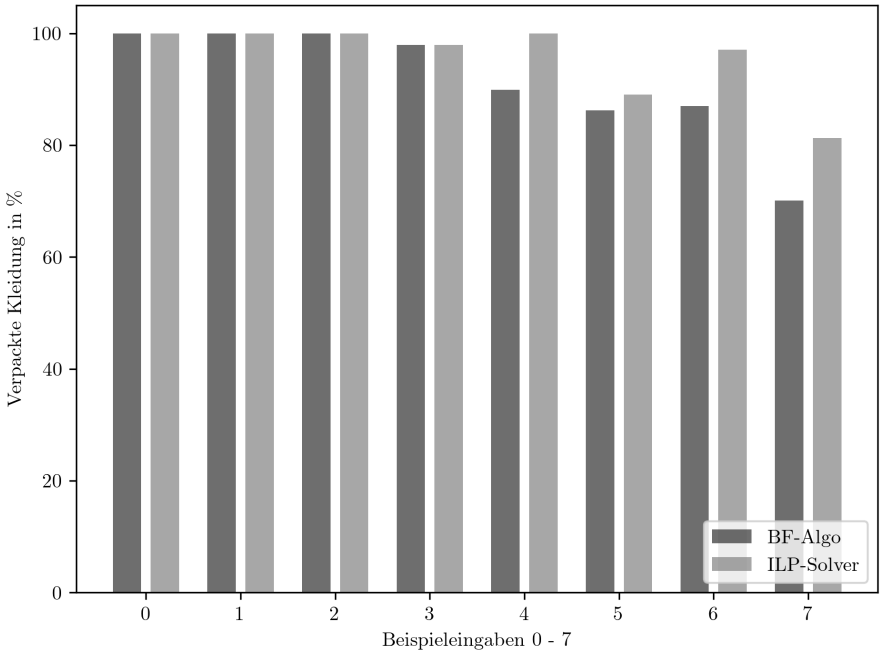

Abbildung 1: Vergleich der Lösungen der Beispielaufgaben 0 - 7. Siehe Anhang 7.2

Während die ersten 4 Eingaben (0-3) an der Qualität die exakt selben

[5]siehe Anhang 7.4
[6]siehe Anhang 7.4, S. 5

Ergebnisse liefern, lässt sich erkennen, dass der BF-Algorithmus für größere Eingaben[7] durch diese Overflow-Toleranz statistisch signifikante Abweichungen aufweist. Dies lässt sich grundlegend nicht stark verändern, da für kleinere Toleranz unverhältnismäßig höhere Laufzeiten anfallen. Nun wird ein weiteres Problem des BF-Ansatz deutlich, denn dieser kann seine Lösung nur in die der idealen einordnen, wenn der gesamte Callstack bearbeitet wurde. Interpretiert bedeutet das, dass dieser nicht weiß, ob es überhaupt möglich ist 100% der Kleidung in Boxen zu verpacken. So kann es durchaus sein, dass für die heuristischen Lösungen (4-7) im Suchbaum bereits eine ideale Lösung existierte, die allerdings nicht als solche identifiziert werden konnte. Durch das LGS des ILP-Solvers ist es hingegen simpel festzustellen, ob es überhaupt möglich ist, alle Kleidungsstücke vollständig zu verpacken. Auch der ILP-Solver hat nicht für alle Beispielaufgaben Lösungen gefunden, die 100% der Kleidung verpacken, da es durch die Konfiguration nicht möglich gewesen wäre, allerdings welche, die den verpackten Anteil maximieren. So ist festzustellen, dass der ILP-Solver bessere Lösungen hervorbringt als der BF-Algorithmus.

4.2 Laufzeitanalyse

Der ILP-Solver liefert maßgeblich bessere Ergebnisse als der BF-Algorithmus. Diese Aussage wird nun unter dem Aspekt der Laufzeit betrachtet, der die vorangegangene Behauptung entweder bestärken oder entkräften kann. Auch

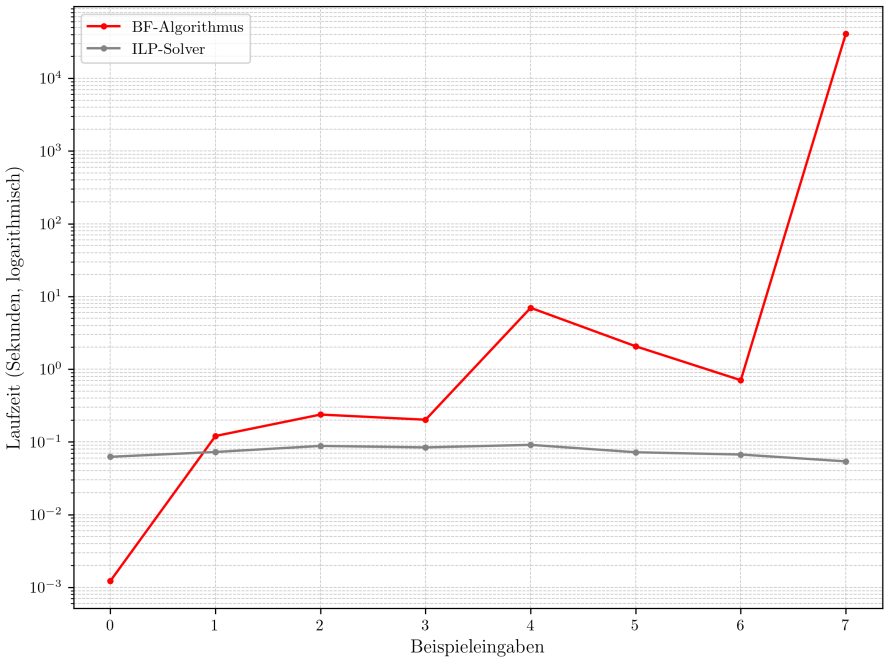

Abbildung 2: Vergleich der Laufzeiten der Beispielaufgaben 0 - 7. Siehe Anhang 7.3

[7]Eingaben sind ungefähr geordnet nach Eingabegröße. Siehe Anhang 7.1

wenn die logarithmische Skala das Ergebnis im Sinne des BF-Algorithmus ein wenig beschönigt, stechen zwei Dinge stark hervor. Zum einen ist, für den BF-Algorithmus, ein Anstieg der Laufzeit mit der Eingabegröße zu erkennen. Zum anderen weist der ILP-Solver für die steigende Eingabegröße so gut wie keine signifikanten Schwankungen und Veränderungen auf. Des Weiteren ist ein Abfall der Laufzeit des BF-Algorithmus' von Beispiel 4-6 zu erkennen.
Folgend sollen sowohl Laufzeit als auch die Ergebnisse an sich in Kontext zueinander gestellt werden und daraus entstehende Schlüsse auf die Verfahren folgen.

4.3 Diskussion

Obwohl sich beide Algorithmen erwiesenermaßen[8] nicht in polynomieller Zeit lösen lassen, kann aus Abbildung 2 eine der konstanten Laufzeitentwicklung ähnlichen abgelesen werden. Dies lässt sich durch die Problemmodellierung aus Kapitel 3.3 und 3.4 begründen, da aus dieser nur wenige Nebenbedingungen mit dementsprechend wenigen Ungleichungen entspringen. Die wachsende Eingabegröße der Stückzahlen von Kleidungsstücken ist durch die Weise, wie das ILP aufgestellt wird, nur wenig relevant. Die Zielfunktion in Gleichung (1) dessen Summe von der Menge N anhängig ist, hängt kausal nicht mit der Stückzahl der Kleidung zusammen, hauptsächlich jedoch mit der Dichte des Stilgraphen aus der die Cliquenmenge C generiert wird, deren Mächtigkeit durch Gleichung (5) durch $C_i \cap T_j$ an Bedeutung. Jegliche Ungleichungen, die aus dieser Formulierung hervorgehen, profitieren von der in den Beispielen niedrigen Anzahl von Stilen und Sorten. Ein Einblick in die Beispieleingaben bestätigt diese Vermutung: Für das Beispiel 4 ist die Laufzeit des ILP geringfügig größer[9], da das Beispiel 4 unter allen die größte Anzahl an Stilen und Sorten vorweist[10]. Erst in der Lösungsrückführung auf's SPP spielt die Stückzahl eine Rolle; diese allerdings ist polynomiell und somit nicht maßgeblich auswirkend für Stückzahlen bis 770[11]. Aus der Zeitkomplexität in Kapitel 3.2.1 ist zu schließen, dass die in Abbildung 1 abgebildete tatsächliche Laufzeit dem entspricht, obgleich sie durch die Overflow-Toleranz bereits abgefälscht wurde. Sowohl Laufzeit als auch generierte Lösung ergeben, dass die Formulierung als ILP dem BF-Ansatz klar überlegen ist, vor allem in Anbetracht der Kontextualisierung der eigentlichen Aufgabe. Mit Bezug auf die Realität, ist die Zahl der Kleidungsstücke nicht eingeschränkt bzw. nur schwach, wohingegen die Stil- und Sortenmenge durch den Anspruch einer realistischen Eingabegröße liminiert ist. Der BF-Algorithmus hingegen ist abhängig von der Eingabegröße q, der Anzahl der Kleidungsstücke, da jedes dieser als einzelne Entität betrachtet wird und somit das Verfahren belastet. Dies impliziert jedoch auch, das kleine Eingaben vom BF-Algorithmus schneller als vom ILP in eine ideale Lösung umgesetzt werden, wie auch aus Abbildung 2 für Beispiel 0 hervorgeht. Dort

[8] siehe Kapitel 3.2.1 und Cormen u. a. [4, S. 853]
[9] siehe Anhang 7.3
[10] siehe Anhang 7.1
[11] siehe Anhang 7.1, Beispiel 6

wird nochmals deutlich, dass der Algorithmus stark von der Eingabegröße q abhängt: für das Beispiel 0 mit $q_0 = 11$[12] und das Beispiel 1 mit $q_1 \approx 3 \cdot {q_0}$[13] divergiert die Laufzeit ca. um den Faktor 100 [14]. Durch den Vergleich von Beispiel 1 und 2 kann zudem festgestellt werden, dass der Einfuss der Anzahl der Stile auch nicht unerheblich für den BF-Algorithmus ist: Für die Eingabegröße s, die Anzahl der Stile weist das Beispiel 1 mit $s_1 = 4, q_1 = 499$[15] im Vergleich mit Beispiel 2 mit $s_2 \approx 2 \cdot s_1, q_2 = \frac{8}{125} q_1$ eine ähnliche Laufzeit[16] vor.

Für das SPP geht der ILP-Solver als klar geeignetere Lösungsmethode hervor.

[12] siehe Anhang 7.1
[13] siehe Anhang 7.1
[14] siehe Abbildung 2
[15] siehe Anhang 7.1
[16] siehe Abbildung 2

5 Bundeswettbewerb Informatik

Der Ursprung dieser BeLl liegt in der Wettbewerbseinreichung für die 2. Runde des 42. Bundeswettbewerb Informatik. Demnach wird im Anschluss die Bewertung der eingereichten Lösung reflektiert, die Lösung in den Kontext einer BwInf-Aufgabe eingeordnet und Rückschlüsse auf Fehler in der Methodik bei der Lösungserstellung gezogen.

5.1 Bewertung & Reflexion

Jede eingereichte Aufgabe der zweiten Runde wird in einem festgelegten Raster bewertet. Dabei startet jede Aufgabe bei einem Grundwert von 20 Punkten. Anschließend können Punkte für jede Kategorie im Bewertungsraster hinzugefügt oder abgezogen werden. Die Bewertungsweise, nach welcher Extrapunkte gegeben, oder Abzüge gemacht werden, ist dabei nicht öffentlich einsehbar und intransparent [10]. Im Anschluss sind die Kategorien und Unterkategorien, auf welche während der Bewertung eine Abweichung der Normpunkte gegeben wurde, knapp angeführt.

Kategorie	20/20
Lösungsweg	
Laufzeit des Verfahrens in Ordnung	-2
Speicherbedarf des Verfahrens in Ordnung	-2
Theoretische Analyse	
Gute Überlegungen zur Laufzeit des Verfahrens	+1
Dokumentation	
Vorgegebene Beispiele Dokumentiert	-1
Ergebnisse nachvollziehbar dargestellt	-2
Gesamtbewertung	13/20

Für die Überlegungen zur Zeitkomplexität[17] wurde +1 Punkt gegeben. In der Ursprungsbearbeitung der Aufgabe wurde für jeden Teilalgorithmus der Gesamtlösung eine kurze Überlegung, eventuell sogar nur ein Kommentar in der „Big O-Notation“ angegeben.[18] Aus dieser Punktegebung geht hervor, dass diese zumindest qualitativ nicht fernab der Wahrheit lagen. Des Weiteren gibt es eine Kategorie in der Wertung, die sich auf die Betrachtung in NP bezieht. Trotz kurzer Betrachtung und eines Versuchs des Beweises des Problems in NP[19] wurden dafür keine Zusatzpunkte erteilt. Dies mag sowohl an der fehlerhaften bzw. mangelhaften Herleitung dieser liegen, als auch an der mangelnden Formalität selbigen Beweises. Wie aus der Zeitkomplexitätsbetrachtung

[17] in der Bewertung „Gute Überlegungen zur Laufzeit des Verfahrens“ genannt
[18] siehe Anhang 7.4, S. 3-5
[19] siehe Anhang 7.4, S. 2

des BF-Ansatzes aus Kapitel 3.2.1 hervorgeht, ist dieser nicht dafür gedacht und geeignet, für größere Eingaben besonders zeiteffizient zu sein. Für rekursive Brute-Force-Algorithmen gilt allgemein im Worst-Case $O(n^k)$ wenn n die Verzweigung und k die maximale Tiefe des Suchbaums ist. Aus der Implementierung[20] geht hervor, dass der Lösungsansatz zusätzlich viele Redundanzen aufweist, welche zusätzlich seine Laufzeit stark verschlechtert. Dementsprechend sind die 2 Punkte Abzug gerechtfertigt. Auch der Speicherbedarf ist durch die rekursive Natur für große Eingaben enorm. Vor allem der Tiefensuchenaspekt sorgt für einen sehr „vollen“ Callstack. Da dieser vor allem für bestimmte Eingaben viele Redundanzen aufzeigt, ist auch dafür der entsprechende Abzug von 2 Punkten gerechtfertigt. Diese Abzüge in der Kategorie „Lösungsweg“ sind durch die Natur des Selbigen nötig. Die insgesamt 3 Punkte Abzug für die Kategorie „Dokumentation“ sind hingegen nicht alle zwingend, im Sinne des Ursprungs der Dokumentation, nötig gewesen. Durch den ineffizienten Lösungsweg waren die letzten beiden und auch größten Beispieleingaben nicht lösbar. Aufgrund der außerordentlich hohen Laufzeit wurden diese frühzeitig terminiert; somit lag für diese keine Lösung vor. Dafür kann der Abzug gerechtferigt werden. Für die Nachvollziehbarkeit der Ergebnisse ist, meines Ermessens nach, die Nummerierung der Kleidungsstücke verantwortlich. Beim Einlesen der Beispieldatei, die jedes Kleidungsstück, beschrieben in Kapitel 2.1, in jenem Format, für jedes unterschiedliche Kleidungsstück eine neue Zeile, darstellt, wird dementsprechend jedes Kleidungsstück gemäß der neuen Zeile von 0 an nummeriert. Im Kapitel 3 „Beispiele“[21] meiner Einreichung, werden die Pakete als Listen mit den entsprechenden Nummern der Kleidungsstücke angegeben. Diese Formatierung geht lediglich aus dem Programm und nicht aus einer erforderlichen Erklärung hervor. Dies hätte durch eine ausführlichere Dokumentationsweise einfach behoben werden können. Des weiteren hätten approximierte Ergebnisse für die Beispiele 6 & 7 angegeben werden können, um somit zwar schlechte, aber dennoch dokumentierte Ergebnisse vorweisen zu können, als auch überhaupt ein Ergebnis für Beispiel 4. Aus der Lösungsdiskussion des Bundeswettbewerbs Informatik geht hervor, dass aufgrund der Einordnung des Problems in ihre Komplexitätsklasse auch nicht perfekte, lediglich gute bzw. befriedigende, durch ihren Lösungsweg begründete Ergebisse, hinreichend sind. So lässt sich sagen, dass zumindest 2 Punkte, womöglich sogar 3 Punkte durch eine präzisere Dokumentation in der Bewertung nicht ins negative Gewicht hätten fallen müssen. Generell hätte ein anderer Lösungsansatz, womöglich heuristischer Natur, nötig sein müssen, um derartige Probleme zu lösen. Diese Erkenntnis hätte womöglich früher getroffen werden können, wenn die eigentlich vorangehende theoretische Analyse der Komplexitätsklasse sorgfältiger durchgeführt worden wäre. Diese Relevanz der Analyse wurde unterschätzt. Des weiteren hätte, auch ohne jene Analyse, nach Testen der Beispiele und vor allem nach der Analyse der Zeitkomplexität des Algorithmus festgestellt werden müssen, dass dieser nicht geeignet für die gegebene Aufgabe war. In der ursprünglichen Wettbewerbseinreichung wurde

[20] siehe Anhang 7.4, S. 3-6

[21] siehe Anhang 7.4, S. 6-10

ein hier nicht weiter erwähnter Greedy-Algorithmus geschrieben, der allerdings schon für kleine und einfache Beispiele, im Vergleich zur Ideallösung, ermittelt durch den BF-Algorithmus, schnell sehr schlechte Ergebnisse geliefert hat. In jedem Fall hätten die Einreichungen von mehr Zeit stark profitiert. Wie im Kapitel 5.3 weiter kommentiert wird, war der Fokus in der Entwicklung der Lösungsansätze, falsch gesetzt, was, trotz guten oder hinreichenden Lösungen für kleinere Eingaben, zu schlechten Bewertungen geführt hat.

5.2 Lösungsdiskussion

Wie schon mehrfach erwähnt und bearbeitet wurde, ist ein DFS-BF-Algorithmus als Lösungsansatz für die Aufgabe verwendet worden. In dieser Lösungsdiskussion soll nicht der Algorithmus in seinen pragmatischen Vor- und Nachteilen analysiert werden, stattdessen in den Kontext des Wettbewerbs und den Anforderungen der Aufgabenstellung und der gegebenen Beispiel gestellt werden. Viele, wenn nicht alle Aufgaben der 2. Runde des Bundeswettbewerbs Informatik fallen in gewisse Muster; entweder sind sie ein NP-schweres Problem oder eine sehr frei gestellte Aufgabe, die individuell modellierbar ist und keine klaren Ansprüche setzt. Die hier behandelte Aufgabe fällt definitiv in die erste Kategorie der Aufgaben. Derartige Aufgaben haben spezifische Anforderungen an ihre Lösungen. Sie sollten in der Regel:

1. die NP-Schwere anerkennen,
2. mit ihr umgehen,
3. die Zeitkomplexität der Lösungsqualität anpassen, wobei
4. die Laufzeit in einem akzeptablen Rahmen liegt.

Zunächst einmal muss die erwähnte NP-Schwere durch den Lösungsansatz anerkannt werden. Das bedeutet, dass eben jener BF-Ansatz nicht geeignet ist, da die Natur des Lösungsansatzes schon offensichtlich diese Uneignung impliziert. Dementsprechend war der gewählte Ansatz von Grund auf nicht für eine derartige BwInf-Aufgabe anwendbar. Eine Lösung die jenes jedoch anerkennt, zieht auch Punkt 2 zwingend nach sich. Jede Lösung die durch andere Weisen versucht, das Problem zu lösen und somit die Komplexität anerkennt, geht auch in dem Rahmen mit ihr um, indem in den meisten Fällen Ergebnisse generiert werden, die zwar nicht in der Menge der idealen Lösungen liegen, jedoch mit der enormen Laufzeit und somit mit der NP-Schwere umgehen, die die Aufgabe impliziert. Anschließend an die Laufzeit muss die Zeitkomplexität stets der Lösungsqualität angepasst sein; das bedeutet, dass eine vergleichsweise schlechtere Lösung zwingend eine geringere Zeitkomplexität aufweisen muss, um als valider Lösungsweg ratifiziert zu werden. Vor allem für gute Lösungswege kann die Laufzeit schnell durch nicht polynomielle Laufzeiten enorm lang werden. Dies ist relevant, um zu verstehen, dass die Lösungsqualität nicht das Maß aller Dinge ist, wenn durch die Zeitkomplexität die Laufzeit stark darunter leidet. All dies hat Relevanz für den Lösungsansatz, weil dieser ohne dieses Wissen bzw, die Erfahrung formuliert wurde. Die Lösungsqualität war,

während der Suche nach einer Lösungsidee, vor allen anderen Metriken die Wichtigste. Fälschlicherweise ist somit vor allem die Laufzeit in den Hintergrund geraten; dass eine nicht ideale Lösung, generiert durch Heuristiken und co. auch begründet und akzeptabel sein kann, war mir nicht bewusst. Rückblickend wären wahrscheinlich, vor allem die großen Eingaben betreffend, andere Wege, wie sie letztendlich auch aus dieser Arbeit hervorgehen, gesucht worden, die deutlich bessere Lösungen erzeugen. Es ist nochmals zu erwähnen, dass der BF-Ansatz an sich ein akzeptabler und valider Lösungsansatz ist - er ist nur für eine Aufgabenumgebung, die derartige BwInf-Aufgaben bieten, meistens nicht geeignet.

5.3 Kommentar zur Aufgabe 3: Siedler

Zu einer Wettbewerbseinreichung für den Bundeswettbewerb Informatik gehören genau zwei bearbeitete Aufgaben. Als zweite Aufgabe wurde dabei SSiedleräusgewählt. Abstrahiert wird das Problem folgendermaßen formuliert:

Siedler-Problem. *Gegeben ist ein n-Polygon P. Es kann zudem ein Kreis K mit dem Radius $r = 85$ und dem Mittelpunkt m in dieses Polygon platziert werden. Nun sollen, innerhalb von K, Kreise mit $r = 5$ und außerhalb K und innerhalb von P, Kreise mit $r = 10$ platziert werden. Wenn x nun die Summe aller platzierten Kreise, ausgeschlossen K darstellt, soll x maximal sein.*

Formuliert wurde das Problem in der Dokumentation als Packungsproblem. „Wie auch in anderen Packungsproblem versuchen wir also, in einen festgelegten Behälter die maximale Menge an Objekten unterzukriegen. Die Ähnlichkeit des Siedler-Problems mit Packungsproblemen kann dabei helfen, die Schwierigkeit des Problems einzuordnen und evtl. einen Beweis auf seine Komplexitätsklasse zu führen.“[22] Das führte bei der Bearbeitung schließlich zu Kreispackungen in der Mathematik in unendlichen Räumen. Es ist bewiesen, dass die maximale Dichte für jene Kreispackung $p = \frac{\pi \cdot \sqrt{3}}{6} \approx 0.907$ beträgt [9]. Dieses Muster wurde durch einen simplen Algorithmus, der von links nach rechts über das Polygon in kleinen Schritten iteriert ist, repliziert, indem immer ein Kreis platziert wurde, wenn es möglich war. Weil eben dieses Muster, dessen genaue Form an dieser Stelle nicht genauer beschrieben wird, durch dieses Verfahren generiert wurde, war dies die geführte Argumentation für den Algorithmus. Für folgendes wurden Punkte abgezogen:

Kategorie	20/20
Lösungsweg	
Verfahren nicht unnötig ineffizient	-3
Speicherbedarf des Verfahrens in Ordnung	-2
Verfahren mit guten Ergebnissen	-4
Gesamtbewertung	11/20

[22] siehe Anhang 7.4, S. 2

Eine genaue Bewertungsreflexion wäre im Rahmen dieser Arbeit nicht angebracht. Allerdings lässt sich sagen, dass sich der 3 Punkte Abzug nicht zwingend durch die Laufzeit zu begründen ist. Die längste Laufzeit für das aufwendigste Beispiel betrug ca. 67 Sekunden. Es ist jedoch ein Kernproblem des Lösungsansatzes hervorzuheben: es gibt keine klare Methodik und wird auch nicht durch den Algorithmus an sich geliefert, um die Qualität einer Lösung festzustellen. Dies sollte auch die 4 Punkte Abzug in der dazugehörigen Unterkategorie begründen. Dieser Sachverhalt hebt einen weiteren Fokuspunkt dieser Aufgabentypen hervor; eine Lösung sollte stets offenlegen, inwiefern die Lösung an eine Lösung der Idealmenge herankommt und warum und auf welche Weise ihre hervorgebrachten Ergebnisse nicht jener Menge entsprechen. Dies lässt sich nachträglich zu den Eigenschaften jener theoretischen Probleme, wie sie in Kapitel 5.2 genannt wurden, hinzufügen.

6 Fazit

Diese Arbeit sollte eine Fortführung der Wettbewerbseinreichung des 42. Bundeswettbewerbs Informatik darstellen, indem für die Aufgabe 2: „Stilvolle Päckchen“ neue und bessere Lösungsansätze formuliert, analysiert und verglichen werden sollten. Durch die Formulierung eines ILP, gelingt es eine Lösungsidee vorzustellen, deren Ergebnisse die ursprüngliche Lösung um mehrere Größenordnungen übertreffen. Es wird erkannt, dass eine starke Abhängigkeit von der Eingabegröße q nicht vorteilhaft für die Ergebnisgewinnung sein kann, da, im Gegensatz zu Sorten- und Stilmenge, die Kleidungsmenge q stark schwankt.[23] Auf dieser Erkenntnis lässt sich begründen, wie die Formulierung des ILP zu so guten Ergebnissen führen konnte, da sie keine (starke) Abhängigkeit von q aufweist, da die Zuordnung der Kleidungsstücke auf die Cliquen nur unerheblich von der Menge q beeinflusst wird. Nach einer Reflexion über die Bewertung der Aufgabe durch den Wettbewerb wird erkannt, dass ein derartiger BF-Algorithmus, wie er sowohl hier als auch in der Einreichung formuliert wurde, nicht geeignet im Kontext des Bundeswettbewerbs Informatik ist und nicht hinreichend mit der NP-Schwere des Problems umgeht. Der ILP hingegen ist durch seine Modellierung angepasst auf die ursprüngliche Problemformulierung, indem er Abhängigkeiten hauptsächlich auf Sorten- und Cliquenmengen bezieht und somit eine fast konstante Laufzeit über alle Beispiele hinweg aufweisen kann.
Wie durch Kapitel 3.4.2 erwähnt, sind Erweiterungen und Optimierungen des ILP's möglich. Zum einen lässt sich dieser zu einem LP relaxieren, um vor allem für größere Eingaben effizientere Laufzeiten zu erzeugen, zum anderen wären vor allem Optimierungen des Rückführungsalgorithmus und des Bron-Kerbosch Algorithmus wie durch Eppstein, Löffler und Strash [5] formuliert, einflussreicher für die Ergebnisgewinnung im Rahmen dieser Arbeit.

[23] siehe Anhang 7.1

Literatur

[1] Coen Bron und Joep Kerbosch. „Algorithm 457: Finding all cliques of an undirected graph“. In: *Communications of the ACM* 16.9 (1973), S. 1–5. DOI: 10.1145/362342.362367.

[2] Randy Carraghan und Panos M. Pardalos. „An exact algorithm for the maximum clique problem“. In: *Operations Research Letters* 9.6 (1990), S. 375–382. ISSN: 0167-6377. DOI: https://doi.org/10.1016/0167-6377(90)90057-C. URL: https://www.sciencedirect.com/science/article/pii/016763779090057C.

[3] CMU$_{S}$*chool*. *Lecture #18: LP Algorithms, and Seidel's Algorithm*. Lecture Notes. 15-451/651: Design & Analysis of Algorithms, Carnegie Mellon University. 2017. URL: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.cs.cmu.edu/~15451-f17/lectures/lec18-lp2.pdf.

[4] Thomas H. Cormen u. a. *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009, S. 850–866. ISBN: 0262033844.

[5] David Eppstein, Maarten Löffler und Darren Strash. *Listing All Maximal Cliques in Sparse Graphs in Near-optimal Time*. arXiv:1006.5440v1 [cs.DS]. 2010. URL: https://arxiv.org/abs/1006.5440v1.

[6] Donald Goldfarb. „On the Complexity of the Simplex Method“. In: *Advances in Optimization and Numerical Analysis*. Hrsg. von Susana Gomez und Jean-Pierre Hennart. Dordrecht: Springer Netherlands, 1994, S. 25–38. ISBN: 978-94-015-8330-5. DOI: 10.1007/978-94-015-8330-5_2. URL: https://doi.org/10.1007/978-94-015-8330-5_2.

[7] John D. C. Little u. a. „An Algorithm for the Traveling Salesman Problem“. In: *Operations Research* 11.6 (1963), S. 1–4. ISSN: 0030364X, 15265463. URL: http://www.jstor.org/stable/167836 (besucht am 16. 12. 2024).

[8] Stuart Mitchell u. a. *Integer Programming*. Email: s.mitchell@auckland.ac.nz, pchtsp@gmail.com. Accessed: 2024-12-18. n.d. URL: https://coin-or.github.io/pulp/main/optimisation_concepts.html#integer-programing.

[9] Gabriele Nebe. *Kugelpackungen*. https://www.math.rwth-aachen.de/~Gabriele.Nebe/papers/Kugelpackungen.pdf. Zugriff am 18. Dezember 2024.

[10] Moritz Schwalm. *Besondere Lernleistung: Einsendung zur 2. Runde des 41. Bundeswettbewerbs Informatik*. Eingereicht am 14.12.2023. 2023.

7 Anhang

7.1 Beispielaufgaben

Beispiele	0	1	2	3	4	5	6	7
Anzahl der Stile	2	4	9	10	24	21	10	10
Anzahl der Sorten	3	3	3	5	7	5	5	5
Anzahl der Kleidung	11	499	32	96	437	174	770	700

7.2 Lösungen

Beispiele	0	1	2	3	4	5	6	7
BF-Algo	2	4	9	10	24	21	10	10
ILP-Solver	3	3	3	5	7	5	5	5

Angaben in %.[24]

7.3 Laufzeiten

Beispiele	0	1	2	3	4	5	6	7
BF-Algo	1.2	120	237	201.	6974	2062	704	40919$\widehat{=}$11 std.
ILP-Solver	62	72	87	83	80	71	66	53

Angaben in ms.[25]

7.4 Wettbewerbseinreichung

[24] Anteil verpackter Kleidung

[25] Wenn nicht anders angegeben.

Aufgabe 2: Stilvolle Päckchen

Teilnahme-ID: 70345

Bearbeiter/-in dieser Aufgabe:
Felix Du

15. April 2024

Inhaltsverzeichnis

1 Lösungsidee

1.1 Betrachtung als Problem der Graphentheorie

Beim Lösen des Stilvolle Päckchen Problems stößt man leicht auf die Frage, wie man die vorgegebenen Daten darstellen sollte. Dabei habe ich mich für einen Graphen G entschieden, der in der Regel durch eine Menge Knoten N und eine Menge Kanten E beschrieben wird. In diesem Fall besitzt jeder Knoten in der Menge N weitere zwei Eigenschaften, Sorte und Stil. Wenn Stile verschiedener Knoten kompatibel miteinander sind zwischen ihnen eine Kante konstruiert. Der Graph ist ungerichtet. Jedes Kleidungsstück ist sein eigener Knoten, das heißt, dass es genau so viele Knoten wie Kleidungsstücke gibt. Wenn nun bei jeder Iteration Pakete gepackt werden, dann wird immer ein Subgraph H aus dem Graph G entfernt. So kann jedes gepackt Paket ebenfalls als Graph dargestellt werden.

Nun kann man die Randbedingungen der Aufgabe auf Eigenschaften eines Graphen übertragen. Alle Kleidungsstücke eines Pakets müssen miteinander kompatibel sein. Weil Knoten immer dann benachbart sind, wenn ihre Stile miteinander kompatibel sind, muss jeder Subgraph eines Pakets ein kompletter Graph, auch bezeichnet als Clique, sein. Das bedeutet, dass ein Knoten n ein Nachbar jedes anderen Knoten in $N\backslash\{n\}$ sein muss. Außerdem muss innerhalb eines Subgraphen jede Sorten vertreten, jedoch nicht öfters als 3 Mal.Da die Pakete so gepackt werden sollen, dass der Verschleiß, sprich Kleidung, die übrig bleibt, minimal sein soll, ist das Optimierungsproblem Stilvolle Päckchen ein Problem der Grpahentheorie, in dem Cliquen mit diesen Vorgaben so entfernt werden sollten, sodass möglichst wenig Knoten übrig bleiben.

1.2 Kommentar zur Komplexitätsklasse

Verwandte Graphen und Cliquen Probleme befinden sich typischerweise mindestens in NP. Es folgt ein Versuch eine gegebene Lösung des Problems zu verifizieren. Es ist außerdem aufgrund der NP-schwere verwandter und unkomplexerer Probleme davon auszugehen, dass dieses Problem ebenfalls mindestens NP-vollständig ist. Die für den beweis erforderliche Reduktion wurde jedoch nicht geführt.

1.2.1 Beweis in NP

Wenn sich ein Problem in NP befindet, dann ist es möglich durch ein Zertifikat eine gegebene Lösung zu verifizieren. :
"The class NP consists of those problems that are 'verifiable' in polynomial time. What do we mean by a problem being verifiable? If you were somehow given a 'certificate' of a solution, then you could verify that the certificate is correct in time polynomial in the size of the input to the problem"[1]
Gegeben sei eine Lösung L durch eine Menge P an gepackten Paketen und eine Menge R an Restbeständen. Wenn die Summe jedes Kleidungsstücks mit der Ursprungssumme übereinstimmt und alle Stilrichtungen in einem Paket miteinander kompatibel sind, ist die Lösung verifiziert. Die Rückführung zum Ursprungsgaphen ist in $O(1)$ umsetzbar und die das Prüfen der Stilrichtungen hängt von der Menge der Pakete P ab, als $O(P)$.

1.3 heuristischer Greedy-Algorithmus

Eine sehr zeiteffiziente Lösung ist die heuristische Lösung durch einen Greedy-Ansatz. Dabei wird für jeden Schritt immer der am wenigsten verbundene Knoten u im Graph genommen, und eine beliebige minimale Clique, die den Bedingungen entspricht und u enthält, aus dem Graphen entfernt, und als Paket verpackt. Die Annahme ist, dass es wahrscheinlicher ist, dass für wenig verbundene Knoten substantielle Partnerbindungen wegfallen, als für vielverbundene Knoten, sodass erst die wenig verbundenen Knoten verarbeitet werden. Da der Ansatz seine eigenen Handlungen im Nachhinein nicht revidieren und bewerten kann, ist dieser Ansatz sowohl in Zeit als auch in Platz äußerst effizient. Allerdings liefert er auch bei Eingaben mit moderater Schwierigkeit nur moderate Lösungen, da er nicht gesichtert nach dem Optimum sucht, sondern nur auf einer Annahme basierend, Heuristiken generiert.

1.3.1 Zeitkomplexität

Wie bereits angesprochen, ist der Algorithmus sowohl zeit- als auch platzeffizient. Der Platz ist unabhängig von der Inputgröße, da immer nur ein Speicher für den Graphen, und ein Speicher für bisherige Ergebnisse verwendet wird und ist somit $O(1)$. Da in jedem Schritt drei Knoten entfernt werden, ist die Zeitkomplexität für n Knoten maximal $O(n/3) = O(n)$.

1.3.2 Pseudocode

1.4 Brute-Force Algorithmus

Aufgrund der Ungenauigkeit des Greedy-Ansatzes habe ich einen Brute-Force Algorithmus entwickelt, der im Sinne eines BF-Algorithmus alle erdenklichen Möglichkeiten berechnet. Genauer besagt berechnet er jegliche Kombination von minimalen Cliquen, die die Packbedindungen erfüllen und fügt anschließend, wenn keine weiteren validen Cliquen mehr extrahierbar sind, die übrigen Knoten in die bereits entstandenen Pakete ein. Wenn es also ideale Lösungen ohne Verschleiß geben sollte, können sie so errechnet

[1] Cormen, Thomas H., Charles Eric Leiserson, and Ronald L. Rivest. Introduction to Algorithms. Cambridge, Mass. : New York: MIT Press ; McGraw-Hill, 1990. Seite 1065

werden. Vor allem für kleine Input-Größen ist dies offensichtlich der sichere und bessere Algorithmus, mit steigender Inputgröße jedoch, müssen Optimierungen und Abbruchbedingungen eingebaut werden, um überhaupt in fassbarer Laufzeit zu terminieren.
Der BF-Algorithmus wird im Endeffekt verwendet, um die Ergebnisse zu generieren.

1.4.1 Pseudocode

Algorithm 1 Find All Minimal Clique Combinations in a Graph

```
1:  global global_storage ← []
2:  function AllMinCliqueCombinations(Graph(N, E), paeckchen : list)
3:      for each vertex x ∈ N do
4:          min_cliques ← GetMinimalCliques(x)                ▷ Returns list
5:          if Length(min_cliques) = 0 then
6:              global_storage.append(paeckchen)
7:          end if
8:          for each clique y ∈ min_cliques do
9:              Copied_Graph ← Graph.copy()
10:             Copied_Graph.remove(y)
11:             new_paeckchen ← paeckchen.copy()
12:             new_paeckchen.append(y)
13:             AllMinCliqueCombinations(Copied_Graph, new_paeckchen)
14:         end for
15:     end for
16:     return global_storage
17: end function
```

1.4.2 Zeitkomplexität

Für die Anzahl an Knoten k und die Menge an minimalen Cliquen m von k , ist die Zeitkomplexität:

$$k \cdot O(m(k)^n) = O(m^n)$$

Dies geht aus der rekursiven Eigenschaft des Algorithmus hervor. Auch für den Platz ist dies kein Kinderspiel, denn für jede Iteration und jede neue Funktion auf dem Call Stack wird eine neue Liste erstellt. das Verfahren ist ganz klar nicht polynomial, weshalb Optimierungen und Abbruchbedingungen essentiell sind.

2 Umsetzung

Die Umsetzung wird in der Programmiersprache Python[2] realisiert. Hier wird nur der BF-Algorithmus für das Problem erläutert und diskutiert, da die generierten Heuristiken des Greedy-Algorithmus nicht zufriedenstellend waren und deshalb vorrangig mit dem BF-Algorithmus gearbeitet wurde. Zunächst wird der Graph für alle gegebenen Stile und ihre Kombinationen erstellt. Dies macht es im Verlauf des Verfahrens einfacher, Kombinationsmöglichkeiten festzustellen, Cliquen herauszuarbeiten oder die Kanten des Hauptgraphs zu erstellen. Illustrationen der Stilgraphen befinden sich im Anhang. Nachdem dies geschehen ist, werden die Kanten des Hauptgraphen mit Hilfe des Stilgraphen konstruiert. Die Graphendarstellung findet in der Python Bibliothek 'NetworkX' statt. Obwohl zunächst eine Adjazenzmatrix geplant war, mussten mit den Eigenschaften 'stil' und 'sorte' zu viele Informationen für einen Knoten gespeichert werden, sodass sich die Bibliothek anbot. Des weiteren wird die Anzahl gleicher Kleidungsstücke nicht mit Hilfe mehrerer Knoten mit gleichen Eigenschaften dargestellt, sondern stattdessen eine dritte Eigenschaft 'zahl' für jeden Knoten hinzugefügt, welche speichert, wie viele Stücke es von einem Kleidungsstück noch zur Verfügung gibt. Wenn gilt: $zahl = 0$ wird der entsprechende Knoten aus dem Graphen entfernt. Außerdem wird eine blacklist eingeführt, die speichert, wenn Knoten keine gültigen Cliquen mehr erzeugen können.

[2] Python 3.11.1

2.1 Erwähnenswerte Funktionen

Die Funktion *get_cliques*() generiert alle minimalen Cliquen mit der root_node n, sprich, alle Cliquen, die den Knoten n enthalten. Trotz schlechterer Laufzeit wurde hier auf ein rekursives Programmierparadigma zurückgegriffen, weil die Input-Größen im akzeptablen Rahmen liegen, sodass sie sich nicht allzu stark auf die Laufzeit auswirkt. Der Algorithmus iteriert über ein Python-Dictionary, welches als Schlüssel alle Sorten hat und als Werte alle benachbarten Knoten mit entsprechender Sorte speichert. Nun werden daraus alle möglichen Kombinationen gebildet; die Anzahl der Knoten einer Clique entspricht der Anzahl der Sorten.

Algorithm 2 Alle minimalen Cliquen mit root node n

```
1: function GetCliques(G : nx.classes.graph.Graph, neighbors_sorten : dict, iterator : int,
   sub_res : list, res : list, blacklist : list)
2:     if iterator = len(neighbors_sorten) then
3:         if Complete(G, sub_res) = False then
4:             pass
5:         else if len(blacklist) = 0 then
6:             res.append(sub_res)
7:         else
8:             bool_copy ← False
9:             for x in blacklist do
10:                if sub_res[0] ∈ x and Sorted(sub_res) = Sorted(x) then
11:                    bool_copy ← True
12:                    break
13:                end if
14:            end for
15:            if bool_copy = False then
16:                res.append(sub_res)
17:            end if
18:        end if
19:    else
20:        keys ← list(neighbors_sorted.keys())
21:        for x in neighbors_sorten[keys[iterator]] do
22:            lst ← sub_res.copy()
23:            lst.append(x)
24:            GetCliques(G, neighbors_sorten, iterator + 1, lst, res, blacklist)
25:        end for
26:    end if
27: end function
```

Output: res: list ⟶ Liste aller Kombinationen
Zeitkomplexität für $k = sorten\_num$ $O(k^n)$

Innerhalb von *get_cliques*() findet die Funktion *complete*() große Verwendung. Sie kontrolliert für jede Kombination an Knoten, ob diese Kombination einen vollständigen Subgraphen abbildet und somit eine Clique ist. Diese Funktion ist essentiell, weil sie unmittelbar kausal mit den Bedingungen zusammenhängt. Da laut Regeln alle Stile der Kleidungsstücke innerhalb eines Pakets miteinander kombinierbar sein müssen, muss auch jedes Kleidungsstück innerhalb eines Pakets benachbart zu allen anderen Knoten sein.

Algorithm 3 komplette-Subgraphen Algorithmus

```
1: function COMPLETE(Graph : nx.classes.graph.Graph, subNodes : list)
2:     sg ← Graph.subgraph(subNodes)
3:     workingGraph ← nx.Graph()
4:     workingGraph.add_nodes_from(n for n in sg.nodes())
5:     workingGraph.add_edges_from(n for n in sg.edges())
6:     for x in subNodes do
7:         if len(workingGraph.edges(x)) < len(subNodes) − 1 then
8:             return False
9:         end if
10:    end for
11: end function
```

Output: boolean ⟵ ist G ein kompletter Graph?

Der BF-Algorithmus ruft für alle Knoten n $get\_cliques(n)$ auf und iteriert über jede dieser verschiedenen Cliquen. Das bedeutet, dass er jeweils die ausgewählte Clique entfernt und innerhalb dieser Schleife über alle Cliquen stets sich selbst aufruft, bis irgendwann alle Knoten des übrig gebliebenen Graphs in der blacklist stehen (siehe 1.4.1 Pseudocode). Wenn dieser Punkt erreicht wurde, greift die Funktion $assign\_rest()$. Sie nimmt nun zum einen die bisherigen Päckchen, als auch die übrig gebliebenen Knoten als Eingabe und ordnet jeden Knoten iterativ nach Möglichkeit in ein Paket zu. Nun muss beachtet werden, dass in ein Paket jeweils nur drei Kleidungsstücke gleicher Sorte gepackt werden dürfen. In welche Pakete die übrig gebliebenen Knoten gepackt werden dürfen, kann anhand zweier Kriterien bestimmt werden:

1. Der Stil des übrig gebliebenen Knoten ist auch im Paket enthalten.
2. Der Stil des übrig gebliebenen Knoten ist mit jedem Stil des Pakets kombinierbar.

Das rein iterative Verfahren füllt zwar die Pakete unregelmäßig auf, jedoch ist dies einerseits nicht Kriterium, andererseits zum Zwecke der Laufzeit. Der Pseudocode soll an dieser Stelle eingespaart werden, da die Funktion zwar essentiell, jedoch keineswegs anspruchsvoll oder gar kompliziert ist; sie ist dafür aber extensiv lang. Es sei gesagt, dass die **Zeitkomplexität:** $O(n)$ für die Summe aller übrig gebliebenen Knoten ist.

Dieser gesamte Prozess ist nur für kleine Input-Größen halbwegs zeitlich ausführbar. Deshalb wurden multiple Abbruchbedingungen eingebaut, die die Laufzeit um ein Vielfaches verkürzen können.

2.2 Optimierungen

Eine offensichtliche Optimierung ist die Abbruchbedingung bei Rest = 0, sprich, wenn nach der Funktion $assign_rest()$ keine Knoten mehr übrig sind. In dem Fall können die Päckchenkonfigurationen als definitive ideale Lösung gewertet werden und alle noch anstehenden Callstack-Aufrufe der Funktion können als irrelevant bezeichnet werden. Diese einfache Implementation ermöglicht es, auch größere Inputgrößen zu verarbeiten, da ideale Lösungen teilweise schon in frühen Iterationen vorkommen. Um zu verhindern, dass die ersten Iterationen durch den Graphen nur die gleichen Cliquen sind, ist die Reihenfolge aller Knoten für jeden Aufruf der Funktion zufällig gewählt.

2.2.1 Overflow Toleranz

Eine weitere essentielle Rolle spielt die Overflow Toleranz, die zu Beginn des Programms eingegeben werden kann. Dabei handelt es sich um eine Prozentzahl, die festlegt, bei wie viel prozentualem Anteil Verschleiß, der Algorithmus terminieren darf. Um ein Beispiel anzuführen: Die Overflow Toleranz wird zu Beginn mit dem Wert 10 deklariert. Das heißt, dass der Algorithmus terminiert, auch wenn keine optimale Päckchenkonfiguration erreicht wurde, solange der prozentuale Verschleiß im Verhältnis zur Gesamtanzahl von Kleidung <= 10 ist. Somit wird es möglich, sich langsam an Konfigurationen mit akzeptabler Laufzeit ranzutasten und moderate Lösungen für große Input-Größen zu bekommen. Diese Optimierung ist essentiell für den sogleich beschriebenen Prozess der Weiteroptimierung.

2.2.2 Feststellen unverwendbarer Nodes

Gegeben seien die bisherigen gepackte Pakete P und die übrig gebliebenen, in den momentanen Paketen nicht mehr unterzubringenden, Knoten N. Nun wird für jeden übrig gebliebenen Knoten m geprüft, ob dieser zwangsweise übrig bleiben muss. Dies geschieht mit folgendem Verfahren: Zunächst werden alle möglichen Cliquen des Stils von m generiert. k sei die eigentliche Gesamtanzahl des übrig gebliebenen Kleidungsstücks. Nun wird für jede Stilkombination geprüft, ob es von jeder anderen Sorte mindestens $ceil(k/3)$[3] gibt. Wenn es nur von einer Sorte in einer Stilkombination zu wenig gibt, muss das Kleidungsstück in der jeweiligen Stilkombination übrig bleiben. Wenn das auf jede mögliche Stilkombination zutrifft, ist bewiesen, dass m zwangläufig übrig bleiben muss, unabhängig von der Packung der Pakete.
Das Prüfen in der jeweiligen Stilkombination ist mit O(1) kein Problem. Da dies für die Anzahl der Stilkombinationen j, j-mal geschieht, ist der Prozess des Prüfens für alle Stilrichtungen $O(j)$. Im worstcase ist die Zeitkomplexität des Generierens für die Stilkombinationen $O(k^{(k-1)})$, wenn k die Anzahl der verbindbaren Stile des Stils von m sind. Für unsere Inputgrößen gilt $0 < k < 25$; das ist zwar eigentlich inakzeptabel, ist jedoch nur der worst-case und wird nicht eintreffen, da ein Kleidungsstück, welches 24 verschiedene Stilpartner hat, mit sehr großer Wahrscheinlichkeit nicht übrig bleiben wird.
Mit dieser Methode können Lösungen, die vorerst nicht ideal scheinen, im Nachhinein als ideal bewiesen werden. Des weiteren gibt es noch einen großen Vorteil: Die generelle Abbruchbedingung liegt bei Verschleißmenge = 0; doch da in manchen Beispielen, wie soeben gezeigt, nicht zwangsläufig Lösungen mit solch einer Verschleißmenge existieren müssen, und somit nie über diese Abbruchbedingung terminieren, ist es schwer, als ideal verifizierte Lösungen zu generieren. Mit diesem Verfahren können auch Lösungen, die über die Overflow Toleranz termierten, als Ideal verifiziert werden. Zumindest kann die reelle Verschleißsumme bestimmt werden, bereinigt von ohnehin nicht verwertbaren Kleidungsstücken. Aus zeitlichen Gründen ist es mir jedoch nicht gelungen dies zu implementieren.

3 Beispiele

Die Kleidungsstücke werden beim Einlesen in Reihenfolge nummeriert, um besser mit ihnen arbeiten zu können. Die Zahlen referieren also in Reihenfolge auf Sorte und Stil in Zeile n+1 der Eingabe der einzelnen Kleidungsstücke. Vor allem Beispiel 5, 6 & 7 sind nicht ideal, hatten jedoch eine zu hohe Laufzeit und wurden dementsprechend mit der Overflow Toleranz generiert.

3.1 Beispiel 0

paeckchen0.txt

Elapsed times:
Elapsed time for creating Graphs & Inputs in sec: 0.0010350000000001192s
Elapsed time for running Algorithm: 0.0012240000000001139s

Ideale Lösung gefunden

Summe der gesamten Kleidung: 11
wie viel verwertet wurde: 11
Wie viel übrig geblieben ist: 0
Wie viele Pakete gepackt wurden: 3
Wie viel Prozent % der Gesamtkleidung verwertet wurde: 100.0%

Wie die Pakete gepackt werden: [2, 0, 3, 0, 1] [0, 2, 3] [1, 2, 3]

3.2 Beispiel 1

paeckchen1.txt

Elapsed times:
Elapsed time for creating Graphs & Inputs in sec: 0.0016700000000000603s
Elapsed time for running Algorithm: 0.12044699999999997s

[3] striktes Aufrunden

Ideale Lösung gefunden

Summe der gesamten Kleidung: 499
wie viel verwertet wurde: 499
Wie viel übrig geblieben ist: 0
Wie viele Pakete gepackt wurden: 124
Wie viel Prozent % der Gesamtkleidung verwertet wurde: 100.0%

Wie die Pakete gepackt werden: [1, 4, 8, 1, 1, 8, 8] [11, 2, 6, 2, 2, 7, 7, 11, 11] [1, 4, 8, 1, 1, 8, 8] [10, 1, 5, 1, 1, 9, 9] [0, 4, 8, 1, 1, 8, 8] [0, 4, 8, 1, 1, 8, 8] [7, 2, 10, 2, 2, 7, 7, 11, 11] [9, 0, 4, 1, 1, 8, 8] [2, 5, 9, 1, 1, 9] [10, 1, 5, 2, 2] [0, 4, 8] [5, 0, 8] [1, 4, 8] [1, 4, 8] [9, 0, 4] [5, 0, 8] [1, 4, 8] [0, 4, 8] [8, 0, 4] [9, 0, 4] [9, 0, 4] [6, 1, 9, 2, 2] [2, 5, 9, 2, 2] [11, 2, 6, 2, 2, 7, 7, 11] [10, 1, 5, 2, 2] [8, 0, 4] [9, 0, 4] [9, 0, 4] [8, 0, 4] [8, 0, 4] [7, 2, 10, 2, 2, 7, 7] [10, 1, 5, 2, 2] [6, 1, 9, 2, 2] [8, 0, 4] [7, 2, 10, 2, 2, 7, 7] [10, 1, 5, 2, 2] [0, 4, 8] [5, 0, 8] [9, 0, 4] [8, 0, 4] [9, 0, 4] [2, 5, 9, 2, 2] [5, 0, 8] [6, 1, 9, 2, 2] [10, 1, 5, 2, 2] [10, 1, 5, 2, 2] [10, 1, 5, 2, 2] [0, 4, 8] [2, 5, 9, 2, 2] [10, 1, 5, 2, 2] [2, 5, 9, 2, 2] [3, 6, 10, 2, 2, 7, 7] [3, 6, 10, 2, 2, 7, 7] [6, 1, 9, 2, 2] [4, 0, 8] [9, 0, 4] [3, 6, 10, 2, 2, 7, 7] [3, 6, 10, 2, 2, 7, 7] [4, 0, 8] [9, 0, 4] [11, 2, 6, 2, 3, 7, 7] [1, 4, 8] [7, 2, 10, 3, 3, 7, 7] [9, 0, 4] [7, 2, 10, 3, 3, 7, 7] [3, 6, 10, 3, 3, 7, 7] [10, 1, 5] [5, 0, 8] [11, 2, 6, 3, 3, 7] [6, 1, 9] [6, 1, 9] [3, 6, 10, 3, 3] [6, 1, 9] [0, 4, 8] [8, 0, 4] [2, 5, 9] [5, 0, 8] [7, 2, 10, 3, 3] [9, 0, 4] [10, 1, 5] [5, 0, 8] [10, 1, 5] [4, 0, 8] [5, 0, 8] [10, 1, 5] [0, 4, 8] [10, 1, 5] [4, 0, 8] [8, 0, 4] [8, 0, 4] [7, 2, 10, 3, 3] [11, 2, 6, 3, 3] [6, 1, 9] [1, 5, 8] [11, 2, 6] [11, 2, 6] [6, 1, 9] [0, 5, 8] [0, 5, 8] [9, 0, 5] [1, 5, 8] [9, 0, 5] [1, 5, 8] [8, 0, 5] [11, 2, 6] [6, 1, 9] [9, 0, 5] [1, 5, 8] [0, 5, 8] [6, 1, 9] [1, 5, 8] [3, 6, 10] [9, 1, 5] [3, 6, 11] [7, 2, 11] [1, 5, 8] [5, 1, 8] [1, 5, 8] [7, 2, 11] [3, 6, 11] [9, 1, 5] [3, 6, 11] [2, 5, 9] [11, 2, 7]

3.3 Beispiel 2

paeckchen2.txt

Elapsed times:
Elapsed time for creating Graphs & Inputs in sec: 0.0016530000000001266s
Elapsed time for running Algorithm: 0.23753899999999994s

Ideale Lösung gefunden

Summe der gesamten Kleidung: 32
wie viel verwertet wurde: 32
Wie viel übrig geblieben ist: 0
Wie viele Pakete gepackt wurden: 6
Wie viel Prozent % der Gesamtkleidung verwertet wurde: 100.0%

Wie die Pakete gepackt werden: [14, 4, 9, 9, 9, 14, 14] [0, 5, 11, 5, 6, 11] [8, 1, 12, 8, 8, 12, 12] [13, 3, 7, 10, 13] [15, 4, 9, 14] [11, 2, 6]

3.4 Beispiel 3

paeckchen3.txt

Elapsed times:
Elapsed time for creating Graphs & Inputs in sec: 0.0025839999999999197
Elapsed time for running Algorithm: 0.2017

perfekte Lösung...

Summe der gesamten Kleidung: 96
wie viel verwertet wurde: 94
Wie viel übrig geblieben ist: 2
Wie viele Pakete gepackt wurden: 10
Wie viel Prozent % der Gesamtkleidung verwertet wurde: 97.91666666666666%

Wie die Pakete gepackt werden: [2, 8, 17, 24, 29, 8, 8, 17, 17, 24, 24, 29, 29] [26, 3, 9, 18, 30, 4, 4, 11, 11, 19, 20, 26, 26, 31, 31] [29, 2, 8, 17, 24, 8, 8, 17, 24] [14, 0, 6, 21, 27, 0, 1, 6, 6, 22, 22, 31] [23, 1, 7, 15, 28, 7, 7, 16, 22, 22] [30, 0, 6, 14, 21, 4, 4, 22] [24, 2, 8, 17, 29, 8, 8] [19, 3, 10, 25, 30, 5, 5, 13, 32, 33] [16, 3, 12, 26, 32] [7, 0, 18, 22, 31]

3.5 Beispiel 4

paeckchen4.txt

Elapsed times:
Elapsed time for creating Graphs & Inputs in sec: 0.006427000000000183s
Elapsed time for running Algorithm: 6.974564000000001s

Es kann evt. eine bessere Lösung geben...

Summe der gesamten Kleidung: 437
wie viel verwertet wurde: 393
Wie viel übrig geblieben ist: 44
Wie viele Pakete gepackt wurden: 27
Wie viel Prozent % der Gesamtkleidung verwertet wurde: 89.93135011441647%

Wie die Pakete gepackt werden: [52, 2, 11, 20, 28, 36, 44, 2, 2, 11, 11, 20, 20, 28, 28, 39, 39, 44, 48] [5, 9, 17, 25, 33, 41, 50, 2, 2, 9, 9, 17, 17, 25, 25, 33, 33, 41, 41] [38, 0, 15, 22, 30, 46, 53, 15, 15, 22, 22, 30, 30, 38, 38, 46, 46, 53, 53] [37, 6, 14, 21, 29, 45, 55, 14, 14, 21, 21, 29, 29, 37, 37, 45, 45, 56, 56] [55, 4, 12, 19, 27, 35, 43, 12, 19, 19, 27, 27, 35, 43, 43] [16, 2, 20, 28, 36, 44, 52, 2, 2, 11, 13, 20, 24, 28, 28, 39, 39, 48, 48] [51, 8, 10, 18, 26, 34, 42, 18, 26, 40, 40, 51, 51] [1, 9, 17, 25, 33, 41, 54, 1, 1, 9, 9, 17, 17, 25, 25, 33, 33, 41, 41] [10, 8, 18, 26, 34, 42, 51, 40, 40, 51, 51] [40, 2, 11, 23, 31, 47, 57, 2, 2, 23, 23, 31, 31, 40, 40, 47, 47, 57, 57] [41, 2, 11, 17, 25, 33, 50, 2, 2, 9, 9, 17, 17, 25, 41, 41] [21, 6, 14, 29, 37, 45, 56, 14, 14, 21, 21, 29, 29, 37, 37, 45, 45, 56, 56] [14, 6, 21, 29, 37, 45, 56, 14, 21, 23, 29, 29, 37, 37, 45, 45, 56, 56] [51, 8, 10, 18, 26, 34, 42, 40, 40, 51, 51] [57, 8, 10, 18, 26, 34, 42, 40, 40, 51, 51] [34, 8, 10, 18, 26, 42, 51, 40, 40, 51, 51] [18, 8, 10, 26, 34, 42, 57, 40, 40, 51, 51] [50, 5, 11, 17, 25, 33, 41, 2, 2, 9, 9, 17] [40, 2, 11, 23, 31, 47, 57, 2, 2, 23, 23, 31, 31, 47, 47, 57, 57] [15, 8, 22, 30, 38, 46, 57, 15, 30, 30, 49, 49] [46, 8, 15, 22, 30, 40, 49, 30, 49, 49] [47, 2, 11, 23, 31, 40, 57, 2, 23, 23, 31, 31, 47, 47, 57, 57] [11, 2, 23, 31, 40, 47, 57, 23, 23, 47, 47, 57, 57] [30, 8, 15, 22, 38, 46, 57, 49, 49] [29, 6, 14, 21, 37, 45, 56, 23, 29, 29, 37, 45, 47, 56, 56] [57, 2, 11, 23, 31, 40, 47, 47, 47, 57, 57] [26, 8, 10, 18, 34, 42, 51, 51, 57]

3.6 Beispiel 5

paeckchen5.txt

Elapsed times:
Elapsed time for creating Graphs & Inputs in sec: 0.002250999999999781s
Elapsed time for running Algorithm: 2.0625999999999998s

Es kann evt. eine bessere Lösung geben...

Summe der gesamten Kleidung: 174
wie viel verwertet wurde: 150
Wie viel übrig geblieben ist: 24
Wie viele Pakete gepackt wurden: 17
Wie viel Prozent % der Gesamtkleidung verwertet wurde: 86.20689655172413%

Wie die Pakete gepackt werden: [25, 4, 12, 16, 29, 4, 4, 12, 12, 25, 25, 29, 29] [3, 8, 18, 24, 30, 8, 8, 15, 15] [20, 6, 11, 22, 32, 6, 6, 11, 11, 20, 20, 22, 22] [28, 0, 7, 14, 21, 7, 7, 14, 21, 21, 28, 28] [25, 4, 12, 16, 29, 29, 29] [5, 9, 15, 26, 33, 5, 5, 17, 17, 26, 26, 33, 33] [10, 3, 15, 24, 30, 8, 10, 18] [3, 10, 18, 22, 31, 1, 1, 10, 22, 31, 31] [9, 5, 17, 26, 33, 17, 17, 26, 33] [15, 5, 9, 26, 33, 17, 17] [15, 3, 8, 24, 30] [10, 1, 18, 22, 31, 1, 1, 31] [14, 0, 7, 21, 28, 7, 7, 21, 28, 28] [1, 10, 18, 22, 31, 1] [10, 1, 18, 22, 31] [10, 1, 18, 22, 31] [34, 2, 13, 19, 23, 2, 2, 13, 34]

3.7 Beipsiel 6

paeckchen6.txt

Elapsed times:
Elapsed time for creating Graphs & Inputs in sec: 0.0015039999999999498s
Elapsed time for running Algorithm: 0.7048380000000001s

Es kann evt. eine bessere Lösung geben...

Summe der gesamten Kleidung: 770
wie viel verwertet wurde: 670
Wie viel übrig geblieben ist: 100
Wie viele Pakete gepackt wurden: 72
Wie viel Prozent % der Gesamtkleidung verwertet wurde: 87.01298701298701%

Wie die Pakete gepackt werden: [12, 3, 8, 19, 20, 3, 3, 8, 8, 12, 12, 20, 20] [14, 3, 8, 19, 20, 3, 3, 8, 9, 12, 12, 20, 20] [20, 0, 5, 10, 15, 2, 2, 5, 5, 10, 10, 17, 17, 20, 20] [19, 0, 5, 10, 20, 2, 2, 5, 5, 10, 10, 17, 17, 20, 20] [2, 5, 10, 15, 20, 2, 2, 5, 5, 10, 10, 17, 17, 20, 20] [3, 8, 12, 19, 20, 3, 3, 9, 9, 12, 12, 20, 20] [19, 0, 5, 10, 20, 2, 2, 5, 5, 10, 17, 17, 20, 20] [18, 0, 5, 10, 20, 5, 5, 11, 11, 16, 16, 20, 20] [12, 3, 8, 19, 20, 3, 3, 9, 9, 14, 14, 20, 20] [11, 0, 5, 15, 20, 5, 5, 11, 11, 16, 16, 20, 20] [1, 5, 10, 15, 20, 2, 2, 5, 5, 17, 17, 20, 20] [11, 0, 5, 15, 20, 5, 6, 11, 11, 16, 16, 20, 20] [5, 0, 10, 15, 20, 2, 2, 7, 7, 17, 17, 20, 20] [5, 0, 10, 15, 20, 2, 2, 7, 7, 17, 17, 20, 20] [3, 8, 12, 19, 20, 3, 3, 9, 9, 14, 14, 20, 20] [5, 0, 10, 15, 20, 2, 2, 7, 7, 17, 17, 20, 20] [6, 0, 10, 15, 20, 4, 4, 6, 6, 13, 13, 18, 18, 20, 20] [20, 0, 5, 10, 15, 2, 2, 7, 7, 17, 17, 20, 20] [9, 3, 12, 19, 20, 3, 3, 9, 9, 14, 14, 20, 20] [7, 0, 10, 15, 20, 2, 2, 7, 17, 17, 20, 20] [1, 5, 10, 15, 20, 2, 2, 17, 17, 20, 20] [12, 3, 8, 19, 20, 3, 3, 9, 9, 14, 14, 20, 20] [6, 0, 10, 15, 20, 4, 4, 6, 6, 13, 13, 18, 18, 20, 20] [3, 8, 12, 19, 20, 3, 3, 9, 9, 14, 14, 20, 20] [14, 3, 8, 19, 20, 3, 3, 9, 9, 14, 14] [1, 5, 10, 15, 20, 2, 2, 17, 17] [11, 0, 5, 15, 20, 6, 6, 11, 11, 16, 16] [1, 5, 10, 15, 20, 2, 2, 17, 17] [19, 0, 5, 10, 20, 2, 2, 17, 17] [5, 0, 10, 15, 20, 2, 2, 17, 17] [5, 0, 10, 15, 20, 2, 2, 17, 17] [14, 3, 8, 19, 20, 3, 3, 9, 9, 14, 14] [17, 1, 5, 10, 20, 2, 2, 17, 17] [1, 5, 10, 17, 20, 2, 2, 17, 17] [18, 4, 6, 13, 20, 4, 4, 6, 6, 18, 18] [4, 6, 10, 18, 20, 4, 4, 6, 6, 18, 18] [7, 1, 10, 17, 20, 2, 2, 17, 17] [5, 1, 10, 15, 20, 2, 2, 17, 17] [4, 6, 13, 15, 20, 4, 4, 6, 6, 18, 18] [10, 1, 7, 15, 20, 2, 2] [17, 2, 5, 10, 20, 2, 2] [5, 1, 10, 17, 20] [17, 1, 7, 10, 20] [17, 1, 5, 10, 20] [1, 5, 10, 15, 20] [5, 1, 10, 15, 20] [7, 1, 10, 17, 20] [4, 6, 10, 15, 20, 4, 4, 6, 6, 18, 18] [6, 4, 10, 15, 20, 4, 4, 18, 18] [10, 2, 5, 15, 20] [7, 2, 10, 17, 20] [7, 2, 10, 17, 20] [17, 2, 5, 10, 20] [6, 4, 10, 15, 20, 4, 4, 18, 18] [17, 2, 5, 10, 20] [20, 2, 7, 10, 17] [5, 2, 10, 15, 20] [6, 4, 13, 15, 20, 4, 4, 18, 18] [6, 4, 10, 18, 20, 18, 18] [7, 2, 10, 17, 20] [18, 4, 6, 13, 20, 18, 18] [7, 2, 10, 15, 20] [17, 2, 5, 10, 20] [4, 6, 10, 15, 20, 18, 18] [10, 2, 5, 17, 20] [20, 2, 5, 10, 17] [13, 4, 6, 15, 20, 18, 18] [13, 4, 6, 18, 20, 18, 18] [6, 4, 13, 15, 20, 18, 18] [20, 4, 6, 10, 18, 18, 18] [20, 2, 7, 10, 17] [18, 4, 6, 10, 20, 18, 18]

3.8 Beispiel 7

paeckchen7.txt

Elapsed times:
Elapsed time for creating Graphs & Inputs in sec: 0.0010620000000001184s
Elapsed time for running Algorithm: 40419.134696999994s

Es kann evt. eine bessere Lösung geben...

Summe der gesamten Kleidung: 700
wie viel verwertet wurde: 491
Wie viel übrig geblieben ist: 209

Wie viele Pakete gepackt wurden: 46
Wie viel Prozent % der Gesamtkleidung verwertet wurde: 70.14285714285714%

Wie die Pakete gepackt werden: [7, 2, 11, 15, 16, 2, 2, 7, 7, 11, 11, 15, 16, 16] [4, 0, 8, 12, 16, 4, 4, 8, 8, 12, 12, 16, 16] [12, 0, 4, 8, 16, 4, 4, 8, 8, 12, 12, 16, 16] [11, 2, 7, 15, 16, 2, 2, 7, 7, 11, 11, 16, 16] [13, 0, 4, 8, 16, 4, 4, 8, 8, 12, 13, 16, 16] [16, 0, 4, 8, 12, 4, 4, 8, 8, 13, 13, 16, 16] [11, 2, 7, 15, 16, 2, 2, 7, 7,

11, 11, 16, 16] [0, 4, 8, 12, 16, 4, 4, 8, 8, 13, 13, 16, 16] [12, 0, 4, 8, 16, 4, 4, 8, 8, 13, 13, 16, 16] [12, 0, 4, 8, 16, 4, 4, 8, 8, 13, 13, 16, 16] [11, 2, 7, 15, 16, 2, 2, 7, 7, 11, 11, 16, 16] [14, 0, 4, 8, 16, 4, 4, 8, 8, 13, 13, 16, 16] [13, 0, 4, 8, 16, 4, 4, 8, 8, 13, 13, 16, 16] [6, 0, 8, 12, 16, 1, 1, 6, 8, 8, 16, 16] [8, 0, 4, 12, 16, 4, 4, 8, 8, 13, 13, 16, 16] [14, 0, 4, 8, 16, 4, 4, 8, 8, 13, 13, 16, 16] [6, 1, 8, 12, 16, 1, 1, 8, 16, 16] [0, 4, 8, 12, 16, 4, 4, 9, 9, 13, 13, 16, 16] [16, 0, 4, 8, 12, 4, 4, 9, 9, 13, 13, 16, 16] [15, 2, 7, 11, 16, 2, 2, 7, 7, 11, 11, 16, 16] [12, 1, 6, 8, 16, 1, 1, 16, 16] [4, 0, 9, 13, 16, 4, 4, 9, 9, 13, 13, 16, 16] [16, 2, 7, 11, 15, 2, 2, 7, 7, 11, 11, 16, 16] [6, 1, 8, 12, 16, 1, 1, 16, 16] [8, 1, 6, 15, 16, 1, 1, 16, 16] [12, 1, 6, 8, 16, 1, 1, 16, 16] [16, 1, 6, 8, 12, 1, 1, 16, 16] [1, 6, 8, 12, 16, 1, 1, 16, 16] [4, 0, 8, 14, 16, 4, 4, 9, 9, 13, 13] [8, 1, 6, 12, 16, 1, 1] [6, 1, 8, 12, 16, 1, 1] [16, 1, 6, 8, 12, 1, 1] [15, 2, 7, 11, 16, 2, 2, 7, 7, 11, 11] [6, 1, 8, 12, 16, 1, 1] [0, 5, 8, 12, 16, 4, 4, 9, 9, 13, 13] [1, 6, 8, 12, 16, 1, 1] [16, 2, 7, 11, 15, 2, 2, 7, 7, 11, 11] [8, 1, 6, 12, 16, 1, 1] [16, 2, 7, 11, 15, 2, 2, 7, 7, 11, 11] [16, 2, 7, 11, 15, 2, 2, 7, 7, 11, 11] [16, 1, 6, 8, 12, 1, 1] [8, 1, 6, 12, 16, 1, 1] [6, 1, 8, 12, 16, 1, 1] [7, 2, 11, 15, 16, 2, 2, 7, 7, 11, 11] [16, 1, 6, 8, 12, 1, 1] [12, 1, 6, 8, 16, 1, 1]

Eigenständigkeitserklärung

Hiermit versichere ich, die vorliegende schriftliche Arbeit mit dem Titel „Eine Optimierung des Stilvolle-Päckchen-Problems durch die Modellierung als ILP“ eigenständig und ohne fremde Hilfe verfasst zu haben und keine anderen als die aufgeführten Hilfsmittel verwendet habe. Den Inhalt des Textes, den ich nicht als Zitat oder Paraphrase ausgewiesen habe, kann ich selbst in vollem Umfang formal wir inhaltlich vertreten.

Meißen, den 18. Dezember 2024

Felix Ying Xin Du