Benutzerhandbuch

Software-Engineering

Logistik-Optimierungsbeispiel

eingereicht an der

Montanuniversität Leoben

Vorgelegt von

Hoffelner Mario, m01614788

Nehl Stefan, m00935188

Betreuer/Gutachter

Univ.-Prof. Dipl.-Ing. Dr.techn. Peter Auer

Leoben, 26. August 2022

Inhaltsverzeichnis

Abbildungs- und Tabellenverzeichnis

1 Aufgabenstellung

Die Aufgabenstellung war es, ein Programm zu schreiben, welches eine optimierte Reihenfolge von Arbeitsaufträgen und deren einzelnen Schritten zurückgibt. Die optimale Reihenfolge bzw. Reihenfolge der Arbeitsvorgänge wurde hierbei so definiert, dass die Maximierung des Gewinnes über eine bestimmte Zeiteinheit als optimal angesehen wird. Für diese Optimierung bekamen wir ein Umfeld vorgegeben, welches in der Abbildung grafisch dargestellt wurde.

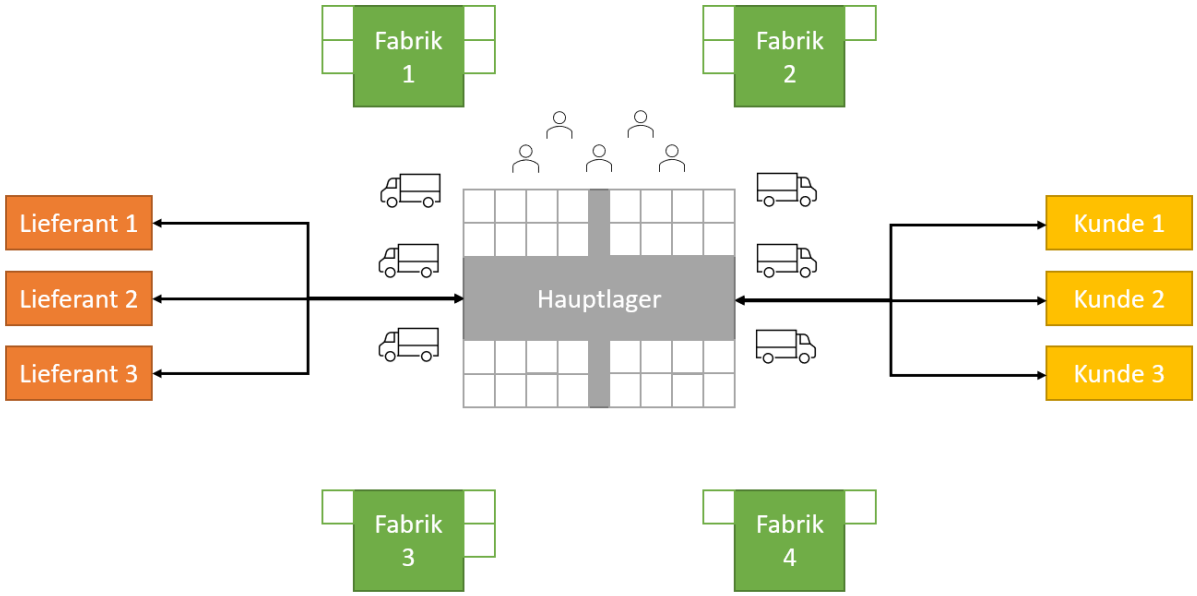

Abbildung 1: Umfeld der Aufgabenstellung

Wie aus der Darstellung ersichtlich, besteht unser Umfeld ausfolgenden Komponenten:

- Lieferanten

Der Ort, wo wir unsere Rohstoffe holen, wird bzw. werden „Lieferanten“ genannt. Im Vergleich zu der normalen Verwendung des Lieferanten Begriffes, holen wir alle unsere Rohstoffe selbst vom „Lieferanten“ ab. Die Rohstoffe haben in unserer Optimierung keinen Kaufpreis. Somit ist der einzigen Faktoren, welche wir hierbei betrachten, die Einschränkungen der Transportmittelkategorien und die dafür benötigte Transportzeit. In der Abbildung 2 wird ein solcher Lieferant beschrieben. Bei ihm können wir mit allen Transportmitteln, welche in der Kategorie „Area“ in 1 fallen, das Rohmaterial Eisenerz mit einer Transportzeit von 30 Sekunden holen.

Transprtmittelkat.		Rohmaterial	Transportzeit
1,x,x		Eisenerz	00:00:30

Abbildung 2: Rohstoffe holen

- Fabriken

In der uns zur Verfügung gestellten Testdatei sind 6 Fabriken vorhanden. Jede Fabrik kann unterschiedliche Produkte fertigen. Die benötigten Produktionszeiten, Rohstoffe und die Losgröße des Endproduktes kann man aus dieser Datei entnehmen. Zusätzlich besitzt jede Fabrik Input bzw. Output Buffer. In diesen können fertige Losgrößen nach der Produktion gelagert werden, bis sie ins Hauptlager bewegt werden. Im Input Buffer können alle Rohmaterialien für einen Batch eines zu produzierenden Produktes gelagert werden. In der Abbildung 3 sehen wir ein Beispiel einer Fabrik. In dieser können wir zwei Produkte fertigen. Zusätzlich ist noch zu erwähnen, dass gleichzeitig immer nur ein Produkt auf einer Maschine gefertigt werden kann.

Fabrik/Puffer	Buffer	Losgröße	Produkte	Prod.-Zeit				
Schmelzhütte	1in/2out	40	Stahl	00:05:00	10	Eisene	30	Kohle
		80	Kupfer	00:30:00	40	Kupfer	40	Kohle

Abbildung 3: Fabrik

Abstrakt betrachtet, beinhaltet eine Firma: einen Namen, Input- und Outputbuffer, und Fertigungsprozesse. Die Anzahl der Input- und Outputbuffer kann hierbei variieren. Fertigungsprozesse besitzen eine Losgröße, ein Produkt, das produziert wird. Eine Produktionszeit und die jeweiligen Rohstoffe oder Zwischenprodukte und die Anzahl, welche für die Produktion der Losgröße benötigt werden.

- Mitarbeitern

Es gibt eine Anzahl von Mitarbeitern, welche die Transporter bedienen. Diese Anzahl wird in unserer Simulation eingegeben und später noch genauer beschrieben.

- Transporter

Weiters erhalten wir aus der gegebenen Excel Datei unsere Transportmittel. Bei den Transportmitteln gibt es Unterscheidungen in 3 Kategorien und der Kapazität eines Transporters. In der Abbildung 4 sehen wir alle Transporter, welche in die Area 1 in der gegebenen Datei fallen und die ersten 3 Transporter, welche in Area 2 fahren können. Jeder Transporter stellt hierbei eine Zeile dar. In der ersten Zeile erkennt man nun einen Transporter, welcher die Einschränkungen Area 1, YE und S beinhaltet und eine Kapazität von 60 umfasst.

#A = Nr Area	(YE,PU,BL,GR)	Eng = (O,E,S)	Kapazität
1	YE	S	60
	PU	S	45
	BL	S	30
	GR	S	20
	GR	S	20
	GR	S	17
	GR	S	15
	PU	S	13
	BL	S	8
2	YE	O	80
	PU	S	45
	YE	E	31

Abbildung 4: Transporter Area 1

- Hauptlager

Das Hauptlager ist das einzige Lager, welches wir verwenden. In diesem werden sowohl Rohstoffe als auch fertige Produkte gelagert. Die Kapazität des Lagers geben wir in unserer Simulation an und wird später noch beschrieben, wie man diese ändert.

- Kunden

Von unseren Kunden erhalten wir Aufträge, diese Aufträge werden in Auftragslisten zusammengefasst und unserer Simulation als Input gegeben. In der Abbildung 5 wird ein Teil einer Auftragsliste abgebildet. Aus dieser kann man die Area, das Produkt, die bestellte Anzahl an Produkten, Transportmittelkategorien für die Auslieferung, den Ertrag und die Auslieferungszeit ablesen.

Aufträge	**3**						
#A	**#P**	**Product**	**Size**	**Ertrag**	**Col**	**Eng**	**T**
5	23	Stahlblech	94	3973	GR	x	30
5	6	Stahl	109	3913	BL	x	30
5	32	Kunststoff	19	3854	YE	x	480
5	31	Kleidersch	106	4960	GR	x	30
5	26	Sperrholz	110	4891	GR	x	30

Abbildung 5: Auftragsliste

2 Aufbau der SRC

Wie in der Abbildung 6 ersichtlich, besteht der *Source-Ordner* nur aus dem *logistikoptimierung-Ordner*.

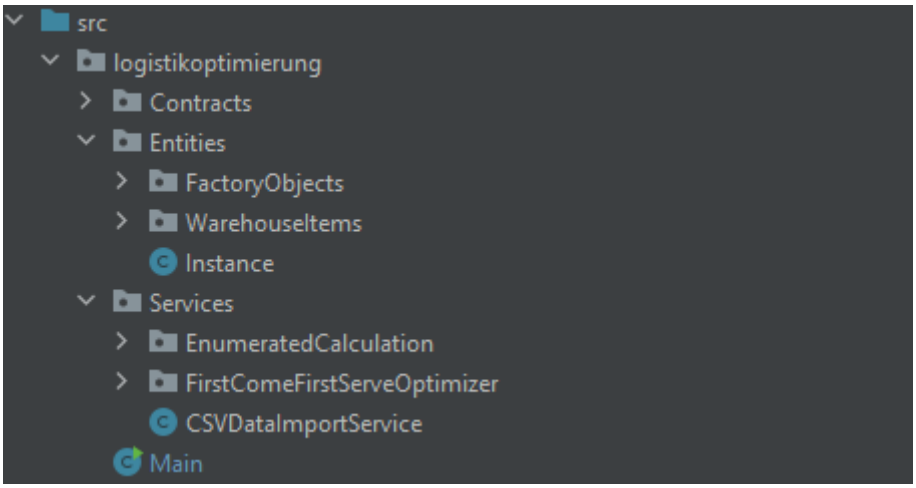

Abbildung 6: Ordnerstruktur der src

2.1 Contracts

Wie in der Abbildung 7 abgebildet, unterteilt sich der Ordner *Contracts* in zwei Klassen, der *IDataService-* und *IOptimizationService-Klasse*. Diese sind zwei Interfaces, welche für den Optimierungsservice und Datenservice implementiert werden müssen.

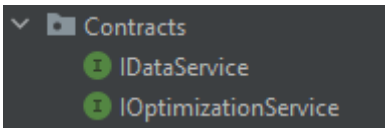

Abbildung 7: Contracts-Ordner

2.2 Entities

Wie aus Abbildung 8 ersichtlich, unterteilt sich der Ordner *Entities* in die Ordner *FactoryObjects*, *WarehouseItems* und in die Klasse *Instance*. In diesen Ordnern befinden sich alle Klassen, welche zur Objekterzeugung benötigt werden. In *FactoryObjects* sind Klassen, welche die aktiven Teile der Simulation darstellen wie z.B. *Driver, Factory* oder auch *Transporter*. Als aktiven Teil bezeichnen wir Teile, die eine Tätigkeit ausführen. Die *WarehouseItems* sind alle Klassen welche die passiven Teile, wie z.B. Material oder Produkt simulieren. Ein Beispiel hierzu wird im nächsten Absatz genauer erläutert. Die genaueren Beschreibungen zu den Klassen bzw. einzelnen Methoden können im *Javadoc* nachgelesen werden.

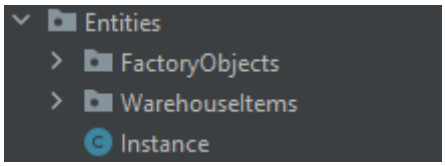

Abbildung 8: Entities-Ordner

FactoryStep

Wir verwenden für unsere Simulation *FactoryStep*'s. Diese *Steps* beinhalten alle Informationen, was mit welchem Objekt, zu welchem Zeitpunkt passieren soll. Ein *FactoryStep* sieht wie in Abbildung 9 aus. Dabei ist das *itemToManipulate* das passive Item. Als Beispiel können wir den Transport von Rohstoffen zum Hauptlager heranziehen. Hierbei ist der aktive Teil, der Transporter und der passive Teil der geholte Rohstoff. Die einzelnen *FactoryStepTyp* sind im Enum FactoryStepTypes hinterlegt. Das Object *factory Konglomerat* stellt die Simulation, also das übergeordnete Objekt, wo die Manipulation durchgeführt werden soll. Die *amountOfItmes* ist die Anzahl an Items, welche manipuliert werden sollen. *doTimeStep* gibt an, wann die Aktion durchgeführt werden soll und *factoryStepsToDoBefore* beschreibt, welche Schritte vor dem aktuellen Schritt ausgeführt werden müssen.

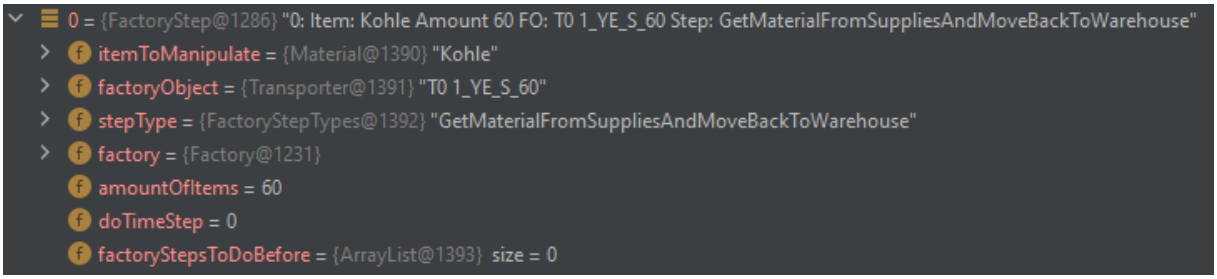

Abbildung 9: FactoryStep

In Abbildung 10 sieht man Beispielhaft die *FactoryStep*'s welche für den Auftrag von 94 Stahlblech benötigt werden. Hierbei starten die *Step*'s mit dem Holen der Rohstoffe und endet mit der Auslieferung der fertigen Produkte an den Kunden.

Alle vorhandenen FactoryStepTypes werden nun genauer beschrieben.

None

Hierbei wird keine Aktion durchgeführt.

GetMaterialFromSuppliesAndMoveBackToWarehouse

Ein Transporter wird losgeschickt, um Material vom Lieferanten zu holen. Als *FactoryObjekt* wird hier ein Transporter benötigt. Weiters wird das Material, welches geholt werden soll, benötigt und die Anzahl.

MoveMaterialFromTransporterToWarehouse

Das geholte Material wird versucht vom Transporter ins Warehouse zu verladen. Als *FactoryObjekt* wird ein Transporter benötigt. Weiters wird das Material benötigt und die Anzahl, welche ins Warehouse verschoben werden soll. Falls das Lager die aktuelle Anzahl nicht aufnehmen kann, schlägt dieser *Step* fehl und unsere *Factory* Konglomerat versucht den *Step* später nochmals durchzuführen. Der Transporter bleibt derweil mit dem Material beladen.

MoveMaterialFromWarehouseToInputBuffer

Material wird in den InputBuffer gelegt. Als *FactoryObjekt* wird eine Fabrik benötigt und welches Produkt produziert werden soll. Für diesen Schritt muss genügend Material im Warehouse vorhanden sein, ansonsten schlägt der *Step* fehl.

Produce

Es wird das Produkt produziert. Als *FactoryObjekt* wird hierbei die Fabrik benötigt und alle Materialien für diesen Batch müssen im InputBuffer vorhanden sein. Sollten die Materialen nicht im InputBuffer sein oder nicht genug Platz im OutputBuffer schlägt dieser Schritt fehl.

MoveProductToOutputBuffer

Die fertigen Produkte werden in den OutputBuffer gelegt. Hierbei ist das *FactoryObjekt* die Fabrik, in der produziert wurde. Weiters wird noch das produzierte Produkt benötigt und im OutputBuffer muss ein Platz frei sein.

MoveProductFromOutputBufferToWarehouse

Die fertigen Produkte werden vom OutputBuffer ins Warehouse befördert. Das *FactoryObjekt* hierbei ist die Fabrik in der produziert wurde. Weiters wird für den Step das produzierte Produkt benötigt und im Warehouse muss genügend Platz vorhanden sein, andernfalls schlägt der *Step* fehl.

ConcludeorderTransportToCustomer

Hierbei werden die bestellten Produkte zum Kunden geliefert. Das *FactoryObjekt* hierbei ist der Transporter, welches die Produkte befördert. Weiters wird hierbei die Anzahl der zu befördernden Materialien benötigt. Sollten nicht genug Produkte im Warehouse vorhanden sein, schlägt der *Step* fehl.

ClosesOrderFromCustomer

Der Auftrag wird abgeschlossen. Das *FactoryObjekt* hier ist der Transporter. Weiters wird das Produkt, welches bestellt wurde, benötigt. Der Parameter der Anzahl wird hierbei ignoriert. Nach diesem Step wird das eingenommene Geld erhöht. Dieser Step schlägt fehl, wenn die benötigte Anzahl an Produkten zum Abschluss des Auftrages noch größer als Null ist.

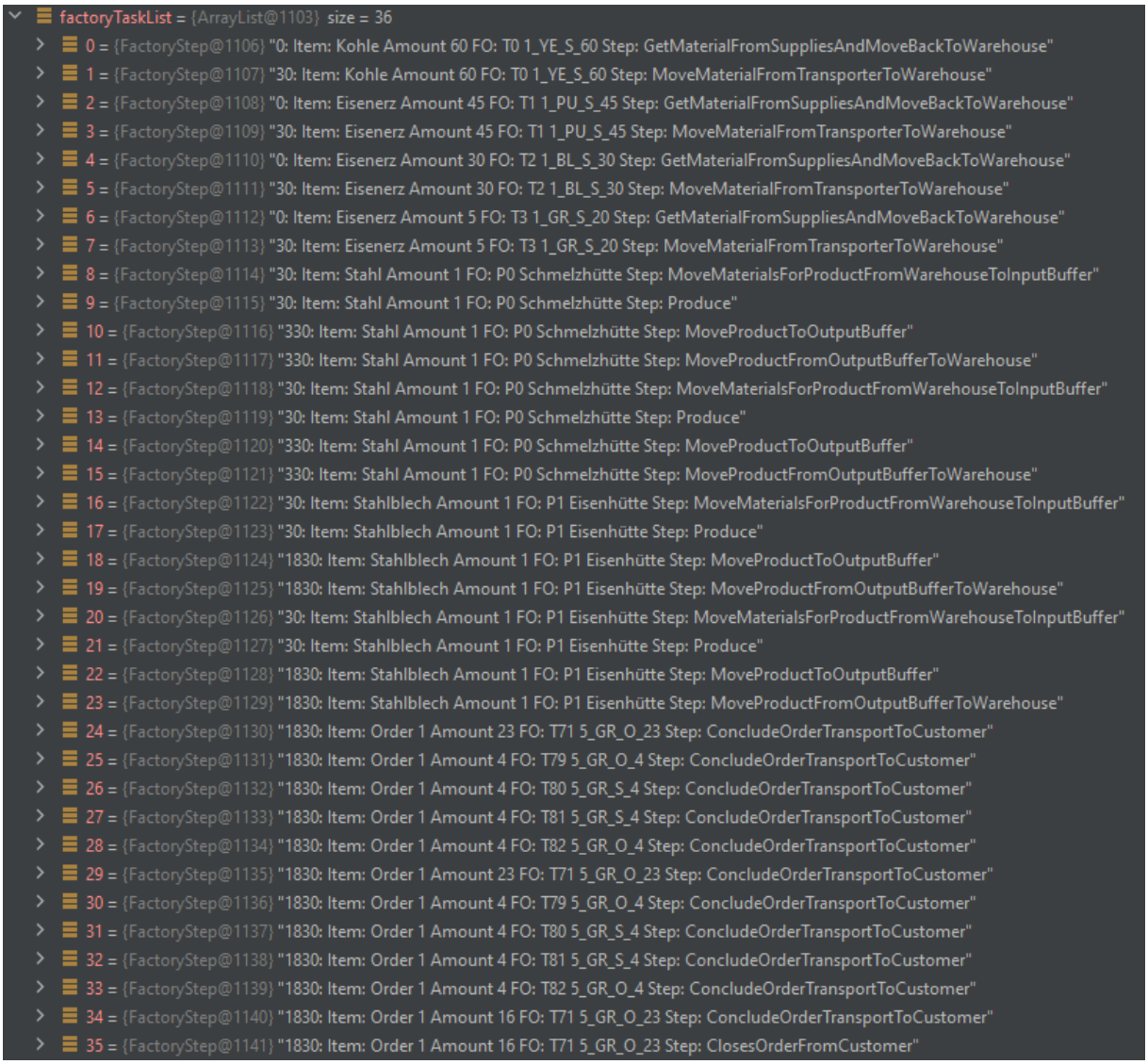

Abbildung 10: Komplette Liste von FactoryStep's für die Produktion von 94 Stahlblech

2.3 Services

Wie in Abbildung 11 ersichtlich, besteht der Ordner *Services* aus den zwei Unterordnern *EnumeratedCalculation* und *FirstComeFirstServeOptimizer* als auch aus der Klasse *CSVDataImportService*.

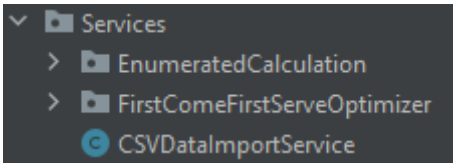

Abbildung 11: Services-Ordner

EnumeratedCalculation - Ordner

Alle Klassen in diesem Ordner sind selbsterklärend bzw. in der Javadoc beschrieben. In dem nächsten Absatz wird noch auf die Klasse *EnumeratedCalculationMain* genauer eingegangen.

In der *EnumeratedCalculationMain* Klasse wird ein Objekt erzeugt, welches dann durch einen rekursiven Aufbau, alle möglichen Kombinationen der Arbeitsschritte durchmischt und berechnet. Die Berechnung erfolgt in unserem Programm durch die Methode *startFactory*. In dieser Methode greifen wir auf die Klasse *logistikoptimierung* → *Entities* → *FactoryObjects* → *Factory* zurück. In dieser, versucht unsere Factory Konglomerates die eingegebene Abfolge der Arbeitsschritte durchzuführen. Am Ende wird entweder die benötigte Zeit zum Abschluss aller Arbeitsschritte zurückgegeben oder die Methode abgebrochen, falls sie eine gewisse Zeit überschreitet.

FirstComeFirstServeOptimizer - Ordner

In der Klasse *FirstComeFirstServeOptimizerMain* wird der FirstComeFirstServe Algorithmus verwendet, um auf ein Simulationsergebnis zu kommen. Hierfür werden *FactorySteps* erzeugt und diese nach dem FirstComeFirstServe Prinzip in die Simulation gepackt. Wir verwenden diese Methode, um einen Startwert zu erhalten, den wir dann mit der Enumeration versuchen zu verbessern.

CSVDataImportService - Klasse

Diese Klasse wird verwendet, um alle Parameter aus der CSV-Datei einzulesen. Die CSV-Dateien sind im *data* Ordner hinterlegt und beinhalten alle Daten, welche zum Erzeugen einer Instanz notwendig sind.

3 Anleitungen

In diesem Kapitel werden bestimmte Anwendungsfälle erklärt, um das Verwenden des Programmes einfacher zu gestalten.

3.1 Einlesen von Dateien

Das Einlesen von Dateien erfolgt über die *CSVDataImportService* Klasse. Diese befindet sich in *logistikoptimierung → Services*. Der Konstruktor dieser Klasse erzeugt ein Objekt, welche das Importieren von CSV Dateien ermöglicht. Die CSV Dateien wurden aus dem uns zur Verfügung gestellten Excel Dokument erzeugt. Als Trennzeichen wurde hier der Strichpunkt genutzt und alle sind in UTF-8 enkodiert. Die für die Simulation notwendigen Daten wurden auf vier CSV Dateien aufgeteilt. Diese sind in Tabelle 1 dargestellt.

Name	Spalten	Beschreibung
Transportmittel.csv	#A = Nr Area	Auslieferungsareal
	(YE,PU,BL,GR)	Typ des Transportmittels.
	Eng = (O,E,S)	Typ des Motors
	Kapazität	Kapazität des Transport- mittels
RohstoffeTransportmittel.csv	Transportmittelkat.	Starten mit der Zahl des Auslieferungsareals, gefolgt durch den Typ des Transportmittels, wobei x für jeden Typ steht. Sollte mehrere Transportmittel geben werden diese in Klammer angegeben mit einem Leerzeichen zwischen den Typen. „(YE PU)“ zum Beispiel für die Typen YE und PU. Auslieferungsareal und Typ ist mit einem Beistrich getrennt. Danach wieder mit einem Beistrich getrennt kommt der Typ des Motors, wobei hier wieder x für jeden Motortypen steht.
	ID	ID des Rohstoffes
	Rohmaterial	Name des Rohstoffes

	Transportzeit	Transportzeit, um den Rohstoff zu beschaffen
Produkte.csv	ID	ID kann leer sein oder eine Zahl beinhalten. Wenn leer, dann steht in der Spalte Rohstoff die Fabrik. Die darauffolgenden Zeilen beinhalten dann eine ID des Produktes oder Rohstoffes, welche in der Fabrik produziert werden können. Abbildung 11 zeigt den Aufbau der CSV Datei.
	Rohstoffe	Name des Rohstoffes, Produktes oder Fabrik.
FabrikenMitPuffer.csv	Fabrik/Puffer	Name der Fabrik. Wenn diese Zelle Leer ist, folgt ein Produktionsprozess, welcher noch zu der Fabrik gehört. Abbildung 12 zeigt diesen Aufbau.
	Buffer	Ein und Ausgangspuffer Größe für die Fabrik. Getrennt durch einen Querstrich.
	Losgröße	Losgröße des Produktes
	Produkte	Produkt welches produziert wird
	Prod.-Zeit	Produktionszeit
	N x 2 -Spalten für die Materialen, wobei N die Anzahl and den Verschiedenen Materialen ist.	Hier stehen die Materialpositionen, welche benötigt werden. Zuerst die Anzahl des Materials dann der Name des Materials.

Tabelle 1: Beschreibung der CSV Dateien

```
;Raffinerie
35;Benzin
36;Klebstoff
37;Plastik
;Lebensmittelbetrieb
39;Mehl
40;Milch
```

Abbildung 12: Aufbau der Produkte.csv

```
Schmelzhütte;1in/2out;40;Stahl;00:05:00;10;Eisenerz;30;Kohle;;
;;80;Kupfer;00:30:00;40;Kupfererz;40;Kohle;;
Eisenhütte;2in/3out;30;Eisenpulver;00:03:00;30;Eisenerz;;;;
;;40;Nägel;00:10:00;40;Stahl;;;;
;;70;Sägeblatt;00:20:00;40;Stahl;30;Eisenpulver;;
;;110;Kupferdraht;00:25:00;80;Kupfer;30;Kupfererz;;
;;70;Stahlblech;00:30:00;40;Stahl;30;Eisenerz;;
```

Abbildung 13: Aufbau der FabrikenMitPuffer.csv

Die möglichen Werte, welche in den Daten vorhanden sind, sind in Tabelle 2 aufgelistet.

Name	Mögliche Werte
Area (Auslieferungsareal)	1, 2, 3, 4, 5
Typ des Transportmittels	YE, PU, BL, GR
Typ des Motors	O, E, S

Tabelle 2: Transportereigentschaften

Die Aufträge wurden in eigenen CSV-Dateien zusammengefasst. Diese werden über die Konstanten in der *CSVDataImportService* aufgerufen. Die Struktur des CSV Dateien für die Aufträge ist in Tabelle 3 dargestellt.

Name	Spalten	Beschreibung
Auftrag.csv (Beispiel Name)	#A	Auslieferungsareal
	#P	ID des Produktes
	Product	Name des Produktes
	Size	Anzahl an Produkten

	Ertrag	Ertrag für den Auftrag
	Col	Typ des Transportmittels
	Eng	Typ des Motors
	T	Transportzeit

Tabelle 3: Beschreibung der Auftrags CSV Datei

Nach dem Erstellen der Klasse *CSVDataImportService* wird mit der Methode *loadDataAndCreateInstance* die Auftragsdatei mit dem Namen, welcher der Methode als Input gegeben wird geladen.

Beispiel:

```
String contractList = CSVDataImportService.PARALLEL_ORDERS;
var dataService = new CSVDataImportService();
var instance = dataService.loadDataAndCreateInstance(contractList);
```

Hierbei definieren wir mit dem String *contractList*, welche Liste im Anschluss geladen werden soll. In der nächsten Zeile erzeugen wir das Objekt der Klasse *CSVDataImportService*. Mit der letzten Zeile laden wir dann alle Aufträge der ausgewählten Auftragsdatei und eine Instanz erzeugt welche das Fabrik Konglomerat, die Auftragsliste, Lagerkapazität und die Anzahl der Fahrer.

3.2 Factory Conglomerate

Das Objekt *FactoryConglomerate* beinhaltet alle Informationen über die aktuelle Simulation. Sie beinhaltet alle Informationen über die Fabriken, die Transporter, die verfügbaren Rohstoffe und auch die Order Liste. Ein Factory Konglomerates kann erst nach dem Einlesen der Daten erstellt werden. Mit der Methode *startSimulation* führt man eine Simulation für die aktuellen Parameter und die übergebene *FactoryStep's* Liste durch. Diese Methode gibt einen long Wert zurück, der entweder die maximale RunTime welche, der Methode als Input gegeben wird oder die Laufzeit, welche bis zum Abschluss aller *FactoryStep's* benötigt wurde.

3.3 Start einer Optimierung

Nachdem man eine Instanz mit Fabrik und Auftragsliste erstellt hat, kann man einen Optimierungsservice erstellen. Hierstehen die zwei zuvor vorgestellten Services zur

Verfügung. Der *FirstComeFirstServeOptimizer* und der EnumeratedCalculation Optimierungsservice. Zum Erstellen dieser Services muss das davor erzeugte Instanzobjekt im Konstruktor übergeben werden. Die Methode *optimize* des jeweiligen Optimierungsservice kann dann aufgerufen werden und ein Integer-Wert mit der Anzahl an Aufträgen übergeben werden. Dieser Parameter der Methode *optimize* ist bestimmt die Anzahl wie viele Aufträge der in der Instanz gespeicherten Liste optimiert werden sollen. Die Optimierungsmethode des EnumeratedCalculation Services erstellt alle möglichen Anordnungen der *FactoryStep*'s. Hierbei beachten wir, dass nur Listen in die Simulation gepackt werden, welche auch logisch sind. Ein Beispiel hierzu wäre: Wir simulieren keine Listen, wo die Auslieferung der Endprodukte vor der Anlieferung der Materialien stattfindet. Im Ablauf der Optimierung werden nun alle möglichen Kombinationen in der Fabrik simuliert und die Ergebnisse in der Konsole ausgegeben und das beste Ergebnis in Form einer Liste mit *FactorySteps* zurückgegeben.

3.4 Main-Klasse

Zu Beginn der Klasse wird die Klasse für den CSV Import erstellt und die Auftragsliste festgelegt und geladen. Als nächstes können wir festlegen, welche Messages wir in der Konsole angezeigt bekommen möchten. Die Einstellungen dafür werden im Record *LogSettings* gespeichert. Die *LogSettings* werden dann in der Instanz gesetzt.

```
var logSettings = new LogSettings(
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false,
        false
);
```

Der nächste Parameter, der verändert werden kann, ist die Anzahl der Fahrer und die Kapazität des Lagers welche auch in der Instanz gesetzt werden. Als weiterer Parameter folgt die Anzahl an Aufträgen, die wir optimieren möchten. Hierbei empfiehlt es sich, eine Zahl ≥ 1 zu wählen. Die String Variable gibt unserem Projekt an, welche Auftragsliste wir optimieren möchten. Der nächste Parameter, der hier abgeändert werden kann, ist die maximale Laufzeit einer Simulation. Diese ist deswegen relevant, da unsere Simulation mit Zeitschritten arbeitet und in jedem

Zeitschritt versucht alle Aufträge, der Reihe nach abzuarbeiten. Diese *maxRuntimeInSeconds* ist jedoch nur für den ersten Durchlauf relevant. Mehr hierzu finden Sie unter dem Kapitel 4.2.

```
int nrOfOrderToOptimize = 5;
long maxRuntimeInSeconds = 10000000;
var maxSystemRunTimeInSeconds = 1800;
boolean fillWarehouseWith20PercentOfNeededMaterials = false;
```

Der nächste Parameter, der gesetzt werden kann, ist *maxSystemRunTimeInSeconds.* Dieser setzt die maximale Berechnungszeit in Sekunden. Nach dieser Zeit wird die Berechnung abgebrochen und derzeit beste gefundene Ergebnis zurückgegeben. Der letzte Parameter ist *fillWarehouseWith20PercentOfNeededMaterials,* dieser legt fest, ob das Lager schon mit 20 Prozent der benötigten Materialen befüllt ist. Diese werden dann bei der Simulation ignoriert. Dies bedeutet, wenn ein Transporter 20 Holz holen muss, diese sich jedoch schon im Lager befinden, so wird dieser Schritt auf erledigt gesetzt und nicht durchgeführt.

Möchte man nun einen neuen Algorithmus verwenden muss dieser einfach im gleichen Schema erzeugt wie die anderen Tests, indem vorher die Daten geladen werden und dann die Instanz dem jeweiligen Algorithmus übergeben werden.

4 Enumeration Calculation Algorithmus

Der von uns angewandte Algorithmus erstellt für die einzelnen Aufträge eine Prozessliste für das Holen des Materials, das Produzieren eines Produktes und die Lieferung zum Kunden. Diese werden dann rekursive zu der verschiedenen Kombination zusammengefügt und simuliert. Da die Anzahl der Auswahlmöglichkeiten faktoriell mit der Anzahl an Prozessschritten steigt haben wir versucht Lösungen wegzuschneiden umso schneller auf eine Lösung zu kommen.

4.1 Erstellen der Prozessschritte - PlanningItems

Für die Erstellung der Prozessschritte wurde die Klasse *PlanningItems* erzeugt. Diese hat drei verschiedene Typen, *Acquire* für das Holen von Materialen, *Produce* für das Produzieren eines Loses und *Deliver* für das ausliefern.

4.1.1 Acquire

Für das Holen von Materialen werden zuerst die Prozessschritte berechnet, welche notwendig sind, um ein Produkt zu produzieren. Dies geschieht unter Berücksichtigung der Losgröße der einzelnen Produktionsprozesse. Aus diesen Prozessschritten werden dann die benötigten Materialen abgeleitet. Die Zuweisung zu einem Transporter geschieht dann in der Transporterauswahl welche im Punkt 3.2.3 näher besprochen wird.

4.1.2 Produce

Für das Produzieren eines benötigten Produktes wird die vorher erstellte Prozessliste wieder herangezogen und ein *PlanningItem* für jedes Los, welches produziert werden muss, erstellt.

4.1.3 Deliver

Für das Ausliefern wurde für jede Lieferung ein *PlanningItem* erzeugt.

4.2 Bounding, Cutting und Transporterauswahl

Um die Laufzeit unseres Programmes zu reduzieren, haben wir versucht so viele vernünftige Boundings und Cutting einzufügen, wie uns möglich.

4.2.1 Einschränkung der Laufzeit - Bounding

In der *main*-Methode übergeben wir zuerst eine *maxRuntimeInSeconds*, diese wird für den ersten Durchlauf der Simulation als Abbruchkriterium herangezogen. Im Anschluss wird diese Laufzeit auf die Laufzeit der aktuellen besten Lösung begrenzt. Diese Begrenzung haben wir eingeführt, da es nicht möglich ist, eine bessere Lösung zu erhalten, wenn die Laufzeit der aktuellen besten Lösung überschritten ist.

4.2.2 Ausschließen von unmöglichen Reihenfolgen - Cutting

Da wir in unserem Programm, alle Reihenfolgen von *FactorySteps* ausprobieren, kann es zu unmöglichen Kombinationen kommen. Ein Beispiel hierzu wäre, wir möchten, dass unser Factory Konglomerat die Endprodukte ausliefert, bevor diese überhaupt produziert wurden. Solche Varianten verfolgen wir nicht weiter. Um unser Vorgehen leichter zu verstehen, haben wir drei sehr vereinfachte Beispiele in Abbildung 14 dargestellt.

Wird getestet	
Auftragsreihenfolge	
Auftrag	Arbeitsschritt
1	Material holen
1	Produzieren
1	Endprodukt ausliefern
2	Material holen
2	Produzieren
2	Endprodukt ausliefern

Wird getestet	
Auftragsreihenfolge	
Auftrag	Arbeitsschritt
2	Material holen
1	Material holen
1	Produzieren
2	Produzieren
2	Endprodukt ausliefern
1	Endprodukt ausliefern

Wird nicht getestet	
Auftragsreihenfolge	
Auftrag	Arbeitsschritt
1	Endprodukt ausliefern
1	Material holen
2	Material holen
2	Produzieren
2	Endprodukt ausliefern
1	Produzieren

Abbildung 14: Vereinfachte FactorySteps-Liste

4.2.3 Verwendung von verschiedenen Transportern

Um eine möglichst breite Lösungsmenge abzudecken, haben wir uns bei der Auswahl der Transporter folgendes überlegt. Da wir es nicht sinnvoll fanden, für jeden Branch, alle Transporter durchzutesten, haben wir für jeden einzelnen Branch, einen guten Transporter bestimmt und somit eine möglichst hohe Variation bei der Auswahl der Transporter, für verschiedene Transportschritte zu gewährleisten. Um unsere Auswahl leichter zu verstehen, finden Sie im Anschluss ein Beispiel hierzu.

Wir suchen für unsere Transporte immer den guten Transporter, diese Suche bedeutet für uns, wir suchen einen Transporter, der die geringste positive Differenz zur benötigten Kapazität besitzt. In der Abbildung 13 suchen wir zuerst für das Holen von Rohstoff R1 einen Transporter. Hier finden wir den Transporter T1, welchen wir durch die Differenz von 0 als den passenden Transporter identifizieren und verwenden. Für das Holen von Rohstoffen R2 haben wir den gleichen Fall und finden den Transporter T2. Für das Holen von Rohstoffen R3 finden wir keinen direkt passenden Transporter. Hier stehen uns nun nur noch die Transporter T3, T4 und T5 zur Verfügung. In unserem Sucherverfahren identifizieren wir nun den Transporter T3 als den Transporter mit der geringsten negativen Differenz. Diesen wählen wir aus und beginnen nun die Suche für einen weiteren Transporter um die noch fehlende Menge abzudecken. Hierbei finden wir den Transporter T5, welche die restliche Differenz genau abdeckt. Für das Holen von Rohstoff R4 finden wir dann zum Schluss noch den Transporter T4.

FactorySteps	benötigtes Material	ausgewählter Transporter
Holen von Rohstoff R1	40	
Holen von Rohstoff R2	39	
Holen von Rohstoff R3	35	
Holen von Rohstoff R4	30	

Verfügbare Transporter	Kapazität
T1	40
T2	39
T3	30
T4	30
T5	5

FactorySteps	benötigtes Material	ausgewählter Transporter
Holen von Rohstoff R1	40	T1
Holen von Rohstoff R2	39	
Holen von Rohstoff R3	35	
Holen von Rohstoff R4	30	

Verfügbare Transporter	Kapazität
T1	40
T2	39
T3	30
T4	30
T5	5

FactorySteps	benötigtes Material	ausgewählter Transporter
Holen von Rohstoff R1	40	T1
Holen von Rohstoff R2	39	T2
Holen von Rohstoff R3	35	
Holen von Rohstoff R4	30	

Verfügbare Transporter	Kapazität
T1	40
T2	39
T3	30
T4	30
T5	5

FactorySteps	benötigtes Material	ausgewählter Transporter
Holen von Rohstoff R1	40	T1
Holen von Rohstoff R2	39	T2
Holen von Rohstoff R3	35	T3 & T5
Holen von Rohstoff R4	30	

Verfügbare Transporter	Kapazität
T1	40
T2	39
T3	30
T4	30
T5	5

FactorySteps	benötigtes Material	ausgewählter Transporter
Holen von Rohstoff R1	40	T1
Holen von Rohstoff R2	39	T2
Holen von Rohstoff R3	35	T3 & T5
Holen von Rohstoff R4	30	T4

Verfügbare Transporter	Kapazität
T1	40
T2	39
T3	30
T4	30
T5	5

Abbildung 15: Auswahl der Transporter Bsp. 1

Um zu zeigen, wie die unterschiedliche Reihenfolge der *FactorySteps* die Auswahl der Transporter beeinflussen kann, haben wir in Abbildung 16 eine andere Reihenfolge gewählt und das gleiche Verfahren angewendet. Der Unterschied in diesem Beispiel ist ab dem Holen von Rohstoff R3 zu sehen. Hier wählen wir einen Transporter aus, der eine positive Differenz hat von +4. Somit transportieren wir hier das erste Mal mit verlorenen Kapazitäten. Bei dem Holen von Rohstoff R2 passiert uns das gleiche, jedoch haben wir hier nur eine positive Differenz von +1, welche wir nicht ausnutzen mit diesem Transport. Bei der Suche für einen Transporter für das Holen von Rohstoff R1 haben wir dann ein Problem. Hier finden wir zuerst den Transporter T4 und haben somit noch eine Differenz von 10 Stück, die wir noch holen müssen. Im Anschluss finden wir für diesen Auftrag, den Transporter T5, womit die noch zu holende Anzahl auf 5 schrumpft. Da wir in diesem Beispiel davon ausgehen, dass der zuerst weggeschickte

Transporter, auch wieder als erstes zurück ist. Müssen wir den Transporter T3 auch noch verwenden, um die benötigten restlichen 5 Stück zu holen.

FactorySteps	benötigtes Material	ausgewählter Transporter
Holen von Rohstoff R4	30	
Holen von Rohstoff R3	35	
Holen von Rohstoff R2	39	
Holen von Rohstoff R1	40	

Verfügbare Transporter	Kapazität
T1	40
T2	39
T3	30
T4	30
T5	5

FactorySteps	benötigtes Material	ausgewählter Transporter
Holen von Rohstoff R4	30	T3
Holen von Rohstoff R3	35	
Holen von Rohstoff R2	39	
Holen von Rohstoff R1	40	

Verfügbare Transporter	Kapazität
T1	40
T2	39
T3	30
T4	30
T5	5

FactorySteps	benötigtes Material	ausgewählter Transporter
Holen von Rohstoff R4	30	T3
Holen von Rohstoff R3	35	T2
Holen von Rohstoff R2	39	
Holen von Rohstoff R1	40	

Verfügbare Transporter	Kapazität
T1	40
T2	39
T3	30
T4	30
T5	5

FactorySteps	benötigtes Material	ausgewählter Transporter
Holen von Rohstoff R4	30	T3
Holen von Rohstoff R3	35	T2
Holen von Rohstoff R2	39	T1
Holen von Rohstoff R1	40	

Verfügbare Transporter	Kapazität
T1	40
T2	39
T3	30
T4	30
T5	5

FactorySteps	benötigtes Material	ausgewählter Transporter
Holen von Rohstoff R4	30	T3
Holen von Rohstoff R3	35	T2
Holen von Rohstoff R2	39	T1
Holen von Rohstoff R1	40	T4 & T5 & T3

Verfügbare Transporter	Kapazität
T1	40
T2	39
T3	30
T4	30
T5	5

Abbildung 16: Auswahl der Transporter Bsp. 2

Um die unterschiedlichen Ergebnisse darzustellen, haben wir die Ergebnisse in Abbildung 17 nochmals gegenübergestellt.

FactorySteps	ausgewählter Transporter
Holen von Rohstoff R1	T1
Holen von Rohstoff R2	T2
Holen von Rohstoff R3	T3 & T5
Holen von Rohstoff R4	T4

FactorySteps	ausgewählter Transporter
Holen von Rohstoff R4	T3
Holen von Rohstoff R3	T1
Holen von Rohstoff R2	T2
Holen von Rohstoff R1	T4 & T5 & T3

Abbildung 17: Vergleich der Transporter

5 Ergebnisse

Unser *Branch and Cutting* Algorithmus ist stark abhängig von den Aufträgen und den möglichen Kombinationen, da die Anzahl der Auswahlmöglichkeiten faktoriell mit der Anzahl an *PlanningItems* steigt. Diese Anzahl hängt von der Tiefe einer Produktion, also wie viele Produktionsschritte notwendig sind, um ein Produkt für die Order herzustellen, von der Anzahl der verschiedenen Materialen und die Anzahl an Produktionen die parallele abgearbeitet werden können. Um dies Abzubilden haben wir verschiedene Auftragslisten erstellt, um unseren Algorithmus zu testen. Eine Liste versucht eine große Variation der Aufträge darzustellen, eine die viele ähnliche Produkte produziert und eine Liste mit verschiedenen Prozessen welche parallel abgearbeitet werden können. Die maximale Anzahl an Aufträgen, welche wir optimieren wollen, wurde von 1 auf 5 gesteigert, um die Änderungen in der Laufzeit zu beobachten. Weiters wurde die Kapazität des Lagers auf 1000 gesetzt und die Anzahl an Fahrer auf 6.

5.1 Durchmischte Liste

Wie schon vorhin erwähnt wurde die durchmischte Liste so erzeugt, um möglichst viele Produkte mit unterschiedlicher Prozesstiefe zu haben. Couch hat eine Prozesstiefe von 3, Nägel und Eisenpulver von 2, Kohle und Stahl, welches wir immer vom Hersteller holen, eine Prozesstiefe von 1. Die Auftragsliste ist in Tabelle 3 zu sehen.

Areal	Produkte	Menge	Ertrag	Col	Engine	Zeit
1	Couch	236	4202	GR	x	60
5	Stahl	101	3638	BL	x	30
5	Kohle	40	702	YE	x	3
4	Nägel	105	1563	GR	x	60
5	Eisenpulver	80	2493	YE	x	15

Tabelle 4: Durchmischte Auftragsliste

Die Ergebnisse der einzelnen Auftragslisten werden in den nächsten Tabellen dargestellt. Dabei wird in der Spalte Anzahl Aufträge die Anzahl wie viele Aufträge der Auftragsliste optimiert wurden, in Rechenzeit die Zeit, die der genutzte Rechner brauchte um zu einem Ergebnis zu kommen, Anzahl *PlanningItems* die Anzahl an *PlanningItems* in welche die Aufträge aufgeteilt wurden, Anzahl an Kombinationen, die Anzahl der verschiedenen Kombinationen, Anzahl Simulationen, die Anzahl Simulationen welche berechnet wurden, Ergebnis EC, das Ergebnis der Simulation mit der *EnumeratedCalculation* Optimierung und Ergebnis FCFS, das Ergebnis der First Come First Serve Optimierung. In Tabelle 4 wird der Anstieg der Rechenzeit mit der Anzahl an Kombinationen bzw. der Anzahl an *PlanningItems* dargestellt. Zwar wird die Anzahl an Simulation stark herunter gebrochen und ist nur ein Bruchteil der Anzahl an Kombination jedoch wächst die Anzahl an möglichen Kombinationen sehr schnell an und damit auch unsere Zeit, mit der unser Algorithmus das Problem optimal löst. Die *EnumeratedCalculation* Optimierung kommt zu einer leicht verbesserten Lösung als die *FirstComeFirstServe* Optimierung.

Anzahl Aufträge	Rechen-zeit	Anzahl PlanningItems	Anzahl Kombinationen	Anzahl Simulationen	Ergebnis EC	Ergebnis FCFS
1	00:00:00	7	5040	30	21607	21608
2	00:00:00	9	362880	336	21607	21608
3	00:00:00	11	39916800	1344	21607	21608
4	00:02:56	18	6,4024E+15	73678	21969	22332
5	00:30:00	25	1,5511E+25	7926809	22000	22332

Tabelle 5: Ergebnisse durchmischte Auftragsliste

5.2 Liste mit ähnlichen Produkten

In der Liste mit ähnlichen Produkten haben wir Couch, Stahl und Nägel hinzugefügt. Hier tritt ein ähnlicher Fall wie in der durchmischten Liste auf. Da die Produktion der Couch, Nägel braucht und Nägeln Stahl brauchen steigt auch hier die Anzahl der Kombinationen an einer Maschine, was zu einer hohen Laufzeit führt. Die Auftragsliste ist in Tabelle 5 abgebildet. Tabelle 6 zeigt die Ergebnisse mit dieser Liste.

Areal	Produkte	Menge	Ertrag	Col	Engine	Zeit
1	Couch	236	4202	GR	x	60
5	Stahl	101	3638	BL	x	30
5	Couch	80	3688	x	S	15
4	Nägel	105	1563	GR	x	60
3	Stahl	189	2284	PU	x	60

Tabelle 6: Ähnliche Produkte

Anzahl Aufträge	Rechen-zeit	Anzahl PlanningItems	Anzahl Kombinationen	Anzahl Simulationen	Ergebnis EC	Ergebnis FCFS
1	00:00:00	7	5040	30	21607	21608
2	00:00:00	9	362880	336	21607	21608
3	00:00:22	16	2,0923E+13	33442	21608	21608
4	00:30:00	23	2,5852E+22	9954413	22150	22753
5	00:30:00	25	1,5511E+25	7564401	24312	24498

Tabelle 7: Ergebnis ähnliche Produkte

In Tabelle 6 ist zu sehen, dass bei einer hohen Anzahl an Produktionen auf einer Maschine bzw. dass so bald mehr als 20 *PlanningItems* zum Lösen benötigt werden, sodass die Rechenzeit des Algorithmus schneller steigt. Hier wurde auch ein Abbruch nach 30 Minuten durchgeführt. Allgemeint sieht man auch hier eine leichte Verbesserung der Ergebnisse mit dem entwickelten Algorithmus.

5.3 Liste mit verschiedenen Prozessen

In der Liste mit verschiedenen Prozessen haben wir Aufträge welche parallel abgearbeitet werden können. Die Liste ist in Tabelle 7 abgebildet. Wir haben uns dazu für die Produkte Eisenpulver, Benzin, Mehl und Kunststoffmöbel entschieden, da diese nur wenige Überschneidungen der Produktionsprozesse haben.

Areal	Produkte	Menge	Ertrag	Col	Engine	Zeit
5	Eisenpulver	80	2493	YE	x	15
5	Benzin	45	2377	GR	x	30
5	Mehl	21	1011	BL	x	30
5	Kunststoffmöbel	96	4024	BL	x	30
5	Eisenpulver	60	1172	GR	x	15

Tabelle 8: Verschiedene Prozesse

Anzahl Aufträge	Rechen-zeit	Anzahl PlanningItems	Anzahl Kombinationen	Anzahl Simulationen	Ergebnis EC	Ergebnis FCFS
1	00:00:00	7	5040	3817	3817	3817
2	00:00:00	10	3628800	5040	5704	5705
3	00:00:02	13	6227020800	25200	5704	5705
4	00:30:00	19	1,2165E+17	15591027	11108	11108
5	00:30:00	24	6,2045E+23	11227093	11109	11109

Tabelle 9: Ergebnis verschiedene Prozesse

Hier kommt ein sehr ähnliches Ergebnis zum Vorschein in Bezug auf die Rechenzeit. Leider wurde hier mit unserem Algorithmus keine signifikante Verbesserung erzielt der Simulationszeit erreicht.

5.4 Zusammenfassen der Materialien

In unseren Algorithmus wurde auch die Möglichkeit eingebaut, das Holen der Materialen zu verdichten damit weniger *PlanningItems* zur Planung entstehen. Wir haben diese Möglichkeit mit der Auftragsliste mit den verschiedenen Prozessen getestet. Hier findet eine leichte Verbesserung für die Berechnung von 5 Aufträgen statt. Die Ergebnisse sind in Tabelle 10 dargestellt.

Anzahl Aufträge	Rechen-zeit	Anzahl PlanningItems	Anzahl Kombinationen	Anzahl Simulationen	Ergebnis EC	Ergebnis FCFS
1	00:00:00	5	120	6	3817	3817
2	00:00:00	8	40320	336	5704	5705
3	00:00:02	11	39916800	25200	5704	5705
4	00:30:00	15	1,3077E+12	16120306	11108	11108
5	00:30:00	18	6,4024E+15	12135256	10664	11109

Tabelle 10: Ergebnis Zusammenfassen der Materialien

5.5 Zusammenfassung

Unser Algorithmus hat gewisse Probleme mit der Laufzeit so bald viele Entscheidungen zu treffen sind. Zwar greifen die Cutting in manchen Fällen, jedoch liegt hier sicher noch Verbesserungspotential. Auch eine Parallelisierung für das Branchen würde die Laufzeit des Algorithmus verbessern.