Aufgabe 2: Stilvolle Päckchen

Máté Tirpák

Teilnahme-ID: 71669

15. April 2024

Inhaltsverzeichnis

1. Lösungsidee

1.1 Problembeschreibung, Intuition und NP-hard Schwere

Gegeben sind Kleidungsstücke verschiedener Typen (/Sorten) und Stile, die in eine beliebige Anzahl Boxen so verpackt werden sollen, dass die Anzahl übrig gebliebener Kleidungsstücke möglichst gering ist. Um eine Box als vollständig zu erklären, muss jeder Typ mindestens 1-Mal und höchstens 3-Mal enthalten sein. Dabei spielt auch die Verteilung der Stile eine entscheidende Rolle, da jeder Stil nur mit bestimmten anderen kombinierbar ist.

Es ließe sich mit großer Sicherheit sagen, dass dieses Problem NP-hard. Es beschreibt eine erheblich kompliziertere Variante des Standard Bin Packing Problems. In diesem müssen n Objekte einer variablen Höhe in eine möglichst geringe Anzahl Behälter einer festen Höhe eingefügt werden, ohne dass die Objekte eines Behälters diese Höhe in der Summe überschreiten [1]. Dabei existieren zwei Varianten des Problems, „online“, in welchem aus der Gesamtzahl Objekte stets nur eins bekannt ist, welches eingefügt werden muss, um das nächste in der Liste zu erhalten, und „offline“, wo alle Objekte zu Beginn bekannt sind. Während die „online“ Version bewiesenermaßen NP-complete ist, entspricht die „offline“ Version der dem Schweregrad NP-hard. Die Problemstellung der stilvollen Päckchen entspricht ebenfalls dem „offline“ Prinzip, da alle Kleidungsstücke bereits zu Beginn vorliegen. Die Zielstellungen sind ähnlich, so soll beim klassischen „bin packing“ die Anzahl Boxen und bei den stilvollen Päckchen die Anzahl unverpackter Kleidungsstücke minimiert werden. Das Ziel des klassischen NP-hard schweren „offline bin packing“ ist dabei nicht schwerer als das der stilvollen Päckchen, da die unterschiedliche Größe der Boxen, 1-3 Kleidungsstücke pro Typ, als zusätzliche Komplexität dient und neue Verfahren bietet, die Anzahl unverpackter Kleidungsstücke zu minimieren. Abgesehen von der Parallele, dass in einer Box höchstens 3 Kleidungsstücke pro Typ vorhanden sein dürfen und beim klassischen „bin packing“ ebenfalls eine Größe der Boxen existiert, ließe sich eine noch geeignetere Parallele ziehen, etwa indem die Kompatibilität der Stile einer Box betrachtet werden muss. Diese ist variabel, ebenso wie die Größe der Boxen, aber weitaus komplexer als eine simple Maximalgröße einer Box. Da das klassische „offline bin packing“ NP-hard ist, ließe sich behaupten, dass „Stillvolle Päckchen“ ebenfalls NP-hard ist.

Um eine möglichst erfolgreiche Verteilung der Kleidungsstücke auf die Boxen zu erreichen, müssen diese zu Beginn mit einem simplen Algorithmus verpackt und anschließend optimiert werden. Dieser Algorithmus sei „First Fit Decreasing“. Bezüglich der Optimierung wären folgende Methoden ersichtlich. Einerseits könnten alle Kleidungsstücke aus unvollständigen Boxen gesammelt und erneut mit „First Fit Decreasing“ einsortiert werden. Andererseits könnten die unvollständigen Boxen auch gefüllt werden, indem überflüssige Kleidungsstücke aus vollständigen Boxen transferiert werden. Für das erneute Einfügen inkorrekt platzierter Kleidung wäre zudem eine Funktion ersichtlich, die übermäßig gefüllte Boxen in verschiedene weniger befüllte aufteilt. So ließe sich eine Box die von jedem Typ mindestens 2 Stücke enthält in 2 separate Boxen aufteilen, und eine mit 3 pro Typ in 3 verschiedene Boxen. So entstehe Platz für zusätzliche Kleidungsstücke. Im Gegensatz zu einem klassischen „Bin Packing Problem“, welches die Anzahl Boxen zu minimieren bestrebt, ist diese in dieser Problemstellung unberücksichtigt zu lassen, sodass diese Methode durchaus schlüssig erscheint.

1.2 Allgemeine Funktionalitäten

1.2.1 Strukturierung der Daten

Als Resultat sollen die Anzahl Kleidungstypen und -Stile gespeichert, ein Verzeichnis mit allen Kombinationsmöglichkeiten der Stile erstellt und alle Kleidungsstücke zum erstmaligen Verpacken in eine geeignete Struktur sortiert werden.

1.2.1.1 Einlesen der Datei

Es sollen die Anzahl der Stile und Typen, aber auch die Stile selbst als Liste gespeichert werden. Zudem sollen die Kleidungsstücke und Kombinationen entnommen werden.

1.2.1.2 Speichern der Stil-Kombinationen

Es sollen eine Python Dictionary (/Hashmap), die jedem Stil alle anderen kompatiblen zuordnet und eine Matrix, die entlang der beiden Indexe repräsentativ für zwei Stile deren Kompatibilität als boolean speichert, erstellt werden. Das Verzeichnis aller Kombinationen wird durchiteriert und die Kombinationen in die Dictionary und Matrix eingetragen. Dabei wird beachtet, dass die Kombinationen in beide Richtungen gewertet werden, so würde bei a ist kombinierbar mit b im Falle der Dictionary, sowohl ein Eintrag b im Index a geschehen als auch ein Eintrag a im Index b. Zudem ist die Kompatibilität unter dem eigenen Stil bereits bei der Initialisierung gewährleistet, so existiert bereits zu Beginn ein Eintrag a im Index a.

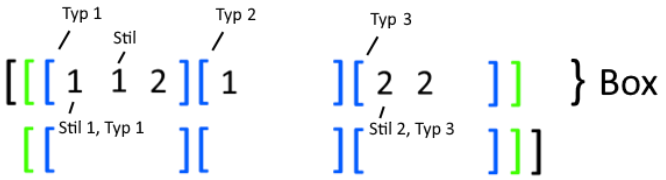
1.2.1.3 Sortieren nach aufsteigender Stil-Kompatibilität

Es wird ein Hilfsarray in der Länge der Stile+1 erstellt, wobei jeder Index einen Stil repräsentiert. Anschließend wird die Hashmap der Stil-Kombinationen durchiteriert und die Anzahl möglicher Kombinationen, die Länge der Kombinationsliste, in den passenden Index des Hilfsarrays eingefügt. Anschließend wird eine neue leere Liste initialisiert und solange der Index, repräsentativ für den Stil, des größten Elements des Hilfsarrays an den Anfang gehangen (und aus dem Hilfsarray entfernt), bis alle Stile abgearbeitet wurden. Als Resultat bleibt eine Liste, die alle Stile, sortiert nach deren möglichen Kombinationen mit anderen Stilen, in steigender Reihenfolge beinhaltet. Dabei gilt, je kleiner die Anzahl möglicher Kombinationen, desto geringer ist auch die Kompatibilität. Die Anzahl möglicher Kombinationen liegt im Intervall [1; Anzahl Stile]. 1 repräsentativ für den Fall, dass der Stil nur mit sich selbst kombinierbar ist, und „Anzahl Stile“ dafür, dass der Stil mit jedem anderen kompatibel ist. Dieses Intervall bietet eine gute Möglichkeit, das Maximum des Hilfsarrays zu entnehmen. Da dies jedoch den momentan kompatibelsten Stil beschreibt, muss dieser zwingend an den Anfang der neuen Liste gehangen werden, sodass es sich von hinten nach vorne absteigend aufbaut und somit von vorne betrachtet steigend vorliegt. Es ließe sich auch von vorne aufbauen, allerdings müsse dazu stets das Minimum des Hilfsarrays gesucht werden und etwa die Indexe auf den maximalen Integer gesetzt werden, sodass aus Anschaulichkeitsgründen darauf verzichtet wurde.

1.2.2 Beziehungen zwischen Kleidungsstücken und Boxen

1.2.2.1 Hierarchie

Eine Box entspricht einem Element einer Liste, die alle Boxen vereint. Die Box beschreibt eine Liste, die so viele Elemente besitzt, wie es Typen gibt. Der Index+1 entspricht dabei dem Typen eines Kleidungsstücks. Diese Elemente, jeweils repräsentativ für einen Typ, bestehen wiederum aus einer Liste, die alle Stile der vorhandenen Kleidungsstücke des Typs beschreibt. Ein Kleidungsstück lässt sich folglich durch seinen Index innerhalb der Box und den Wert des Elements innerhalb der Liste des Indexes beschreiben.



1.2.2.2 Stile einer Box

Es sollen alle Stile einer Box abfragbar sein. Diese werden in einem „set“ gespeichert, welches stets nur einzigartige Elemente enthält. Für jeden Typ wird jeder Stil betrachtet und in das „set“ eingefügt. Dieses wird als Liste zurückgegeben und beinhaltet alle Stile der betrachteten Box.

1.2.2.3 Kompatible Stile einer Box

Die Grundlage bietet eine Liste, die alle Stile enthält, aus welcher im folgenden Verlauf inkompatible entfernt werden sollen, sodass diese schlussendlich zurückgegeben werden kann. Es wird sich eine Liste über alle in der Box enthaltenen Stile beschafft und diese anschließend iteriert. Für jeden der Stile der Box wird die Kompatibilität mit anderen Stilen geprüft. Es wird die anfängliche Liste, welche bei Initialisierung alle Stile beinhaltet von hinten iteriert und ein Stil entfernt, sobald dieser nicht mit dem momentan betrachteten nicht kombinierbar ist.

1.2.2.4 Fehlende Kleidungsstücke einer Box

Es sollen alle fehlenden Typen und kompatiblen Stile ermittelt werden, sodass daraus ein Rückschluss darauf gezogen werden kann, ob ein betrachtetes Kleidungsstück für eine bestimmte Box gesucht wird. Die kompatiblen Stile lassen sich bereits mit der vorherigen Funktionalität abfragen. Zusätzlich dazu müssen die Typ-Listen iteriert werden, wobei jeder Typ gespeichert wird, dessen Liste leer ist. Die kompatiblen Stile und fehlenden Typen werden zurückgegeben.

1.2.2.5 Kompatibilität Kleidungsstück-Box

Zu Beginn wird überprüft, ob die Box nicht bereits 3 Kleidungsstücke des einzufügenden Typs enthält. Falls dies noch nicht zum Abbruch führt, wird überprüft, ob der Stil des einzufügenden Kleidungsstücks mit allen anderen der Box kompatibel ist.

1.2.2.6 Einfügen eines Kleidungsstücks

Gegeben eine Box und Typ und Stil eines Kleidungsstücks wird dessen Stil in den Index Typ-1 der Box eingefügt. Es sollen keine Überprüfungen auf Korrektheit stattfinden und der Prozess ist in-place.

1.2.2.7 Überprüfung des Resultats

Es müssen zwei Aspekte überprüft werden. Die Menge der Kleidungsstücke und die Kompatibilität aller Stile einer Box. Um ersteres zu überprüfen wir das beim Öffnen der Datei erstellte Verzeichnis aller Kleidungsstücke geöffnet. Für jeden Eintrag, der Typ, Stil und Anzahl repräsentiert, wird geprüft, ob dies mit dem Resultat übereinstimmt, indem die Boxen iteriert und in jeder das Vorkommen des Kleidungsstücks gezählt wird. Zweiteres wird kontrolliert, indem alle Stile der Box abgerufen werden und anschließend mithilfe der Kombinationsmatrix überprüft wird, ob jeder Stil mit jedem anderen kombinierbar ist.

1.2.2.8 Auswertung des Resultats

Es sollen die Anzahl korrekter und inkorrekter Boxen, und die Anzahl verpackter und unverpackter Kleidungsstücke gezählt werden. Eine Box ist inkorrekt, wenn sie einen Typen ohne zugeordnete Kleidungsstücke enthält. Es werden alle Kleidungsstücke der Box zusammengezählt und zu verpackten oder unverpackten hinzuaddiert. Die Heuristik, welche die Qualität eines Resultats angibt, soll berechnet werden, indem die Anzahl unverpackter Kleidungsstücke durch die Gesamtzahl der Kleidungsstücke geteilt wird. Sie beschreibt die Fehlerquote und kann somit zum Vergleichen verschiedener Resultate genutzt werden.

1.3 Algorithmen

1.3.1 First Fit Decreasing

Für das erste Verpacken der Kleidungsstücke soll eine abgewandelte Version des „First Fit Decreasing“ Algorithmus [1] verwendet werden. Im klassischen „bin packing problem“ müssen n Objekte einer variablen Höhe in eine möglichst geringe Anzahl Behälter einer festen Höhe eingefügt werden, ohne dass die Objekte eines Behälters diese Höhe in der Summe überschreiten. Eine bekannte Methode ist „First Fit“, welche besagt, dass die Behälter iteriert werden, wobei einzufügende Objekte stets in den erstmöglichen Platz eingefügt werden. Die Problemstellung der stilvollen Päckchen entspricht der „offline“ Version des Bin-Packing-Problems, in der, im Gegensatz zur „online“ Version, wo stets nur das aktuelle Objekt bekannt ist, alle Objekte bereits zu Beginn vorliegen. Deswegen lässt sich diese Methode zu „First Fit Decreasing“ erweitern. Die Objekte werden dabei, in absteigender Größe sortiert, in den Behältern gefügt. Der Hintergrundgedanke ist dabei, dass sehr große Objekte, die bereits alleine fast einen gesamten Behälter beanspruchen, zuerst eingefügt werden sollen, sodass zum Ende des Verlaufs immer kleinere Objekte als Ergänzung zu den größeren dienen.

Dieses Prinzip lässt sich auf das Verpacken der Kleidungsstücke übertragen. Die Größe der Objekte soll dabei durch die Kompatibilität der Stile repräsentiert werden. Hat ein Stil eine größere Anzahl Kombinationsmöglichkeiten als ein anderer, so ist er bezogen auf diesen kompatibler. So wie es das Ziel ist, große Objekte zu Beginn in eigene Behälter zu verpacken, sollen auch schlecht kompatible Stile möglichst in eigene Boxen bzw. Boxen mit geringer Stilvielfalt gefügt werden. Stile, die mit vielen anderen kombinierbar und sehr kompatibel sind, sollen zum Schluss verpackt werden und als Ergänzung für die weniger kompatiblen Stile dienen. Somit sollen die Kleidungsstücke in aufsteigender Kompatibilität ihrer Stile verpackt werden. Die aufsteigende Natur widerspricht bei oberflächlicher Betrachtung zwar dem Prinzip des Decreasings, allerdings ist dies damit zu erklären, dass die Schwere des Verpackens eines Stils im antiproportionalen Verhältnis zur Anzahl Kombinationsmöglichkeiten steht.

Der Algorithmus soll wie folgt aussehen. Es sollen alle Kleidungsstücke in aufsteigender Kompatibilität ihrer Stile in die erstmögliche Box verpackt werden. In einem Zyklus werden immer alle Kleidungsstücke eines bestimmten Stils verpackt. Dabei folgt diese Reihenfolge der steigenden Kompatibilität. Die Boxen werden für jedes Kleidungsstück iteriert, bis ein passender Platz gefunden oder schlussendlich eine neue Box erstellt wurde und das Kleidungsstück wird eingefügt. Dabei werden zuerst alle Kleidungen des ersten Typs mit dem gegebenen Stil und zuletzt alle Kleidungen des letzten Typs in die Boxen geordnet. Durch diesen Vorgang werden schwer kombinierbare Stile einerseits gruppiert und später durch kompatiblere Stile zu vollständigen Boxen aufgefüllt. So sollen schwer kombinierbare Stile mit sehr reinen Boxen die Grundlage bilden und leicht kombinierbare Stile diese Boxen schlussendlich ergänzen.

1.3.2 Füllen unvollständiger Boxen

Das Ziel ist es, unvollständige Boxen mithilfe überflüssiger Kleidungsstücke anderer Boxen, zu füllen. Die Boxen werden iteriert und unvollständige einzeln betrachtet. Dort werden die fehlenden Typen und kompatiblen Style abgefragt. Alle Boxen werden erneut einzeln betrachtet, vollständige sowie unvollständige. Wenn eine Box von einem gesuchten Typ mehr als 1 Kleidungsstück besitzt und dieses mit der zu füllenden Box kompatibel ist, wird das Stück entfernt und gespeichert. Wenn alle gesuchten Kleidungsstücke gefunden wurden oder alle Boxen einmal durchgeschaut wurden, werden die gefundenen Stück abschließend in die zu füllende Box eingefügt. Wichtig zu beachten ist dabei, dass eine vollständige Füllung nicht garantiert wird und stets die erstmöglichen Kleidungsstücke entnommen werden. Daher wird dem nächsten Algorithmus eine umso größere Rolle zugeschrieben.

1.3.3 Auflösen und Neuverpacken unvollständiger Boxen

Im Fall, dass eine Box nicht gefüllt werden kann, soll sie aufgelöst und ihre Kleidungsstücke in bereits vollständige Boxen hinzugefügt werden, um die verschwendete Anzahl Stücke zu minimieren, wobei nicht garantiert ist, dass alle Kleidungsstücke einen neuen Platz finden. Gegebenenfalls muss eine ebenfalls unvollständige Box erstellt werden, jedoch mit weniger Kleidungsstücken, sodass sich diese Optimierung auch in dem Fall durchaus lohnt.

Die Boxen werden iteriert und sobald eine unvollständige gefunden wurde, wird jedes Kleidungsstück der Box gespeichert, diese anschließend gelöscht, und die Suche weitergeführt. Nach diesem Prozess werden die Kleidungsstücke nach aufsteigender Kompatibilität ihrer Stile sortiert und mit „First Fit Decreasing“ erneut verpackt.

1.3.4 Teilung vollständiger Boxen

Der eben beschriebene Algorithmus des Auflösens und Neuverpackens der Kleidungsstücke unvollständiger Boxen kann optimiert werden, indem vollständige Boxen auf kleinere, aber weiterhin vollständige, Boxen aufgeteilt werden und somit Plätze zum Einfügen neuer Kleidungsstücke geschaffen wird.

Dabei zu beachten ist jedoch, dass sich beim potenziell anschließenden Füllen der jetzt kleineren Boxen deren Stilvielfalt deutlich erhöht und es daher nicht mehr möglich ist, diese annähernd zu ihren ursprünglichen Boxen zu vereinen. Es resultieren Boxen mit wenigen Kleidungsstücken, weshalb unvollständige Boxen in Zukunft nur noch stark erschwert durch das Suchen überflüssiger

Kleidungsstücke gefüllt werden können. Folglich ist die Teilung vollständiger Boxen in der Endphase des Ablaufs anzuwenden.

1.3.5 Simulated Annealing

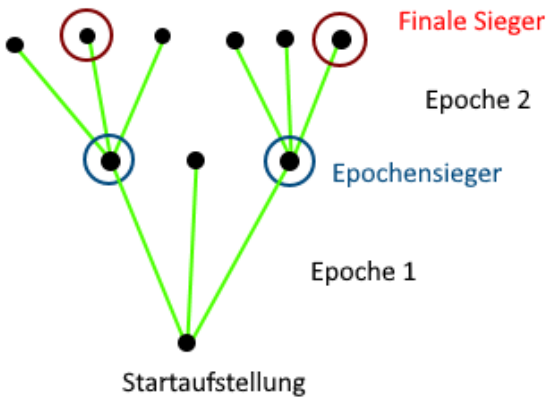
Die Idee ist es, einen Algorithmus zu erstellen, welcher durch zufällige Veränderungen aus einer ursprünglichen Aufstellung der Boxen viele neue unterschiedliche Ergebnisse erstellt. Diese zufälligen Veränderungen könnten etwa durch ständiges Tauschen zweier Kleidungsstücke realisiert werden.

Eine stark vereinfachte Version dieser Vorgehensweise könnte wie folgt aussehen. Die Kleidungsstücke werden in Boxen verpackt, sodass eine erste Verteilung vorliegt. Auf Grundlage dieser, werden 10 neue Ergebnisse erschaffen, indem die ursprüngliche Aufstellung jeweils zufällig verändert wird. Das beste Ergebnis dieser 10 neuen Resultate wird zum Gewinner erklärt, gespeichert und die restlichen verworfen.

Dies lässt sich weiter ausbauen. Zum einen würde sich die Heuristik, der Anteil unverpackter Kleidungsstücke aus der Gesamtmenge, welcher zum Vergleichen der Ergebnisse genutzt werden soll, durch reines Tauschen nie ändern, da nur die Stile durchmischt werden und die Anzahl Kleidungsstücke mit Beachtung der Typen in jeder Box gleichbleibt. Somit muss nach Abschluss aller Tausche eine Optimierung der Boxen erfolgen. Dabei sollen alle unvollständigen Boxen nach dem üblichen Prinzip mit überflüssigen Kleidungsstücken gefüllt werden, danach alle überbesetzten Boxen und einzelne kleinere weiterhin vollständige Boxen aufgeteilt und danach erneut unvollständige Boxen gefüllt werden. Zum Abschluss werden weiterhin übrig gebliebene Kleidungsstücke in bereits vollständige Boxen verpackt. Der Prozess des vorherigen Tauschens soll wie folgt ablaufen. Jedes Kleidungsstück aller Boxen soll in Betracht gezogen werden, wobei unvollständige Boxen übersprungen werden. Wenn eine festgelegte Wahrscheinlichkeit erfüllt wird, etwa durch einen Zufallsgenerator, wird ein Tausch des momentanen Kleidungsstücks angestrebt. Dabei werden erneut alle Kleidungsstücke iteriert und überprüft, ob dieses kompatibel für den Tausch ist. Falls dies erfüllt ist, wird erneut ein Zufallsgenerator durchlaufen und bei Erfolg der Tausch durchgeführt. Der Prozess endet, wenn die ursprüngliche Iteration ans Ende der Boxen gelangt. Nach der darauffolgenden Optimierung werden die Boxen nach der Heuristik ausgewertet und zusammen mit dieser gespeichert.

Das eingeführte Beispiel, in welchem eine Ursprungsform zu 10 neuen erweitert wurde, lässt sich weiter ausbauen. So soll die beste Aufstellung nicht direkt akzeptiert und simulated annealing beendet werden, sondern anhand dieser weitere 10 neue Aufstellungen erstellt werden. So ließen sich etwa 10 Epochen durchspielen, in denen stets das beste Resultat weitergeführt wird. Allerdings ergeben sich dabei bedenken, denn es ist riskant stets nur eine einzige Aufstellung zu betrachtet, denn obwohl diese die momentan attraktivste sei, könnte sie für weitere Betrachtungen nachteilig sein. So ließe sich etwa sagen, dass stets die 3 besten Aufstellungen jeder Epoche weitergeführt werden und für jede dieser 3 Aufstellungen je 10 neue erstellt und betrachtet werden. Bei dieser Betrachtung sollen die Ursprünglichen allerdings nicht ausgeschlossen werden. Durch systematisches Probieren ergaben sich folgende Werte als besonders geeignet. Die Epochenzahl beläuft sich dabei auf 3, da in darüber hinauslaufenden Epochen meist keine besseren Fälle mehr gefunden werden können. Die Anzahl zufälliger Stichproben aus einer einzelnen Aufstellung beträgt wie auch im Beispiel 10, sodass ein großes Maß an weiterführenden Resultaten betrachtet werden kann. Dabei reicht es, immer die beiden besten Ergebnisse weiterzuführen. So wird ein Tunnelblick auf eine einzige Aufstellung vermieden, allerdings die Laufzeit nicht unnötig in die Höhe getrieben.

Folgendes Bild visualisiert den Algorithmus.



1.4 Programmablauf

Zu Beginn wird die Datei eingelesen, die Kombinationsmöglichkeiten der Stile als Python Dictionary gespeichert und das erste Verpacken der Kleidungsstücke vorbereitet, indem die Kleidungsstücke nach aufsteigender Kompatibilität ihrer Stile sortiert werden. Anschließend werden diese mit „First Fit Decreasing“ verpackt. Folgend werden alle unvollständigen Boxen mithilfe überflüssiger Kleidungsstücke vollständiger Boxen gefüllt, in der Hoffnung, dass diese ebenfalls zu vollständigen werden. Für den Fall, dass diese Boxen zwar befüllt, aber nicht gefüllt werden, sollen die unvollständigen Boxen aufgelöst und deren Inhalt neu verteilt werden, damit sie in der Vielfalt und Größe sinken. Anschließend wird Simulated Annealing angewandt. Abschließend wird versucht, unvollständige Boxen zu füllen, und darauffolgend werden alle vollständigen Boxen aufgeteilt und alle Kleidungsstücke unvollständiger Boxen in die neuen kleineren Boxen eingefügt, sodass möglichst wenige Kleidungsstücke in unvollständigen Boxen übrigbleiben. Die Anzahl Boxen vergrößert sich dadurch erheblich, genauso aber auch die Anzahl erfolgreich platzierter Kleidungsstücke, sodass hervorragende Ergebnisse erzielt werden können.

2. Zeit- und Platzkomplexität

Die Strukturierung der Daten, ausgelesen aus der Datei besteht aus dem Erstellen einer Dictionary und Matrix zum Speichern aller Kombinationsmöglichkeiten, dem Erstellen eines Rankings für die Kompatibilität der Stile und dem letztendlichen Sortieren der Kleidungsstücke auf Basis dieses Rankings. Die Komplexitäten sind dabei sehr niedrig, so beläuft sich die Laufzeit beim Erstellen der Dictionary und Matrix auf O(s+n) für s Stile und n Kombinationen, da die Kompatibilität jedes Stils mit sich selbst zu Beginn verzeichnet und anschließend die Kombinationen eingetragen werden. Die Platzkomplexität wird von der Matrix dominiert, deren Achsenlängen die Anzahl Stile betragen, somit O(s^2) für s Stile. Die Platzkomplexität des Erstellens des Kompatibilitäts-Rankings unterbietet dies mit O(s) für s Stile, da nur eine Ausgabeliste und eine Hilfsliste zum Zählen der Anzahl Kombinationsmöglichkeiten in der Länge der Anzahl Stile erstellt werden müssen. Die Zeitkomplexität erreicht allerdings ein Maximum, denn das Hilfsarray wird äquivalent häufig zu seiner Länge iteriert, um das Maximum herauszufinden. Dies beläuft sich auf O(s^2). Beim Sortieren der Kleidungsstücke betragen sowohl Zeit- und Platzkomplexität O(k) für k Kleidungsstücke, denn als zusätzlicher Speicher müssen ein Hilfsarray und eine Ausgabeliste erstellt werden, und die Zeitkomplexität ergibt sich dadurch, dass jedes Kleidungsstück in das Hilfsarray in linearer Zeit eingefügt und anschließend in ebenfalls linearer Zeit entnommen werden muss.

Anschließend können die Kleidungsstücke mithilfe von „First Fit Decreasing“ eingefügt werden. Sie werden Stil nach Stil eingefügt, wobei die Kleidungsstücke in Listen vorliegen, deren Indexe den Typen

entsprechen. Aus diesen Listen müssen so lange Kleidungsstücke verpackt werden, bis der Stil ausgeschöpft ist. Für jeden Stil müssen alle Typen betrachtet werden und für den bestimmten Stil innerhalb dieses Typs muss seine Liste und parallel dazu die Boxen zum Verpacken iteriert werden. Die Zeitkomplexität ist folglich O(s*t*(k+b)) für s Stile, t Typen, k Kleidungsstücke des Typen und b Boxen und die Platzkomplexität O(n) für n Kleidungsstücke. Darauffolgend werden unvollständige Boxen gefüllt. Es werden alle Boxen nacheinander auf eine leere Typ Liste untersucht und anschließend die Gesamtheit aller Kleidungsstücke der Boxen nach einem passenden Stück durchsucht, welches anschließend eingefügt werden kann. Die Zeitkomplexität wäre O(b*k*s) für b Boxen, k als Menge Kleidungsstücke aller Boxen und s kompatible Stile und die Platzkomplexität O(m) für m gesammelte Kleidungsstücke, da diese zwischengespeichert werden. Es folgt das Sammeln und Neueinfügen aller Kleidungsstücke aus unvollständigen Boxen. Der aufwendigste Prozess dabei ist eindeutig das Einfügen durch „First Fit Decreasing“. Sowohl Zeit- als auch Platzkomplexität werden dadurch bestimmt. Darauf folgt „simulated annealing“. Die Kernfunktion ist dabei das zufällige Tauschen von Kleidungsstücken. Dazu müssen alle Boxen inklusive all ihrer Kleidungsstücke iteriert werden. Dabei müssen die kompatiblen Stile der Box abgefragt werden. Um das zweite Kleidungsstück für den Tausch zu finden, müssen die Boxen erneut iteriert werden, allerdings ist der Typ bereits bekannt, sodass sich die Suche innerhalb einer einzigen Box deutlich verschnellert. Die Zeitkomplexität wäre daher O(b^2*k*s*n) für b Boxen, k Kleidungsstücke einer Box, s Stile und n Stile einer Box. Da eine Kopie aller Boxen zu Beginn erstellt werden muss, beläuft sich die Platzkomplexität auf O(b*k) für Boxen und k Kleidungsstücke der Boxen. Des weiteres muss die Optimierung im letzten Schritt des Prozesses betrachtet werden. Diese besteht allerdings aus den bereits bekannten Verfahren, dem Füllen unvollständiger Boxen und das Einfügen von Kleidung aus unvollständigen Boxen, mit der Ausnahme des Teilens der Boxen, welches neu ist. Die Zeitkomplexität für diese Funktionalität beläuft sich auf O(b*t) für b Boxen und t Typen und die Platzkomplexität auf O(k) für k entnommene Kleidungsstücke. Somit unterbiete auch diese Funktionalität beide Komplexitäten des zufälligen Tauschens, In „simulated annealing“ wird der Prozess des Tauschens wie folgt verwendet. Es gibt Epochen, in denen für jede momentan gespeicherte Lösung eine bestimmte Anzahl neuer Lösungen durch zufällige Tausche geschaffen wird. Wenn man diese 3 Elemente als Faktoren hinzubezieht, ergibt sich eine Zeitkomplexität O(e*n*m*b^2*k*s*o) für e Epochen, n gespeicherte Lösungen pro Epoche, m neue Lösungen pro Ursprung, b Boxen, k Kleidungsstücke einer Box, s Stile und o Stile einer Box. Diese erscheint zuerst problematisch, ist allerdings akzeptabel, da im üblichen Fall nur b und eventuell k die einzigen hohen Werte sind. Die Platzkomplexität beläuft sich auf O(n*m*k) für n gespeicherte Lösungen pro Epoche und m neue Lösungen pro Ursprung und k Kleidungsstücke in den Boxen. Nach diesem Vorgang werden als Zusatz alle unvollständigen Boxen zu füllen versucht, alle vollständigen Boxen aufgespalten und Kleidungsstücke aus unvollständigen Boxen neu verteilt. Deren Komplexitäten stellen allerdings wie bereits analysiert keine neuen Maxima dar.

Zuletzt sollen die Boxen ausgewertet und das Ergebnis validiert werden. Die Laufzeit der Auswertung beträgt O(b*m) für b Boxen und m Typen, da jede Box betrachtet und anschließend die Anzahl Kleidungsstücke gezählt werden muss. Diese werden gezählt, indem die Längen der Typ-Listen zusammengezählt werden. Die Platzkomplexität ist dabei konstant, O(1). Die Zeitkomplexität der Überprüfung des Ergebnisses beläuft sich auf O(b*k+b*s^2) für b Boxen, s Stile einer Box und k einzigartige Kleidungsstücke. Zuerst muss die Anzahl Kleidungsstücke jeweils überprüft werden. Dabei zählen nur einzigartige Kleidungsstücke, denn im Format der Datei liegen Kleidungsstücke gleichen Typs und Stils zusammengefasst unter einer Zahl, repräsentativ für die Anzahl, vor. Für jedes dieser Stücke müssen alle Boxen iteriert werden, und alle Stile der bestimmten Typ-Liste überprüft werden, wobei sich diese auf 0-3 belaufen, sodass dieser Faktor konstant ist. Dies ergibt O(b*k) für b Boxen und k einzigartige Kleidungsstücke. Als zweiter Schritt soll die Kompatibilität aller Stile einer Box für alle Boxen bestätigt werden. Dazu muss jede Box einzeln betrachtet, und jeder darin vorkommende Stil mit

jedem anderen vorkommenden auf Kombinierbarkeit überprüft werden. Es ergäbe sich O(b*s^2) für b Boxen und s Stile einer Box. Das ursprüngliche O(b*k+b*s^2) kann zu O(b*(k+s^2)) umgeschrieben werden. Obwohl die Anzahl Stile einer Box meist deutlich geringer ist, als die Anzahl Kleidungsstücke gelistet in der Datei, überwiegt diese dennoch aufgrund des Quadrats, woraus die Zeitkomplexität O(b*s^2) für b Boxen und s Stile einer Box resultiert. Die Platzkomplexität ist dabei linear für die Stile einer Box, O(s), da ansonsten nur Integer fürs Zählen oder Iterieren verwendet werden.

„simulated annealing“ dominiert eindeutig beide Komplexitäten. So betrage die gesamte Zeitkomplexität des Programms dadurch O(e*n*m*b^2*k*s*o) für e Epochen, n gespeicherte Lösungen pro Epoche, m neue Lösungen pro Ursprung, b Boxen, k Kleidungsstücke einer Box, s Stile und o Stile einer Box und die Platzkomplexität O(n*m*k) für n gespeicherte Lösungen pro Epoche und m neue Lösungen pro Ursprung und k Kleidungsstücke in den Boxen.

3. Beispiele inklusive Bedienung des Programms

3.1 Bedienung

Beim Start des Programms durch __main__.py wird nach dem Pfad eines Ordners gefragt. Ein Beispiel einer möglichen Eingabe wäre:

C:\informatik\bwinf\

Wichtig dabei zu beachten ist, dass die Eingabe mit einem „/“ beendet wird. Anschließend wird einem die Option gegeben, alle vorgegebenen Beispiele kollektiv zu lösen, Voraussetzung dafür ist, dass alle Beispiele 0-7 unter dem unveränderten Namen im angegebenen Ordner vorliegen. Wenn diese Option abgelehnt wird, muss man einen Dateinamen eingeben, etwa „beispiel1.txt“, welche anschließend gelöst wird.

3.2 Vorgegebene Beispiele

Wenn die Option des Auswertens aller Beispiele gewählt wird, sieht die Ausgabe wie folgt aus.

```
wasted pieces / all pieces:  0.0000          runtime:  0.00 s
wasted pieces / all pieces:  0.0000          runtime:  0.21 s
wasted pieces / all pieces:  0.0000          runtime:  0.00 s
wasted pieces / all pieces:  0.0208          runtime:  0.02 s
wasted pieces / all pieces:  0.0000          runtime:  0.28 s
wasted pieces / all pieces:  0.1092          runtime:  0.11 s
wasted pieces / all pieces:  0.0286          runtime:  0.85 s
wasted pieces / all pieces:  0.1871          runtime:  0.78 s
```

Anderenfalls erhält der Nutzer beim Lösen einer einzigen Datei eine detaillierte Übersicht, die für die vorgegebenen Beispiele wie folgt aussieht.

3.2.1 Beispiel 0

```
paeckchen0.txt
------------------
Korrekte Boxen: 3
Inkorrekte Boxen: 0
Verpackte Kleidungsstücke: 11
Übrige Kleidungsstücke: 0
Übrige / alle Kleidungsstücke: 0.0000
Laufzeit:  0.00 s
------------------
Boxen:
Eine Zeile entspricht einer Box, die wie folgt aufgebaut ist, S entspricht einem beliebigen Stil: [Typ 1:[S, S, S], Typ 2:[S, S, S], ..., Typ n:[S, S, S]]

Muster: Box N: Typ X: [Stil 1, Stil 2, Stil 3]

Box 1:  Typ 1: [2, 2]   Typ 2: [2]   Typ 3: [2]
Box 2:  Typ 1: [1, 1]   Typ 2: [2]   Typ 3: [2]
Box 3:  Typ 1: [1]   Typ 2: [2]   Typ 3: [2]
```

3.2.2 Beispiel 1

```
paeckchen1.txt
------------------
Korrekte Boxen: 137
Inkorrekte Boxen: 0
Verpackte Kleidungsstücke: 499
Übrige Kleidungsstücke: 0
Übrige / alle Kleidungsstücke: 0.0000
Laufzeit:  0.24 s
------------------
Boxen:
Eine Zeile entspricht einer Box, die wie folgt aufgebaut ist, S entspricht einem beliebigen Stil: [Typ 1:[S, S, S], Typ 2:[S, S, S], ..., Typ n:[S, S, S]]

Muster: Box N: Typ X: [Stil 1, Stil 2, Stil 3]

Box 1:  Typ 1: [4, 4, 3]   Typ 2: [3]   Typ 3: [4]
Box 2:  Typ 1: [4, 4]   Typ 2: [4]   Typ 3: [4, 4]
Box 3:  Typ 1: [4, 4]   Typ 2: [4]   Typ 3: [4, 4]
Box 4:  Typ 1: [3]   Typ 2: [4]   Typ 3: [4]
Box 5:  Typ 1: [3, 4]   Typ 2: [4, 4]   Typ 3: [4]
Box 6:  Typ 1: [4, 4]   Typ 2: [4, 4]   Typ 3: [4]
Box 7:  Typ 1: [4, 4]   Typ 2: [4, 4]   Typ 3: [4]
Box 8:  Typ 1: [4, 4]   Typ 2: [4, 4]   Typ 3: [4]
Box 9:  Typ 1: [4]   Typ 2: [4]   Typ 3: [3]
Box 10:         Typ 1: [3, 3, 4]   Typ 2: [4, 4]   Typ 3: [3]
Box 11:         Typ 1: [4, 3, 4]   Typ 2: [4, 4, 4]   Typ 3: [3]
Box 12:         Typ 1: [3, 3, 3]   Typ 2: [4, 4, 4]   Typ 3: [3]
Box 13:         Typ 1: [3, 3, 3]   Typ 2: [4]   Typ 3: [3]
Box 14:         Typ 1: [1, 2, 1]   Typ 2: [1]   Typ 3: [1]
Box 15:         Typ 1: [2, 2, 1]   Typ 2: [1]   Typ 3: [1]
Box 16:         Typ 1: [1, 2, 1]   Typ 2: [2]   Typ 3: [1]
Box 17:         Typ 1: [1, 1, 2]   Typ 2: [1]   Typ 3: [1]
Box 18:         Typ 1: [1, 1]   Typ 2: [1]   Typ 3: [1]
Box 19:         Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 20:         Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 21:         Typ 1: [1]   Typ 2: [2]   Typ 3: [1]
Box 22:         Typ 1: [1]   Typ 2: [2]   Typ 3: [1]
Box 23:         Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 24:         Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 25:         Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 26:         Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 27:         Typ 1: [1, 1]   Typ 2: [2]   Typ 3: [1, 1]
Box 28:         Typ 1: [1]   Typ 2: [2]   Typ 3: [1]
Box 29:         Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 30:         Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 31:         Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 32:         Typ 1: [2]   Typ 2: [2]   Typ 3: [1]
Box 33:         Typ 1: [3]   Typ 2: [4]   Typ 3: [3]
Box 34:         Typ 1: [3]   Typ 2: [4]   Typ 3: [3]
Box 35:         Typ 1: [3]   Typ 2: [3]   Typ 3: [3]
Box 36:         Typ 1: [3]   Typ 2: [3]   Typ 3: [2]
Box 37:         Typ 1: [3]   Typ 2: [3]   Typ 3: [2]
Box 38:         Typ 1: [3]   Typ 2: [3]   Typ 3: [2]
Box 39:         Typ 1: [3]   Typ 2: [3]   Typ 3: [2]
Box 40:         Typ 1: [3]   Typ 2: [3]   Typ 3: [2]
Box 41:         Typ 1: [3]   Typ 2: [3]   Typ 3: [2]
Box 42:         Typ 1: [3]   Typ 2: [2]   Typ 3: [2]
Box 43:         Typ 1: [2]   Typ 2: [2]   Typ 3: [2]
Box 44:         Typ 1: [3]   Typ 2: [2]   Typ 3: [2]
Box 45:         Typ 1: [3]   Typ 2: [2]   Typ 3: [2]
Box 46:         Typ 1: [3]   Typ 2: [2]   Typ 3: [2]
Box 47:         Typ 1: [3]   Typ 2: [2]   Typ 3: [2]
Box 48:         Typ 1: [3, 3, 2]   Typ 2: [2, 2]   Typ 3: [2]
Box 49:         Typ 1: [3, 3, 3]   Typ 2: [2, 2, 2]   Typ 3: [2]
Box 50:         Typ 1: [3, 3, 2]   Typ 2: [2, 2, 2]   Typ 3: [3]
Box 51:         Typ 1: [3, 3, 3]   Typ 2: [2]   Typ 3: [2]
Box 52:         Typ 1: [3, 3, 3]   Typ 2: [4]   Typ 3: [4]
Box 53:         Typ 1: [3, 4, 3]   Typ 2: [4]   Typ 3: [4]
Box 54:         Typ 1: [2, 3, 3]   Typ 2: [2]   Typ 3: [2]
Box 55:         Typ 1: [3, 2, 2]   Typ 2: [3]   Typ 3: [2]
Box 56:         Typ 1: [3, 3, 2]   Typ 2: [2]   Typ 3: [3]
Box 57:         Typ 1: [2, 1, 2]   Typ 2: [1]   Typ 3: [1]
Box 58:         Typ 1: [2, 1, 2]   Typ 2: [2]   Typ 3: [1]
Box 59:         Typ 1: [2, 1, 2]   Typ 2: [1]   Typ 3: [1]
```

```
Box 60:         Typ 1: [2, 2, 2]   Typ 2: [1]   Typ 3: [1]
Box 61:         Typ 1: [2, 2, 2]   Typ 2: [1]   Typ 3: [1]
Box 62:         Typ 1: [2, 2, 2]   Typ 2: [1]   Typ 3: [1]
Box 63:         Typ 1: [2, 2, 2]   Typ 2: [2]   Typ 3: [1]
Box 64:         Typ 1: [2, 2, 2]   Typ 2: [1]   Typ 3: [1]
Box 65:         Typ 1: [2, 2, 2]   Typ 2: [1]   Typ 3: [1]
Box 66:         Typ 1: [3, 3, 2]   Typ 2: [3]   Typ 3: [3]
Box 67:         Typ 1: [3, 3, 2]   Typ 2: [3]   Typ 3: [3]
Box 68:         Typ 1: [2, 2, 2]   Typ 2: [3]   Typ 3: [3, 3]
Box 69:         Typ 1: [2, 2]   Typ 2: [2]   Typ 3: [3, 3]
Box 70:         Typ 1: [2]   Typ 2: [2]   Typ 3: [2]
Box 71:         Typ 1: [2]   Typ 2: [2]   Typ 3: [3]
Box 72:         Typ 1: [3]   Typ 2: [2]   Typ 3: [2]
Box 73:         Typ 1: [3]   Typ 2: [2]   Typ 3: [2]
Box 74:         Typ 1: [3]   Typ 2: [2]   Typ 3: [2]
Box 75:         Typ 1: [3]   Typ 2: [3]   Typ 3: [2]
Box 76:         Typ 1: [2]   Typ 2: [2]   Typ 3: [2]
Box 77:         Typ 1: [3]   Typ 2: [2]   Typ 3: [2]
Box 78:         Typ 1: [3]   Typ 2: [3]   Typ 3: [2]
Box 79:         Typ 1: [3]   Typ 2: [2]   Typ 3: [2]
Box 80:         Typ 1: [3]   Typ 2: [2]   Typ 3: [2]
Box 81:         Typ 1: [3]   Typ 2: [2]   Typ 3: [3]
Box 82:         Typ 1: [3]   Typ 2: [3]   Typ 3: [3]
Box 83:         Typ 1: [2]   Typ 2: [3]   Typ 3: [2]
Box 84:         Typ 1: [3]   Typ 2: [3]   Typ 3: [2]
Box 85:         Typ 1: [3]   Typ 2: [3]   Typ 3: [2]
Box 86:         Typ 1: [3]   Typ 2: [2]   Typ 3: [2]
Box 87:         Typ 1: [3]   Typ 2: [3]   Typ 3: [3]
Box 88:         Typ 1: [3]   Typ 2: [3]   Typ 3: [2]
Box 89:         Typ 1: [3]   Typ 2: [3]   Typ 3: [2]
Box 90:         Typ 1: [3]   Typ 2: [3]   Typ 3: [2]
Box 91:         Typ 1: [3]   Typ 2: [3]   Typ 3: [2]
Box 92:         Typ 1: [3]   Typ 2: [3]   Typ 3: [2]
Box 93:         Typ 1: [3]   Typ 2: [3]   Typ 3: [2]
Box 94:         Typ 1: [3]   Typ 2: [3]   Typ 3: [2]
Box 95:         Typ 1: [2]   Typ 2: [3]   Typ 3: [3]
Box 96:         Typ 1: [3]   Typ 2: [4]   Typ 3: [3]
Box 97:         Typ 1: [3]   Typ 2: [3]   Typ 3: [3]
Box 98:         Typ 1: [3]   Typ 2: [4]   Typ 3: [3]
Box 99:         Typ 1: [3]   Typ 2: [3]   Typ 3: [4]
Box 100:        Typ 1: [2]   Typ 2: [2]   Typ 3: [1]
Box 101:        Typ 1: [2]   Typ 2: [2]   Typ 3: [1]
Box 102:        Typ 1: [2]   Typ 2: [2]   Typ 3: [1]
Box 103:        Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 104:        Typ 1: [2]   Typ 2: [2]   Typ 3: [1]
Box 105:        Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 106:        Typ 1: [1]   Typ 2: [2]   Typ 3: [1]
Box 107:        Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 108:        Typ 1: [2]   Typ 2: [2]   Typ 3: [1]
Box 109:        Typ 1: [1]   Typ 2: [2]   Typ 3: [1]
Box 110:        Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 111:        Typ 1: [1]   Typ 2: [2]   Typ 3: [1]
Box 112:        Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 113:        Typ 1: [2]   Typ 2: [1]   Typ 3: [1]
Box 114:        Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 115:        Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 116:        Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 117:        Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 118:        Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 119:        Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 120:        Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 121:        Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 122:        Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 123:        Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 124:        Typ 1: [1]   Typ 2: [2]   Typ 3: [1]
Box 125:        Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 126:        Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 127:        Typ 1: [1]   Typ 2: [1]   Typ 3: [1]
Box 128:        Typ 1: [4]   Typ 2: [4]   Typ 3: [3]
Box 129:        Typ 1: [4]   Typ 2: [4]   Typ 3: [3]
Box 130:        Typ 1: [3]   Typ 2: [4]   Typ 3: [3]
Box 131:        Typ 1: [4]   Typ 2: [4]   Typ 3: [3]
Box 132:        Typ 1: [4]   Typ 2: [4]   Typ 3: [4]
Box 133:        Typ 1: [4]   Typ 2: [4]   Typ 3: [4]
Box 134:        Typ 1: [4]   Typ 2: [4]   Typ 3: [4]
Box 135:        Typ 1: [4]   Typ 2: [4]   Typ 3: [4]
Box 136:        Typ 1: [4]   Typ 2: [4]   Typ 3: [4]
Box 137:        Typ 1: [4]   Typ 2: [4]   Typ 3: [4]
```

3.2.3 Beispiel 2

```
paeckchen2.txt
------------------
Korrekte Boxen: 6
Inkorrekte Boxen: 0
Verpackte Kleidungsstücke: 32
Übrige Kleidungsstücke: 0
Übrige / alle Kleidungsstücke: 0.0000
Laufzeit:  0.00 s
------------------
Boxen:
Eine Zeile entspricht einer Box, die wie folgt aufgebaut ist, S entspricht einem beliebigen Stil: [Typ 1:[S, S, S], Typ 2:[S, S, S], ..., Typ n:[S, S, S]]

Muster: Box N: Typ X: [Stil 1, Stil 2, Stil 3]

Box 1:  Typ 1: [7]   Typ 2: [7, 7, 7]   Typ 3: [7, 9, 9]
Box 2:  Typ 1: [7]   Typ 2: [7]   Typ 3: [9, 9]
Box 3:  Typ 1: [1]   Typ 2: [4, 4, 5]   Typ 3: [5, 5]
Box 4:  Typ 1: [2]   Typ 2: [6, 6, 6]   Typ 3: [8, 8, 2]
Box 5:  Typ 1: [6]   Typ 2: [8, 2]   Typ 3: [2, 2]
Box 6:  Typ 1: [3]   Typ 2: [5]   Typ 3: [5]
```

3.2.4 Beispiel 3

```
paeckchen3.txt
------------------
Korrekte Boxen: 12
Inkorrekte Boxen: 1
Verpackte Kleidungsstücke: 94
Übrige Kleidungsstücke: 2
Übrige / alle Kleidungsstücke: 0.0208
Laufzeit:  0.02 s
------------------
Boxen:
Eine Zeile entspricht einer Box, die wie folgt aufgebaut ist, S entspricht einem beliebigen Stil: [Typ 1:[S, S, S], Typ 2:[S, S, S], ..., Typ n:[S, S, S]]

Muster: Box N: Typ X: [Stil 1, Stil 2, Stil 3]

Box 1:  Typ 1: [4]   Typ 2: [5, 5, 5]   Typ 3: [4, 4, 4]   Typ 4: [5, 5]   Typ 5: [4, 4]
Box 2:  Typ 1: [4]   Typ 2: [5, 5, 5]   Typ 3: [4, 4]   Typ 4: [5, 5, 5]   Typ 5: [4, 4]
Box 3:  Typ 1: [4]   Typ 2: [5, 5, 5]   Typ 3: [4]   Typ 4: [5]   Typ 5: [4]
Box 4:  Typ 1: [9, 9]   Typ 2: [9]   Typ 3: [3, 3]   Typ 4: [3, 9]   Typ 5: [3, 9]
Box 5:  Typ 1: [1, 2]   Typ 2: [1, 1]   Typ 3: [1]   Typ 4: [1, 1]   Typ 5: [6]
Box 6:  Typ 1: [1, 1]   Typ 2: [1, 2]   Typ 3: [6]   Typ 4: [2, 2]   Typ 5: [8, 8]
Box 7:  Typ 1: [8, 8]   Typ 2: [10, 8]   Typ 3: [10]   Typ 4: [9, 9]   Typ 5: [10, 8]
Box 8:  Typ 1: [8, 7, 7]   Typ 2: [2, 2, 6]   Typ 3: [2]   Typ 4: [2, 2]   Typ 5: [2]
Box 9:  Typ 1: [8]   Typ 2: [8]   Typ 3: [9]   Typ 4: [6]   Typ 5: [6]
Box 10:         Typ 1: [2]   Typ 2: [2]   Typ 3: [6]   Typ 4: [2]   Typ 5: [6]
Box 11:         Typ 1: [1]   Typ 2: [1]   Typ 3: [1]   Typ 4: [2]   Typ 5: [8]
Box 12:         Typ 1: [7]   Typ 2: [7]   Typ 3: [9]   Typ 4: [9]   Typ 5: [9]
Box 13:         Typ 1: []   Typ 2: [5, 5]   Typ 3: []   Typ 4: []   Typ 5: []
```

3.2.5 Beispiel 4

```
paeckchen4.txt
------------------
Korrekte Boxen: 31
Inkorrekte Boxen: 0
Verpackte Kleidungsstücke: 437
Übrige Kleidungsstücke: 0
Übrige / alle Kleidungsstücke: 0.0000
Laufzeit:  0.30 s
------------------
Boxen:
Eine Zeile entspricht einer Box, die wie folgt aufgebaut ist, S entspricht einem beliebigen Stil: [Typ 1:[S, S, S], Typ 2:[S, S, S], ..., Typ n:[S, S, S]]

Muster: Box N: Typ X: [Stil 1, Stil 2, Stil 3]

Box 1:  Typ 1: [12, 12, 10]   Typ 2: [22, 22, 22]   Typ 3: [22, 22, 22]   Typ 4: [22, 22, 22]   Typ 5: [22, 22, 22]   Typ 6: [12, 22, 22]   Typ 7: [8]
Box 2:  Typ 1: [10, 12, 12]   Typ 2: [12, 12, 22]   Typ 3: [22, 22, 12]   Typ 4: [22, 22, 22]   Typ 5: [22, 22, 22]   Typ 6: [22, 12, 22]   Typ 7: [8]
Box 3:  Typ 1: [19, 19, 12]   Typ 2: [12, 12, 12]   Typ 3: [12, 12, 12]   Typ 4: [12, 12, 12]   Typ 5: [12]   Typ 6: [12, 21]   Typ 7: [20, 20]
Box 4:  Typ 1: [17, 17, 17]   Typ 2: [4, 4, 4]   Typ 3: [4, 4, 4]   Typ 4: [4, 4]   Typ 5: [4, 4]   Typ 6: [4, 4, 4]   Typ 7: [2]
Box 5:  Typ 1: [17, 17, 17]   Typ 2: [4, 4, 4]   Typ 3: [4, 4, 4]   Typ 4: [4, 4, 4]   Typ 5: [4, 4, 4]   Typ 6: [4, 4, 4]   Typ 7: [2]
Box 6:  Typ 1: [17, 17, 17]   Typ 2: [4, 4, 4]   Typ 3: [4, 4, 4]   Typ 4: [4, 4, 4]   Typ 5: [4, 4]   Typ 6: [4, 4, 4]   Typ 7: [2]
Box 7:  Typ 1: [15]   Typ 2: [11, 11]   Typ 3: [11, 11, 11]   Typ 4: [11, 11, 11]   Typ 5: [11, 11]   Typ 6: [11, 11, 11]   Typ 7: [18, 18]
Box 8:  Typ 1: [6, 6, 6]   Typ 2: [4, 10, 10]   Typ 3: [4, 4]   Typ 4: [4]   Typ 5: [4]   Typ 6: [4]   Typ 7: [14]
Box 9:  Typ 1: [18]   Typ 2: [13, 13]   Typ 3: [13, 13, 13]   Typ 4: [13, 13, 13]   Typ 5: [13, 13, 13]   Typ 6: [13, 13, 13]   Typ 7: [20, 20, 20]
Box 10:         Typ 1: [18]   Typ 2: [13, 13, 13]   Typ 3: [13, 13, 13]   Typ 4: [13, 13, 13]   Typ 5: [13, 13, 13]   Typ 6: [13, 13, 13]   Typ 7: [20, 20, 20]
Box 11:         Typ 1: [18]   Typ 2: [13, 13, 13]   Typ 3: [13, 13, 13]   Typ 4: [13, 13, 13]   Typ 5: [13, 13, 13]   Typ 6: [13, 13, 13]   Typ 7: [20, 20, 20]
Box 12:         Typ 1: [18]   Typ 2: [13]   Typ 3: [21, 21]   Typ 4: [13, 13, 13]   Typ 5: [13, 13]   Typ 6: [13, 13]   Typ 7: [20, 20]
Box 13:         Typ 1: [3]   Typ 2: [16, 16]   Typ 3: [16]   Typ 4: [16]   Typ 5: [16, 23, 16]   Typ 6: [16]   Typ 7: [9, 9, 9]
Box 14:         Typ 1: [24]   Typ 2: [5]   Typ 3: [5]   Typ 4: [5]   Typ 5: [5, 23, 23]   Typ 6: [5]   Typ 7: [7, 7, 7]
Box 15:         Typ 1: [24]   Typ 2: [5, 5]   Typ 3: [5, 5]   Typ 4: [5, 5]   Typ 5: [23, 23, 5]   Typ 6: [5, 5]   Typ 7: [7, 7, 7]
Box 16:         Typ 1: [24]   Typ 2: [5]   Typ 3: [5, 5]   Typ 4: [5, 5]   Typ 5: [5, 5]   Typ 6: [5]   Typ 7: [7, 7, 7]
Box 17:         Typ 1: [24]   Typ 2: [5]   Typ 3: [5]   Typ 4: [5]   Typ 5: [5]   Typ 6: [5]   Typ 7: [7, 7, 7]
Box 18:         Typ 1: [24]   Typ 2: [5]   Typ 3: [5]   Typ 4: [5]   Typ 5: [23, 5]   Typ 6: [5]   Typ 7: [7, 7, 7]
Box 19:         Typ 1: [24]   Typ 2: [16, 16]   Typ 3: [16, 16]   Typ 4: [16, 16, 16]   Typ 5: [16, 23]   Typ 6: [16, 16]   Typ 7: [1, 1, 1]
Box 20:         Typ 1: [24]   Typ 2: [16]   Typ 3: [16]   Typ 4: [16, 16, 16]   Typ 5: [16, 23]   Typ 6: [16]   Typ 7: [1, 1, 1]
Box 21:         Typ 1: [24]   Typ 2: [16]   Typ 3: [16]   Typ 4: [16]   Typ 5: [23, 23]   Typ 6: [16]   Typ 7: [1, 1, 1]
Box 22:         Typ 1: [10, 10, 10]   Typ 2: [10]   Typ 3: [21]   Typ 4: [21]   Typ 5: [23]   Typ 6: [21]   Typ 7: [23, 23, 23]
Box 23:         Typ 1: [10, 10, 10]   Typ 2: [10, 10]   Typ 3: [21, 21, 21]   Typ 4: [21, 21]   Typ 5: [23]   Typ 6: [21, 21, 21]   Typ 7: [23, 23, 23]
Box 24:         Typ 1: [10, 10, 10]   Typ 2: [10, 10]   Typ 3: [21, 21, 21]   Typ 4: [21, 21, 21]   Typ 5: [23]   Typ 6: [21, 21, 21]   Typ 7: [23, 23, 23]
Box 25:         Typ 1: [10, 10, 10]   Typ 2: [10]   Typ 3: [21, 21, 21]   Typ 4: [21, 21]   Typ 5: [23]   Typ 6: [21, 21, 21]   Typ 7: [23, 23, 23]
Box 26:         Typ 1: [24]   Typ 2: [5]   Typ 3: [5]   Typ 4: [5]   Typ 5: [23, 5]   Typ 6: [5]   Typ 7: [7, 7, 7]
Box 27:         Typ 1: [24]   Typ 2: [16]   Typ 3: [16]   Typ 4: [16]   Typ 5: [16, 23]   Typ 6: [16]   Typ 7: [1, 1, 1]
Box 28:         Typ 1: [10, 10, 10]   Typ 2: [10]   Typ 3: [21]   Typ 4: [21]   Typ 5: [23]   Typ 6: [21, 21]   Typ 7: [23, 23, 23]
Box 29:         Typ 1: [10, 10, 10]   Typ 2: [10]   Typ 3: [21]   Typ 4: [21]   Typ 5: [23, 23]   Typ 6: [21]   Typ 7: [23, 23, 23]
Box 30:         Typ 1: [10, 10, 10]   Typ 2: [10]   Typ 3: [21]   Typ 4: [21]   Typ 5: [23]   Typ 6: [21]   Typ 7: [23, 23]
Box 31:         Typ 1: [12]   Typ 2: [12]   Typ 3: [12]   Typ 4: [12, 12, 12]   Typ 5: [12]   Typ 6: [21]   Typ 7: [20]
```

3.2.6 Beispiel 5

```
paeckchen5.txt
------------------
Korrekte Boxen: 17
Inkorrekte Boxen: 7
Verpackte Kleidungsstücke: 155
Übrige Kleidungsstücke: 19
Übrige / alle Kleidungsstücke: 0.1092
Laufzeit:  0.11 s
------------------
Boxen:
Eine Zeile entspricht einer Box, die wie folgt aufgebaut ist, S entspricht einem beliebigen Stil: [Typ 1:[S, S, S], Typ 2:[S, S, S], ..., Typ n:[S, S, S]]

Muster: Box N: Typ X: [Stil 1, Stil 2, Stil 3]

Box 1:  Typ 1: [1]   Typ 2: [1, 1, 1]   Typ 3: [2, 2]   Typ 4: [2, 2, 2]   Typ 5: [2, 2, 2]
Box 2:  Typ 1: [1]   Typ 2: [1, 1, 1]   Typ 3: [2]   Typ 4: [2, 2]   Typ 5: [2, 2, 2]
Box 3:  Typ 1: [21, 21, 21]   Typ 2: [16, 16, 16]   Typ 3: [21, 21, 21]   Typ 4: [21, 21, 21]   Typ 5: [17]
Box 4:  Typ 1: [13, 13, 13]   Typ 2: [20, 20]   Typ 3: [20]   Typ 4: [14]   Typ 5: [20, 20]
Box 5:  Typ 1: [19, 19, 19]   Typ 2: [9]   Typ 3: [10, 10, 10]   Typ 4: [19, 19, 19]   Typ 5: [19, 19, 19]
Box 6:  Typ 1: [19]   Typ 2: [9]   Typ 3: [10, 10, 10]   Typ 4: [19, 19]   Typ 5: [19, 19]
Box 7:  Typ 1: [19]   Typ 2: [9]   Typ 3: [10, 10, 10]   Typ 4: [19]   Typ 5: [19]
Box 8:  Typ 1: [18, 18, 18]   Typ 2: [18, 18, 18]   Typ 3: [5]   Typ 4: [18, 18, 18]   Typ 5: [6, 6, 6]
Box 9:  Typ 1: [18]   Typ 2: [18]   Typ 3: [5]   Typ 4: [18]   Typ 5: [6, 6, 6]
Box 10:         Typ 1: [15, 8]   Typ 2: [15]   Typ 3: [15, 15]   Typ 4: [11]   Typ 5: [12, 12, 12]
Box 11:         Typ 1: [15]   Typ 2: [15]   Typ 3: [15]   Typ 4: [11]   Typ 5: [12]
Box 12:         Typ 1: [8, 8, 8]   Typ 2: [15, 15]   Typ 3: [15]   Typ 4: [11, 11, 11]   Typ 5: [12]
Box 13:         Typ 1: [8, 8, 8]   Typ 2: [15]   Typ 3: [15]   Typ 4: [11, 11, 11]   Typ 5: [12]
Box 14:         Typ 1: [15]   Typ 2: [3, 3, 3]   Typ 3: [4, 4, 4]   Typ 4: [15]   Typ 5: [7, 7]
Box 15:         Typ 1: [15]   Typ 2: [3, 15, 3]   Typ 3: [4, 4, 4]   Typ 4: [15]   Typ 5: [7]
Box 16:         Typ 1: [8]   Typ 2: [15]   Typ 3: [15]   Typ 4: [15]   Typ 5: [12]
Box 17:         Typ 1: [8]   Typ 2: [15]   Typ 3: [15]   Typ 4: [11]   Typ 5: [12]
Box 18:         Typ 1: []   Typ 2: []   Typ 3: [21, 21, 21]   Typ 4: [21, 21, 21]   Typ 5: []
Box 19:         Typ 1: []   Typ 2: []   Typ 3: [21]   Typ 4: [21, 21, 21]   Typ 5: []
Box 20:         Typ 1: [13, 13, 13]   Typ 2: []   Typ 3: []   Typ 4: []   Typ 5: []
Box 21:         Typ 1: [13]   Typ 2: []   Typ 3: []   Typ 4: []   Typ 5: []
Box 22:         Typ 1: []   Typ 2: []   Typ 3: [10, 10, 10]   Typ 4: []   Typ 5: []
Box 23:         Typ 1: []   Typ 2: [1]   Typ 3: []   Typ 4: []   Typ 5: []
Box 24:         Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: []   Typ 5: [6]
```

3.2.7 Beispiel 6

```
paeckchen6.txt
------------------
Korrekte Boxen: 88
Inkorrekte Boxen: 4
Verpackte Kleidungsstücke: 748
Übrige Kleidungsstücke: 22
Übrige / alle Kleidungsstücke: 0.0286
Laufzeit:  0.84 s
------------------
Boxen:
Eine Zeile entspricht einer Box, die wie folgt aufgebaut ist, S entspricht einem beliebigen Stil: [Typ 1:[S, S, S], Typ 2:[S, S, S], ..., Typ n:[S, S, S]]

Muster: Box N: Typ X: [Stil 1, Stil 2, Stil 3]

Box 1:  Typ 1: [7, 7, 7]   Typ 2: [9, 7, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 2:  Typ 1: [7, 7, 7]   Typ 2: [7, 9, 9]   Typ 3: [9, 7, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 3:  Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 4:  Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 5:  Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 6:  Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [7, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 7:  Typ 1: [7, 7, 7]   Typ 2: [9, 7, 9]   Typ 3: [7, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 8:  Typ 1: [7, 7, 7]   Typ 2: [9, 9, 7]   Typ 3: [9, 9, 7]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 9:  Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 10:         Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [7, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 11:         Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10]
Box 12:         Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 7, 9]   Typ 4: [10]   Typ 5: [10]
Box 13:         Typ 1: [7, 7]   Typ 2: [9, 9, 7]   Typ 3: [9, 9, 7]   Typ 4: [10]   Typ 5: [10]
Box 14:         Typ 1: [8, 8, 8]   Typ 2: [4, 4, 4]   Typ 3: [8, 8, 8]   Typ 4: [4, 4, 4]   Typ 5: [10]
Box 15:         Typ 1: [8, 8, 8]   Typ 2: [4, 4, 4]   Typ 3: [8, 8, 8]   Typ 4: [4, 4, 4]   Typ 5: [10]
Box 16:         Typ 1: [8, 8, 8]   Typ 2: [4]   Typ 3: [8, 8, 1]   Typ 4: [4, 4, 4]   Typ 5: [10]
Box 17:         Typ 1: [8, 8, 8]   Typ 2: [4]   Typ 3: [8, 8, 1]   Typ 4: [4, 4, 4]   Typ 5: [10]
Box 18:         Typ 1: [8, 8, 8]   Typ 2: [4]   Typ 3: [1, 1, 1]   Typ 4: [4, 4, 4]   Typ 5: [10]
Box 19:         Typ 1: [8, 8, 8]   Typ 2: [4]   Typ 3: [1]   Typ 4: [4, 4, 4]   Typ 5: [10]
Box 20:         Typ 1: [8, 8, 8]   Typ 2: [4]   Typ 3: [1]   Typ 4: [4, 1, 4]   Typ 5: [10]
Box 21:         Typ 1: [8, 8, 8]   Typ 2: [4]   Typ 3: [1]   Typ 4: [4, 4, 4]   Typ 5: [10]
Box 22:         Typ 1: [8, 8, 8]   Typ 2: [4]   Typ 3: [1]   Typ 4: [4, 4, 4]   Typ 5: [10]
Box 23:         Typ 1: [8, 8, 8]   Typ 2: [4]   Typ 3: [1]   Typ 4: [4, 4, 4]   Typ 5: [10]
Box 24:         Typ 1: [8, 8, 8]   Typ 2: [4]   Typ 3: [1, 1, 8]   Typ 4: [4, 4, 4]   Typ 5: [10]
Box 25:         Typ 1: [6, 6, 6]   Typ 2: [5, 3, 3]   Typ 3: [1]   Typ 4: [1, 3, 3]   Typ 5: [10]
Box 26:         Typ 1: [6, 5, 5]   Typ 2: [5, 3]   Typ 3: [1]   Typ 4: [3, 3, 1]   Typ 5: [10]
Box 27:         Typ 1: [6, 5, 5]   Typ 2: [5]   Typ 3: [1]   Typ 4: [3, 1, 1]   Typ 5: [10]
Box 28:         Typ 1: [6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [3, 1, 1]   Typ 5: [10]
Box 29:         Typ 1: [6]   Typ 2: [3]   Typ 3: [1]   Typ 4: [3, 1, 1]   Typ 5: [10]
Box 30:         Typ 1: [6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [3, 1, 1]   Typ 5: [10]
Box 31:         Typ 1: [6]   Typ 2: [3]   Typ 3: [1]   Typ 4: [3, 1, 1]   Typ 5: [10]
Box 32:         Typ 1: [6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [3, 1, 1]   Typ 5: [10]
Box 33:         Typ 1: [6]   Typ 2: [3]   Typ 3: [1]   Typ 4: [1, 1, 1]   Typ 5: [10]
Box 34:         Typ 1: [5]   Typ 2: [3]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 35:         Typ 1: [6]   Typ 2: [3]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 36:         Typ 1: [6]   Typ 2: [3]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 37:         Typ 1: [6, 6, 6]   Typ 2: [3]   Typ 3: [1]   Typ 4: [3, 3, 3]   Typ 5: [10]
Box 38:         Typ 1: [1]   Typ 2: [4]   Typ 3: [2, 2, 2]   Typ 4: [2]   Typ 5: [10]
Box 39:         Typ 1: [1]   Typ 2: [4]   Typ 3: [2, 2, 2]   Typ 4: [4]   Typ 5: [10]
Box 40:         Typ 1: [1]   Typ 2: [4]   Typ 3: [2, 2, 2]   Typ 4: [4]   Typ 5: [10]
Box 41:         Typ 1: [6, 6, 6]   Typ 2: [3]   Typ 3: [1]   Typ 4: [3, 3, 3]   Typ 5: [10]
Box 42:         Typ 1: [6, 6, 6]   Typ 2: [3, 3]   Typ 3: [1]   Typ 4: [3, 3, 3]   Typ 5: [10, 10]
Box 43:         Typ 1: [6, 6, 6]   Typ 2: [3, 3, 3]   Typ 3: [1]   Typ 4: [3, 3, 3]   Typ 5: [10, 10]
Box 44:         Typ 1: [6, 6, 6]   Typ 2: [3, 3, 3]   Typ 3: [1]   Typ 4: [3, 3, 3]   Typ 5: [10, 10]
Box 45:         Typ 1: [6, 6, 6]   Typ 2: [3, 3, 3]   Typ 3: [1]   Typ 4: [3, 3, 3]   Typ 5: [10, 10]
Box 46:         Typ 1: [6, 6, 6]   Typ 2: [3, 3, 3]   Typ 3: [1]   Typ 4: [3, 3, 3]   Typ 5: [10, 10]
Box 47:         Typ 1: [6, 6, 5]   Typ 2: [3, 3, 3]   Typ 3: [1]   Typ 4: [3, 1, 3]   Typ 5: [10, 10]
Box 48:         Typ 1: [6, 6, 6]   Typ 2: [3, 3, 3]   Typ 3: [1]   Typ 4: [3, 1, 1]   Typ 5: [10, 10]
Box 49:         Typ 1: [5, 5, 5]   Typ 2: [3]   Typ 3: [1]   Typ 4: [1, 1, 1]   Typ 5: [10, 10]
Box 50:         Typ 1: [5, 5]   Typ 2: [3]   Typ 3: [1]   Typ 4: [1, 1, 1]   Typ 5: [10, 10]
Box 51:         Typ 1: [1]   Typ 2: [4, 4]   Typ 3: [2, 2, 2]   Typ 4: [2, 2, 2]   Typ 5: [10]
Box 52:         Typ 1: [1]   Typ 2: [4, 4]   Typ 3: [2, 2, 2]   Typ 4: [2, 2, 2]   Typ 5: [10]
Box 53:         Typ 1: [1]   Typ 2: [4, 4]   Typ 3: [2, 2, 2]   Typ 4: [2, 2, 2]   Typ 5: [10]
Box 54:         Typ 1: [1]   Typ 2: [4, 4]   Typ 3: [2, 2, 2]   Typ 4: [4, 4, 4]   Typ 5: [10]
Box 55:         Typ 1: [1]   Typ 2: [4]   Typ 3: [2, 2, 2]   Typ 4: [4, 4, 4]   Typ 5: [10, 10]
Box 56:         Typ 1: [1]   Typ 2: [4, 4]   Typ 3: [2, 2, 2]   Typ 4: [4, 4, 4]   Typ 5: [10]
Box 57:         Typ 1: [1]   Typ 2: [4, 4]   Typ 3: [2, 2, 2]   Typ 4: [4, 4, 1]   Typ 5: [10]
Box 58:         Typ 1: [1]   Typ 2: [4]   Typ 3: [2, 2, 2]   Typ 4: [1, 1, 4]   Typ 5: [10, 10]
Box 59:         Typ 1: [1]   Typ 2: [3]   Typ 3: [2, 2, 2]   Typ 4: [4]   Typ 5: [10]
Box 60:         Typ 1: [1]   Typ 2: [3]   Typ 3: [2, 2, 2]   Typ 4: [4]   Typ 5: [10]
Box 61:         Typ 1: [1]   Typ 2: [3]   Typ 3: [2, 2, 2]   Typ 4: [4]   Typ 5: [10]
Box 62:         Typ 1: [1]   Typ 2: [3]   Typ 3: [2, 2, 2]   Typ 4: [4]   Typ 5: [10]
Box 63:         Typ 1: [1]   Typ 2: [3]   Typ 3: [2, 2, 2]   Typ 4: [2]   Typ 5: [10]
Box 64:         Typ 1: [1]   Typ 2: [3]   Typ 3: [2, 2, 2]   Typ 4: [2]   Typ 5: [10]
Box 65:         Typ 1: [6]   Typ 2: [3]   Typ 3: [1]   Typ 4: [1]   Typ 5: [10]
```

```
Box 66:         Typ 1: [5]   Typ 2: [3]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 67:         Typ 1: [6]   Typ 2: [3]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 68:         Typ 1: [6]   Typ 2: [3]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 69:         Typ 1: [6]   Typ 2: [3]   Typ 3: [1]   Typ 4: [1]   Typ 5: [10]
Box 70:         Typ 1: [6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 71:         Typ 1: [5]   Typ 2: [3]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 72:         Typ 1: [6]   Typ 2: [3]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 73:         Typ 1: [6]   Typ 2: [3]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 74:         Typ 1: [6]   Typ 2: [3]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 75:         Typ 1: [5]   Typ 2: [5]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 76:         Typ 1: [6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 77:         Typ 1: [6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 78:         Typ 1: [6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 79:         Typ 1: [6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 80:         Typ 1: [6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 81:         Typ 1: [6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [1]   Typ 5: [10]
Box 82:         Typ 1: [6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 83:         Typ 1: [5]   Typ 2: [3]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 84:         Typ 1: [6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 85:         Typ 1: [6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 86:         Typ 1: [6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 87:         Typ 1: [6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 88:         Typ 1: [6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [3]   Typ 5: [10]
Box 89:         Typ 1: []   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: []   Typ 5: []
Box 90:         Typ 1: []   Typ 2: [7, 7, 7]   Typ 3: [9, 9, 9]   Typ 4: []   Typ 5: []
Box 91:         Typ 1: []   Typ 2: [7, 7, 7]   Typ 3: [7, 7, 7]   Typ 4: []   Typ 5: []
Box 92:         Typ 1: []   Typ 2: [7]   Typ 3: [7, 7, 7]   Typ 4: []   Typ 5: []
```

3.2.8 Beispiel 7

```
paeckchen7.txt
------------------
Korrekte Boxen: 50
Inkorrekte Boxen: 30
Verpackte Kleidungsstücke: 569
Übrige Kleidungsstücke: 131
Übrige / alle Kleidungsstücke: 0.1871
Laufzeit:  0.80 s
------------------
Boxen:
Eine Zeile entspricht einer Box, die wie folgt aufgebaut ist, S entspricht einem beliebigen Stil: [Typ 1:[S, S, S], Typ 2:[S, S, S], ..., Typ n:[S, S, S]]

Muster: Box N: Typ X: [Stil 1, Stil 2, Stil 3]

Box 1:  Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 2:  Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 3:  Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 4:  Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 5:  Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 6:  Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 7:  Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 8:  Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 9:  Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 10:         Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 11:         Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 12:         Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 13:         Typ 1: [7, 7]   Typ 2: [9]   Typ 3: [9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 14:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1, 1, 1]   Typ 5: [10, 10, 10]
Box 15:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1, 1, 1]   Typ 5: [10, 10, 10]
Box 16:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1, 1, 1]   Typ 5: [10, 10, 10]
Box 17:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1, 1, 1]   Typ 5: [10, 10, 10]
Box 18:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1, 1, 1]   Typ 5: [10, 10, 10]
Box 19:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1, 1, 1]   Typ 5: [10, 10, 10]
Box 20:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1, 1]   Typ 5: [10, 10, 10]
Box 21:         Typ 1: [1]   Typ 2: [4, 4, 4]   Typ 3: [2, 2, 2]   Typ 4: [3, 4, 4]   Typ 5: [10, 10, 10]
Box 22:         Typ 1: [1]   Typ 2: [4, 4, 4]   Typ 3: [2, 2, 2]   Typ 4: [3, 4, 4]   Typ 5: [10, 10, 10]
Box 23:         Typ 1: [1]   Typ 2: [4, 4, 4]   Typ 3: [2, 2, 2]   Typ 4: [3, 4, 4]   Typ 5: [10, 10, 10]
Box 24:         Typ 1: [1]   Typ 2: [4, 4, 4]   Typ 3: [2, 2, 2]   Typ 4: [4, 3, 4]   Typ 5: [10, 10, 10]
Box 25:         Typ 1: [1]   Typ 2: [4, 4, 4]   Typ 3: [2, 2, 2]   Typ 4: [3, 3, 4]   Typ 5: [10, 10, 10]
Box 26:         Typ 1: [1]   Typ 2: [4, 4, 3]   Typ 3: [2, 2, 2]   Typ 4: [4, 4, 3]   Typ 5: [10, 10, 10]
Box 27:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1]   Typ 5: [10]
Box 28:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1]   Typ 5: [10]
Box 29:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1]   Typ 5: [10]
Box 30:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1]   Typ 5: [10]
Box 31:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1]   Typ 5: [10]
Box 32:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1]   Typ 5: [10]
Box 33:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1]   Typ 5: [10]
Box 34:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1]   Typ 5: [10]
Box 35:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1]   Typ 5: [10]
Box 36:         Typ 1: [1]   Typ 2: [4, 4, 4]   Typ 3: [2, 2, 2]   Typ 4: [4, 4, 3]   Typ 5: [10]
Box 37:         Typ 1: [1]   Typ 2: [4, 3, 3]   Typ 3: [2, 2, 2]   Typ 4: [4, 3, 4]   Typ 5: [10]
Box 38:         Typ 1: [1]   Typ 2: [4, 4, 3]   Typ 3: [2, 2, 2]   Typ 4: [4, 4, 3]   Typ 5: [10]
Box 39:         Typ 1: [1]   Typ 2: [4, 4, 3]   Typ 3: [2, 2, 2]   Typ 4: [4, 4, 3]   Typ 5: [10]
Box 40:         Typ 1: [1]   Typ 2: [4, 3, 3]   Typ 3: [2, 2, 2]   Typ 4: [4, 3, 4]   Typ 5: [10]
Box 41:         Typ 1: [1]   Typ 2: [4, 4, 4]   Typ 3: [2, 2, 2]   Typ 4: [4, 4, 4]   Typ 5: [10]
Box 42:         Typ 1: [1]   Typ 2: [3, 4, 3]   Typ 3: [2, 2, 2]   Typ 4: [3, 4, 4]   Typ 5: [10]
Box 43:         Typ 1: [1]   Typ 2: [3, 3, 4]   Typ 3: [2, 2, 2]   Typ 4: [4, 3, 4]   Typ 5: [10]
Box 44:         Typ 1: [1]   Typ 2: [3, 3, 4]   Typ 3: [2, 2, 2]   Typ 4: [3, 3, 4]   Typ 5: [10]
Box 45:         Typ 1: [1]   Typ 2: [3, 3, 3]   Typ 3: [2, 2, 2]   Typ 4: [3, 3, 3]   Typ 5: [10]
Box 46:         Typ 1: [1]   Typ 2: [3, 3, 3]   Typ 3: [2, 2, 2]   Typ 4: [3, 3, 3]   Typ 5: [10]
Box 47:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1]   Typ 5: [10]
Box 48:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1]   Typ 5: [10]
Box 49:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1]   Typ 5: [10]
Box 50:         Typ 1: [6, 6]   Typ 2: [5]   Typ 3: [1, 1]   Typ 4: [1]   Typ 5: [10]
Box 51:         Typ 1: [8, 8, 8]   Typ 2: []   Typ 3: [8, 8, 8]   Typ 4: []   Typ 5: []
Box 52:         Typ 1: [8, 8, 8]   Typ 2: []   Typ 3: [8, 8, 8]   Typ 4: []   Typ 5: []
Box 53:         Typ 1: [8, 8, 8]   Typ 2: []   Typ 3: [8, 8, 8]   Typ 4: []   Typ 5: []
Box 54:         Typ 1: [8, 8, 8]   Typ 2: []   Typ 3: [8, 8]   Typ 4: []   Typ 5: []
Box 55:         Typ 1: [8, 8, 8]   Typ 2: []   Typ 3: []   Typ 4: []   Typ 5: []
Box 56:         Typ 1: [8, 8, 8]   Typ 2: []   Typ 3: []   Typ 4: []   Typ 5: []
Box 57:         Typ 1: [8, 8, 8]   Typ 2: []   Typ 3: []   Typ 4: []   Typ 5: []
Box 58:         Typ 1: [8, 8, 8]   Typ 2: []   Typ 3: []   Typ 4: []   Typ 5: []
Box 59:         Typ 1: [8, 8, 8]   Typ 2: []   Typ 3: []   Typ 4: []   Typ 5: []
Box 60:         Typ 1: [8, 8, 8]   Typ 2: []   Typ 3: []   Typ 4: []   Typ 5: []
Box 61:         Typ 1: [8, 8, 8]   Typ 2: []   Typ 3: []   Typ 4: []   Typ 5: []
Box 62:         Typ 1: []   Typ 2: [3, 3, 3]   Typ 3: []   Typ 4: [4, 4, 4]   Typ 5: []
Box 63:         Typ 1: []   Typ 2: [3, 3, 3]   Typ 3: []   Typ 4: [4, 4, 4]   Typ 5: []
Box 64:         Typ 1: []   Typ 2: [3, 3, 3]   Typ 3: []   Typ 4: [4, 4, 4]   Typ 5: []
Box 65:         Typ 1: []   Typ 2: [3, 3, 3]   Typ 3: []   Typ 4: [4, 4, 4]   Typ 5: []
Box 66:         Typ 1: []   Typ 2: [3, 3, 3]   Typ 3: []   Typ 4: [4, 4, 4]   Typ 5: []
Box 67:         Typ 1: []   Typ 2: [3, 3, 3]   Typ 3: []   Typ 4: [4, 4, 4]   Typ 5: []
Box 68:         Typ 1: []   Typ 2: [3, 3, 3]   Typ 3: []   Typ 4: [4, 4, 4]   Typ 5: []
```

```
Box 69:         Typ 1: []   Typ 2: [3, 3, 3]   Typ 3: []   Typ 4: [3, 3, 3]   Typ 5: []
Box 70:         Typ 1: []   Typ 2: [3, 3, 3]   Typ 3: []   Typ 4: [3, 3, 3]   Typ 5: []
Box 71:         Typ 1: []   Typ 2: [3, 3, 3]   Typ 3: []   Typ 4: [3, 3, 3]   Typ 5: []
Box 72:         Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [3, 3, 3]   Typ 5: []
Box 73:         Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [3, 3, 3]   Typ 5: []
Box 74:         Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [3, 3, 3]   Typ 5: []
Box 75:         Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [3, 3, 3]   Typ 5: []
Box 76:         Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [3, 3, 3]   Typ 5: []
Box 77:         Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [3, 3, 3]   Typ 5: []
Box 78:         Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [3, 3, 3]   Typ 5: []
Box 79:         Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [3, 3, 3]   Typ 5: []
Box 80:         Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [3, 3, 3]   Typ 5: []
```

3.3 Zusätzliche Beispiele

Da „simulated annealing“ die Zeitkomplexität erheblich steigert, soll dessen Profit analysiert werden. So sollen alle vorgegebenen Beispiele mit und ohne dieses Verfahren durchlaufen werden.

Ohne „simulated annealing“:

```
wasted pieces / all pieces:  0.0000          runtime:  0.00 s
wasted pieces / all pieces:  0.0000          runtime:  0.00 s
wasted pieces / all pieces:  0.0000          runtime:  0.00 s
wasted pieces / all pieces:  0.0208          runtime:  0.00 s
wasted pieces / all pieces:  0.0549          runtime:  0.00 s
wasted pieces / all pieces:  0.1379          runtime:  0.00 s
wasted pieces / all pieces:  0.0623          runtime:  0.00 s
wasted pieces / all pieces:  0.3129          runtime:  0.01 s
```

Mit Verwendung von „simulated annealing“:

```
wasted pieces / all pieces:  0.0000          runtime:  0.00 s
wasted pieces / all pieces:  0.0000          runtime:  0.23 s
wasted pieces / all pieces:  0.0000          runtime:  0.01 s
wasted pieces / all pieces:  0.0208          runtime:  0.02 s
wasted pieces / all pieces:  0.0000          runtime:  0.30 s
wasted pieces / all pieces:  0.1092          runtime:  0.11 s
wasted pieces / all pieces:  0.0286          runtime:  0.89 s
wasted pieces / all pieces:  0.1871          runtime:  0.81 s
```

Die Laufzeiten erhöhen sich durchaus erheblich, allerdings liegen sie weiterhin im Bruchteil einer Sekunde, weshalb sie vernachlässigt werden können. Es lässt sich zudem eine nennenswerte Leistungssteigerung verzeichnen. So konnte im 4. Beispiel, „paeckchen4.txt ein perfektes Ergebnis erzielt und in den Beispielen 5-7 deutliche Verbesserungen gemacht werden. 7 ist dabei der Extremfall, welcher eine Steigerung von 60% beschreibt. Somit erweist sich die Nutzung von „simulated annealing“ als geeignet und rechtfertig definitiv die hohe Zeit- und Platzkomplexität.

Anschließend sollen schwerere Beispiele getestet werden.

Das Beispiel „paeckchen7.txt“ soll zur Basis genommen und die Anzahl Stile verdoppelt werden. Dabei sollen diese 10 neuen Stile nur sehr schwer zu kombinieren sein. Das Die neuen Daten sehen wie folgt aus, wobei Veränderungen grün markiert sind.

```
5 20
1 2
1 3
1 4
```

```
1 5
1 6
1 8
1 10
2 3
2 4
2 8
2 10
3 4
3 10
4 10
5 6
5 10
6 10
7 9
7 10
8 10
9 10
11 12
12 14
15 16
17 18
19 20
4 12
9 17
1 20
8 17
15 18
17 15

1 1 17
1 6 59
1 7 38
1 8 33
2 3 49
2 4 32
2 5 20
2 9 37
3 1 59
3 2 51
3 8 11
3 9 38
4 1 33
4 3 58
4 4 50
4 10 13
5 10 102
1 10 20
1 12 15
1 13 20
2 15 20
2 16 10
```

```
2 17 20
3 18 30
3 19 35
3 10 20
4 11 15
4 12 20
4 13 30
4 14 10
5 15 15
1 17 20
5 18 20
4 15 20
```

```
------------------
Korrekte Boxen: 77
Inkorrekte Boxen: 56
Verpackte Kleidungsstücke: 826
Übrige Kleidungsstücke: 214
Übrige / alle Kleidungsstücke: 0.2058
Laufzeit:  2.09 s
------------------
Boxen:
Eine Zeile entspricht einer Box, die wie folgt aufgebaut ist, S entspricht einem beliebigen Stil: [Typ 1:[S, S, S], Typ 2:[S, S, S], ..., Typ n:[S, S, S]]

Muster: Box N: Typ X: [Stil 1, Stil 2, Stil 3]

Box 1:  Typ 1: [17]   Typ 2: [15, 15, 17]   Typ 3: [18, 18, 18]   Typ 4: [15, 15, 15]   Typ 5: [18, 15, 15]
Box 2:  Typ 1: [17]   Typ 2: [15, 15, 17]   Typ 3: [18, 18, 18]   Typ 4: [15, 15, 15]   Typ 5: [15, 15, 15]
Box 3:  Typ 1: [17, 17]   Typ 2: [15, 15, 17]   Typ 3: [18, 18, 18]   Typ 4: [15, 15, 15]   Typ 5: [18, 18, 15]
Box 4:  Typ 1: [17, 17]   Typ 2: [15, 17, 17]   Typ 3: [18, 18]   Typ 4: [15, 15, 15]   Typ 5: [18, 15, 15]
Box 5:  Typ 1: [17, 17, 17]   Typ 2: [15, 15, 15]   Typ 3: [18, 18, 18]   Typ 4: [15]   Typ 5: [18, 18, 18]
Box 6:  Typ 1: [17, 17, 17]   Typ 2: [15, 15, 15]   Typ 3: [18, 18, 18]   Typ 4: [15]   Typ 5: [18, 18, 15]
Box 7:  Typ 1: [17]   Typ 2: [15, 17, 17]   Typ 3: [18, 18]   Typ 4: [15]   Typ 5: [18, 18, 18]
Box 8:  Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 9:  Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 10:         Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 11:         Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 12:         Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 13:         Typ 1: [7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 14:         Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 15:         Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 10]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 16:         Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 17:         Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 18:         Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 19:         Typ 1: [7, 7, 7]   Typ 2: [9, 9, 9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10, 10]
Box 20:         Typ 1: [7, 7, 7]   Typ 2: [9]   Typ 3: [9, 9, 9]   Typ 4: [10]   Typ 5: [10, 10]
Box 21:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1, 1, 1]   Typ 5: [10]
Box 22:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1, 1, 1]   Typ 5: [10]
Box 23:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1, 1, 1]   Typ 5: [10]
Box 24:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1, 1, 1]   Typ 4: [1, 1, 1]   Typ 5: [10]
Box 25:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [10, 10, 10]   Typ 4: [1, 1, 1]   Typ 5: [10]
Box 26:         Typ 1: [6, 6, 6]   Typ 2: [5, 5]   Typ 3: [10, 10, 10]   Typ 4: [1, 1, 1]   Typ 5: [10]
Box 27:         Typ 1: [6, 6, 6]   Typ 2: [5, 5]   Typ 3: [1, 10]   Typ 4: [1, 1, 1]   Typ 5: [10]
Box 28:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [10]   Typ 4: [1, 1]   Typ 5: [10]
Box 29:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [10]   Typ 4: [1]   Typ 5: [10]
Box 30:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [1]   Typ 5: [10]
Box 31:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [10]   Typ 4: [1]   Typ 5: [10]
Box 32:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [1]   Typ 5: [10]
Box 33:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [1]   Typ 5: [10]
Box 34:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [1]   Typ 5: [10]
Box 35:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [1]   Typ 5: [10]
Box 36:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [1]   Typ 5: [10]
Box 37:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [1]   Typ 5: [10]
Box 38:         Typ 1: [6, 6, 6]   Typ 2: [5]   Typ 3: [1]   Typ 4: [1]   Typ 5: [10]
Box 39:         Typ 1: [1]   Typ 2: [3, 3, 3]   Typ 3: [1, 2, 1]   Typ 4: [4, 3, 3]   Typ 5: [10]
Box 40:         Typ 1: [1]   Typ 2: [3, 3, 3]   Typ 3: [1, 2, 1]   Typ 4: [3, 3, 4]   Typ 5: [10]
Box 41:         Typ 1: [1]   Typ 2: [3, 3, 3]   Typ 3: [2, 2, 2]   Typ 4: [3, 3, 3]   Typ 5: [10]
Box 42:         Typ 1: [1]   Typ 2: [3, 3, 4]   Typ 3: [1, 1, 2]   Typ 4: [3, 4, 4]   Typ 5: [10]
Box 43:         Typ 1: [10]   Typ 2: [3, 4, 4]   Typ 3: [1, 2, 2]   Typ 4: [3, 3, 3]   Typ 5: [10]
Box 44:         Typ 1: [1]   Typ 2: [4, 4, 4]   Typ 3: [2, 2, 2]   Typ 4: [4, 4, 4]   Typ 5: [10]
Box 45:         Typ 1: [10]   Typ 2: [3, 4, 4]   Typ 3: [2, 2, 2]   Typ 4: [4, 4, 3]   Typ 5: [10]
Box 46:         Typ 1: [1]   Typ 2: [3, 4, 4]   Typ 3: [2, 2, 2]   Typ 4: [3, 4, 4]   Typ 5: [10]
Box 47:         Typ 1: [1]   Typ 2: [4, 4, 4]   Typ 3: [2, 1, 10]   Typ 4: [3, 3, 4]   Typ 5: [10]
Box 48:         Typ 1: [1]   Typ 2: [3, 4, 4]   Typ 3: [2, 2, 2]   Typ 4: [3, 3, 4]   Typ 5: [10]
Box 49:         Typ 1: [10]   Typ 2: [3, 4, 4]   Typ 3: [1, 2, 2]   Typ 4: [3, 3, 3]   Typ 5: [10]
Box 50:         Typ 1: [1, 1, 10]   Typ 2: [3, 3, 3]   Typ 3: [2, 10, 10]   Typ 4: [3, 3, 3]   Typ 5: [10]
Box 51:         Typ 1: [10]   Typ 2: [3, 3, 3]   Typ 3: [2, 2, 2]   Typ 4: [3, 3, 4]   Typ 5: [10]
Box 52:         Typ 1: [17, 17]   Typ 2: [15, 15, 17]   Typ 3: [18, 18, 18]   Typ 4: [15]   Typ 5: [18, 18, 18]
Box 53:         Typ 1: [17, 17]   Typ 2: [15, 17, 17]   Typ 3: [18, 18, 18]   Typ 4: [15]   Typ 5: [18, 18, 15]
Box 54:         Typ 1: [17]   Typ 2: [17, 17, 17]   Typ 3: [18, 18, 18]   Typ 4: [15]   Typ 5: [18, 18, 15]
Box 55:         Typ 1: [1]   Typ 2: [3, 3, 4]   Typ 3: [1, 1, 2]   Typ 4: [3, 3, 3]   Typ 5: [10, 10]
Box 56:         Typ 1: [10]   Typ 2: [3, 3, 3]   Typ 3: [1, 1, 2]   Typ 4: [3, 4, 3]   Typ 5: [10, 10]
Box 57:         Typ 1: [1]   Typ 2: [3, 3, 4]   Typ 3: [10, 2, 2]   Typ 4: [3, 3, 3]   Typ 5: [10, 10]
Box 58:         Typ 1: [1]   Typ 2: [4, 4, 4]   Typ 3: [1, 2, 2]   Typ 4: [3, 3, 3]   Typ 5: [10]
Box 59:         Typ 1: [10]   Typ 2: [3, 4, 4]   Typ 3: [1, 1, 2]   Typ 4: [3, 3, 3]   Typ 5: [10]
```

```
Box 60:     Typ 1: [10]   Typ 2: [3, 4, 4]   Typ 3: [2, 2, 1]   Typ 4: [3, 4, 4]   Typ 5: [10]
Box 61:     Typ 1: [10]   Typ 2: [3, 4]   Typ 3: [1, 2, 1]   Typ 4: [3, 3, 3]   Typ 5: [10, 10]
Box 62:     Typ 1: [10]   Typ 2: [3, 3, 4]   Typ 3: [2, 1, 2]   Typ 4: [3, 3, 3]   Typ 5: [10, 10]
Box 63:     Typ 1: [10]   Typ 2: [3, 3]   Typ 3: [10, 1, 2]   Typ 4: [3, 3, 3]   Typ 5: [10, 10]
Box 64:     Typ 1: [1]   Typ 2: [3, 3]   Typ 3: [2, 10, 1]   Typ 4: [4, 3, 3]   Typ 5: [10, 10]
Box 65:     Typ 1: [10]   Typ 2: [3, 3]   Typ 3: [1, 1, 2]   Typ 4: [3, 4, 4]   Typ 5: [10, 10]
Box 66:     Typ 1: [10]   Typ 2: [3, 3, 3]   Typ 3: [1, 1, 1]   Typ 4: [3, 4, 4]   Typ 5: [10]
Box 67:     Typ 1: [10]   Typ 2: [4]   Typ 3: [1, 2, 1]   Typ 4: [4, 4, 4]   Typ 5: [10]
Box 68:     Typ 1: [10]   Typ 2: [4]   Typ 3: [1, 2, 1]   Typ 4: [4, 4, 4]   Typ 5: [10]
Box 69:     Typ 1: [10, 1]   Typ 2: [4]   Typ 3: [1, 2, 2]   Typ 4: [3, 4, 4]   Typ 5: [10]
Box 70:     Typ 1: [10]   Typ 2: [3]   Typ 3: [1, 10, 1]   Typ 4: [4, 4, 4]   Typ 5: [10]
Box 71:     Typ 1: [1]   Typ 2: [3, 3]   Typ 3: [2, 10, 1]   Typ 4: [4, 4, 4]   Typ 5: [10]
Box 72:     Typ 1: [17]   Typ 2: [15, 15, 17]   Typ 3: [18]   Typ 4: [15]   Typ 5: [15, 15, 15]
Box 73:     Typ 1: [1]   Typ 2: [3]   Typ 3: [2, 2]   Typ 4: [4, 4, 3]   Typ 5: [10, 10]
Box 74:     Typ 1: [10]   Typ 2: [3]   Typ 3: [1, 2]   Typ 4: [4, 3, 3]   Typ 5: [10, 10]
Box 75:     Typ 1: [10]   Typ 2: [4]   Typ 3: [2, 1]   Typ 4: [4, 4, 3]   Typ 5: [10, 10]
Box 76:     Typ 1: [10]   Typ 2: [3]   Typ 3: [1, 10]   Typ 4: [4, 4, 4]   Typ 5: [10, 10]
Box 77:     Typ 1: [17]   Typ 2: [15, 17, 17]   Typ 3: [18]   Typ 4: [15]   Typ 5: [18, 15]
Box 78:     Typ 1: [13, 13, 13]   Typ 2: []   Typ 3: []   Typ 4: [13, 13, 13]   Typ 5: []
Box 79:     Typ 1: [13, 13, 13]   Typ 2: []   Typ 3: []   Typ 4: [13, 13, 13]   Typ 5: []
Box 80:     Typ 1: [13, 13, 13]   Typ 2: []   Typ 3: []   Typ 4: [13, 13, 13]   Typ 5: []
Box 81:     Typ 1: [13, 13, 13]   Typ 2: []   Typ 3: []   Typ 4: [13, 13, 13]   Typ 5: []
Box 82:     Typ 1: [13, 13, 13]   Typ 2: []   Typ 3: []   Typ 4: [13, 13, 13]   Typ 5: []
Box 83:     Typ 1: [13, 13, 13]   Typ 2: []   Typ 3: []   Typ 4: [13, 13, 13]   Typ 5: []
Box 84:     Typ 1: [13, 13]   Typ 2: []   Typ 3: []   Typ 4: [13, 13, 13]   Typ 5: []
Box 85:     Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [13, 13, 13]   Typ 5: []
Box 86:     Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [13, 13, 13]   Typ 5: []
Box 87:     Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [13, 13, 13]   Typ 5: []
Box 88:     Typ 1: []   Typ 2: []   Typ 3: [19, 19, 19]   Typ 4: []   Typ 5: []
Box 89:     Typ 1: []   Typ 2: []   Typ 3: [19, 19, 19]   Typ 4: []   Typ 5: []
Box 90:     Typ 1: []   Typ 2: []   Typ 3: [19, 19, 19]   Typ 4: []   Typ 5: []
Box 91:     Typ 1: []   Typ 2: []   Typ 3: [19, 19, 19]   Typ 4: []   Typ 5: []
Box 92:     Typ 1: []   Typ 2: []   Typ 3: [19, 19, 19]   Typ 4: []   Typ 5: []
Box 93:     Typ 1: []   Typ 2: []   Typ 3: [19, 19, 19]   Typ 4: []   Typ 5: []
Box 94:     Typ 1: []   Typ 2: []   Typ 3: [19, 19, 19]   Typ 4: []   Typ 5: []
Box 95:     Typ 1: []   Typ 2: []   Typ 3: [19, 19, 19]   Typ 4: []   Typ 5: []
Box 96:     Typ 1: []   Typ 2: []   Typ 3: [19, 19, 19]   Typ 4: []   Typ 5: []
Box 97:     Typ 1: []   Typ 2: []   Typ 3: [19, 19, 19]   Typ 4: []   Typ 5: []
Box 98:     Typ 1: []   Typ 2: []   Typ 3: [19, 19, 19]   Typ 4: []   Typ 5: []
Box 99:     Typ 1: []   Typ 2: []   Typ 3: [19, 19]   Typ 4: []   Typ 5: []
Box 100:    Typ 1: []   Typ 2: [16, 16, 16]   Typ 3: []   Typ 4: []   Typ 5: []
Box 101:    Typ 1: []   Typ 2: [16, 16, 16]   Typ 3: []   Typ 4: []   Typ 5: []
Box 102:    Typ 1: []   Typ 2: [16, 16, 16]   Typ 3: []   Typ 4: []   Typ 5: []
Box 103:    Typ 1: []   Typ 2: [16]   Typ 3: []   Typ 4: []   Typ 5: []
Box 104:    Typ 1: [12, 12, 12]   Typ 2: []   Typ 3: []   Typ 4: [14, 14, 14]   Typ 5: []
Box 105:    Typ 1: [12, 12, 12]   Typ 2: []   Typ 3: []   Typ 4: [14, 14, 14]   Typ 5: []
Box 106:    Typ 1: [12, 12, 12]   Typ 2: []   Typ 3: []   Typ 4: [14, 14, 14]   Typ 5: []
Box 107:    Typ 1: [12, 12, 12]   Typ 2: []   Typ 3: []   Typ 4: [14, 12, 12]   Typ 5: []
Box 108:    Typ 1: [12, 12, 12]   Typ 2: []   Typ 3: []   Typ 4: [11, 11, 11]   Typ 5: []
Box 109:    Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [11, 11, 11]   Typ 5: []
Box 110:    Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [11, 11, 11]   Typ 5: []
Box 111:    Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [11, 11, 11]   Typ 5: []
Box 112:    Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [11, 11, 11]   Typ 5: []
Box 113:    Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [12, 12, 12]   Typ 5: []
Box 114:    Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [12, 12, 12]   Typ 5: []
Box 115:    Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [12, 12, 12]   Typ 5: []
Box 116:    Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [12, 12, 12]   Typ 5: []
Box 117:    Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [12, 12, 12]   Typ 5: []
Box 118:    Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [12, 12, 12]   Typ 5: []
Box 119:    Typ 1: [6, 6, 6]   Typ 2: []   Typ 3: []   Typ 4: []   Typ 5: []
Box 120:    Typ 1: [6, 6]   Typ 2: []   Typ 3: []   Typ 4: []   Typ 5: []
Box 121:    Typ 1: [8, 8, 8]   Typ 2: [17, 17, 17]   Typ 3: [8, 8, 8]   Typ 4: []   Typ 5: []
Box 122:    Typ 1: [8, 8, 8]   Typ 2: [17]   Typ 3: [8, 8, 8]   Typ 4: []   Typ 5: []
Box 123:    Typ 1: [8, 8, 8]   Typ 2: []   Typ 3: [8, 8, 8]   Typ 4: []   Typ 5: []
Box 124:    Typ 1: [8, 8, 8]   Typ 2: []   Typ 3: [8, 8]   Typ 4: []   Typ 5: []
Box 125:    Typ 1: [8, 8, 8]   Typ 2: []   Typ 3: []   Typ 4: []   Typ 5: []
Box 126:    Typ 1: [8, 8, 8]   Typ 2: []   Typ 3: []   Typ 4: []   Typ 5: []
Box 127:    Typ 1: [8, 8, 8]   Typ 2: []   Typ 3: []   Typ 4: []   Typ 5: []
Box 128:    Typ 1: [8, 8, 8]   Typ 2: []   Typ 3: []   Typ 4: []   Typ 5: []
Box 129:    Typ 1: [8, 8, 8]   Typ 2: []   Typ 3: []   Typ 4: []   Typ 5: []
Box 130:    Typ 1: [8, 8, 8]   Typ 2: []   Typ 3: []   Typ 4: []   Typ 5: []
Box 131:    Typ 1: [8, 8, 8]   Typ 2: []   Typ 3: []   Typ 4: []   Typ 5: []
Box 132:    Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [4, 4, 4]   Typ 5: []
Box 133:    Typ 1: []   Typ 2: []   Typ 3: []   Typ 4: [4, 4, 4]   Typ 5: []
```

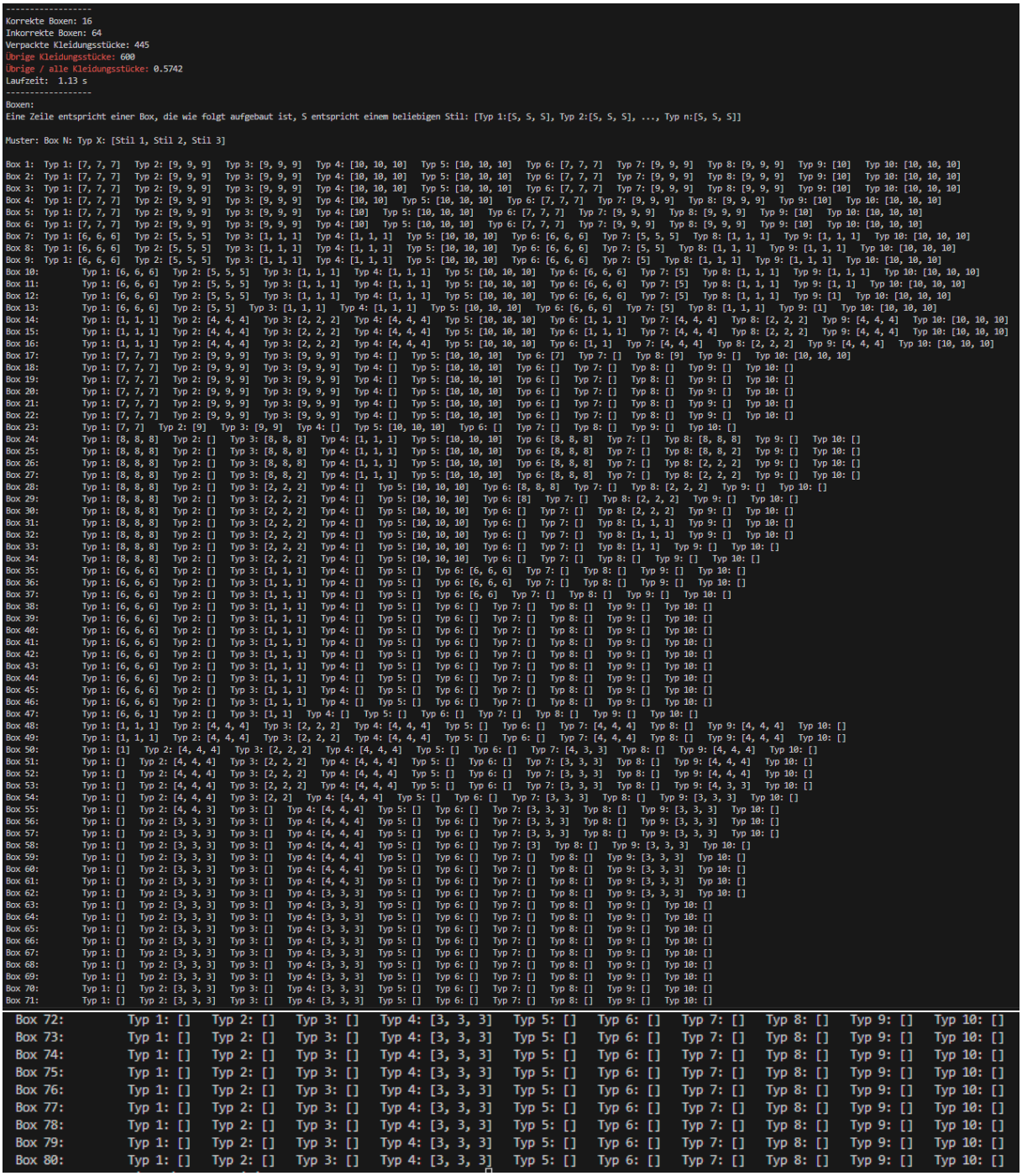
Das Ergebnis ist mit einem Verhältnis 0,2058 der übrigen zu allen Kleidungsstücken sehr zufriedenstellend, besonders wenn beachtet wird, dass dieses Verhältnis vor der Erschwerung bereits nur 0,1871 betrug. Die neuen Stile besitzen kaum Kombinationsmöglichkeiten und die Vielfalt der Typen, die diesen Stil besitzen ist ebenfalls gering, die Anzahl der Kleidungsstücke jedoch sehr hoch, sodass diese durchaus ein nennenswertes Problem darstellen. Wenn man die Lösung genauer betrachtet, wird zudem ersichtlich, dass sich die schwer kombinierbaren Kleidungsstücke erfolgreich in großer Menge in den ersten Boxen angeordnet haben, sodass dem Programm ein großer Erfolg zuzuschreiben ist.

Ein weiterer Fall, der getestet werden soll, ist ein solcher, in dem viele Typen vorliegen, wobei ein Großteil dieser nur gering besetzt ist. So sollen wieder mit Basis „paeckchen7.txt“ die Anzahl Typen verdoppelt werden und diese 5 neuen Typen nur von einer geringen Menge Kleidungsstücke beschrieben werden. Die bereits gegebenen Kleidungsstücke sollen übernommen werden, der Typ um 5 erhöht werden, um sie zu den neuen Typen zu transformieren, und die Anzahl halbiert werden. So lauten die neuen Daten wie folgt.

```
10 10
1 2
1 3
1 4
1 5
1 6
1 8
1 10
2 3
2 4
2 8
2 10
3 4
3 10
4 10
5 6
5 10
6 10
7 9
7 10
8 10
9 10

1 1 17
1 6 59
1 7 38
1 8 33
2 3 49
2 4 32
2 5 20
2 9 37
3 1 59
3 2 51
```

```
3 8 11
3 9 38
4 1 33
4 3 58
4 4 50
4 10 13
5 10 102
6 1 8
6 6 29
6 7 19
6 8 16
7 3 24
7 4 16
7 5 10
7 9 18
8 1 29
8 2 25
8 8 5
8 9 19
9 1 16
9 3 29
9 4 25
9 10 6
10 10 51
```



Zwar erscheint das Verhältnis 0.5742 erschreckend, allerdings ist dieses durchaus akzeptabel. Die anfänglichen Kleidungsstücke wurden gespiegelt und die Spiegelungen in ihrer Anzahl halbiert. Folglich würde man erwarten, dass die Hälfte der Kleidungsstücke näherungsweise, übrigbleibt. Das Programm versucht hauptsächlich durch das Spalten großer Boxen die Anzahl gering vorkommender Typen in jeder Box zu reduzieren und die der häufig vorkommenden zu erhöhen. Diesem Teilproblem wurde allerdings eine große Herausforderung gestellt, denn es müssen ganze 5 fehlende Typen mit nur der Hälfte der Spiegelungen gefüllt werden, wobei diese Kleidungsstücke zudem in ihrer großen Menge beim Übrigbleiben eine erhebliche Verschlechterung herbeirufen. Eine solche des Verhältnisses von 0,3871 ist folglich sehr akzeptabel und erfolgreich.

4. Implementation

4.1 create_combinations_dict_and_matrix

```
def create_combinations_dict_and_matrix(combinations,
styles_amount):
    '''Erstellt eine Dictionary, in welcher für jeden Stil alle
anderen kombinierbaren gespeichert sind und eine Matrix, die die
Kompatibilität zweier Stile als bool beinhaltet'''
    '''Time Complexity: O(s+n) für s Stile und n Kombinationen,
Auxiliary Space: O(s^2) für s Stile'''
    comb_dict = dict()
    comb_matrix = [[False] * (styles_amount + 1) for _ in
range(styles_amount + 1)]

    # Für jeden Stil wird die Kompatibilität mit sich selbst
eingetragen
    for style in range(1, styles_amount+1):
        comb_dict[style] = [style]
        comb_matrix[style][style] = True
    # Jede Kombination wird in die Verzeichnisse beider beteiligter
Stile eingetragen
    for comb in combinations:
        # Die beiden kombinierbaren Stile
        style_one = int(comb[0])
        style_two = int(comb[1])

        # Eintragen in die Dictionary und Matrix, Berücksichtigung
beider Richtungen
        comb_dict[style_one] += [style_two]
        comb_dict[style_two] += [style_one]

        comb_matrix[style_one][style_two] = True
        comb_matrix[style_two][style_one] = True

    return comb_dict, comb_matrix
```

4.2 create_compatibility_ranking

```
def create_compatibility_ranking(comb_dict, styles_amount):
    '''Erstellt eine Liste mit aufsteigender Kompatibilität der
Stile, bzw. Anzahl möglicher Kombinationen mit anderen'''
    '''Time Complexity: O(s^2) für s Stile, Auxiliary Space: O(s)
für s Stile'''
    # Es sollen hier die Stile in aufsteigender Kompatibilität
eingetragen werden, je mehr Kombinationsmöglichkeiten ein Stil hat,
desto kompatibler ist er
    style_compatibility_ranking = []
    # Es wird ein Hilfsarray zum Zählen erstellt
    style_compatibility_counter = [0] * (styles_amount+1)
```

```
    # Es wird die Anzahl Kombinationsmöglichkeiten jedes Stils im
dem eigenen Index gespeichert
    for key, value in comb_dict.items():
        style_compatibility_counter[key] = len(value)
    # Der Stil mit dem größten Wert wird an den Anfang der leeren
Liste gehangen, dann aus dem Hilfsarray entfernt und der Prozess
wiederholt
    for _ in range(styles_amount):
        # Index des Maximums ermitteln
        maximum = max(style_compatibility_counter)
        index = style_compatibility_counter.index(maximum)
        # Anfügen des Stils an den Beginn der Lösungsliste und
Entfernen im Hilfsarray
        style_compatibility_ranking.insert(0, index)
        style_compatibility_counter[index] = 0
    # Die Kompatibilität liegt in aufsteigender Reihenfolge vor
    return style_compatibility_ranking
```

4.3 sort_clothes

```
def sort_clothes(clothes, style_compatibility_ranking, types_amount,
styles_amount):
    '''Sortiert die Kleidung (im noch unveränderten Format der txt-
Datei) nach aufsteigender Kompatibilität der Stile mit
Berücksichtigung des Typen'''
    '''Time Complexity: O(k) für k Kleidungsstücke, Auxiliary Space:
O(k) für k Kleidungsstücke'''
    # Es wird ein Hilfsarray erstellt, die Indexe entsprechen den
Kleidungsstilen
    counting_arr = [[]] * (styles_amount+1)
    # Die Kleidungsstücke werden in das Hilfarray einsortiert
    for piece in clothes:
        counting_arr[int(piece[1])] = counting_arr[int(piece[1])] +
[piece]
    # Kleidungsstücke werden in der Reihenfolge der Kompatibilität
der Stile in eine 2D Liste zusammengefügt. Der Index+1 entspricht
dem Typen und der Wert dem Stil
    types_desc_compatibility = [[]] * (types_amount)
    # Folgende Schleife entspricht weiterhin einer Iteration der
Kleidungsstücke, O(n), nur mit besonderer Betrachtung ihrer Stile
    for style in style_compatibility_ranking:
        # Alle Kleidungsstücke des Stiles werden betrachtet
        for piece in counting_arr[style]:
            # In den Index, repräsentativ für den Typen, wird der
Stil so oft eingefügt, wie es das Kleidungsstück gibt
            types_desc_compatibility[int(piece[0])-1] =
types_desc_compatibility[int(piece[0])-1] + [int(piece[1])] *
int(piece[2])

    return types_desc_compatibility
```

4.4 styles_in_bin

```
def styles_in_bin(bin):
    '''Ermittelt alle Stile innerhalb einer Box'''
    '''Time Complexity: O(k) für k Kleidungsstücke der Box,
Auxiliary Space: O(s) für s Stile der Box'''
    # Nur einzigartige Einträge werden zugelassen
    styles = set()
    # Iterieren und Speichern aller Stile
    for style_list in bin:
        for style in style_list:
            styles.add(style)
    # Zurückgeben als Liste anstatt eines Sets
    return list(styles)
```

4.5 compatible_styles_with_bin

```
def compatible_styles_with_bin(bin, style_combinations_matrix,
styles_amount, extra_styles = []):
    '''Ermittelt alle Stile kombinierbar mit einer Box in O(n*m*k)
für n Stile der Box, m Stile die momentan noch zur Auswahl stehen
und k kompatible Stile mit der Box'''
    '''Time Complexity: O(n*s) für n Stile der Box und s als Menge
aller Stile, Auxiliary Space: O(s) für s als Menge aller Stile'''
    # Alle möglichen Stile
    all_compatible_styles = [x for x in range(1,styles_amount+1)]
    # Alle momentanen Stile
    bin_styles = styles_in_bin(bin) + extra_styles

    # Iterieren aller Stile der Box
    for style in bin_styles:
        i = 0
        while i < len(all_compatible_styles):
            # Jeder Stil der nicht mit dem momentan betrachteten
kombinierbar ist, wird entfernt
            if not
style_combinations_matrix[all_compatible_styles[i]][style]:
                all_compatible_styles.pop(i)
            else:
                i += 1
    # Übrig gebliebene Stile sind kompatibel und werden
zurückgegeben
    return all_compatible_styles
```

4.6 missing_elements

```
def missing_elements(bin, style_combinations_matrix, styles_amount,
extra_styles = []):
    '''Ermittelt alle fehlenden Typen einer Box, eingeschlossen der
kompatiblen Stile in O(n*m*k) für die Funktion
compatible_styles_with_bin'''
```

```
    '''Time Complexity: O(n*s) für n Stile der Box und s als Menge
aller Stile, Auxiliary Space: O(s+t) für s Stile und t Typen'''
    # Kompatible Stile werden ermittelt
    compatible_styles = compatible_styles_with_bin(bin,
style_combinations_matrix, styles_amount, extra_styles=extra_styles)
    # Fehlende Typen werden ermittelt
    types = []
    for i, style_list in enumerate(bin):
        if len(style_list) == 0:
            types.append(i+1)
    return types, compatible_styles
```

4.7 is_insertable

```
def is_insertable(bin, piece, style_combinations_matrix):
    '''Überprüft, ob ein Kleidungsstück in eine Box passt in O(n*m)
für n Stile der Box und m für das Suchen des Stils in einer
Kombinationsliste'''
    '''Time Complexity: O(k) für k Kleidungsstücke der Box,
Auxiliary Space: O(s) für s Stile der Box'''
    piece_type, piece_style = piece
    # Überprüft, ob die Box noch Platz für den Typen bietet
    if len(bin[piece_type-1]) >= 3:
        return False
    # Untersucht, ob der Stil mit allen anderen der Box kombinierbar
ist
    styles_bin = styles_in_bin(bin)
    for style in styles_bin:
        if not style_combinations_matrix[style][piece_style]:
            return False
    return True
```

4.8 insert_piece

```
def insert_piece(bin, piece):
    '''Fügt ein Kleidungsstück in-place in eine Box ein'''
    '''Time Complexity: O(1), Auxiliary Space: O(1)'''
    piece_type, piece_style = piece
    bin[piece_type-1] = bin[piece_type-1] + [piece_style]
```

4.9 evaluate_bins

```
def evaluate_bins(bins):
    '''Wertet das Ergebnis aus'''
    '''Time Complexity: O(b*m) für b Boxen und m Typen, Auxiliary
Space: O(1)'''
    correct_bins = 0
    used_pieces = 0
    wasted_pieces = 0
    for bin in bins:
        if [] not in bin:
            correct_bins += 1
```

```
            used_pieces += sum(map(len, bin))
        else:
            wasted_pieces += sum(map(len, bin))

    incorrect_bins = len(bins)-correct_bins
    return (correct_bins, incorrect_bins, used_pieces,
wasted_pieces, wasted_pieces/(used_pieces+wasted_pieces))
```

4.10 bins_are_correct

```
def bins_are_correct(bins, clothes, style_combinations_matrix):
    '''Überprüft, ob das Ergebnis korrekt ist'''
    '''Time Complexity: O(b*s^2) für b Boxen und s Stile einer Box,
Auxiliary Space: O(s) für s Stile einer Box'''
    # Überprüft die Menge der Kleidungsstücke
    for data in clothes:
        type, style, amount = data
        type, style, amount = int(type), int(style), int(amount)

        cnt = 0
        for bin in bins:
            for c_style in bin[type-1]:
                if c_style == style:
                    cnt+=1
        if cnt != amount:
            print('amount of clothes incorrect')
            return False
    # Überprüft die Kompatibilität der Stile der Boxen
    for bin in bins:
        bin_styles = styles_in_bin(bin)
        i = 0
        while i < len(bin_styles):
            style = bin_styles[i]
            j = i + 1
            while j < len(bin_styles):
                if not
style_combinations_matrix[bin_styles[j]][style]:
                    print('style incompatibility error')
                    print(bins)
                    print('----')
                    print(bin)
                    return False
                j += 1
            i += 1
    return True
```

4.11 display_result

```
def display_result(result, bins, runtime):
    '''Gibt das Ergebnis aus'''
    correct_bins, incorrect_bins, used_pieces, wasted_pieces,
waste_ratio = result
```

```
    print("------------------")
    print("Korrekte Boxen:", correct_bins)
    print("Inkorrekte Boxen:", incorrect_bins)
    print("Verpackte Kleidungsstücke:", used_pieces)
    print(colored("Übrige Kleidungsstücke:", "red"), wasted_pieces)
    print(colored("Übrige / alle Kleidungsstücke:", "red"), "%.4f" %
waste_ratio)
    print("Laufzeit: ", "{:.2f}".format(runtime), "s")
    print("------------------")
    print("Boxen:")
    print("Eine Zeile entspricht einer Box, die wie folgt aufgebaut
ist, S entspricht einem beliebigen Stil: [Typ 1:[S, S, S], Typ 2:[S,
S, S], ..., Typ n:[S, S, S]]")
    print("\nMuster: Box N: Typ X: [Stil 1, Stil 2, Stil 3]\n")
    for i, bin in enumerate(bins):
        print(f"Box {i+1}: ", end="\t")
        for j, type_list in enumerate(bin):
            print(f"Typ {j+1}:", type_list, end="   ")
        print()
```

4.12 first_fit_decreasing

```
def first_fit_decreasing(bins, clothes_desc_compatibility,
style_compatibility_ranking, style_combinations_matrix,
types_amount):
    '''Einfügen der Kleidungsstücke in den erstmöglichen Platz in
aufsteigender Reihenfolge der Kompatibilität der Stile'''
    '''Time Complexity: O(s*t*(k+b)) für s Stile, t Typen, k
Kleidungsstücke des Typen und b Boxen, Auxiliary Space: O(n) für n
Kleidungsstücke'''
    for general_style in style_compatibility_ranking:
        # Einfügen der Kleidungsstücke Stil nach Stil, in der
Reihenfolge der Kompatibilität der Stile
        for type, style_list in
enumerate(clothes_desc_compatibility):
            # Iterieren der Kleidungsstücke eines Types, solange
diese den gesuchten Stil besitzten
            index = 0
            j = 0
            while j < len(style_list):
                style = style_list[j]

                if style != general_style:
                    # Stil wurde bereits ausgeschöpft
                    break

                # Einfügen des Kleidungsstücks in den erstmöglichen
Platz
                piece = [type+1, style]
                while index < len(bins):
                    if is_insertable(bins[index], piece,
style_combinations_matrix):
```

```
                        break
                    index += 1
                if index == len(bins):
                    # Es muss eine neue Box erstellt werden
                    new_bin = [[]] * (types_amount)
                    bins.append(new_bin)
                # Einfügen
                insert_piece(bins[index], piece)
                j+=1
            clothes_desc_compatibility[type] =
clothes_desc_compatibility[type][j:]
    return
```

4.13 fill_wrong_bins

```
def fill_wrong_bins(bins, style_combinations_matrix, styles_amount,
enable_swapping=False):
    '''Unvollständige Boxen werden so weit es geht gefüllt'''
    '''Time Complexity: O(b*k*s) für b Boxen, k Kleidungsstücke und
s kompatible Stile, Auxiliary Space: O(m) für m gesammelte
Kleidungsstücke'''
    # Einzelnes Betrachten der Boxen
    for i, bin in enumerate(bins):
        # Box ist vollständig und wird übersprungen
        if [] not in bin:
            continue
        # Es werden fehlende Typen und die kompatiblen Stile mit der
Box gepeichert
        missing_types, compatible_styles = missing_elements(bin,
style_combinations_matrix, styles_amount)
        # Es werden die fehlenden Kleidungsstücke in anderen Boxen
gesucht und dort entnommen
        pieces, success = search_pieces(bins, i, missing_types,
compatible_styles, style_combinations_matrix, styles_amount)
        # Die Kleidungsstücke werden in die Box eingefügt
        for piece in pieces:
            insert_piece(bins[i], piece)
    return
```

4.14 search_pieces

```
def search_pieces(bins, to_be_filled_i, missing_types,
compatible_styles, style_combinations_matrix, styles_amount):
    '''Es wird ein passendes Kleidungsstück aus einer beliebigen Box
gesucht'''
    '''Time Complexity: O(k*s) für k Kleidungsstücke und s
kompatible Stile, Auxiliary Space: O(m) für m gesammelte
Kleidungsstücke'''
    new_styles = []
    pieces = []
    # Jede Box wird durchsucht
    for bin_i, bin in enumerate(bins):
```

```
        # Die Box entspricht der, für die die Kleidungsstücke
gesucht werden, daher überspringen
        if bin_i == to_be_filled_i:
            continue
        # Die Typen der Box werden nacheinander betrachtet
        for type, style_list in enumerate(bin):
            # Es kann kein passendes Kleidungsstück geben, da
entweder der Typ nicht gesucht wird oder die Menge nicht zum
Entnehmen ausreicht
            if len(style_list) <=1 or type+1 not in missing_types:
                continue
            # Gesuchtes Kleidungsstück existiert potenziell
            for style_i, style in enumerate(style_list):
                # Keine Kompatibilität mit der Box, für die die
Kleidungsstücke gesucht werden, daher überspringen
                if style not in compatible_styles:
                    continue
                # Kleidungsstück ist passend, Entnehmen
                piece_style = style_list.pop(style_i)
                # Entnommenes Stück speichern
                pieces.append([type+1, piece_style])
                # Gesuchte Elemente aktualisieren
                new_styles.append(piece_style)
                missing_types, compatible_styles =
missing_elements(bins[to_be_filled_i], style_combinations_matrix,
styles_amount, extra_styles=new_styles)
                break
            if len(missing_types) == len(pieces):
                # es konnten alle fehlenden Kleidungsstücke gefunden
werden und diese werden zurückgegeben
                return pieces, True
    # Es konnte nur ein Teil der nötigen fehlenden Kleidungsstücke
gefunden werden, diese werden zurückgegeben
    return pieces, False
```

4.15 collect_wrong_pieces

```
def collect_wrong_pieces(bins):
    '''Sammelt alle Kleidungsstücke aus unvollständigen Boxen'''
    '''Time Complexity: O(b*k) für b Boxen und k Kleidungsstücke
einer Box, Auxiliary Space: O(n) für n entnommene Kleidungsstücke'''
    pieces = []
    i = 0
    # Einzelne Betrachtung aller Boxen
    while i < len(bins):
        bin = bins[i]
        # Box ist unvollständig
        if [] in bin:
            # Iterieren und Speichern aller Kleidungsstücke
            for type, style_list in enumerate(bin):
                for style in style_list:
                    pieces.append([type+1, style, 1])
```

```
            # Entfernen der Box
            bins.pop(i)
        else:
            i += 1
    return pieces
```

4.16 insert_pieces_from_wrong_bins

```
def insert_pieces_from_wrong_bins(bins, style_compatibility_ranking,
style_combinations_matrix, types_amount, styles_amount):
    '''Sammelt alle Kleidungsstücke aus unvollständigen Boxen und
fügt diese erneut mit first fit decreasing ein'''
    '''Time Complexity: O(s*t*(k+b)) für s Stile, t Typen, k
Kleidungsstücke des Typen und b Boxen, Auxiliary Space: O(k) für k
einzufügende Kleidungsstücke'''
    # Es werden alle Kleidungsstücke aus unvollständigen Boxen
entfernt
    clothes = collect_wrong_pieces(bins)
    # Die Kleidungsstücke werden erneut eingefügt
    sorted_pieces = sort_clothes(clothes,
style_compatibility_ranking, types_amount, styles_amount)
    first_fit_decreasing(bins, sorted_pieces,
style_compatibility_ranking, style_combinations_matrix,
types_amount)
    return
```

4.17 split_bins

```
def split_bins(bins):
    '''Jede völlständige Box wird auf möglichst viele kleinere, neue
und ebenfalls korrekte Boxen aufgeteilt'''
    '''Time Complexity: O(b*t) für b Boxen und t Typen, Auxiliary
Space: O(k) für k entnommene Kleidungsstücke'''
    i = len(bins)-1
    # Jede Box wird einzeln betrachtet, dabei wird von hinten
iteriert, sodass neue Boxen hinten angehangen werden können, ohne
die Iteration zu stören
    while i >= 0:
        bin = bins[i]
        # Eine Schicht, "layer", liegt vor, wenn von jedem
Kleidungsstück 1 vorhanden ist, folglich entspricht die Anzahl
Schichten der geringsten Anzahl Stücke eines beliebigen Typen
        layers = min(map(len, bin))

        # Jede überflüssige Schicht wird als neue Box am Ende
angehängt
        while layers > 1:
            new_bin = []
            for type, _ in enumerate(bin):
                new_bin.append([bin[type].pop()])
            bins.append(new_bin)
            layers-=1
```

```
        i -= 1
    return
```

4.18 random_solution_swapping

```
def random_solution_swapping(old_bins, probability,
style_combinations_matrix, style_compatibility_ranking,
types_amount, styles_amount):
    '''Es werden zufällige Tausche zwischen je zwei Kleidungsstücken
durchgeführt und anschließend Optimierungen durchgeführt'''
    '''Time Complexity: O(b^2*k*s*n) für b Boxen, k Kleidungsstücke
einer Box, s Stile und n Stile einer Box, Auxiliary Space: O(b*k)
für Boxen und k Kleidungsstücke der Boxen, aufgrund der deepcopy'''
    bins = deepcopy(old_bins)
    # Alle Boxen werden iteriert
    for bin_i, bin in enumerate(bins):
        # Nur vollständige Boxen werden zunächst für Tausche in
Erwägung gezogen, um die Festigkeit derer Elemente zu bekämpfen, da
volle Boxen in den meisten anderen Algorithmen unberührt bleiben
        if [] in bin:
            continue
        for type_i, style_list in enumerate(bin):
            for style_i, style in enumerate(style_list):
                # Zufallsgenerator
                if 1 != randint(1,int(1/probability)):
                    continue
                # Für das Kleidungsstück wird ein Tausch angestrebt
                success=False
                comp_styles = compatible_styles_with_bin(bin,
style_combinations_matrix, styles_amount)

                # Es wird nach einem zweiten Tauschobjekt gesucht
                s_bin_i = bin_i
                while s_bin_i < len(bins)-1:
                    s_bin_i += 1
                    s_bin = bins[s_bin_i]

                    comp_styles_two =
compatible_styles_with_bin(s_bin, style_combinations_matrix,
styles_amount)
                    if style not in comp_styles_two:
                        continue
                    for s_style_i, s_style in
enumerate(s_bin[type_i]):
                        # Kleidungsstücke
                        if s_style==style:
                            continue
                        if s_style in comp_styles:
                            # Zufallsgenerator
                            if 1 != randint(1,int(1/probability)):
                                continue
                            # Tausch wird durchgeführt
```

```
                            success=True
                            break
                    if success:
                        break
                if not success:
                    continue
                # Tausch wird realisiert
                bins[bin_i][type_i][style_i],
bins[s_bin_i][type_i][s_style_i]=s_style, style

    # Optimierung
    fill_wrong_bins(bins, style_combinations_matrix, styles_amount)
    split_bins(bins)
    fill_wrong_bins(bins, style_combinations_matrix, styles_amount)
    insert_pieces_from_wrong_bins(bins, style_compatibility_ranking,
style_combinations_matrix, types_amount, styles_amount)
    return bins
```

4.19 simulated_annealing

```
def simulated_annealing(bins, epochs, saved_sols, samples,
style_combinations_matrix, style_compatibility_ranking,
types_amount, styles_amount):
    '''Durch zufällige Tausche zwischen Kleidungsstücken soll in
einem strukturierten Prozess die bestmöglichste Aufstellung der
Boxen gefunden werden'''
    '''Time Complexity: O(e*n*m*b^2*k*s*o) für e Epochen, n
gespeicherte Lösungen pro Epoche, m neue Lösungen pro Ursprung, b
Boxen, k Kleidungsstücke einer Box, s Stile und o Stile einer Box
       Auxiliary Space: O(n*m*k) für n gespeicherte Lösungen pro
Epoche und m neue Lösungen pro Ursprung und k Kleidungsstücke in den
Boxen'''
    # Zu Beginn wird der momentane Zustand als bester betrachtet
    cur_bests=[bins]
    # Hier sollen die Lösungen sortiert nach ihrer Qualität
gespeichert werden
    solutions=[]
    heapify(solutions)
    # Mit einer ID werden Fehlermeldungen des Heaps vermieden, wenn
der Wert übereinstimmt und darauffolgend versucht wird, die Liste zu
vergleichen
    id = 0
    for e in range(epochs):
        # Jedes gespeicherte Ergebnis wird einzeln betrachtet
        for i in range(saved_sols):
            # Erster Zyklus der Funktion kann nur ein Ergebnis
beinhalten, überspringen
            if e==0 and i>0:
                break
            # Es werden Stichproblem an dem Ergebnis durchgeführt
und im Heap gespeichert
            for _ in range(samples):
```

```
                sol = random_solution_swapping(cur_bests[i], 0.15,
style_combinations_matrix, style_compatibility_ranking,
types_amount, styles_amount)
                _,_,_,_, val = evaluate_bins(sol)
                id+=1
                heappush(solutions, (val, id, sol))
        # Letzte epoch wurde erreicht, bestes Ergebnis wird
zurückgegeben
        if e==epochs-1:
            val, _, best=heappop(solutions)
            return val, best
        # Die besten Exemplare sollen aktualisiert werden
        cur_bests = []
        # Heap für den nächsten Zyklus
        new_solutions=[]
        heapify(new_solutions)
        # Die besten Exemplare werden für den nächsten Zyklus
gespeichert und zudem in den Heap eingefügt
        for _ in range(saved_sols):
            val, tmp_id, best=heappop(solutions)
            cur_bests.append(best)
            heappush(new_solutions, (val, tmp_id, best))
        # Heap wird überschrieben
        solutions=new_solutions

    # Input-Werte sind falsch
    return 1, None
```

4.20 solve

```
def solve(style_combinations, clothes, types_amount, styles_amount):
    '''Lösungsvorgang eines Beispiels'''
    '''Time Complexity: O(e*n*m*b^2*k*s*o) für e Epochen, n
gespeicherte Lösungen pro Epoche, m neue Lösungen pro Ursprung, b
Boxen, k Kleidungsstücke einer Box, s Stile und o Stile einer Box
       Auxiliary Space: O(n*m) für n gespeicherte Lösungen pro
Epoche und m neue Lösungen pro Ursprung'''
    start_time = time.time()

    # Die Daten werden strukturiert
    # Erstellen eines Verzeichnisses für mögliche Kombinationen
    style_combinations_dict, style_combinations_matrix =
create_combinations_dict_and_matrix(style_combinations,
styles_amount)
    # Sortieren der Kleidungsstücke
    style_compatibility_ranking =
create_compatibility_ranking(style_combinations_dict, styles_amount)
    clothes_desc_compatibility = sort_clothes(clothes,
style_compatibility_ranking, types_amount, styles_amount)

    # Einfügen der Kleidungsstücke
    bins = []
```

```
    first_fit_decreasing(bins, clothes_desc_compatibility,
style_compatibility_ranking, style_combinations_matrix,
types_amount)
    # Unvollständige Boxen werden mithilfe überflüssiger
Kleidungsstücke aus vollständigen Boxen gefüllt
    fill_wrong_bins(bins, style_combinations_matrix, styles_amount)
    # Falls unvollständige Boxen übrig bleiben, sollen derren
Kleidungsstücke entnommen und in bereits vollständige gepackt werden
    insert_pieces_from_wrong_bins(bins, style_compatibility_ranking,
style_combinations_matrix, types_amount, styles_amount)

    _, bins=simulated_annealing(bins, 3, 2, 10,
style_combinations_matrix, style_compatibility_ranking,
types_amount, styles_amount)

    # Füllen unvollständiger Boxen
    fill_wrong_bins(bins, style_combinations_matrix, styles_amount)
    # Vollständige Boxen werden auf möglichst viele kleinere,
ebenfalls vollständige Boxen aufgeteilt
    split_bins(bins)
    # Einfügen der Stücke in bereits vollständige Boxen
    insert_pieces_from_wrong_bins(bins, style_compatibility_ranking,
style_combinations_matrix, types_amount, styles_amount)

    # Auswerten der Lösung
    result = evaluate_bins(bins)
    # Überprüfen der Lösung
    correct_sol = bins_are_correct(bins, clothes,
style_combinations_matrix)

    end_time = time.time()
    elapsed_time = end_time - start_time

    return result, correct_sol, bins, elapsed_time
```

4.21 solve_single_file

```
def solve_single_file(file):
    '''Löst ein bestimmtes Beispiel mit einer detaillierten
Ergebnis-Übersicht'''
    style_combinations, clothes, types_amount, styles_amount =
read_file(file)
    result, correct_sol, bins, elapsed_time =
solve(style_combinations, clothes, types_amount, styles_amount)
    if not correct_sol:
        print("Die Überprüfung des Ergebnisses ergibt einen Fehler")
        exit()
    else:
        display_result(result, bins, elapsed_time)
```

4.22 solve_all_examples

```
def solve_all_examples(dir):
    '''Löst alle Standard-Beispiele'''
    solutions = []
    for i in range(0,8):
        file = dir + f'paeckchen{i}.txt'
        style_combinations, clothes, types_amount, styles_amount =
read_file(file)
        result, correct_sol, _, elapsed_time =
solve(style_combinations, clothes, types_amount, styles_amount)
        if correct_sol:
            solutions.append((result[-1], elapsed_time))
        else:
            solutions.append(("error", elapsed_time))

    for sol in solutions:
        print("wasted pieces / all pieces: ",
"{:.4f}".format(sol[0]), end="\t\t")
        print("runtime: ", "{:.2f}".format(sol[1]), "s")
```

5. Literatur

[1] Albert-Ludwigs-Universität Freiburg, „Bin Packing“

https://ac.informatik.uni-freiburg.de/lak_teaching/ws11_12/combopt/notes/bin_packing.pdf

[2] Prof. Dr. Rolf Klein, Dr. Elmar Langetepe, Dipl. Inform. Thomas Kamphans, „Online-Algorithmen“ (2000)

https://www.eurocg.org/tizian.cs.uni-bonn.de/Lehre/Seminare/ProSem0001/Themen/Ausarbeitung/BinPacking.pdf