Universität Leipzig

Fachbereich Informatik/Mathematik

Diplomarbeit

im Studiengang Wirtschaftsmathematik

Thema: Flowshop-Probleme mit Puffern:
Analyse und Entwurf eines Ameisenalgorithmus

eingereicht von: Martin Winkel ‹winkelmhc@gmx.de›

eingereicht am: 27. November 2015

Betreuer: Herr Prof. Dr. Martin Middendorf
Herr Tom Hartmann

Inhaltsverzeichnis

Abstract

The aim of this thesis is to design and evaluate an ant algorithm for a certain class of (hybrid) flowshop problems.
These problems consist of a two-staged production process with two identical machines on the second station and buffers in between the stages. For a given number of jobs, each consisting of a processing time and a deadline, the goal is to create a feasible partition and permutation of these jobs which minimizes the total tardiness. Furthermore, a theoretical investigation on the placement and size of the buffers and its impact on the resulting solution space was done.
In order to further improve the swarm intelligence it was combined with a set of different heuristics and pheromone evaluation rules. The resulting variations of the algorithm were later tested and evaluated on a set of benchmark instances. Subsequently, the obtained theoretical insights on the flowshop problems were analyzed in a practical matter and proved to be valuable. Finally, the quality of the solutions found by the designed ant algorithm was analyzed on smaller problem instances. It became apparent that at least for such smaller problems the algorithm creates very good solutions which only differ little from the optimum.

Notation

$1^{st}(\sigma)$	Erster Job aus σ
$(.)$	Leere Queue
$\|M\|$	Kardinalität der Menge M
$\sigma \setminus \sigma'$	σ ohne die Jobs aus σ'
$\geq^{\sigma}$	Ordnung zwischen Jobs bezüglich σ
$C((\sigma_1, \sigma_2))$	Gesamtkosten eines Outputs
$C^P(\sigma, \sigma')$	Pufferkosten um von σ zu σ' überzugehen
$C^Q(\sigma)$	Kosten von σ an einer zweiten Maschine
$\gamma(t)$	Queue der zum Zeitpunkt t bearbeiteten Jobs
$D(j)$	Deadline von j
$\delta^{(k_1,k_2,k_3)}$	Bearbeitungsreihenfolge einer Queue
$\eta_{i,j}$	Heuristische Werte
$GP_i((k_1, k_2, k_3))$	Größter Puffer vor Maschine $i-1$ im Setting
$\mathcal{J}$	Menge der Jobs
$J(\sigma)$	Menge der Jobs aus σ
$j = (t, D)$	Job j mit Bearbeitungszeit t und Deadline D
(k_1, k_2, k_3)	Setting mit Puffern den k_i entsprechenden Größen
$(k_1, k_2, k_3)(\sigma)$	Outputmenge des Settings (k_1, k_2, k_3)
$K^x(j)$	Wert des Jobs j der Kriteriumsfunktion K^x
L	Lösunspopulation des pACO
$\mathbb{N}_0$	$\mathbb{N} \cup \{0\}$
$N_{\geq 2}$	Natürliche Zahlen größer als 1
$nrj(\delta^{(k_1,k_2,k_3)}, \gamma(t))$	Next Required Job
$\mathbb{O}(\sigma)$	Outputmenge von σ
$\mathbb{O}$	Menge aller Outputs
$\mathbb{P}$	Pheromonmatrix
$P_i^{(k_1,k_2,k_3)}$	Puffer des Settings (k_1, k_2, k_3) an der Stelle i
$\mathcal{P}(M)$	Potenzmenge der Menge M
$P(\sigma)$	Puffermenge bezüglich σ und P
$PK((k_1, k_2, k_3))$	Pufferkapazität des Settings
$Pos(j, \sigma)$	Position des Jobs j innerhalb von σ
Q	Menge der Queues
$S(\sigma)$	Split einer Queue σ bezüglich einem speziellen Output
σ	Folge von Jobs
$t(j)$	Bearbeitungszeit von j
$\tau_{i,j}$	Einträge der Pheromonmatrix

1 Einleitung

1.1 Motivation

Durch die immer größer werdende Leistungsfähigkeit moderner Computer und die daraus resultierende Rechenleistung ist der Mensch inzwischen in der Lage, sehr komplexe Strukturen zu simulieren und zu analysieren. Beispielsweise müssen während der Vorhersage des Wetters eine sehr große Anzahl von Daten erfasst, bearbeitet und anschließend sehr aufwendig ausgewertet werden. Ziel einer mathematischen Modellierung war es ursprünglich, die treibenden Kräfte von Prozessen verständlich zu machen. Da die Simulationen und Berechnungen heutiger Probleme jedoch für den menschlichen Verstand zu komplex sind, um nachvollzogen werden zu können, wandelt sich die Mathematik an dieser Stelle von einem klassischen Kausalverständnis in ein Systemverstehen (vgl. [Len15]).
Strukturen dieser Art können nach [Gon11, S.4] auch in der Wirtschaft, beispielsweise bei der Auto- oder Stahlherstellung, gefunden werden. Um die Wettbewerbsfähigkeit zu steigern und Resourcen zu schonen, wird dabei versucht Produktionsabläufe und -reihenfolgen zu verbessern. Die auftretenden Probleme müssen dabei in kurzer Zeit möglichst gut gelöst werden, wodurch es häufig zur Anwendung von Metaheuristiken kommt.
Ziel der vorliegenden Arbeit ist der Entwurf einer solchen naturinspirierten Heuristik, einem Ameisenalgorithmus, zur Optimierung eines zweistufigen Fertigungsprozesses. Dabei müssen sogenannte Jobs, beispielsweise verschiedene Autoteile, zwei unterschiedliche Bearbeitungsstufen durchlaufen, wobei ihre Reihenfolge zwischen den Stationen vertauschbar ist. Aus der Vielzahl von möglichen Produktionsreihenfolgen soll anschließend eine effiziente Sequenz ausgewählt werden, welche die Gesamtverspätung möglichst minimiert.

1.2 Ablaufplanung und Flowshop-Probleme

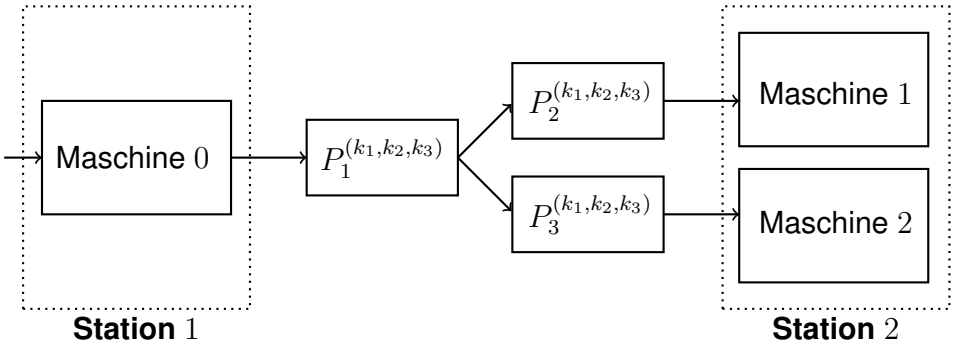
Unter Ablaufplanung bzw. Scheduling versteht man ein Problem, in welchem "eine begrenzte Anzahl von Ressourcen bzw. Maschinen zur Verfügung [stehen] und mit deren Hilfe [...] eine Menge von Aufträgen (oder Jobs) möglichst kostengünstig zu bearbeiten [ist]." [Gon11, S. 4] Das heißt es muss aus allen, durch eine Kostenfunktion bewerteten, zulässigen Zuteilungen zwischen Aufträgen und Maschinen eine günstige ausgewählt werden. Der Begriff "Maschine" kann im übertragenen Sinne verstanden werden und für jegliche Art von Ressourcen, beispielsweise Personal, stehen. Zu den zahlreichen möglichen Anwendungs-

gebieten zählt somit neben den bereits genannten auch etwa das Management eines Flughafens (vgl. [BDG06]).
Flowshop-Probleme bzw. Hybrid-Flowshop-Probleme sind eine spezielle Klasse solcher Scheduling-Probleme, bei denen jeder Job eine bestimmte Anzahl von Maschinenstufen in einer festen Reihenfolge durchlaufen muss. Eine Maschinenstufe kann aus mehreren parallelen Maschinen bestehen, wobei jeder Job lediglich von einer Maschine pro Stufe bearbeitet werden muss (vgl. [Gon11, S. 17]).

1.3 Problemstellung

In den hier betrachteten Flowshop-Problemen werden n zu produzierende Aufträge in zwei Fertigungsstufen und dazwischen liegenden Puffern bearbeitet. Wie in Abbildung 1 zu sehen ist, es existiert eine Maschine in der erste Station (Maschine 0) und zwei identische zweite Maschinen in der zweiten Station (Maschine 1 und 2). Zwischen den Fertigungsstufen erfolgt eine Aufteilung bzw. ein Split, welcher Job zu welcher Maschine der zweiten Fertigungsstufe entsandt wird. Ein Puffer kann sich sowohl vor als auch nach dieser Aufteilung befinden. Die Puffer haben eine feste Größe, das heißt eine endliche Anzahl an Plätzen bzw. Slots, in denen Jobs geladen und vertauscht werden können. Probleme dieser Art werden im Weiteren als Setting bezeichnet.

Abbildung 1: Darstellung der in dieser Arbeit betrachteten Flowshop-Probleme.



Da die einzelnen Settings sich nur durch Pufferpositionen und -größen unterscheiden, werden sie als Tripel (k_1, k_2, k_3) mit $k_1, k_2, k_3 \in \mathbb{N}_0$ dargestellt. Dabei entspricht die erste Komponente der Größe des Puffers P_1, die zweite der Größe des Puffers P_2 und die dritte der Größe des Puffers P_3. Falls an einer Stelle i kein Puffer vorhanden ist, so gilt $k_i = 0$ oder $k_i = 1$, da ein Puffer mit ge-

nau einem Slot zwar einen Job laden könnte, allerdings keine Vertauschungen durchführen kann. Ansonsten soll $k_i \geq 2$ gelten.
Da die Maschinen 1 und 2 identisch sind, sind die Fälle (k_1, k_2, k_3) und (k_1, k_3, k_2) für $k_1, k_2, k_3 \in \mathbb{N}_0$ gleichwertig. O.B.d.A. werden nur Tripel (k_1, k_2, k_3) mit $k_2 \geq k_3$ betrachtet.

Ziel ist es, jeweils günstige Reihenfolgen und Aufteilungen der Jobs zu finden. Kosten entstehen durch Verspätungen der Jobs und Vertauschungen in den Puffern. Eventuelle Kosten aus der ersten Fertigungsstufe können weder durch einen Puffer, noch durch den Split beeinflusst werden. O.B.d.A. werden sie daher vernachlässigt.

1.4 Begriffe und Notationen

Mit $\mathbb{R}_{\geq 0} := \{r \in \mathbb{R} | r \geq 0\}$ versteht man unter einem Auftrag oder Job ein Tupel

$$j = (t, D) \in \mathbb{R}^2_{\geq 0} =: \mathcal{J},$$

wobei t der Fertigungsdauer an Station 2 und D der Deadline des Jobs entspricht. Mit $\mathcal{J}$ wird die Menge aller Aufträge bezeichnet.
Betrachte weiterhin die beiden Funktionen t und D:

$$t : \mathcal{J} \to \mathbb{R}_{\geq 0}, (t, D) \mapsto t,$$

$$D : \mathcal{J} \to \mathbb{R}_{\geq 0}, (t, D) \mapsto D.$$

Für $j = (5, 2)$ ist demnach $t(j) = 5$ und $D(j) = 2$.

Eine Queue $\sigma = (j_1, \ldots, j_n)$ mit $n \in \mathbb{N}$ ist eine Folge von Jobs aus $\mathcal{J}$. Mit $(.)$ werde die leere Queue bezeichnet. Für eine Queue σ ist $J(\sigma) = \{j_1, \ldots, j_n\}$ die Menge der Aufträge aus σ und $|\sigma| := |J(\sigma)| = n$ entspreche der Länge von σ. Es gilt $J((.)) = \emptyset$ und $|(.)| = 0$. Weiterhin sei Q die Menge aller Queues.

Ordnungsrelation

Für $\sigma \in Q$ und $j \in J(\sigma)$ bildet die Funktion

$$Pos : J(\sigma) \times Q \to \{1, \ldots, |\sigma|\} \tag{1}$$

auf die Position des Jobs j in σ ab. Für $\sigma^1 = (j_1, \ldots, j_n)$ und $\sigma^2 = (j_n, \ldots, j_1)$ gilt beispielsweise $Pos(j_1, \sigma^1) = 1$ und $Pos(j_1, \sigma^2) = n$.

Seien $\sigma \in Q$ und $x, y \in J(\sigma)$. Dann ist

$$x \succeq^\sigma y :\iff Pos(x, \sigma) \geq Pos(y, \sigma). \tag{2}$$

Für $\sigma^1 = (j_1, \ldots, j_n)$ gilt beispielsweise $j_5 \succeq^{\sigma_1} j_3$ oder $j_4 \succeq^{\sigma_1} j_4$.

Behauptung: Für $\sigma \in Q$ ist $(J(\sigma), \succeq^\sigma)$ eine total geordnete Menge.
Beweis:

1. *Reflexivität:*
   Sei $x \in J(\sigma)$ und $Pos(x, \sigma) = s \in \{1, \ldots, |\sigma|\}$.
   Es gilt $s \geq s \overset{(2)}{\iff} x \succeq^\sigma x$.

2. *Antisymmetrie:*
   Seien $x, y \in J(\sigma)$, $s, r \in \{1, \ldots, |\sigma|\}$, $Pos(x, \sigma) = s$ und $Pos(y, \sigma) = r$. Gelte $x \succeq^\sigma y$ und $y \succeq^\sigma x$. Laut (2) gilt $s \geq r$ und $r \geq s$, also $s = r$. Somit gilt $x = y$, da auf einer Position in σ nur genau ein Job sein kann.

3. *Transitivität:*
   Seien $x, y, z \in J(\sigma)$, $s, r, l \in \{1, \ldots, |\sigma|\}$, $Pos(x, \sigma) = s$, $Pos(y, \sigma) = r$ und $Pos(z, \sigma) = l$. Nach (2) gilt $s \geq r$ und $r \geq l$. Nach der Transitivität von $(\{1, \ldots, |\sigma|\}, \geq)$ folgt $s \geq l$ und somit $x \succeq^\sigma z$.

4. *Totalität:*
   Da $(\{1, \ldots, |\sigma|\}, \geq)$ eine total geordnete Menge ist, gilt für $s, r \in \{1, \ldots, |\sigma|\}$ $s \geq r$ oder $r \geq s$. Für $x, y \in J(\sigma)$ mit $Pos(x, \sigma), Pos(y, \sigma) \in \{1, \ldots, |\sigma|\}$ gilt $Pos(x, \sigma) \geq Pos(y, \sigma)$ oder $Pos(x, \sigma) \leq Pos(y, \sigma)$ und somit $x \succeq^\sigma y$ oder $y \succeq^\sigma x$.

□

Outputs

Für ein $\sigma \in Q$ versteht man unter einem zugehörigen Output ein Tupel (σ_1, σ_2) mit

(i) $\sigma_1, \sigma_2 \in Q$,

(ii) $J(\sigma_1) \cup J(\sigma_2) = J(\sigma)$ und

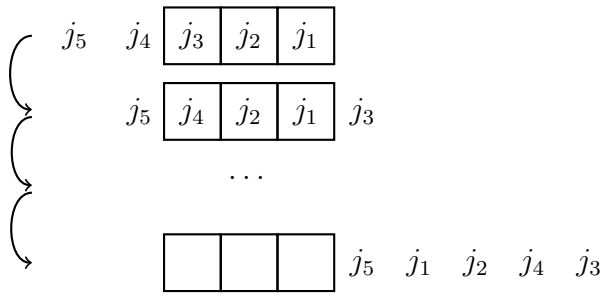
(iii) $J(\sigma_1) \cap J(\sigma_2) = \emptyset$.

Beispielsweise sind $((6,5,4,3,2,1),(.))$ und $((4,2,1),(5,3,6))$ zu $(1,2,3,4,5,6)$ gehörige Outputs. Mit $\mathbb{O}(\sigma)$ werde die Menge aller zu σ gehörigen Outputs bezeichnet. Weiterhin sei $\mathbb{O} := \bigcup_{\sigma \in Q} \mathbb{O}(\sigma)$.

Puffer

Ein Puffer P der Größe $k \in \mathbb{N}$ ordnet die Aufträge einer Queue $\sigma \in Q$ um. Falls $|\sigma| \leq k$ ist, so können alle Jobs gleichzeitig in P geladen und in beliebiger Reihenfolge wieder ausgegeben werden. Falls hingegen $|\sigma| > k$ gilt, so muss die Queue den Puffer sukzessiv durchlaufen:
Wie in Abbildung 2 dargestellt, werden zu Beginn die ersten k Jobs der Queue geladen, eventuell sortiert und anschließend wird der vorderste Job ausgegeben. Immer wenn ein Auftrag aus dem Puffer ausgegeben wird, wird anschließend der nächste Job aus σ geladen bis schließlich alle Jobs aus σ den Puffer durchlaufen haben. Am Ende dieses Prozesses ist eine Permutation der Jobs aus $J(\sigma)$ entstanden.

Abbildung 2: Darstellung des sukzessiven Übergangs der Queue $(j_1, j_2, j_3, j_4, j_5)$ zu $(j_3, j_4, j_2, j_1, j_5)$ durch einen Puffer der Größe 3.



Für jede Nachbarvertauschung entstehen Pufferkosten in Höhe von 1. O.B.d.A. wird auf die Betrachtung von redundanten Vertauschungen verzichtet. Redundante Vertauschungen sind Umsortierungen, welche im Laufe des Pufferns wieder rückgängig gemacht werden. Solche Vertauschungen hätten keinen Einfluss auf die entstehende Umordnung, würden allerdings Pufferkosten erzeugen und somit die Gesamtkosten erhöhen.

Eine Umordnung ändert lediglich die Reihenfolge der Jobs, das heißt die Jobmengen bleiben gleich. Weiterhin gilt, dass wenn ein Auftrag j mit Position $s := Pos(j, \sigma)$ in den Puffer geladen wird, so wird der nächste ausgegebene

Job an Position $(s-k)+1$ gesetzt. Demzufolge kann j durch einen Puffer der Größe k nicht auf eine frühere Position als $s-k+1$ gesetzt werden. Andererseits kann ein geladener Job beliebig spät ausgegeben werden. Mit Hilfe dieser Überlegungen ergibt sich die Menge der Queues, welche durch einen Puffer aus σ erzeugt werden können, als eine Abbildung

$$P : Q \to \mathcal{P}(Q) \tag{3}$$

mit $\sigma' \in P(\sigma)$ falls

1. $J(\sigma) = J(\sigma')$ und
2. $\forall j \in J(\sigma) : Pos(j,\sigma) - Pos(j,\sigma') < k$.

$Pos(j,\sigma')$ darf folglich beliebig größer als $Pos(j,\sigma)$ sein, falls allerdings $Pos(j,\sigma) > Pos(j,\sigma')$ gilt, darf die Differenz aus oben genannten Gründen nicht größer als k sein. Die Menge $P(\sigma)$ wird als Puffermenge bezüglich σ und P bezeichnet.

Die Pufferkosten lassen sich rückwirkend aus den Queues σ und σ' berechnen. Betrachte

$$C^P : Q \times Q \to \mathbb{N}_0 \tag{4}$$

mit

$$C^P(\sigma,\sigma') = \sum_{x \in J(\sigma')} |\{y \in J(\sigma) | Pos(y,\sigma') > Pos(x,\sigma') \text{ und } Pos(y,\sigma) < Pos(x,\sigma)\}|.$$

Für einen Job $x \in J(\sigma')$ gibt der Ausdruck $|\{y \in J(\sigma) | Pos(y,\sigma') > Pos(x,\sigma')$ und $Pos(y,\sigma) < Pos(x,\sigma)\}|$ an, wie viele Aufträge in σ' nach x positioniert sind, in σ jedoch vor x standen. Da für jeden solchen Job eine Vertauschung mit Kosten in Höhe von 1 stattgefunden haben muss, repräsentiert dies die Vertauschungskosten des Jobs x. Summiert man diesen Ausdruck über alle Jobs aus $J(\sigma')$ auf, so ergeben sich gerade die gesamten Pufferkosten, um von σ zu σ' überzugehen.
Betrachte beispielsweise die Queues $\sigma^1 = (j_1,\ldots,j_5)$, $\sigma^2 = (j_5,\ldots,j_1)$, $\sigma^3 = (j_3,j_4,j_2,j_5,j_1)$ und einen Puffer P der Größe 4. σ^2 kann nicht durch P aus σ^1 hervorgehen, da $Pos(j_5,\sigma^1) = 5$ und $Pos(j_5,\sigma^2) = 1$ ist. Allerdings gilt $5-1 = 4 \not< 4$, was nach Definition des Puffers nicht zulässig ist. Hingegen gilt $\sigma^3 \in P(\sigma^1)$.

Split

Im Split wird für einen Job j aus einer Queue $\sigma \in Q$ entschieden, ob dieser zu Maschine 1 oder Maschine 2 entsandt wird.
Für einen Output $(\sigma_1, \sigma_2) \in \mathbb{O}(\sigma)$ kann der zugehörige Split als eine Bipartition der Queue σ aufgefasst werden.

$$S : Q \to Q \times Q. \tag{5}$$

Sei σ' diejenige Queue, welche aus σ hervorgeht, nachdem der Puffer P_1 durchlaufen wurde. Der zugehörige Split ergibt sich als

$$S(\sigma') := (\sigma'_1, \sigma'_2).$$

Dabei gilt

1. $J(\sigma'_1) = J(\sigma_1)$ und $J(\sigma'_2) = J(\sigma_2)$,
2. $J(\sigma'_1) \cup J(\sigma'_2) = J(\sigma')$,
3. $J(\sigma'_1) \cap J(\sigma'_2) = \emptyset$ und
4. Sei $i \in \{1, 2\}$: $\forall x, y \in J(\sigma'_i) : x \succeq^{\sigma'_i} y \Rightarrow x \succeq^{\sigma'} y$.

Der Split spaltet σ' in zwei Queues auf, wobei sich ein Job aus σ' nur in genau einer der beiden Queues σ'_1, σ'_2 befinden darf. Weiterhin dürfen die Reihenfolgen der Jobs innerhalb von σ'_1 bzw. σ'_2 nicht im Widerspruch zur Reihenfolge von σ' stehen.
Beispielsweise erfüllt $S((1, \ldots, 5)) = ((1, 3, 4), (2, 5))$ diese Eigenschaften. Falls $S'((1, \ldots, 5)) = ((1, 4, 3), (2, 5))$ gilt, so handelt es sich bei S' allerdings nicht um einen zugehörigen Split, da $4 \succeq^{(1,\ldots,5)} 3$ und $3 \succeq^{(1,4,3)} 4$ gilt.

Settings

Für $k_1, k_2, k_3 \in \mathbb{N}$ ist ein Setting (k_1, k_2, k_3) somit eine Abbildung

$$(k_1, k_2, k_3) : Q \to \mathbb{O}, \sigma \mapsto (k_1, k_2, k_3)(\sigma) \subseteq \mathbb{O}(\sigma), \tag{6}$$

wobei

$$(k_1, k_2, k_3)(\sigma) = \{(\sigma_1, \sigma_2) \in \mathbb{O}(\sigma) | \exists \overline{\sigma} \in P_1(\sigma) \text{ und ein zu } (\sigma_1, \sigma_2) \text{ gehöriger Split } S(\overline{\sigma}) = (\overline{\sigma}', \overline{\sigma}''), \text{ sodass } \sigma_1 \in P_2(\overline{\sigma}') \text{ und } \sigma_2 \in P_3(\overline{\sigma}'')\}$$

die Outputmenge des Settings (k_1, k_2, k_3) bezüglich σ ist.

1.5 Formulierung als Optimierungsproblem

Die Kosten eines Outputs (σ_1, σ_2) ergeben sich als Summe der Kosten der einzelnen Queues und der Pufferkosten:

$$C((\sigma_1, \sigma_2)) := C^P + C^Q(\sigma_1) + C^Q(\sigma_2). \tag{7}$$

C^P entspricht, analog zu (4), der Anzahl der Nachbarvertauschungen in den Puffern, welche notwenig sind, um von σ zu σ_1 bzw σ_2 überzugehen. Für $i \in \{1, 2\}$ ist

$$C^Q(\sigma_i) := \sum_{l=1}^{|\sigma_i|} \max\{0, time(j_l) - D(j_l)\}, \tag{8}$$

wobei $time(j_l) := \sum_{s=1}^{l} t(j_s)$ der Fertigstellungszeitpunkt des Jobs j_l ist.

Für ein $\sigma \in Q$ und ein Setting (k_1, k_2, k_3) mit $k_1, k_2, k_3 \in \mathbb{N}$ ergibt sich somit das Optimierungsproblem:

$$\begin{aligned} C((\sigma_1, \sigma_2)) &\to \min! \\ (\sigma_1, \sigma_2) &\in (k_1, k_2, k_3)(\sigma). \end{aligned} \tag{9}$$

1.6 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich wie folgt: Im zweiten Kapitel findet zunächst eine theoretische Betrachtung der Settings anhand ihrer realisierbaren Outputs statt. Es werden Zusammenhänge und Inklusionsbeziehungen der Outputmengen in Abhängigkeit der Puffergrößen aufgezeigt, wodurch tiefere Einblicke in die Struktur der Probleme gewonnen werden. Der zur Lösung der Flowshop-Probleme verwendete Ameisenalgorithmus, samt Puffer-Heuristiken und Pheromon-Evaluationsverfahren, wird im dritten Kapitel vorgestellt. Die Ergebnisse aus diesen zwei Abschnitten werden anschließend in Kapitel 4 angewandt und mit Beispielen genauer betrachtet. Es wird auf die Implementierung der Metaheuristik, die verwendeten Beispielsets sowie auf die Lösungsqualität der verwendeten Algorithmen eingangen. Das fünfte Kapitel fasst abschließend die gewonnen Erkenntnisse zusammen und gibt einen Ausblick auf weitere Betrachtungen, welche im Rahmen dieser Arbeit nicht mehr vollzogen werden konnten.

2 Vergleich der Settings anhand ihrer Lösungsmengen

In diesem Kapitel werden allgemeine Pufferpositionen verglichen, ohne dass die Puffer dabei einer fixierten Strategie (einem Pufferalgorithmus) folgen. Die Fragestellung dabei ist, ob Reihenfolgen bzw. Aufteilungen existieren, welche in bestimmten Settings realisiert werden können, in anderen jedoch nicht. Mittels solcher Untersuchungen können tiefere Einblicke in die Vor- und Nachteile der verschiedenen Platzierungen der Puffer erhalten werden.
Etwaige Pufferkosten werden in diesem Kapitel vernachlässigt.

2.1 Notation

Im Weiteren werden Jobs durch ihre Indizes abgekürzt, sodass jeder Job eindeutig durch eine natürliche Zahl repräsentiert ist. Für ein $\sigma \in Q$ gilt

$$f : J(\sigma) \to \{1, \ldots, |\sigma|\}, j_i \mapsto i.$$

Einige Besonderheiten des Bearbeitungsprozesses einer Queue werden im Folgenden aufgelistet.

- Ein Puffer gibt genau dann einen Job aus, wenn er über keinen freien Slot verfügt oder keine weiteren Aufträge existieren, welche er laden kann.
- Ein Moment, Zeitpunkt oder Augenblick $t \in \mathbb{N}_0$ ist wie folgt definiert:
  - $t = 0$:
    Noch kein Job aus σ wurde an einer Maschine bearbeitet oder in einem Puffer geladen.
  - $t \to t + 1$:
    Es wird von einem neuen Moment gesprochen, falls ein Puffer (mindestens) einen neuen Job geladen oder ein Job an einer der Maschinen bearbeitet wurde und zusätzlich eine der folgenden Bedingungen erfüllt ist:
    a) Ein Puffer verfügt über keinen freien Platz.
    b) Alle Jobs aus σ befinden sich in den Puffern oder wurden an Maschine 1 oder 2 entsandt.
    c) Ein Job wird direkt an Maschine 3 entsandt, ohne dabei einen Puffer zu durchlaufen (nur in $(0, k_2, 0)$ möglich).

- Für einen Zeitpunkt t ist $\gamma(t)$ die Queue der an den Maschinen bearbeiteten Jobs (in der Reihenfolge, in der diese ausgegeben worden).

- Seien $k_1, k_2, k_3 \in \mathbb{N}$ und (k_1, k_2, k_3) ein Setting. Unter einer Bearbeitungsreihenfolge eines zu $\sigma \in Q$ gehörigen Outputs $(\sigma_1, \sigma_2) \in (k_1, k_2, k_3)(\sigma)$ versteht man eine Queue $\delta^{(k_1,k_2,k_3)}((\sigma_1, \sigma_2)) \in Q$ mit

  (i) $J(\delta^{(k_1,k_2,k_3)}((\sigma_1, \sigma_2))) = J(\sigma_1) \cup J(\sigma_2)$.

  (ii) Sei $i \in \{1, 2\}$. $\forall x, y \in J(\sigma_i)$ gilt: $x \succeq^{\sigma_i} y \Rightarrow x \succeq^{\delta^{(k_1,k_2,k_3)}((\sigma_1,\sigma_2))} y$.

  Das heißt $\delta^{(k_1,k_2,k_3)}((\sigma_1, \sigma_2))$ gibt eine Reihenfolge für die Bearbeitung von σ an, wobei den Reihenfolgen aus σ_1 bzw. σ_2 nicht widersprochen werden darf. Meist existieren mehrere Bearbeitungsreihenfolgen für eine Queue σ und einen Output (σ_1, σ_2). Ist klar um welchen Output es sich handelt, wird $\delta^{(k_1,k_2,k_3)}$ anstelle von $\delta^{(k_1,k_2,k_3)}((\sigma_1, \sigma_2))$ geschrieben.

Betrachte beispielsweise $\sigma = (1, \ldots, 5)$ in $(3, 0, 0)$. Tabelle 2 zeigt, dass $((1, 3, 5), (2, 4))$ sowohl in der Reihenfolge $(1, 2, 3, 4, 5)$, als auch in der Reihenfolge $(1, 3, 5, 2, 4)$ entstehen kann. Beide Varianten unterscheiden sich lediglich dadurch, wann die jeweiligen Aufträge an Maschine 1 bzw. 2 entsandt werden.

Tabelle 2: Unterschiedliche Bearbeitungsreihenfolgen bei gleichem Output

	$\delta^{(3,0,0)} = (1, 2, 3, 4, 5)$				$\delta^{(3,0,0)} = (1, 3, 5, 2, 4)$			
t	$P_1^{(3,0,0)}$	M1	M2	$\gamma(t)$	$P_1^{(3,0,0)}$	M1	M2	$\gamma(t)$
0	[. , . , .]			$(.)$	[. , . , .]			$(.)$
1	[1, 2, 3]			$(.)$	[1, 2, 3]			$(.)$
2	[2, 3, 4]	1		(1)	[2, 3, 4]	1		(1)
3	[3, 4, 5]	1	2	$(1, 2)$	[2, 4, 5]	$1, 3$		$(1, 3)$
4	[4, 5, .]	$1, 3$	2	$(1, 2, 3)$	[2, 4, .]	$1, 3, 5$		$(1, 3, 5)$
5	[5, . , .]	$1, 3$	$2, 4$	$(1, 2, 3, 4)$	[4, . , .]	$1, 3, 5$	2	$(1, 3, 5, 2)$
6	[. , . , .]	$1, 3, 5$	$2, 4$	$(1, 2, 3, 4, 5)$	[. , . , .]	$1, 3, 5$	$2, 4$	$(1, 3, 5, 2, 4)$

Der Output $((.), (1, 2, 3)) \in (0, k_2, 0)((1, 2, 3))$ mit $k_2 \in \mathbb{N}$ kann in $(0, k_2, 0)$ hingegen nur in der Reihenfolge $\delta^{(0,k_2,0)} = (1, 2, 3)$ entstehen. Weil die Jobs keinen Puffer durchlaufen, können sie auch nicht in verschiedenen Reihenfolgen ausgegeben und bearbeitet werden, wodurch $\delta^{(0,k_2,0)}$ eindeutig ist.

Zum Löschen von Elementen aus Queues wird folgende Notation benötigt. Seien $\sigma', \sigma'' \in Q$ zwei beliebige Queues, die evtl. gleiche Jobs beinhalten. $\sigma' \setminus \sigma''$ ist genau die Queue, welche folgende Bedingungen erfüllt:

(i) $J(\sigma' \setminus \sigma'') = J(\sigma') \setminus J(\sigma'')$.

(ii) $\forall x, y \in J(\sigma' \setminus \sigma'')$ gilt: $x \succeq^{\sigma' \setminus \sigma''} y \iff x \succeq^{\sigma'} y$.

$\sigma' \setminus \sigma''$ ergibt sich somit als diejenige Queue, welche alle Jobs aus σ' enthält, welche nicht in σ'' vorkommen, und deren Jobs die Reihenfolge aus σ' besitzen. Betrachte zur Veranschaulichung das folgende Beispiel:

$$(1, 2, 3, 4, 5) \setminus (5, 7, 4, 1) = (2, 3).$$

Im Folgenden bildet die 1^{st} Funktion auf den ersten Job einer Queue ab, genauer gilt:

$$1^{st} : Q \to \mathbb{N} \text{ mit } \sigma = (f_1, \ldots, f_n) \mapsto f_1, \text{ mit } f_1, \ldots, f_n \in \mathbb{N}.$$

Betrachte zu einem Zeitpunkt t eine Queue σ, welche ein Setting (k_1, k_2, k_3) (mit $k_1, k_2, k_3 \in \mathbb{N}$) durchläuft. Weiterhin sei $(\sigma_1, \sigma_2) \in (k_1, k_2, k_3)(\sigma)$, $\gamma(t)$ die Queue der bereits bearbeiteten Jobs und $\delta^{(k_1,k_2,k_3)}$ eine Bearbeitungsreihenfolge.
Soll (σ_1, σ_2) in der Reihenfolge aus $\delta^{(k_1,k_2,k_3)}$ erstellt werden, so existiert in jedem Zeitpunkt t ein Job, welcher als nächstes an Maschine 1 bzw. 2 ausgeben werden muss. Dieser Auftrag werde als "Next Required Job" bzw. als $nrj(\delta^{(k_1,k_2,k_3)}, \gamma(t))$ bezeichnet. Explizit gilt

$$nrj(\delta^{(k_1,k_2,k_3)}, \gamma(t)) = 1^{st}(\delta^{(k_1,k_2,k_3)} \setminus \gamma(t)). \tag{10}$$

Betrachte für $k_1, k_2, k_3 \in \mathbb{N}$ und ein Setting (k_1, k_2, k_3) die folgenden Funktionen:

$$PK((k_1, k_2, k_3)) = \max\{\sum_{j=1}^{3} k_j - 2, k_1 + k_2 - 1, k_2 + k_3 - 1, k_1, k_2\} \tag{11}$$

sei die Puffer-Kapazität eines Settings. Sie gibt die maximale Anzal von Jobs an, welche sich gleichzeitig in den Puffern eines Settings befinden können, sodass

jeder der geladenen Jobs als nächstes ausgegeben werden kann.
Für $i \in \{2, 3\}$ beschreibt

$$GP_i((k_1, k_2, k_3)) = \max\{k_1 + k_i - 1, k_1, k_i\} \tag{12}$$

den größten Puffer vor Maschine $i - 1$. Falls die Aufträge auf ihrem Weg zu Maschine $i - 1$ lediglich einen Puffer durchlaufen, ergibt sich GP_i als k_1 bzw. k_i, ansonsten durch Kombination der beiden Puffer als $k_1 + k_i - 1$.

2.2 Lemmata

Im Folgenden werden einige Hilfssätze vorgestellt, welche allgemeine Resultate bezüglich der Settings liefern. Viele Behauptungen innerhalb weiterer Betrachtungen ergeben sich als direkte Anwendungen dieser Lemmata.

Das folgende Lemma garantiert, dass in einem Setting nicht weniger Outputs generiert werden können, falls die Puffer jeweils vergrößert werden.

Erweiterungs-Lemma:

Seien $k_1, k_2, k_3, k_1', k_2', k_3' \in \mathbb{N}$ und (k_1, k_2, k_3), (k_1', k_2', k_3') zwei Settings mit $k_i \geq k_i'$ für $i = 1, 2, 3$.
Dann gilt für ein beliebiges $\sigma \in Q$

$$(k_1, k_2, k_3)(\sigma) \supseteq (k_1', k_2', k_3')(\sigma).$$

Beweis: *Nach Satz 3.2 in [Her13, S. 26 f.] kann der Puffer $P_i^{(k_1,k_2,k_3)}$ jegliche Vertauschungen ausführen, welche in $P_i^{(k_1',k_2',k_3')}$ stattfinden können. Da sich die beiden Settings nur durch die jeweiligen Größen der Puffer unterscheiden, folgt die Behauptung.* □

Das Split-Lemma beschreibt, wie der Split zu wählen ist, falls ein bestimmter Output erzeugt werden soll. Es wird im Unterkapitel 2.3 nützlich sein, da sich der Split als unabhängig vom Setting ergibt und somit aus den dortigen Betrachtungen ausgelassen werden kann.

Split-Lemma:

Seien $\sigma \in Q$, $k_1, k_2, k_3 \in \mathbb{N}$, (k_1, k_2, k_3) ein Setting und $(\sigma_1, \sigma_2) \in (k_1, k_2, k_3)(\sigma)$ ein zu σ gehöriger Output, der in (k_1, k_2, k_3) erzeugt werden kann. Weiterhin sei σ' die Queue, welche aus σ hervorgeht, nachdem σ den Puffer $P_1^{(k_1,k_2,k_3)}$ passiert hat.

i) Der zugehörige Split $S(\sigma') = (\sigma_1', \sigma_2')$, welcher notwendig ist, um (σ_1, σ_2) in (k_1, k_2, k_3) zu erzeugen, ist eindeutig. Es gilt

$$S(\sigma') = (\sigma_1', \sigma_2') = (\sigma' \setminus \sigma_2, \sigma' \setminus \sigma_1). \tag{13}$$

ii) Dieser Split ist unabhängig vom Setting bestimmt.

Beweis: i) *Angenommen, es existiert ein Split* $\overline{S}(\sigma') = (\overline{\sigma'_1}, \overline{\sigma'_2}) \neq S(\sigma')$*, welcher in* (k_1, k_2, k_3) *auch zum Output* (σ_1, σ_2) *führt.*
Nach der Definition eines zu (σ_1, σ_2) *gehörigen Splits gilt zunächst* $J(\overline{\sigma'_1}) = J(\sigma_1) = J(\sigma'_1)$ *und* $J(\overline{\sigma'_2}) = J(\sigma_2) = J(\sigma'_2)$*. Weiterhin kann der Split die Reihenfolgen der Jobs aus* σ' *nicht ändern. Das heißt, dass sich weder* $\overline{\sigma'_1}$ *und* σ'_1*, noch* $\overline{\sigma'_2}$ *und* σ'_2 *durch die Reihenfolgen ihrer Jobs unterscheiden können. Zusammenfassend gilt somit* $\overline{S}(\sigma') = S(\sigma')$*. Widerspruch!*
Der Split ist demnach eindeutig (auf die oben beschriebene Art) bestimmt.

ii) *Da für den Beweis an keiner Stelle die Puffer des Settings benutzt werden, ist der Split unabhängig vom Setting bestimmt.* □

Falls in einem Setting ein bestimmter Output erzeugt werden soll, so gibt das Output-Lemma eine Bedingung, unter welcher dieser Output tatsächlich realisierbar ist.

Output-Lemma:

Seien $\sigma \in Q$*,* $k_1, k_2, k_3 \in \mathbb{N}$*,* (k_1, k_2, k_3) *ein Setting,* $(\sigma_1, \sigma_2) \in \mathbb{O}(\sigma)$ *und* $\delta^{(k_1,k_2,k_3)}$ *eine zugehörige Bearbeitungsreihenfolge.*
Wenn in jedem Moment t*, in welchem anschließend ein Job an Maschine 1 bzw. 2 entsandt werden muss, jeweils* $nrj(\delta^{(k_1,k_2,k_3)}, \gamma(t))$ *entsandt wird, so entsteht* (σ_1, σ_2)*.*

Beweis: *Werde also jeweils* $nrj(\delta^{(k_1,k_2,k_3)}, \gamma(t))$ *an Maschine 1 bzw. Maschine 2 entsandt und es entstehe ein Output* $(\overline{\sigma_1}, \overline{\sigma_2})$ *in der Reihenfolge aus* $\delta^{(k_1,k_2,k_3)}$*.*
Angenommen es gilt $(\overline{\sigma_1}, \overline{\sigma_2}) \neq (\sigma_1, \sigma_2)$*:*
Da der Split - nach Split-Lemma - eindeutig bestimmt ist, können sich beide Outputs höchstens durch die jeweilige Reihenfolge der Jobs unterscheiden.
Dies führt zum Widerspruch zur Definition einer Bearbeitungsreihenfolge, denn diese darf die Reihenfolgen aus σ_1 *und* σ_2 *nicht verletzen.* □

Auswurf-Lemma:

Seien $k_1, k_2, k_3 \in \mathbb{N}$ und (k_1, k_2, k_3) ein Setting. Durchlaufe eine Queue $\sigma \in Q$ das Setting (k_1, k_2, k_3), wobei zu einem Zeitpunkt t die Plätze der Puffer wie folgt belegt sind:

1.) $k_1, k_2, k_3 \geq 2$.
$P_1^{(k_1,k_2,k_3)}$ hat maximal k_1 Jobs geladen und $P_2^{(k_1,k_2,k_3)}$ und $P_3^{(k_1,k_2,k_3)}$ haben

mindestens einen freien Platz.

2.) $k_3 = 0$ und $k_1, k_2 \geq 2$.
$P_1^{(k_1,k_2,k_3)}$ hat maximal k_1 Jobs geladen und $P_2^{(k_1,k_2,k_3)}$ hat mindestens einen freien Platz.

Sei j ein beliebiger Job aus σ, welcher sich zum angegebenen Zeitpunkt in einem der Puffer befindet. Dann ist es möglich j als nächstes an Maschine 1 bzw. 2 auszugeben.

Beweis: *Falls sich j in $P_2^{(k_1,k_2,k_3)}$ bzw. $P_3^{(k_1,k_2,k_3)}$ befindet, kann er direkt an Maschine 1 bzw. 2 ausgegeben werden.*

Sei j in $P_1^{(k_1,k_2,k_3)}$ geladen. Tausche ihn an erste Stelle, gib ihn aus und sende ihn (dem Split entsprechend) an $P_2^{(k_1,k_2,k_3)}$, $P_3^{(k_1,k_2,k_3)}$ bzw. direkt an Maschine 2:

1.) $k_1, k_2, k_3 \geq 2$.
Da $P_2^{(k_1,k_2,k_3)}$ bzw. $P_3^{(k_1,k_2,k_3)}$ jeweils über einen freien Platz verfügen, kann j geladen werden. Tausche j an erste Stelle und sende ihn anschließend an Maschine 1 bzw. 2.

2.) $k_3 = 0$ und $k_1, k_2 \geq 2$.
Sollte j an Maschine 2 entsandt werden, ist die Behaupung gezeigt.
Werde j an $P_2^{(k_1,k_2,k_3)}$ gesandt. Da dieser laut Voraussetzung über einen freien Platz verfügt, kann j geladen, direkt wieder ausgegeben und an Maschine 1 geleitet werden.

Damit ist die Behauptung gezeigt. □

Das Größter-Puffer-Lemma zeigt, dass sich durch einen größeren Puffer mehr Vertauschungen durchführen lassen.

Größter-Puffer-Lemma:

Seien $k_1, k_2, k_3, k_1', k_2', k_3' \in \mathbb{N}$ und (k_1, k_2, k_3), (k_1', k_2', k_3') zwei Settings mit

$$GP_i((k_1, k_2, k_3)) > GP_i((k_1', k_2', k_3')) \text{ für } i = 2 \text{ oder } i = 3.$$

Dann existiert eine Queue $\sigma^0 \in Q$ und ein zugehöriger Output $(\sigma_1^0, \sigma_2^0) \in \mathbb{O}(\sigma)$, sodass:

(i) $(\sigma_1^0, \sigma_2^0) \in (k_1, k_2, k_3)(\sigma^0)$ und

(ii) $(\sigma_1^0, \sigma_2^0) \notin (k_1', k_2', k_3')(\sigma^0)$ gilt.

Beweis: *Sei* $u := GP_i((k_1, k_2, k_3)) = \max\{k_1 + k_i - 1, k_1, k_i\}$ *und* $v := GP_i((k'_1, k'_2, k'_3)) = \max\{k'_1 + k'_i - 1, k'_1, k'_i\}$ *. Betrachte*

$$\sigma^0 := (1, 2, \ldots, u) \tag{14}$$

und den zugehörigen Output

$$(\sigma^0_1, \sigma^0_2) := \begin{cases} ((u, 1, 2, \ldots, u-1), (.)), & \textit{falls } i = 2 \\ ((.), (u, 1, 2, \ldots, u-1)), & \textit{falls } i = 3. \end{cases} \tag{15}$$

1. $u = k_1 + k_i - 1$.
$(\sigma^0_1, \sigma^0_2) \in (k_1, k_2, k_3)(\sigma^0)$*:*
Lasse die ersten $k_i - 1$ *Jobs* $P_1^{(k_1,k_2,k_3)}$ *passieren und lade sie in* $P_i^{(k_1,k_2,k_3)}$*. Die restlichen* k_1 *Aufträge werden in* $P_1^{(k_1,k_2,k_3)}$ *geladen. Anschließend befinden sich alle* u *Aufträge der Queue in den Puffern und können nach Auswurf-Lemma beliebig ausgegeben werden. Wähle daher die Reihenfolge aus* σ^0_{i-1}*.*

1.1. $v = k'_1 + k'_i - 1$.
$(\sigma^0_1, \sigma^0_2) \notin (k'_1, k'_2, k'_3)(\sigma^0)$*:*
Wenn der Job v *in* $P_1^{(k'_1,k'_2,k'_3)}$ *geladen wird, müssen sich genau* k'_1 *Jobs in* $P_1^{(k'_1,k'_2,k'_3)}$ *und genau* $k'_i - 1$ *Jobs in* $P_i^{(k'_1,k'_2,k'_3)}$ *befinden. Da* $P_1^{(k'_1,k'_2,k'_3)}$ *voll geladen ist, muss er einen Job ausgeben. Dieser Job muss anschließend von* $P_i^{(k'_1,k'_2,k'_3)}$ *geladen werden. Da nun wiederum* $P_i^{(k'_1,k'_2,k'_3)}$ *einen seiner geladenen Aufträge ausgeben muss, wird ein Job* $j \leq v < u = 1^{st}(\sigma^0_{i-1})$ *an Maschine* $i - 1$ *entsandt.*
σ^0_{i-1} *- und damit auch der Output* (σ^0_1, σ^0_2) *- ist in diesem Setting folglich nicht realisierbar.*

1.2. $v = k'_1$.
$(\sigma^0_1, \sigma^0_2) \notin (k'_1, k'_2, k'_3)(\sigma^0)$*:*
Die ersten $k'_1 = v$ *Jobs werden in* $P_1^{(k'_1,k'_2,k'_3)}$ *geladen, wovon anschließend ein Job* j *wieder ausgegeben werden muss. Es gilt* $k_i = 0$*, sonst wäre* $k_i \geq 2$ *und* $k_1 + k_i - 1 > k_1$*. Es existiert also keine weitere Möglichkeit eine Vertauschung durchzuführen, d.h.* $j \leq v < u = 1^{st}(\sigma^0_{i-1})$*.*
σ^0_{i-1} *- und damit auch der Output* (σ^0_1, σ^0_2) *- ist in diesem Setting folglich nicht realisierbar.*

1.3. $v = k'_i$.
$(\sigma^0_1, \sigma^0_2) \notin (k'_1, k'_2, k'_3)(\sigma^0)$*:*

Es gilt $k'_1 = 0$, sonst wäre $k'_1 \geq 2$ und $k'_1 + k'_i - 1 > k'_i$. Die Jobs aus σ werden also direkt zu $P_i^{(k'_1,k'_2,k'_3)}$ geleitet. Dieser Puffer kann maximal die ersten v Jobs laden, eventuell sortieren und davon einen Auftrag j wieder ausgegeben, welcher an Maschine $i-1$ gesandt wird. Damit gilt $j \leq v < u = 1^{st}(\sigma^0_{i-1})$.
σ^0_{i-1} - und damit auch der Output (σ^0_1, σ^0_2) - ist in diesem Setting folglich nicht realisierbar.

2. $u = k_1$.
$(\sigma^0_1, \sigma^0_2) \in (k_1, k_2, k_3)(\sigma^0)$:
Der Puffer $P_1^{(k_1,k_2,k_3)}$ lädt alle Elemente aus σ, welche anschließend in beliebiger Reihenfolge wieder ausgeben werden können. Es wird daher die Reihenfolge aus σ^0_{i-1} gewählt. Im Split werden abschließend alle Jobs zu Maschine $i-1$ entsandt und es entsteht der Output (σ^0_1, σ^0_2). Ein Puffer $P_i^{(k_1,k_2,k_3)}$ kann in diesem Fall nicht existieren, da sonst $k_1 + k_i - 1 > k_1$ wäre.

2.1. $v = k'_1 + k'_i - 1$.
$(\sigma^0_1, \sigma^0_2) \notin (k'_1, k'_2, k'_3)(\sigma^0)$:
Beweis analog zu 1.1.

2.2. $v = k'_1$.
$(\sigma^0_1, \sigma^0_2) \notin (k'_1, k'_2, k'_3)(\sigma^0)$:
Beweis analog zu 1.2.

2.3. $v = k'_i$.
$(\sigma^0_1, \sigma^0_2) \notin (k'_1, k'_2, k'_3)(\sigma^0)$:
Beweis analog zu 1.3.

3. $u = k_i$.
$(\sigma^0_1, \sigma^0_2) \in (k_1, k_2, k_3)(\sigma^0)$:
Es gilt $k_1 = 0$, weil ansonsten $k_1 + k_i - 1 > k_i$ gelten würde. Im Split werden alle Jobs an $P_i^{(k_1,k_2,k_3)}$ gesendet. Da $|\sigma| = u$ gilt, können alle Jobs gleichzeitig geladen und anschließend in beliebiger Sortierung ausgegeben werden. Wähle daher die Reihenfolge aus σ^0_{i-1}.

3.1. $v = k'_1 + k'_i - 1$.
$(\sigma^0_1, \sigma^0_2) \notin (k'_1, k'_2, k'_3)(\sigma^0)$:
Beweis analog zu 1.1.

3.2. $v = k'_1$.
$(\sigma^0_1, \sigma^0_2) \notin (k'_1, k'_2, k'_3)(\sigma^0)$:
Beweis analog zu 1.2.

3.3. $v = k'_i$.
$(\sigma^0_1, \sigma^0_2) \notin (k'_1, k'_2, k'_3)(\sigma^0)$*:*
Beweis analog zu 1.3.

Da nach Definition von u *bzw.* v *einer der aufgeführten Fälle eintreten muss, folgt die Behauptung.* □

Übergangs-Lemma:

Seien $k_1, k_2, k_3, k'_1, k'_2, k'_3 \in \mathbb{N}$, $(k_1, k_2, k_3), (k'_1, k'_2, k'_3)$ Settings, $\sigma \in Q$, $(\sigma_1, \sigma_2) \in (k_1, k_2, k_3)(\sigma)$, $(\sigma'_1, \sigma'_2) \in (k'_1, k'_2, k'_3)(\sigma)$.
σ durchlaufe simultan (k_1, k_2, k_3) und (k'_1, k'_2, k'_3), wobei (σ_1, σ_2) bzw. (σ'_1, σ'_2) entstehen und $\delta^{(k'_1,k'_2,k'_3)}$ sei eine Bearbeitungsreihenfolge von (σ'_1, σ'_2).
Für einen beliebigen Zeitpunkt t (bzw. t') bezeichne $\gamma^{(k_1,k_2,k_3)}(t)$ (bzw. $\gamma^{(k'_1,k'_2,k'_3)}(t')$) die bearbeiteten Jobs in (k_1, k_2, k_3) (bzw. (k'_1, k'_2, k'_3)).

Betrachte zu einem Zeitpunkt t_0 alle Jobs in den Puffern $P_1^{(k_1,k_2,k_3)}$, $P_2^{(k_1,k_2,k_3)}$ und $P_3^{(k_1,k_2,k_3)}$ und bezeichne die Menge dieser geladenen Jobs mit $\hat{L}$. Gelte in diesem Moment

(i) $|L| \geq PK((k'_1, k'_2, k'_3))$

(ii) und es existiere ein Moment t'_0 für (k'_1, k'_2, k'_3) mit $\gamma^{(k'_1,k'_2,k'_3)}(t'_0) = \gamma^{(k_1,k_2,k_3)}(t_0)$.

Dann ist

$$nrj(\delta^{(k'_1,k'_2,k'_3)}, \gamma(t'_0)) \in L.$$

Die beschriebene Situation ist zur Veranschlichung in Abbildung 3 dargestellt.

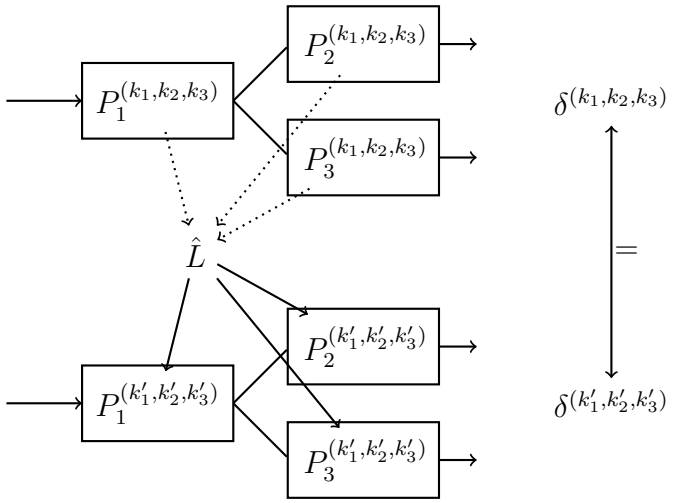
Beweis: *Angenommen, zum Zeitpunkt* t' *in* (k'_1, k'_2, k'_3) *wird als nächstes ein Job* j *mit* $j \notin L$ *an Maschine 1 bzw. 2 ausgegeben.*

Da $\gamma^{(k'_1,k'_2,k'_3)}(t'_0) = \gamma^{(k_1,k_2,k_3)}(t_0)$ *ist, kann kein Job aus* $\hat{L}$ *in* (k'_1, k'_2, k'_3) *bereits an Maschine* 1 *oder* 2 *bearbeitet worden sein. Desweiteren befinden sich alle Jobs aus* $\hat{L}$ *in* (k_1, k_2, k_3) *in den Puffern des Settings und* j *nicht. Es gilt folglich*

$$\forall x \in L : x < j \tag{16}$$

Demnach müssen sich alle Elemente von $\hat{L}$ *in den Puffern von* (k'_1, k'_2, k'_3) *befinden, damit* j *als nächstes an eine der beiden Maschinen gesandt werden kann. Für die einzelnen Settings folgt:*

Abbildung 3: Darstellung der Betrachtungen des Übergangs-Lemmas. Die Jobs aus den Puffern des Settings (k_1, k_2, k_3) werden im Setting (k_1', k_2', k_3') betrachtet. Dabei soll $\delta^{(k_1,k_2,k_3)} = \delta^{(k_1',k_2',k_3')}$ gelten.



1. $(k_1', 0, 0)$.
   Wegen $|L| \geq PK((k_1', 0, 0)) = k_1'$ *und den vorangegangenen Betrachtungen, muss* $P_1^{(k_1',k_2',k_3')}$ *genau* k_1' *Aufträge aus* $\hat{L}$ *geladen haben. Folglich muss* $P_1^{(k_1',k_2',k_3')}$ *einen Job aus* $\hat{L}$ *als nächstes an Maschine 1 oder 2 ausgeben. Widerspruch!*

2. $(0, k_2', 0)$.
   Wegen $|L| \geq PK((0, k_2', 0)) = k_2'$ *und den vorangegangenen Betrachtungen, muss* $P_2^{(k_1',k_2',k_3')}$ *genau* k_2' *Aufträge aus* $\hat{L}$ *geladen haben. Folglich muss* $P_2^{(k_1',k_2',k_3')}$ *einen Job aus* $\hat{L}$ *als nächstes an Maschine 1 oder 2 ausgeben. Widerspruch!*

3. $(0, k_2', k_3')$.
   Wegen $|L| \geq PK((0, k_2', k_3')) = k_2' + k_3' - 1$ *und* (16) *muss einer der beiden Puffer komplett mit Jobs aus* $\hat{L}$ *belegt sein, bevor* j *geladen werden kann. Demnach muss auch ein Job von* $\hat{L}$ *ausgegeben worden sein, bevor* j *geladen wird. Widerspruch!*

4. $(k_1', k_2', 0)$.
   Da $|L| \geq PK((k_1', k_2', 0)) = k_1' + k_2' - 1$ *gilt, muss einer der beiden folgenden Fälle eintreten:*

   4.1. $P_2^{(k_1',k_2',k_3')}$ *hat* k_2' *der Elemente von* $\hat{L}$ *geladen.*
   Der Puffer muss einen dieser Aufträge wieder ausgeben, wodurch

ein Job aus $\hat{L}$ an Maschine 1 gesandt werden muss. Ein etwaiger, wie oben beschriebener Job j kann wegen (16) *erst danach in einen der Puffer gelangen. Es ergibt sich folglich ein Widerspruch zur Annahme.*

4.2. $P_1^{(k'_1,k'_2,k'_3)}$ *hat* k'_1 *und* $P_2^{(k'_1,k'_2,k'_3)}$ *hat* $k'_2 - 1$ *Elemente aus* $\hat{L}$ *geladen.*
Damit j in $P_1^{(k'_1,k'_2,k'_3)}$ geladen werden kann, muss dieser Puffer zunächst einen seiner Jobs ausgeben. Läuft dieser anschließend zu Maschine 2, so ergibt sich ein Widerspruch. Werde dieser soeben ausgegebene Job also an $P_2^{(k'_1,k'_2,k'_3)}$ gesandt und dort anschließend geladen. Nun sind die Bedingungen des Falls 4.1. erfüllt und es ergibt sich ein Widerspruch zur Annahme.

5. (k'_1, k'_2, k'_3)*.*
Wegen $|L| \geq PK((k'_1, k'_2, k'_3)) = \sum_{j=1}^{3} k'_j - 2$ muss einer der folgenden Fälle eintreten:

5.1. $P_2^{(k'_1,k'_2,k'_3)}$ *hat* k'_2 *oder* $P_3^{(k'_1,k'_2,k'_3)}$ *hat* k'_3 *der Elemente von* $\hat{L}$ *geladen.*
Einer der beiden Puffer muss einen der Aufträge wieder ausgeben, wodurch ein Job aus $\hat{L}$ an Maschine 1 gesandt werden muss. Ein etwaiger, wie oben beschriebener Job j kann wegen (16) *erst danach in einen der Puffer gelangen. Es ergibt sich ein Widerspruch zur Annahme.*

5.2. $P_1^{(k'_1,k'_2,k'_3)}$ *hat* k'_1*,* $P_2^{(k'_1,k'_2,k'_3)}$ *hat* $k'_2 - 1$ *und* $P_3^{(k'_1,k'_2,k'_3)}$ *hat* $k'_3 - 1$ *Elemente aus* $\hat{L}$ *geladen.*
Damit j in $P_1^{(k'_1,k'_2,k'_3)}$ geladen werden kann, muss dieser Puffer zunächst einen seiner Jobs ausgeben. Dieser soeben ausgegebene Job wird anschließend an $P_2^{(k'_1,k'_2,k'_3)}$ oder $P_3^{(k'_1,k'_2,k'_3)}$ gesandt und dort geladen. Nun sind die Bedingungen des Falls 5.1. erfüllt und es ergibt sich ein Widerspruch zur Annahme.

Es ergibt sich also in jedem Fall ein Widerspruch.

In (k'_1, k'_2, k'_3) wird demnach zum Zeitpunkt t'_0 als nächstes ein Job aus $\hat{L}$ an Maschine 1 bzw. 2 entsandt, was gleichbedeutend mit $nrj(\delta^{(k'_1,k'_2,k'_3)}, \gamma(t'_0)) \in L$ ist. □

γ-Lemma:

Seien $\sigma \in Q$, $k_1, k_2, k_3, k_1', k_2', k_3' \in \mathbb{N}$, $(k_1, k_2, k_3), (k_1', k_2', k_3')$ zwei Settings, $(\sigma_1, \sigma_2) \in (k_1, k_2, k_3)(\sigma)$, $(\sigma_1', \sigma_2') \in (k_1', k_2', k_3')(\sigma)$ und $\delta^{(k_1', k_2', k_3')}$ eine Bearbeitungsreihenfolge zu (σ_1', σ_2').

Betrachte σ während des Bearbeitungsprozesses in (k_1, k_2, k_3). Kann für einen beliebigen Zeitpunkt t aus der Forderung nach einem Moment t' in (k_1', k_2', k_3') mit $\gamma^{(k_1,k_2,k_3)}(t) = \gamma^{(k_1',k_2',k_3')}(t')$ gefolgert werden, dass $nrj(\delta^{(k_1',k_2',k_3')}, \gamma(t'))$ in (k_1, k_2, k_3) als nächstes an Maschine 1 bzw. Maschine 2 ausgegeben wird.
Dann existiert für jeden Moment t in (k_1, k_2, k_3) ein solcher Zeitpunkt t' in (k_1', k_2', k_3') mit

$$\gamma^{(k_1,k_2,k_3)}(t) = \gamma^{(k_1',k_2',k_3')}(t').$$

Beweis: *Im Moment $t = 0$, kann $t' = 0$ gewählt werden und es gilt $\gamma^{(k_1,k_2,k_3)}(0) = \gamma^{(k_1',k_2',k_3')}(0) = (.)$, da noch kein Job bearbeitet wurde.*
Anschließend wird nach Voraussetzung in beiden Settings jeweils $nrj(\delta^{(k_1',k_2',k_3')}, \gamma^{(k_1',k_2',k_3')}(t'))$ an eine der Maschinen ausgegeben. Es erfolgt somit jeweils ein Übergang von t, t' mit $\gamma^{(k_1,k_2,k_3)}(t) = \gamma^{(k_1',k_2',k_3')}(t')$ zu Zuständen $\overline{t}$, $\overline{t'}$ mit $\gamma^{(k_1,k_2,k_3)}(\overline{t}) = \gamma^{(k_1',k_2',k_3')}(\overline{t'})$. □

Puffer-Kombinations-Lemma:

Seien $k_1 \in \mathbb{N}_{\geq 2}$ und $k_2, k_3, k_2', k_3' \in \mathbb{N}$. Weiterhin seien (k_1, k_2, k_3), $(0, k_2', k_3')$ zwei Settings, $\sigma \in Q$ und $(\sigma_1, \sigma_2) \in \mathbb{O}(\sigma)$ mit $J(\sigma_1) = \emptyset$ oder $J(\sigma_2) = \emptyset$.

1.) Sei $J(\sigma_2) = \emptyset$, $k_2, k_2' \geq 2$ und $k_1 + k_2 - 1 = k_2'$. Dann ist

$$(\sigma_1, \sigma_2) \in (k_1, k_2, k_3)(\sigma) \iff (\sigma_1, \sigma_2) \in (0, k_2', k_3')(\sigma) \tag{17}$$

2.) Sei $J(\sigma_1) = \emptyset$, $k_3, k_3' \geq 2$ und $k_1 + k_3 - 1 = k_3'$. Dann ist

$$(\sigma_1, \sigma_2) \in (k_1, k_2, k_3)(\sigma) \iff (\sigma_1, \sigma_2) \in (0, k_2', k_3')(\sigma) \tag{18}$$

Beweis: *Die Settings unterscheiden sich nur durch die Größen ihrer Puffer. Es ist jeweils zu zeigen, dass die Kombination der Puffer $P_1^{(k_1,k_2,k_3)}$ und $P_2^{(k_1,k_2,k_3)}$ bzw. $P_3^{(k_1,k_2,k_3)}$ die gleichen Vertauschungen vollziehen kann, wie $P_2^{(0,k_2',k_3')}$ bzw. $P_3^{(0,k_2',k_3')}$.*
In einem Setting mit nur einer zweiten Maschine ist dies möglich [nach Her13,

S. 29].

Es kann eine Reduktion der Aufgabenstellung durchgeführt werden:

a) *Der Split kann vernachlässigt werden.*
Da $J(\sigma_1) = \emptyset$ oder $J(\sigma_2) = \emptyset$ gilt und der Split keine Auswirkungen auf die Reihenfolge hat, gilt für eine Queue σ':

$$S(\sigma') = \begin{cases} (\sigma', (.)), & \textit{falls } J(\sigma_2) = \emptyset \\ ((.), \sigma'), & \textit{falls } J(\sigma_1) = \emptyset. \end{cases} \tag{19}$$

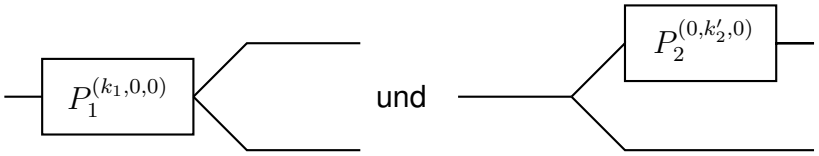
b) *Falls $J(\sigma_1) = \emptyset$ können $P_2^{(k_1,k_2,k_3)}$ und $P_2^{(0,k_2',k_3')}$ vernachlässigt werden.*
Falls $J(\sigma_2) = \emptyset$ können $P_3^{(k_1,k_2,k_3)}$ und $P_3^{(0,k_2',k_3')}$ vernachässigt werden.

Wegen a) kann kein Job in einen der jeweiligen Puffer geladen werden, wodurch dort auch keine Vertauschungen durchgeführt werden können.

Mit diesen Einschränkungen unterscheiden sich die betrachteten Settings nicht von einem Setting mit lediglich einer zweiten Maschine und es folgt die Behauptung. □

2.3 Vergleich der Settings

$(k_1, 0, 0)$ **und** $(0, k_2', 0)$



Satz: Seien $k_1, k_2' \in \mathbb{N}_{\geq 2}$. Dann gilt:

(1) Es existieren $\sigma^0 \in Q$ und $(\sigma_1^0, \sigma_2^0) \in \mathbb{O}(\sigma^0)$, sodass:

(i) $(\sigma_1^0, \sigma_2^0) \in (k_1, 0, 0)(\sigma^0)$ und

(ii) $(\sigma_1^0, \sigma_2^0) \notin (0, k_2', 0)(\sigma^0)$ gilt.

(2) Im Fall $k_1 < k_2'$ gilt zusätzlich:
Es existieren $\sigma^1 \in Q$ und $(\sigma_1^1, \sigma_2^1) \in \mathbb{O}(\sigma^1)$, sodass:

(i) $(\sigma_1^1, \sigma_2^1) \notin (k_1, 0, 0)(\sigma^1)$ und

(ii) $(\sigma_1^1, \sigma_2^1) \in (0, k_2', 0)(\sigma^1)$ gilt.

(3) Im Fall $k_1 \geq k_2'$ gilt zusätzlich: Für ein beliebiges $\sigma \in Q$ ist

$$(k_1, 0, 0)(\sigma) \supseteq (0, k_2', 0)(\sigma)$$

Beweis: (1) Folgt mit

$$GP_3((k_1, 0, 0)) = k_1 > 0 = GP_3((0, k_2, 0))$$

aus dem Größter-Puffer-Lemma.

(2) Durch

$$GP_2((k_1, 0, 0)) = k_1 < k_2' = GP_2((0, k_2, 0))$$

folgt auch diese Behauptung aus dem Größter-Puffer-Lemma.

(3) Sei zunächst $k_1 = k_2' =: k$. Betrachte ein $\sigma \in Q$ und einen zugehörigen, in $(0, k, 0)$ erzeugten Output

$$(\sigma_1^{(0,k,0)}, \sigma_2^{(0,k,0)}) \in (0, k, 0)(\sigma).$$

Dieser Output kann auch in $(k, 0, 0)$ erzeugt werden:
σ muss zunächst den Puffer $P_1^{(k,0,0)}$ durchlaufen. Der darauffolgende Split ist (unabhängig vom Setting) durch das Split-Lemma bereits eindeutig festgelegt. Die Frage ist also, wie $P_1^{(k,0,0)}$ agieren muss, um den gewünschten Output zu realisieren.

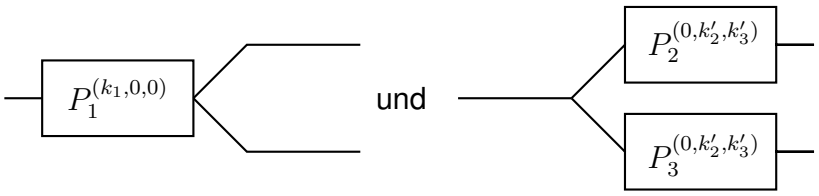
Der Puffer hat zu jedem Zeitpunkt maximal k Jobs geladen und kann nach folgendem Schema vorgehen:

1. *Im Puffer befindet sich ein Job aus* $\sigma_2^{(0,k,0)}$.
   Wähle unter den Jobs, welche zu $\sigma_2^{(0,k,0)}$ gehören, den minimalen (bezüglich seiner Nummerierung) aus und tausche ihn an die erste Stelle, sodass er als nächstes ausgegeben wird. Die anderen Aufträge behalten ihre Reihenfolge bei.
   Innerhalb von $\sigma_2^{(0,k,0)}$ müssen die Jobs aufsteigend sortiert sein, da sie in $(0, k, 0)$ keinen Puffer durchlaufen haben und der Split keine Auswirkungen auf die Sortierung hat. Auf diese Weise entsteht somit $\sigma_2^{(0,k,0)}$.

2. *Es sind nur Jobs aus* $\sigma_1^{(0,k,0)}$ *geladen*.
   Betrachte $\sigma \setminus \sigma_2^{(0,k,0)}$, das heißt die Jobs aus $\sigma_1^{(0,k,0)}$ in der Reihenfolge aus σ. Jegliche Vertauschungen, welche in $(0, k, 0)$ stattgefunden haben, um von $\sigma \setminus \sigma_2^{(0,k,0)}$ zu $\sigma_1^{(0,k,0)}$ überzugehen, fanden in einem Puffer der Größe k statt. Da $P_1^{(k,0,0)}$ ebenfalls die Größe k besitzt, müssen diese Vertauschungen auch in $(k, 0, 0)$ ausgeführt werden können. Somit ist auch $\sigma_1^{(0,k,0)}$ erzeugbar.

Zusammenfassend entsteht der gewünschte Output $(\sigma_1^{(0,k,0)}, \sigma_2^{(0,k,0)})$.
Da diese Sortierungen in $(k, 0, 0)$ mit einem Puffer der Größe k möglich waren, folgt durch das Erweiterungs-Lemma die Behauptung für $k_1 \geq k_2$. □

$(k_1, 0, 0)$ **und** $(0, k_2', k_3')$



Satz: Seien $k_1, k_2', k_3' \in \mathbb{N}_{\geq 2}$ (mit $k_2' \geq k_3'$). Dann gilt

(1) Sei $k_1 > k_3'$. Dann existieren $\sigma^0 \in Q$ und $(\sigma_1^0, \sigma_2^0) \in \mathbb{O}(\sigma^0)$, sodass:

(i) $(\sigma_1^0, \sigma_2^0) \in (k_1, 0, 0)(\sigma^0)$ und

(ii) $(\sigma_1^0, \sigma_2^0) \notin (0, k_2', k_3')(\sigma^0)$ gilt.

(2) Sei $k_1 \leq k_3'$. Dann gilt für ein beliebiges $\sigma \in Q$

$$(k_1, 0, 0)(\sigma) \subseteq (0, k_2', k_3')(\sigma).$$

(3) Sei $k_1 < k_2' + k_3' - 1$. Dann existieren $\sigma^1 \in Q$ und $(\sigma_1^1, \sigma_2^1) \in \mathbb{O}(\sigma^1)$, sodass:

(i) $(\sigma_1^1, \sigma_2^1) \notin (k_1, 0, 0)(\sigma^0)$ und

(ii) $(\sigma_1^1, \sigma_2^1) \in (0, k_2', k_3')(\sigma^0)$ gilt.

(4) Sei $k_1 \geq k_2' + k_3' - 1$. Dann gilt für ein beliebiges $\sigma \in Q$

$$(k_1, 0, 0)(\sigma) \supseteq (0, k_2', k_3')(\sigma).$$

Beweis: (1) Folgt mittels

$$GP_3((k_1, 0, 0)) = k_1 > k_3' = GP_3((0, k_2', k_3'))$$

aus dem Größter-Puffer-Lemma.

(2) Nach Konvention gilt $k_3' \leq k_2'$ und somit auch $k_1 \leq k_2'$.
Sei zunächst $k_1 = k_2' = k_3' =: k$. Betrachte ein $\sigma \in Q$ und einen zugehörigen, in $(k, 0, 0)$ erzeugten Output

$$(\sigma_1^{(k,0,0)}, \sigma_2^{(k,0,0)}) \in (k, 0, 0)(\sigma).$$

Dieser Output kann in $(0, k, k)$ erzeugt werden:
Der Split ist durch das Split-Lemma bereits eindeutig festgelegt. Es bleibt zu betrachten, wie $P_1^{(k,0,0)}$ agieren muss, um den gewünschten Output zu realisieren.

1. Betrachte zunächst $\sigma \setminus \sigma_2^{(k,0,0)}$:
   Jegliche Vertauschungen, welche in $(k, 0, 0)$ stattgefunden haben, um von $\sigma \setminus \sigma_2^{(k,0,0)}$ zu $\sigma_1^{(k,0,0)}$ überzugehen, fanden in einem Puffer der Größe k statt. Da $P_2^{(0,k,k)}$ auch die Größe k besitzt, müssen diese Vertauschungen auch im Setting $(0, k, k)$ ausgeführt werden können.

2. Die gleiche Argumentation kann für $\sigma_2^{(k,0,0)}$ geführt werden, da auch $P_3^{(0,k,k)}$ die Größe k besitzt.

Insgesamt kann also auch in $(0, k, k)$ der Output $(\sigma_1^{(k,0,0)}, \sigma_2^{(k,0,0)})$ entstehen.
Die Verallgemeinerung auf $k_2', k_3' \geq k_1$ folgt aus dem Erweiterungs-Lemma.

(3) Betrachte die Queue

$$\sigma^1 := (1, 2, \ldots, w) \in Q \text{ mit } w := k_2' + k_3'$$

und den Output

$$(\sigma_1^1, \sigma_2^1) = ((w, k_2' - 1, k_2' - 2, \ldots, 1), (w - 1, w - 2, \ldots, \underbrace{w - k_3'}_{=k_2'})) \in \mathbb{O}(\sigma).$$

(i) *Durch $(k_1, 0, 0)$ kann dieser Output nicht generiert werden:*
Der Puffer $P_1^{(k_1,0,0)}$ lädt zunächst die Elemente 1 bis k_1 und muss anschließend eines davon wieder ausgeben. Unter diesen Jobs befindet sich weder $1^{st}(\sigma_1^1) = w$, noch $1^{st}(\sigma_2^1) = w-1$, da nach Voraussetzung $k_1 < k_2' + k_3' - 1 = w-1$ gilt. Im Anschluss gibt es keine Möglichkeit zur Umsortierung, wodurch der gewünschte Output nicht erzeugbar ist.

(ii) *In $(0, k_2', k_3')$ ist der Output erzeugbar:*
Da $|\sigma_1^1| = k_2'$ und $|\sigma_2^1| = k_3'$ gilt, kann anschließend zum Split jede beliebige Reihenfolge innerhalb der Queues realisiert werden. Wähle die Reihenfolge aus σ_1^1 bzw. σ_2^1 und es folgt die Behauptung.

(4) Betrachte $\sigma \in Q$, einen beliebigen, dazugehörigen, in $(0, k_2', k_3')$ erzeugten Output

$$(\sigma_1^{(0,k_2',k_3')}, \sigma_2^{(0,k_2',k_3')}) \in (0, k_2', k_3')(\sigma),$$

und eine zugehörige Bearbeitungsreihenfolge $\delta^{(0,k_2',k_3')}$.

Dieser Output kann auch in $(k_1, 0, 0)$ erzeugt werden:
Wieder muss nur die Puffervorgehensweise betrachtet werden, da der Split eindeutig durch das Split-Lemma vorgegeben ist.

Der Puffer $P_1^{(k_1,0,0)}$ hat zu einem Zeitpunkt t maximal $k_1 \geq k_2' + k_3' - 1 = PK((0, k_2', k_3'))$ Jobs geladen.

a) Habe $P_1^{(k_1,0,0)}$ genau k_1 Jobs geladen.
Betrachte σ in $(0, k_2', k_3')$, in einem Moment t', in welchem $\gamma^{(0,k_2',k_3')}(t') = \gamma^{(k_1,0,0)}(t)$ gilt $(*)$. Durch Anwendung des Übergangs-Lemmas folgt, dass sich $nrj(\delta^{(0,k_2',k_3')}, \gamma^{(0,k_2',k_3')}(t')) = nrj(\delta^{(0,k_2',k_3')}, \gamma^{(k_1,0,0)}(t))$ in $P_1^{(k_1,0,0)}$ befinden muss. Gib diesen Job als nächstes an Maschine 1 bzw. 2 aus.

b) Befinden sich weniger als k_1 Elemente im Puffer, so handelt es sich dabei um die letzten Jobs aus σ - es muss demnach $nrj(\delta^{(0,k_2',k_3')}, \gamma^{(k_1,0,0)}(t))$ darunter sein. Wähle diesen Auftrag aus, tausche ihn an erste Position und sende ihn an Maschine 1 bzw. 2.

Zusammenfassend folgt durch Anwendung des γ-Lemmas, dass die Forderung nach $(*)$ legitim ist. Das Output-Lemma garantiert, dass $(\sigma_1^{(0,k_2',k_3')}, \sigma_2^{(0,k_2',k_3')})$ entsteht. □

Bemerkungen:

1.) (4) impliziert (1):

$$k_1 \overset{(4)}{\geq} k_2' + k_3' - 1 = k_3 + (\underbrace{k_2'}_{\geq 2} - 1) > k_3'.$$

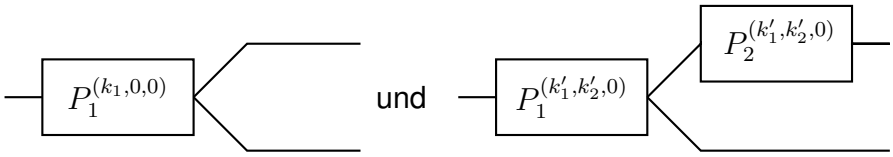
2.) (2) impliziert (3): Aus $k_1 \leq k_3'$ folgt mit $k_3' \leq k_2'$ auch $k_1 \leq k_2'$ und

$$k_2' + k_3' - 1 \geq 2 \cdot k_1 - 1 > k_1, \text{ da } k_1 \geq 2.$$

3.) Man beachte, dass in (3) nicht das Größter-Puffer-Lemma verwendet wird. Dieses würde $k_1 < k_2'$ als Voraussetzung verlangen, was auch $k_1 < k_2' + k_3' - 1$ implizieren würde, da $k_3' \geq 2$ ist.
Betrachte beispielsweise die Settings $(4, 0, 0)$ und $(0, 3, 3)$:

Es gilt $3+3-1=5>4$, aber jeweils $3<4$. Durch die schwächere Voraussetzung entstehen somit echt mehr Fälle, in denen der Satz Anwendung findet.

$(k_1, 0, 0)$ **und** $(k_1', k_2', 0)$



Satz: Seien $k_1, k_1', k_2' \in \mathbb{N}_{\geq 2}$. Dann gilt

(1) Sei $k_1 > k_1'$. Dann existieren $\sigma^0 \in Q$ und $(\sigma_1^0, \sigma_2^0) \in \mathbb{O}(\sigma^0)$, sodass:

(i) $(\sigma_1^0, \sigma_2^0) \in (k_1, 0, 0)(\sigma^0)$ und

(ii) $(\sigma_1^0, \sigma_2^0) \notin (k_1', k_2', 0)(\sigma^0)$ gilt.

(2) Sei $k_1 \leq k_1'$. Dann gilt für ein beliebiges $\sigma \in Q$

$$(k_1, 0, 0)(\sigma) \subseteq (k_1', k_2', 0)(\sigma).$$

(3) Sei $k_1 < k_1' + k_2' - 1$. Dann existieren $\sigma^1 \in Q$ und $(\sigma_1^1, \sigma_2^1) \in \mathbb{O}(\sigma^1)$, sodass:

(i) $(\sigma_1^0, \sigma_2^0) \notin (k_1, 0, 0)(\sigma^0)$ und

(ii) $(\sigma_1^0, \sigma_2^0) \in (k_1', k_2', 0)(\sigma^0)$ gilt.

(4) Sei $k_1 \geq k_1' + k_2' - 1$. Dann gilt für ein beliebiges $\sigma \in Q$

$$(k_1, 0, 0)(\sigma) \supseteq (k_1', k_2', 0)(\sigma).$$

Beweis: (1) Folgt mit

$$GP_3((k_1, 0, 0)) = k_1 > k_1' = GP_3((k_1', k_2', 0))$$

aus dem Größter-Puffer-Lemma.

(2) Die Behauptung folgt mit $k_1' \geq k_1$, $k_2' \geq 2 > 0$ und $0 \geq 0$ aus dem Erweiterungs-Lemma.

(3) Folgt mit

$$GP_2((k_1, 0, 0)) = k_1 < k_1' + k_2' - 1 = GP_2((k_1', k_2', 0))$$

auch aus dem Größter-Puffer-Lemma.

(4) Betrachte $\sigma \in Q$, einen zugehörigen, in $(k_1', k_2', 0)$ erzeugten Output

$$(\sigma_1^{(k_1',k_2',0)}, \sigma_2^{(k_1',k_2',0)}) \in (k_1', k_2', 0)(\sigma)$$

und eine zugehörige Bearbeitungsreihenfolge $\delta^{(k_1',k_2',0)}$.

Dieser Output kann auch in $(k_1, 0, 0)$ entstehen:
Es muss lediglich die Puffervorgehensweise betrachtet werden, da der Split eindeutig durch das Split-Lemma bestimmt ist.
$P_1^{(k_1,0,0)}$ hat zu einem beliebigen Zeitpunkt t maximal $k_1 \geq k_1' + k_2' - 1 = PK((k_1', k_2', 0))$ Jobs geladen.

a) Habe $P_1^{(k_1,0,0)}$ genau k_1 Jobs geladen.
Betrachte σ in $(k_1', k_2', 0)$ in einen Moment t', in welchem $\gamma^{(k_1',k_2',0)}(t') = \gamma^{(k_1,0,0)}(t)$ gilt $(*)$. Nach Übergangs-Lemma befindet sich $nrj(\delta^{(k_1',k_2',0)}, \gamma^{(k_1',k_2',0)}(t')) = nrj(\delta^{(k_1',k_2',0)}, \gamma^{(k_1,0,0)}(t))$ in $P_1^{(k_1,0,0)}$. Wähle diesen Job aus, sortiere ihn an erste Stelle und gib ihn anschließend an Maschine 1 bzw. 2 aus.

b) Falls sich weniger als k_1 Elemente im Puffer befinden, so handelt es sich um alle restlichen Jobs aus σ - also ist auch $nrj(\delta^{(k_1',k_2',0)}, \gamma^{(k_1,0,0)}(t))$ darunter. Tausche diesen an erste Stelle und sende ihn an Maschine 1 bzw. 2.

Durch das γ-Lemma folgt, dass die Forderung nach $(*)$ legitim ist. Mit Hilfe des Output-Lemmas folgt $(\sigma_1^{(k_1',k_2',0)}, \sigma_2^{(k_1',k_2',0)})$. □

Bemerkungen:

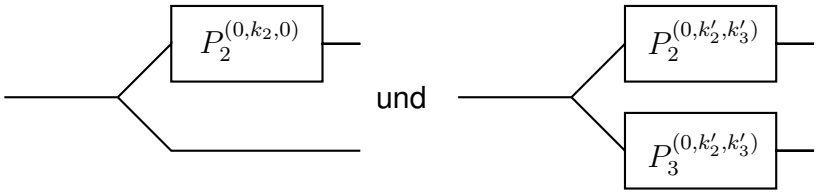
1) (2) impliziert (3):

$$k_1 \overset{(2)}{\leq} k_1' < k_1' + k_2' - 1, \text{ da } k_2' \geq 2$$

2) (4) impliziert (1):

$$k_1 \overset{(4)}{\geq} k_1' + k_2' - 1 > k_1', \text{ da } k_2' \geq 2$$

$(0, k_2, 0)$ **und** $(0, k_2', k_3')$



Satz: Seien $k_2, k_2', k_3' \in \mathbb{N}_{\geq 2}$ (mit $k_2' \geq k_3'$). Dann gilt

(1) Es existieren $\sigma^0 \in Q$ und $(\sigma_1^0, \sigma_2^0) \in \mathbb{O}(\sigma^0)$, sodass:

(i) $(\sigma_1^0, \sigma_2^0) \notin (0, k_2, 0)(\sigma^0)$ und

(ii) $(\sigma_1^0, \sigma_2^0) \in (0, k_2', k_3')(\sigma^0)$ gilt.

(2) Im Fall $k_2 > k_2'$ gilt zusätzlich:
Es existieren $\sigma^1 \in Q$ und $(\sigma_1^1, \sigma_2^1) \in \mathbb{O}(\sigma^1)$, sodass:

(i) $(\sigma_1^1, \sigma_2^1) \in (0, k_2, 0)(\sigma^1)$ und

(ii) $(\sigma_1^1, \sigma_2^1) \notin (0, k_2', k_3')(\sigma^1)$ gilt.

(3) Im Fall $k_2 \leq k_2'$ gilt zusätzlich: Für ein beliebiges $\sigma \in Q$ ist

$$(0, k_2, 0)(\sigma) \subseteq (0, k_2', k_3')(\sigma).$$

Beweis: (1) Folgt mit

$$GP_3((0, k_2, 0)) = 0 < k_3' = GP_3((0, k_2', k_3'))$$

aus dem Größter-Puffer-Lemma.

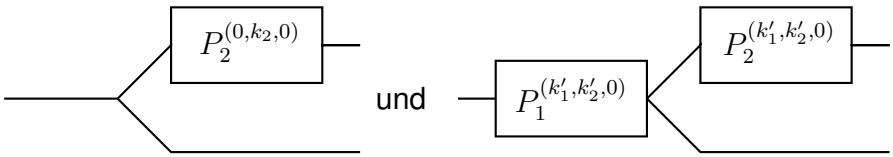
(2) Folgt mit

$$GP_2((0, k_2, 0)) = k_2 > k_2' = GP_2((0, k_2', k_3'))$$

aus dem Größter-Puffer-Lemma.

(3) Da $0 \geq 0$, $k_2' \geq k_2$ und $k_3' \geq 2 > 0$ gilt, folgt die Behauptung direkt aus dem Erweiterungs-Lemma. □

$(0, k_2, 0)$ **und** $(k_1', k_2', 0)$



Satz: Seien $k_2, k_1', k_2' \in \mathbb{N}_{\geq 2}$. Dann gilt

(1) Es existieren $\sigma^0 \in Q$ und $(\sigma_1^0, \sigma_2^0) \in \mathbb{O}(\sigma^0)$, sodass:

(i) $(\sigma_1^0, \sigma_2^0) \notin (0, k_2, 0)(\sigma^0)$ und

(ii) $(\sigma_1^0, \sigma_2^0) \in (k_1', k_2', 0)(\sigma^0)$ gilt.

(2) Im Fall $k_2 > k_1' + k_2' - 1$ gilt zusätzlich:
Es existieren $\sigma^1 \in Q$ und $(\sigma_1^1, \sigma_2^1) \in \mathbb{O}(\sigma^1)$, sodass:

(i) $(\sigma_1^1, \sigma_2^1) \in (0, k_2, 0)(\sigma^1)$ und

(ii) $(\sigma_1^1, \sigma_2^1) \notin (k_1', k_2', 0)(\sigma^1)$ gilt.

(3) Im Fall $k_2 \leq k_1' + k_2' - 1$ gilt zusätzlich: Für ein beliebiges $\sigma \in Q$ ist

$$(0, k_2, 0)(\sigma) \subseteq (k_1', k_2', 0)(\sigma).$$

Beweis: (1) Folgt mit

$$GP_3((0, k_2, 0)) = 0 < k_1' = GP_3((k_1', k_2', 0))$$

aus dem Größter-Puffer-Lemma.

(2) Folgt mit

$$GP_2((0, k_2, 0)) = k_2 > k_1' + k_2' - 1 = GP_2((k_1', k_2', 0))$$

aus dem Größter-Puffer-Lemma.

(3) Sei zunächst $k_1' + k_2' - 1 = k_2$. Betrachte $\sigma \in Q$ und einen zugehörigen, in $(0, k_2, 0)$ erzeugten Output

$$(\sigma_1^{(0,k_2,0)}, \sigma_2^{(0,k_2,0)}) \in (0, k_2, 0)(\sigma).$$

In $(k_1', k_2', 0)$ kann dieser Output auch erzeugt werden:
Der Split ist (unabhängig vom Setting) nach dem Split-Lemma eindeutig festgelegt. Betrachte die Vorgehensweise von $P_1^{(k_1', k_2', 0)}$:

Der Puffer hat zu jedem Zeitpunkt maximal k_1' Jobs geladen und kann wie folgt agieren:

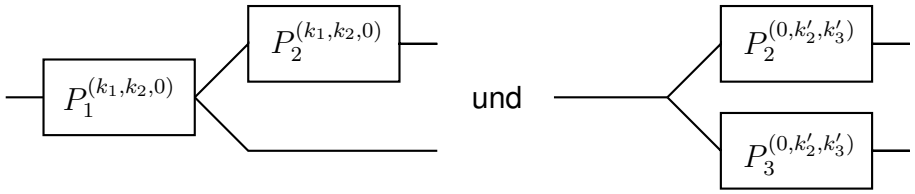
1. *Es befindet sich ein Job aus $\sigma_2^{(0,k_2,0)}$ im Puffer:*
   Wähle unter den Jobs, welche zu $\sigma_2^{(0,k_2,0)}$ gehören, den minimalen (bezüglich seiner Nummerierung) aus und tausche ihn an die erste Stelle, sodass er als nächstes ausgegeben wird. Die anderen Aufträge behalten ihre Reihenfolge bei.
   Innerhalb von $\sigma_2^{(0,k_2,0)}$ müssen die Jobs aufsteigend sortiert sein, da sie in $(0, k_2, 0)$ keinen Puffer durchlaufen haben und der Split keine Auswirkungen auf die Sortierung hat. Auf diese Weise entsteht somit $\sigma_2^{(0,k_2,0)}$.

2. *Im Puffer befindet sich ein Job aus $\sigma_1^{(0,k_2,0)}$ und $P_2^{(k_1', k_2', 0)}$ hat weniger als $k_2' - 1$ Aufträge geladen.*
   Lasse diesen Auftrag den Puffer passieren und lade ihn anschließend in $P_2^{(k_1', k_2', 0)}$.

3. *Es befinden sich keine Jobs aus $\sigma_2^{(0,k_2,0)}$ im Puffer und $P_2^{(k_1', k_2', 0)}$ hat maximal $k_2' - 1$ Aufträge geladen.*
   Jegliche Vertauschungen, welche in $(0, k_2, 0)$ stattgefunden haben, um von $\sigma \setminus \sigma_2^{(0,k_2,0)}$ zu $\sigma_1^{(0,k_2,0)}$ überzugehen, fanden in einem Puffer der Größe k_2 statt.
   Da sich nach 1. keine Jobs aus $\sigma_2^{(0,k_2,0)}$ mehr in σ befinden, kann anstelle von σ die Queue $\sigma \setminus \sigma_2^{(0,k_2,0)}$ mit dem Output $(\sigma_1^{(0,k_2,0)}, (.))$ betrachtet werden:
   Mit $k_1' + k_2' - 1 = k_2$ müssen nach Puffer-Kombinations-Lemma die gleichen Vertauschungen in $(k_1', k_2', 0)$ ausgeführt werden können. $\sigma_1^{(0,k_2,0)}$ kann folglich erzeugt werden.

Zusammenfassend entsteht $(\sigma_1^{(0,k_2,0)}, \sigma_2^{(0,k_2,0)})$. Aus dem Erweiterungs-Lemma folgt die Behauptung für beliebige k_1', k_2' mit $k_1' + k_2' - 1 \geq k_2$. □

$(k_1, k_2, 0)$ **und** $(0, k_2', k_3')$



Satz: Seien $k_1, k_2, k_2', k_3' \in \mathbb{N}_{\geq 2}$ (mit $k_2' \geq k_3'$). Dann gilt

(1) Sei $k_1 + k_2 - 1 > k_2'$ oder $k_1 > k_3'$.
Dann existieren $\sigma^0 \in Q$ und $(\sigma_1^0, \sigma_2^0) \in \mathbb{O}(\sigma^0)$, sodass:

(i) $(\sigma_1^0, \sigma_2^0) \in (k_1, k_2, 0)(\sigma^0)$ und

(ii) $(\sigma_1^0, \sigma_2^0) \notin (0, k_2', k_3')(\sigma^0)$ gilt.

(2) Sei $k_1 + k_2 - 1 \leq k_2'$ und $k_1 \leq k_3'$.
Dann ist für ein beliebiges $\sigma \in Q$

$$(k_1, k_2, 0)(\sigma) \subseteq (0, k_2', k_3')(\sigma).$$

(3) Sei $k_1 < k_3'$ oder $k_1 + k_2 < k_2' + k_3'$.
Dann existieren $\sigma^1 \in Q$ und $(\sigma_1^1, \sigma_2^1) \in \mathbb{O}(\sigma^1)$, sodass:

(i) $(\sigma_1^0, \sigma_2^0) \notin (k_1, k_2, 0)(\sigma^0)$ und

(ii) $(\sigma_1^0, \sigma_2^0) \in (0, k_2', k_3')(\sigma^0)$ gilt.

(4) Sei $k_1 \geq k_3'$ und $k_1 + k_2 \geq k_2' + k_3'$.
Dann ist für ein beliebiges $\sigma \in Q$

$$(k_1, k_2, 0)(\sigma) \supseteq (0, k_2', k_3')(\sigma).$$

Beweis: (1) Sei zunächst $k_1 + k_2 - 1 > k_2'$. Dann gilt

$$GP_2((k_1, k_2, 0)) = k_1 + k_2 - 1 > k_2' = GP_2((0, k_2', k_3')).$$

Sei andererseits $k_1 > k_3'$. Dann gilt

$$GP_3((k_1, k_2, 0)) = k_1 > k_3' = GP_3((0, k_2', k_3')).$$

In beiden Fällen folgt die Behauptung jeweils mit dem Größter-Puffer-Lemma.

(2) Sei zunächst $k'_2 = k_1 + k_2 - 1$ und $k'_3 = k_1$ Betrachte $\sigma \in Q$ und einen zugehörigen, in $(k_1, k_2, 0)$ erzeugten Output

$$(\sigma_1^{(k_1,k_2,0)}, \sigma_2^{(k_1,k_2,0)}) \in (k_1, k_2, 0)(\sigma).$$

Dieser Output kann in $(0, k'_2, k'_3)$ erzeugt werden:
Der zu $(\sigma_1^{(k_1,k_2,0)}, \sigma_2^{(k_1,k_2,0)})$ gehörige Split ist laut Split-Lemma eindeutig und unabhängig vom Setting festgelegt. Weiterhin können $\sigma_1^{(k_1,k_2,0)}$ und $\sigma_2^{(k_1,k_2,0)}$ separat betrachtet werden, da in $(0, k'_2, k'_3)$ jeder Job lediglich genau einen Puffer durchläuft:

1. Betrachte $\sigma \setminus \sigma_2^{(k_1,k_2,0)}$ und $(\sigma_1^{(k_1,k_2,0)}, (.))$:
   Jegliche Vertauschungen, die in $(k_1, k_2, 0)$ stattgefunden haben, um von $\sigma \setminus \sigma_2^{(k_1,k_2,0)}$ zu $\sigma_1^{(k_1,k_2,0)}$ überzugehen, fanden in den Puffern $P_1^{(k_1,k_2,0)}$ und $P_2^{(k_1,k_2,0)}$ statt.
   Da $k'_2 = k_1 + k_2 - 1$ gilt, kann das Puffer-Kombinations-Lemma angewandt werden. $\sigma_1^{(k_1,k_2,0)}$ ist folglich erzeugbar.

2. Betrachte $\sigma \setminus \sigma_1^{(k_1,k_2,0)}$ und $((.), \sigma_2^{(k_1,k_2,0)})$:
   Jegliche Vertauschungen, welche in $(k_1, k_2, 0)$ stattgefunden haben, um von $\sigma \setminus \sigma_1^{(k_1,k_2,0)}$ zu $\sigma_2^{(k_1,k_2,0)}$ überzugehen, fanden in einem Puffer der Größe k_1 statt. Da $P_3^{(0,k'_2,k'_3)}$ auch die Größe $k_3 = k_1$ besitzt, müssen diese Vertauschungen auch im Setting $(0, k'_2, k'_3)$ ausgeführt werden können. Folglich ist $\sigma_2^{(k_1,k_2,0)}$ realisierbar.

Zusammenfassend kann $(\sigma_1^{(k_1,k_2,0)}, \sigma_2^{(k_1,k_2,0)})$ in $(0, k'_2, k'_3)$ erzeugt werden. Durch das Erweiterungs-Lemma folgt die Behauptung für $k'_2 \geq k_1 + k_2 - 1$ und $k'_3 \geq k_1$.

(3) Sei zunächst $k_1 < k'_3$. Dann folgt die Behauptung durch das Größter-Puffer-Lemma, denn es gilt

$$GP_3((k_1, k_2, 0)) = k_1 < k'_3 = GP_3((0, k'_2, k'_3)).$$

Sei nun $k_1 + k_2 < k'_2 + k'_3$ und $k_1 \geq k'_3$. Aus $k_1 + k_2 < k'_2 + k'_3$ und $k_1 \geq k'_3$ folgt direkt, dass $k'_2 \geq k_2$ ist. Betrachte

$$\sigma^1 := (1, 2, \ldots, w) \in Q \text{ mit } w := k'_2 + k'_3$$

und den Output

$$(\sigma_1^1, \sigma_2^1) := ((w, k_2' - 1, k_2' - 2, \ldots, 1), (w - 1, w - 2, \ldots, \underbrace{w - k_3'}_{=k_2'})) \in \mathbb{O}(\sigma).$$

zu (i): *Angenommen, der Output könnte realisiert werden:*
Die ersten $k_2' - 1$ Jobs aus σ gehören zu σ_1^1. Lasse davon die ersten $k_2 - 1$ Aufträge den Puffer $P_1^{(k_1,k_2,0)}$ passieren und lade sie in $P_2^{(k_1,k_2,0)}$. Anschließend lädt $P_1^{(k_1,k_2,0)}$ die Jobs k_2 bis $k_1 + k_2 - 1$, wodurch er keinen freien Slot mehr hat und anschließend einen Auftrag ausgeben muss. Dieser ausgegebene Job wird entweder zu $P_2^{(k_1,k_2,0)}$ oder direkt zu Maschine 2 gesandt. $P_2^{(k_1,k_2,0)}$ würde diesen Job laden und anschließend auch einen seiner Aufträge ausgeben müssen.
Da $k_1 + k_2 - 1 < k_2' + k_3' - 1 = w - 1$ ist, wird demnach ein Job j mit $j \neq 1^{st}(\sigma_1^1), 1^{st}(\sigma_2^1)$ an eine der Maschinen entsandt. Widerspruch! (σ_1^1, σ_2^1) kann folglich nicht erzeugt werden.

zu (ii): *In $(0, k_2, k_3)$ ist der Output aber erzeugbar*:
Da $|\sigma_1^1| = k_2'$ und $|\sigma_2^1| = k_3'$ gilt, kann anschließend zum Split jede beliebige Reihenfolge innerhalb der Queues realisiert werden. Wähle daher gerade die Reihenfolgen aus σ_1^1 bzw. σ_2^1.

(4) Betrachte $\sigma \in Q$, einen zugehörigen, in $(0, k_2', k_3')$ erzeugten Output

$$(\sigma_1^{(0,k_2',k_3')}, \sigma_2^{(0,k_2',k_3')}) \in (0, k_2', k_3')(\sigma)$$

und sei $\delta^{(0,k_2',k_3')}$ eine zugehörige Bearbeitungsreihenfolge.

Der Output kann auch in $(k_1, k_2, 0)$ realisiert werden:
Der Split ist nach Split-Lemma eindeutig festgelegt. Betrachte $P_1^{(k_1,k_2,0)}$ in einem Moment t, in welchem er genau k_1 Jobs geladen hat:

1.) *Es befindet sich ein Job aus $\sigma_1^{(0,k_2',k_3')}$ im Puffer und $P_2^{(k_1,k_2,0)}$ hat weniger als $k_2 - 1$ Aufträge geladen.*
Lasse diesen Job den Puffer passieren und lade ihn in $P_2^{(k_1,k_2,0)}$.

2.) *Es befinden sich nur Jobs aus $\sigma_2^{(0,k_2',k_3')}$ im Puffer und $P_1^{(k_1,k_2,0)}$ und $P_2^{(k_1,k_2,0)}$ haben zusammen weniger als $k_2' + k_3' - 1$ Jobs geladen.*
Wegen $k_1 \geq k_3'$ müssen sich weniger als $k_2' - (k_1 - k_3' + 1)$ Jobs in $P_2^{(k_1,k_2,0)}$ befinden. (#)

Bezeichne L_3 die Menge der Jobs in den beiden Puffern. Betrachte die Aufträge aus L_3 in $(0, k_2', k_3')$ in einem Moment t' mit $\gamma^{(0,k_2',k_3')}(t') = \gamma^{(k_1,k_2,0)}(t)$ $(*)$:

i) $P_3^{(0,k_2',k_3')}$ hat nur Jobs aus L_3 geladen und keinen Slot mehr frei, denn es befinden sich k_1 Jobs aus $\sigma_2^{(0,k_2',k_3')}$ in L_2 und es gilt $k_1 > k_3'$.

ii) $P_2^{(0,k_2',k_3')}$ hat mindestens einen freien Platz.
Wegen $(\#)$ können sich keine k_2' Jobs aus L_3 in $P_2^{(0,k_2',k_3')}$ befinden. Ein Job $j \notin L_2$ kann erst zu einem späteren Zeitpunkt geladen werden, da dieser größer (bezüglich seiner Nummerierung) sein muss, als die Jobs in L_2 und $P_3^{(0,k_2',k_3')}$ somit früher einen Job ausgeben muss.

Folglich muss $nrj(\delta^{(0,k_2',k_3')}, \gamma^{(0,k_2',k_3')}(t')) = nrj(\delta^{(0,k_2',k_3')}, \gamma^{(k_1,k_2,0)}(t)) \in L_3$ sein. Nach Auswurf-Lemma kann er als nächstes an Maschine 2 gesandt werden.

3.) $P_1^{(k_1,k_2,0)}$ *und* $P_2^{(k_1,k_2,0)}$ *haben zusammen mindestens* $k_2' + k_3' - 1 = PK((0, k_2', k_3'))$ *Aufträge geladen.*
Bezeichne $\hat{L}$ die Menge der Jobs in den beiden Puffern. Betrachte die Aufträge aus $\hat{L}$ in $(0, k_2', k_3')$ in einem Moment t' mit $\gamma^{(0,k_2',k_3')}(t') = \gamma^{(k_1,k_2,0)}(t)$ $(*)$. Nach Übergangs-Lemma gilt $nrj(\delta^{(0,k_2',k_3')}, \gamma^{(0,k_2',k_3')}(t')) = nrj(\delta^{(0,k_2',k_3')}, \gamma^{(k_1,k_2,0)}(t)) \in L$, welcher nach Auswurf-Lemma an Maschine 1 bzw. 2 ausgegeben werden kann.

Habe $P_1^{(k_1,k_2,0)}$ nun zum Zeitpunkt $\bar{t}$ weniger als k_1 Jobs geladen. Es muss sich dabei um die letzten Jobs aus σ handeln, wodurch sich auch $nrj(\delta^{(0,k_2',k_3')}, \gamma^{(k_1,k_2,0)}(\bar{t}))$ in einem der Puffer befinden muss.
Analog zur vorhergehenden Puffervorgehensweise von $P_1^{(k_1,k_2,0)}$ kann angenommen werden, dass $P_2^{(k_1,k_2,0)}$ mindestens einen freien Slot hat. Aus dem Auswurf-Lemma folgt, dass $nrj(\delta^{(0,k_2',k_3')}, \gamma^{(k_1,k_2,0)}(\bar{t}))$ als nächstes ausgegeben werden kann.

Zusammenfassend folgt durch das γ-Lemma, dass die Forderung nach $(*)$ legitim ist. Mit dem Output-Lemma folgt somit $(\sigma_1^{(0,k_2',k_3')}, \sigma_2^{(0,k_2',k_3')}) \in (k_1, k_2, 0)(\sigma)$.

Bemerkungen:

1.) (2) *impliziert* (3):

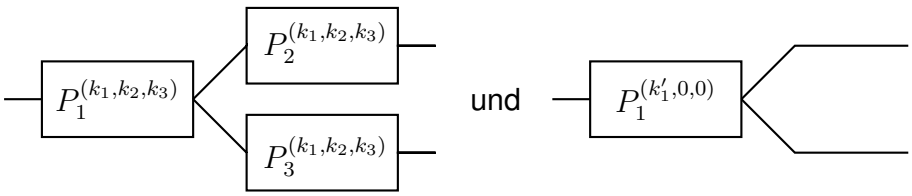
Angenommen es gelte (2) und nicht (3). Dann gilt

$$\begin{aligned} k_1 + k_2 - 1 &\leq k_2' \\ k_1 &\leq k_3' \\ k_1 &\geq k_3' \\ k_1 + k_2 &\geq k_2' + k_3'. \end{aligned}$$

Aus $k_1 \leq k_3'$ und $k_1 \geq k_3'$ folgt zunächst $k_1 = k_3'$. Über $k_1 + k_2 \geq k_2' + k_3'$ folgt daraus $k_2 \geq k_2'$, wodurch $k_1 - 1 + k_2 > k_2'$ ist, da $k_1 \in \mathbb{N}_{\geq 2}$. Dies liefert einen Widerspruch zu $k_1 + k_2 - 1 \leq k_2'$. Also gilt (2) $\Rightarrow$ (3).

2.) (4) *impliziert* (1):
Es gilt (1) $\iff \neg$(2) und (3) $\iff \neg$(4). Nach Bemerkung 1.) gilt (2)$\Rightarrow$(3), was demnach gleichbedeutend mit $\neg$(1)$\Rightarrow \neg$(4) ist. Negiert man diesen Ausdruck erhält man (4)$\Rightarrow$(1).

(k_1, k_2, k_3) **und** $(k_1', 0, 0)$



Satz: Seien $k_1, k_2, k_3, k_1' \in \mathbb{N}_{\geq 2}$ (mit $k_2 \geq k_3$). Dann gilt

(1) Sei $k_1 + k_3 - 1 < k_1'$.
Dann existieren $\sigma^0 \in Q$ und $(\sigma_1^0, \sigma_2^0) \in \mathbb{O}(\sigma^0)$, sodass:

(i) $(\sigma_1^0, \sigma_2^0) \notin (k_1, k_2, k_3)(\sigma^0)$ und

(ii) $(\sigma_1^0, \sigma_2^0) \in (k_1', 0, 0)(\sigma^0)$ gilt.

(2) Sei $k_1 + k_3 - 1 \geq k_1'$. Dann ist für ein beliebiges $\sigma \in Q$

$$(k_1, k_2, k_3)(\sigma) \supseteq (k_1', 0, 0)(\sigma).$$

(3) Sei $\sum_{i=1}^{3} k_i - 2 > k_1'$.
Dann existieren $\sigma^1 \in Q$ und $(\sigma_1^1, \sigma_2^1) \in \mathbb{O}(\sigma^1)$, sodass:

(i) $(\sigma_1^1, \sigma_2^1) \in (k_1, k_2, k_3)(\sigma^1)$ und

(ii) $(\sigma_1^1, \sigma_2^1) \notin (k_1', 0, 0)(\sigma^1)$ gilt.

(4) Sei $\sum_{i=1}^{3} k_i - 2 \leq k_1'$. Dann ist für ein beliebiges $\sigma \in Q$

$$(k_1, k_2, k_3)(\sigma) \subseteq (k_1', 0, 0)(\sigma).$$

Beweis: (1) Mit

$$GP_3((k_1, k_2, k_3)) = k_1 + k_3 - 1 < k_1' = GP_3((k_1', 0, 0))$$

folgt dies aus dem Größter-Puffer-Lemma.

(2) Betrachte $\sigma \in Q$, einen zugehörigen, in $(k_1', 0, 0)$ erzeugten Output

$$(\sigma_1^{(k_1',0,0)}, \sigma_2^{(k_1',0,0)}) \in (k_1', 0, 0)(\sigma)$$

und eine Bearbeitungsreihenfolge $\delta^{(k_1',0,0)}$.

Dieser Output kann in (k_1, k_2, k_3) erzeugt werden:
Es muss lediglich die Puffervorgehensweise betrachtet werden, da der Split durch das Split-Lemma eindeutig festgelegt ist. Der Puffer $P_1^{(k_1,k_2,k_3)}$ hat zu einem beliebigen Zeitpunkt t maximal k_1 Jobs geladen. Er kann wie folgt agieren:

1. *Unter den geladenen Jobs befindet sich ein Job aus* $\sigma_1^{(k_1',0,0)}$ *und* $P_2^{(k_1,k_2,k_3)}$ *hat mehr als einen freien Slot.*
   Lasse diesen den Puffer passieren und lade ihn in $P_2^{(k_1,k_2,k_3)}$.

2. *Unter den geladenen Jobs befindet sich ein Job aus* $\sigma_2^{(k_1',0,0)}$ *und* $P_3^{(k_1,k_2,k_3)}$ *hat mehr als einen freien Slot.*
   Lasse diesen den Puffer passieren und lade ihn in $P_3^{(k_1,k_2,k_3)}$.

3. $P_1^{(k_1,k_2,0)}$, $P_2^{(k_1,k_2,0)}$ *und* $P_3^{(k_1,k_2,0)}$ *haben zusammen mindestens* $k_1' = PK((k_1', 0, 0))$ *Aufträge geladen.*
   Falls $P_1^{(k_1,k_2,k_3)}$ weniger als k_1 Jobs geladen hat, so muss es sich um die letzten Aufträge aus σ handeln, wodurch $nrj(\delta^{(k_1',0,0)}, \gamma^{(k_1,k_2,k_3)}(t))$ in einem der drei Puffer sein muss. Nach Auswurf-Lemma kann dieser als nächstes an Maschine 1 bzw. 2 ausgegeben werden. Gebe $nrj(\delta^{(k_1',0,0)}, \gamma^{(k_1,k_2,k_3)}(t))$ als nächstes aus.

   Habe $P_1^{(k_1,k_2,k_3)}$ genau k_1 Jobs geladen. Da nach 1. bzw. 2. auch $P_2^{(k_1,k_2,k_3)}$ genau $k_2 - 1 \geq k_3 - 1$ oder $P_3^{(k_1,k_2,k_3)}$ genau $k_3 - 1$ Jobs geladen hat, befinden sich mindestens $k_1 + k_3 - 1 \geq k_1' = PK((k_1', 0, 0))$ Jobs in den Puffern des Settings (k_1, k_2, k_3).
   Betrachte diese Aufträge in $(k_1', 0, 0)$ in einem Moment t' mit $\gamma^{(k_1',0,0)}(t') = \gamma^{(k_1,k_2,k_3)}(t)$ $(*)$. Nach Übergangs-Lemma befindet sich $nrj(\delta^{(k_1',0,0)}, \gamma^{(k_1',0,0)}(t')) = nrj(\delta^{(k_1',0,0)}, \gamma^{(k_1,k_2,k_3)}(t))$ in einem der Puffer von (k_1, k_2, k_3), welcher nach Auswurf-Lemma jeweils an Maschine 1 bzw. 2 gesandt werden kann. Gebe $nrj(\delta^{(k_1',0,0)}, \gamma^{(k_1,k_2,k_3)}(t))$ als nächstes aus. Durch das γ-Lemma ist die Forderung nach $(*)$ folglich gerechtfertigt.

Durch die Vorgehensweisen des Puffers und das Output-Lemma folgt, dass $(\sigma_1^{(k_1',0,0)}, \sigma_2^{(k_1',0,0)})$ generiert wird.

(3) Betrachte

$$\sigma^1 := (1, \ldots, u) \text{ mit } u := \sum_{i=1}^{3} k_i - 1$$

und einen zugehörigen Output

$$(\sigma_1^1, \sigma_2^1) \in \mathbb{O}(\sigma) \text{ mit } 1^{st}(\sigma_1^1) = u \text{ und } 1^{st}(\sigma_2^1) = u - 1.$$

(i) Man kann einen Output, welcher oben genannte Kriterien erfüllt, in (k_1, k_2, k_3) erzeugen:
Speichere dazu die ersten $k_1 - 1$ Jobs in $P_1^{(k_1,k_2,k_3)}$ zwischen und lasse die anderen Aufträge passieren. Die Jobs k_1 bis $k_1 + k_2 - 2$ können in $P_2^{(k_1,k_2,k_3)}$ geladen werden und die Jobs $k_1 + k_2 - 1$ bis $k_1 + k_2 + k_3 - 2$ in $P_3^{(k_1,k_2,k_3)}$. Damit verfügen sowohl $P_2^{(k_1,k_2,k_3)}$ als auch $P_3^{(k_1,k_2,k_3)}$ noch über genau einen freien Platz. Lade daher den Job $u - 1$ in $P_2^{(k_1,k_2,k_3)}$, sortiere ihn nach vorn und sende ihn anschließend an Maschine 1. Analog kann $u - 1$ in $P_3^{(k_1,k_2,k_3)}$ geladen und an Maschine 2 gesandt werden. Somit gilt $1^{st}(\sigma_1^1) = u$ und $1^{st}(\sigma_2^1) = u - 1$.

(ii) *Im Setting* $(k_1', 0, 0)$ *ist dies nicht möglich:*
Der Puffer lädt zu Beginn die Jobs 1 bis k_1' und muss anschließend einen davon wieder ausgeben. Damit ist

$$1^{st}(\sigma_2^1), 1^{st}(\sigma_1^1) \leq k_1' < k_1 + k_2 + k_3 - 2 = u - 1.$$

In $(k_1', 0, 0)$ kann folglich kein Output der oben genannten Art entstehen.

(4) Betrachte $\sigma \in Q$, einen zugehörigen, in (k_1, k_2, k_3) erzeugten Output

$$(\sigma_1^{(k_1,k_2,k_3)}, \sigma_2^{(k_1,k_2,k_3)}) \in (k_1, k_2, k_3)(\sigma)$$

und sei $\delta^{(k_1,k_2,k_3)}$ eine zugehörige Bearbeitungsreihenfolge.

Dieser Output kann auch in $(k_1', 0, 0)$ erzeugt werden:
Der Split ist durch das Split-Lemma eindeutig vorgegeben. Der Puffer in $(k_1', 0, 0)$ hat zu einem beliebigen Moment t maximal $k_1' \geq \sum_{i=1}^{3} k_i - 2 = PK((k_1, k_2, k_3))$ Jobs geladen.

a) *Es befinden sich genau* k_1' *Jobs in* $P_1^{(k_1',0,0)}$*:*
Bezeichne $\hat{L}$ die Menge dieser geladenen Aufträge. Betrachte die Jobs aus $\hat{L}$ in (k_1, k_2, k_3) in einem Moment t' mit $\gamma^{(k_1',0,0)}(t') = \gamma^{(k_1,k_2,k_3)}(t)$ $(*)$. Durch das Übergangs-Lemma folgt, dass $nrj(\delta^{(k_1,k_2,k_3)}, \gamma^{(k_1,k_2,k_3)}(t)) = nrj(\delta^{(k_1,k_2,k_3)}, \gamma^{(k_1',0,0)}(t')) \in L$ ist. Tausche diesen Job an erste Stelle und gib ihn als nächstes an Maschine 1 bzw. 2 aus.

b) $P_1^{(k_1',0,0)}$ *hat mindestens einen freien Slot:*
Es muss sich dabei um die letzten Aufträge aus σ handeln, wodurch sich $nrj(\delta^{(k_1,k_2,k_3)}, \gamma^{(k_1,k_2,k_3)}(t))$ im Puffer befindet. Gib diesen als nächstes an Maschine 1 bzw. 2 aus.

Da in beiden Fällen $nrj(\delta^{(k_1,k_2,k_3)}, \gamma^{(k_1,k_2,k_3)}(t))$ an Maschine 1 bzw. 2 ausgegeben werden kann, folgt mit dem γ-Lemma, dass die Voraussetzung $(*)$ jeweils erfüllt ist. Mit dem Output-Lemma folgt, dass $(\sigma_1^{(k_1,k_2,k_3)}, \sigma_2^{(k_1,k_2,k_3)})$ entsteht. □

Bemerkung:

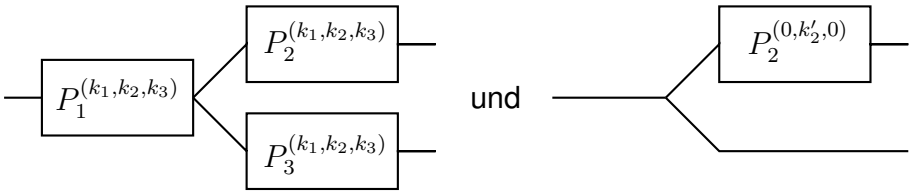
1) (4) impliziert (1):

$$k_1 + k_3 - 1 \overset{(3)}{\leq} k_1' + \underbrace{(1 - k_2)}_{<0} < k_1'.$$

2) (2) impliziert (3):

$$\sum_{i=1}^{3} k_i - 2 = (k_1 + k_3 - 1) + k_2 - 1 \overset{(4)}{\geq} k_1' + (\underbrace{k_2}_{\geq 2} - 1) > k_1'.$$

(k_1, k_2, k_3) **und** $(0, k_2', 0)$



Satz: Seien $k_1, k_2, k_3, k_2' \in \mathbb{N}_{\geq 2}$ (mit $k_2 \geq k_3$). Dann gilt

(1) Es existieren $\sigma^0 \in Q$ und $(\sigma_1^0, \sigma_2^0) \in \mathbb{O}(\sigma^0)$, sodass:

(i) $(\sigma_1^0, \sigma_2^0) \notin (0, k_2', 0)(\sigma^0)$ und

(ii) $(\sigma_1^0, \sigma_2^0) \in (k_1, k_2, k_3)(\sigma^0)$ gilt.

(2) Im Fall $k_2' > k_1 + k_2 - 1$ gilt zusätzlich:
Es existieren $\sigma^1 \in Q$ und $(\sigma_1^1, \sigma_2^1) \in \mathbb{O}(\sigma^1)$, sodass:

(i) $(\sigma_1^1, \sigma_2^1) \in (0, k_2', 0)(\sigma^1)$ und

(ii) $(\sigma_1^1, \sigma_2^1) \notin (k_1, k_2, k_3)(\sigma^1)$ gilt.

(3) Im Fall $k_2' \leq k_1 + k_2 - 1$ gilt zusätzlich: Für ein beliebiges $\sigma \in Q$ ist

$$(0, k_2', 0)(\sigma) \subseteq (k_1, k_2, k_3)(\sigma).$$

Beweis: (1) Folgt mit

$$GP_3((k_1, k_2, k_3) = k_1 + k_3 - 1 \geq 2 + 2 - 1 > 0 = GP_3((0, k_2', 0))$$

aus dem Größter-Puffer-Lemma.

(2) Folgt mit

$$GP_2((k_1, k_2, k_3) = k_1 + k_2 - 1 < k_2' >= GP_2((0, k_2', 0))$$

aus dem Größter-Puffer-Lemma.

(3) Betrachte $\sigma \in Q$ und einen zugehörigen, in $(0, k_2', 0)$ erzeugten Output

$$(\sigma_1^{(0,k_2',0)}, \sigma_2^{(0,k_2',0)}) \in (0, k_2', 0)(\sigma).$$

Sei zunächst $k_1 + k_2 - 1 = k_2'$. Dieser Output ist auch in (k_1, k_2, k_3) realisierbar: Der Split ist (unabhängig vom Setting) nach dem Split-Lemma eindeutig festgelegt. $P_1^{(k_1,k_2,k_3)}$ hat zu jedem Zeitpunkt maximal k_1 Jobs geladen und kann nach folgendem Schema vorgehen:

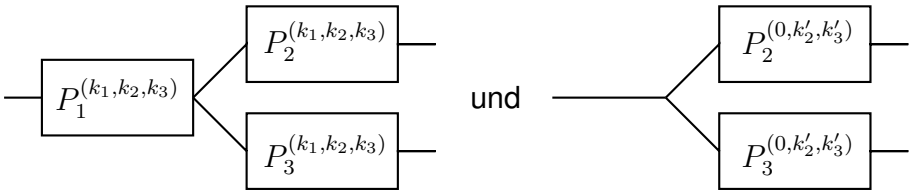
1. *Im Puffer befindet sich ein Job aus* $\sigma_2^{(0,k_2',0)}$.
   Wähle unter den Jobs, welche zu $\sigma_2^{(0,k,0)}$ gehören, den minimalen (bezüglich seiner Nummerierung) aus und tausche ihn an die erste Stelle, sodass er als nächstes ausgegeben wird. Die anderen Aufträge behalten ihre Reihenfolge bei. $P_2^{(k_1,k_2,k_3)}$ muss anschließend keine Sortierung vornehmen und kann die Jobs in dieser Reihenfolge an Maschine 3 ausgeben.
   Innerhalb von $\sigma_2^{(0,k,0)}$ müssen die Aufträge aufsteigend sortiert sein, da sie in $(0, k, 0)$ keinen Puffer durchlaufen haben und der Split keine Auswirkungen auf die Sortierung hat. Auf diese Weise entsteht somit $\sigma_2^{(0,k,0)}$.

2. *Unter den geladenen Jobs befinden sich nur Jobs aus* $\sigma_1^{(0,k_2',0)}$ *und* $P_2^{(k_1,k_2,k_3)}$ *hat weniger als* $k_2 - 1$ *Aufträge geladen*.
   Lasse solange Aufträge den Puffer passieren, bis sich $k_2 - 1$ Jobs in $P_2^{(k_1,k_2,k_3)}$ befinden.

3. *Es befindet sich ein Jobs aus* $\sigma_1^{(0,k_2',0)}$ *im Puffer und* $P_2^{(k_1,k_2,k_3)}$ *hat einen freien Slot*.
   Jegliche Vertauschungen, welche in $(0, k_2', 0)$ stattgefunden haben, um von
   $\sigma \setminus \sigma_2^{(0,k_2',0)}$ zu $\sigma_1^{(0,k_2',0)}$ überzugehen, fanden in einem Puffer der Größe k_2' statt.
   Da sich nach 1. keine Jobs aus $\sigma_2^{(0,k_2',0)}$ mehr in σ befinden, kann anstelle von σ die Queue $\sigma \setminus \sigma_2^{(0,k_2',0)}$ mit dem Output $(\sigma_1^{(0,k_2',0)}, (.))$ betrachtet werden:
   Mit $k_1 + k_2 - 1 = k_2'$ müssen nach Puffer-Kombinations-Lemma die gleichen Vertauschungen in (k_1, k_2, k_3) ausgeführt werden können. $\sigma_1^{(0,k_2',0)}$ kann folglich erzeugt werden.

Zusammenfassend entsteht durch diese Arbeitsweise $(\sigma_1^{(0,k_2',0)}, \sigma_2^{(0,k_2',0)})$. Durch das Erweiterungs-Lemma folgt die Behauptung für $k_1 + k_2 - 1 \geq k_2'$ □

(k_1, k_2, k_3) **und** $(0, k_2', k_3')$



Satz: Seien $k_1, k_2, k_3, k_2', k_3' \in \mathbb{N}_{\geq 2}$ (mit $k_2 \geq k_3$ und $k_2' \geq k_3'$). Dann ist

(1) Gelte eine der folgenden Bedingungen:

a) $\sum_{j=1}^{3} k_j - 2 < k_2' + k_3' - 1$,

b) $k_1 + k_2 - 1 < k_2'$ oder

c) $k_1 + k_3 - 1 < k_3'$.

Dann existieren $\sigma^0 \in Q$ und $(\sigma_1^0, \sigma_2^0) \in \mathbb{O}(\sigma^0)$, sodass:

(i) $(\sigma_1^0, \sigma_2^0) \notin (k_1, k_2, k_3)(\sigma^0)$ und

(ii) $(\sigma_1^0, \sigma_2^0) \in (0, k_2', k_3')(\sigma^0)$ gilt.

(2) Gelten folgende Bedingungen:

a) $\sum_{j=1}^{3} k_j - 2 \geq k_2' + k_3' - 1$,

b) $k_1 + k_2 - 1 \geq k_2'$ und

c) $k_1 + k_3 - 1 \geq k_3'$.

Dann ist für ein beliebiges $\sigma \in Q$

$$(k_1, k_2, k_3)(\sigma) \supseteq (0, k_2', k_3')(\sigma).$$

(3) Sei $k_1 + k_2 - 1 > k_2'$ oder $k_1 + k_3 - 1 > k_3'$.
Dann existieren $\sigma^1 \in Q$ und $(\sigma_1^1, \sigma_2^1) \in \mathbb{O}(\sigma^1)$, sodass:

(i) $(\sigma_1^1, \sigma_2^1) \in (k_1, k_2, k_3)(\sigma^1)$ und

(ii) $(\sigma_1^1, \sigma_2^1) \notin (0, k_2', k_3')(\sigma^1)$ gilt.

(4) Sei $k_1 + k_2 - 1 \leq k_2'$ und $k_1 + k_3 - 1 \leq k_3'$. Dann ist für ein beliebiges $\sigma \in Q$

$$(k_1, k_2, k_3)(\sigma) \subseteq (0, k_2', k_3')(\sigma).$$

Beweis: (1) Falls $k_1 + k_i - 1 < k'_i$ für $i = 2; 3$ gilt, so folgt mit

$$GP_i((k_1, k_2, k_3)) = k_1 + k_i - 1 < k'_i = GP_i((0, k'_2, k'_3))$$

und dem Größter-Puffer-Lemma die Behauptung.

Sei nun $\sum_{j=1}^{3} k_j - 2 < k'_2 + k'_3 - 1$. Betrachte

$$\sigma^1 := (1, 2, \ldots, w) \text{ mit } w := \sum_{j=1}^{3} k_j - 1$$

und den Output

$$(\sigma_1^1, \sigma_2^1) := ((w-1, w-2, \ldots, k_2+k_3-1, k_2-1, k_2-2, \ldots, 1), (w, k_2+k_3-2, \ldots, k_2)).$$

zu (i): *Angenommen in* $(0, k'_2, k'_3)$ *kann dieser Output generiert werden*:
Die Puffer $P_2^{(0,k'_2,k'_3)}$ und $P_3^{(0,k'_2,k'_3)}$ können zunächst jeweils maximal $k'_2 - 1$ bzw. $k'_3 - 1$ Jobs laden. Spätestens, wenn der Job $(k'_2 - 1) + (k'_3 - 1) + 1$ an einen der Puffer gesandt wird, verfügt ein Puffer über keinen freien Slot mehr und muss einen Job j an Maschine 1 bzw. 2 ausgeben. Es gilt

$$j \leq (k'_2 - 1) + (k'_3 - 1) + 1 = k'_2 + k'_3 - 1 < \sum_{j=1}^{3} k_j - 2 = w - 1.$$

Das heißt, dass sowohl $j \neq 1^{st}(\sigma_1^1)$, als auch $j \neq 1^{st}(\sigma_2^1)$ ist. Widerspruch! (σ_1^1, σ_2^1) kann nicht erzeugt werden.

zu (ii): *In* (k_1, k_2, k_3) *ist der Output erzeugbar*:
Lasse die ersten $k_2 - 1$ Jobs den ersten Puffer passieren und lade sie in $P_2^{(k_1,k_2,k_3)}$. Analog werden die Jobs k_2 bis $k_2 + k_3 - 2$ in $P_3^{(k_1,k_2,k_3)}$ geladen. Anschließend nimmt $P_1^{(k_1,k_2,k_3)}$ die Jobs $k_2 + k_3 - 2$ bis $w - 1$ auf und kann $w - 1$ nach Auswurf-Lemma an Maschine 1 ausgeben. Abschließend lädt $P_1^{(k_1,k_2,k_3)}$ den Job w. Nun befinden sich alle verbleibenden Jobs in den drei Puffern des Settings und können jeweils nach Auswurf-Lemma in beliebiger Reihenfolge ausgegeben werden. Wählt man die Reihenfolgen aus σ_1^1 bzw. σ_2^1, so entsteht (σ_1^1, σ_2^1).

(2) Betrachte $\sigma \in Q$, einen zugehörigen, in $(0, k'_2, k'_3)$ erzeugten Output

$$(\sigma_1^{(0,k'_2,k'_3)}, \sigma_2^{(0,k'_2,k'_3)}) \in (0, k'_2, k'_3)(\sigma)$$

und eine zugehörige Bearbeitungsreihenfolge $\delta^{(0,k_2',k_3')}$.

Dieser Output kann in (k_1, k_2, k_3) erzeugt werden:
Durch das Split-Lemma ist der Split eindeutig bestimmt. Betrachte $P_1^{(k_1,k_2,k_3)}$ zu einem Zeitpunkt t, in welchem er genau k_1 Jobs geladen hat:

1. *Im Puffer befindet sich ein Job aus $\sigma_1^{(0,k_2',k_3')}$ und $P_2^{(k_1,k_2,k_3)}$ hat noch mehr als einen Slot frei.*
   Lasse diesen Job den Puffer passieren und lade ihn in $P_2^{(k_1,k_2,k_3)}$.

2. *Im Puffer befindt sich ein Job aus $\sigma_2^{(0,k_2',k_3')}$ und $P_3^{(k_1,k_2,k_3)}$ hat noch mehr als einen Slot frei.*
   Lasse diesen Job den Puffer passieren und lade ihn in $P_3^{(k_1,k_2,k_3)}$.

3. *Im Puffer befinden sich nur Jobs aus $\sigma_1^{(0,k_2',k_3')}$, $P_2^{(k_1,k_2,k_3)}$ hat nur noch einen freien Slot und $P_1^{(k_1,k_2,k_3)}$, $P_2^{(k_1,k_2,k_3)}$ und $P_3^{(k_1,k_2,k_3)}$ haben insgesamt weniger als $k_2' + k_3' - 1$ Jobs geladen.*
   Wegen $k_1 + k_2 - 1 \geq k_2'$ müssen sich weniger als $k_3' - (k_1 + k_2 - k_2') < k_3'$ Jobs in $P_3^{(k_1,k_2,0)}$ befinden. $(\#)$
   Bezeichne L_2 die Menge der Jobs in den drei Puffern. Betrachte die Aufträge aus L_2 in $(0, k_2', k_3')$ in einem Moment t' mit $\gamma^{(0,k_2',k_3')}(t') = \gamma^{(k_1,k_2,k_3)}(t)$ $(*)$:

   i) $P_2^{(0,k_2',k_3')}$ hat ausschließlich Jobs aus L_2 geladen und keinen Slot mehr frei, denn es befinden sich $k_1 + k_2 - 1$ Jobs aus $\sigma_1^{(0,k_2',k_3')}$ in L_2 und es gilt $k_1 + k_2 - 1 > k_2'$.

   ii) $P_3^{(0,k_2',k_3')}$ hat mindestens einen freien Platz.
   Wegen $(\#)$ können sich keine k_3' Jobs aus L_2 in $P_3^{(0,k_2',k_3')}$ befinden. Ein Job $j \notin L_2$ kann erst zu einem späteren Zeitpunkt geladen werden, da dieser größer (bezüglich seiner Nummerierung) sein muss, als die Jobs in L_2 und $P_2^{(0,k_2',k_3')}$ somit früher einen Job ausgeben muss.

   Folglich muss $nrj(\delta^{(0,k_2',k_3')}, \gamma^{(k_1,k_2,k_3)}(t)) = nrj(\delta^{(0,k_2',k_3')}, \gamma^{(0,k_2',k_3')}(t')) \in L_2$ sein. Gib ihn als nächstes an Maschine 1 aus, da dies nach Auswurf-Lemma möglich ist.

4. *Im Puffer befinden sich nur Jobs aus $\sigma_2^{(0,k_2',k_3')}$ und $P_3^{(k_1,k_2,k_3)}$ hat nur noch einen freien Slot.*
   Durch eine zu 3. analoge Argumentation befindet sich $nrj(\delta^{(0,k_2',k_3')}, \gamma^{(k_1,k_2,k_3)}(t))$ in $P_1^{(k_1,k_2,k_3)}$ oder $P_3^{(k_1,k_2,k_3)}$, da $k_1 + k_3 - 1 \geq$

k_3' gilt. Gib diesen Job als nächstes an Maschine 2 aus, da dies nach Auswurf-Lemma möglich ist.

5. $P_1^{(k_1,k_2,k_3)}$, $P_2^{(k_1,k_2,k_3)}$ *und* $P_3^{(k_1,k_2,k_3)}$ *haben zusammen mindestens* $k_2'+k_3'-1 = PK((0,k_2',k_3'))$ *Jobs geladen.*
   Die Menge dieser in den drei Puffern befindlichen Jobs werde mit $\hat{L}$ bezeichnet. Betrachte die Aufträge aus L in $(0,k_2',k_3')$ in einem Moment t' mit $\gamma^{(0,k_2',k_3')}(t') = \gamma^{(k_1,k_2,k_3)}(t)$ $(*)$. Aus dem Übergangs-Lemma folgt $nrj(\delta^{(0,k_2',k_3')}, \gamma^{(k_1,k_2,k_3)}(t)) = nrj(\delta^{(0,k_2',k_3')}, \gamma^{(0,k_2',k_3')}(t')) \in J(\sigma^L)$. Nach Auswurf-Lemma kann dieser Job als nächstes an Maschine 1 bzw. 2 ausgegeben werden.

Habe $P_1^{(k_1,k_2,k_3)}$ nun zum Zeitpunkt $\bar{t}$ weniger als k_1 Jobs geladen. Es muss sich dabei um die letzten Jobs aus σ handeln, wodurch sich auch $nrj(\delta^{(0,k_2',k_3')}, \gamma^{(k_1,k_2,k_3)}(\bar{t}))$ in einem der Puffer befinden muss.
Wenn auch weiterhin 1. und 2. von oben respektiert werden, kann davon ausgegangen werden, dass $P_2^{(k_1,k_2,k_3)}$ und $P_3^{(k_1,k_2,k_3)}$ mindestens einen freien Slot hat. Aus dem Auswurf-Lemma folgt, dass $nrj(\delta^{(0,k_2',k_3')}, \gamma^{(k_1,k_2,k_3)}(\bar{t}))$ als nächstes ausgegeben werden kann.

Zusammenfassend folgt durch das γ-Lemma, dass die Forderung nach $(*)$ legitim ist. Mit dem Output-Lemma folgt somit $(\sigma_1^{(0,k_2',k_3')}, \sigma_2^{(0,k_2',k_3')}) \in (k_1,k_2,k_3)(\sigma)$.

(3) Für $i \in \{2,3\}$ ist

$$GP_i((k_1,k_2,k_3)) = k_1 + k_i - 1 > k_i' = GP_i((0,k_2',k_3')).$$

Zusammen mit dem Größter-Puffer-Lemma folgt die Behauptung.

(4) Sei zunächst $k_2' = k_1 + k_2 - 1$ und $k_3' = k_1 + k_3 - 1$. Betrachte $\sigma \in Q$ und einen zugehörigen, in (k_1,k_2,k_3) erzeugten Output

$$(\sigma_1^{(k_1,k_2,k_3)}, \sigma_2^{(k_1,k_2,k_3)}) \in (k_1,k_2,k_3)(\sigma).$$

Dieser Output ist in $(0,k_2',k_3')$ erzeugbar:
Analog zu den vorhergehenden Unterkapiteln ist der Split bereits eindeutig durch das Split-Lemma vorgegeben. Weiterhin können $\sigma_1^{(k_1,k_2,k_3)}$ und $\sigma_2^{(k_1,k_2,k_3)}$ separat betrachtet werden, da in $(0,k_2',k_3')$ jeder Job lediglich genau einen Puffer durchläuft:

1. Betrachte $\sigma \setminus \sigma_2^{(k_1,k_2,k_3)}$ und $(\sigma_1^{(k_1,k_2,k_3)}, (.))$:
   Jegliche Vertauschungen, die in (k_1, k_2, k_3) stattgefunden haben, um von $\sigma \setminus \sigma_2^{(k_1,k_2,k_3)}$ zu $\sigma_1^{(k_1,k_2,k_3)}$ überzugehen, fanden in den Puffern $P_1^{(k_1,k_2,k_3)}$ und $P_2^{(k_1,k_2,k_3)}$ statt.
   Da $k_2' = k_1 + k_2 - 1$ gilt, kann das Puffer-Kombinations-Lemma angewandt werden. $\sigma_1^{(k_1,k_2,k_3)}$ ist folglich erzeugbar.

2. Betrachte $\sigma \setminus \sigma_1^{(k_1,k_2,k_3)}$ und $((.), \sigma_2^{(k_1,k_2,k_3)})$:
   Durch eine zu 1. analoge Argumentation und $k_3' = k_1 + k_3 - 1$ folgt, dass $\sigma_2^{(k_1,k_2,k_3)}$ realisierbar ist.

Zusammenfassend kann $(\sigma_1^{(k_1,k_2,k_3)}, \sigma_2^{(k_1,k_2,k_3)})$ in $(0, k_2', k_3')$ erzeugt werden. Durch das Erweiterungs-Lemma folgt die Behauptung für $k_2' \geq k_1 + k_2 - 1$ und $k_3' \geq k_1 + k_3 - 1$. □

Bemerkungen:

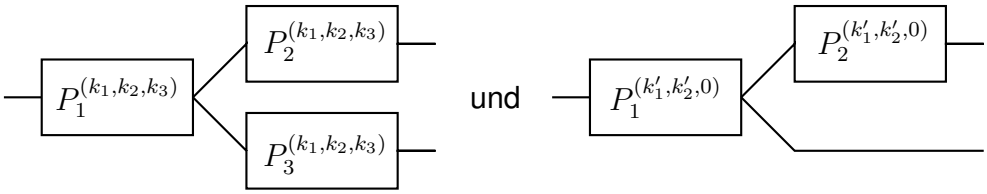
1. (2) *impliziert* (3):
   Angenommen es gelte (2) und nicht (3). Dann gilt

$$\sum_{j=1}^{3} k_j - 2 \geq k_2' + k_3' - 1$$
$$k_1 + k_2 - 1 \geq k_2'$$
$$k_1 + k_3 - 1 \geq k_3'$$
$$k_1 + k_2 - 1 \leq k_2'$$
$$k_1 + k_3 - 1 \leq k_3'.$$

   Aus den unteren vier Ungleichungen folgt zunächst $k_2' = k_1 + k_2 - 1$ und $k_3' = k_1 + k_3 - 1$. Damit muss $k_2' + k_3' - 1 = 2k_1 + k_2 + k_3 - 2 > \sum_{j=1}^{3} k_j - 2$ gelten, da $k_1 \in \mathbb{N}_{\geq 2}$ ist. Widerspruch zur ersten Ungleichung! Aus (2) folgt somit (3).

2. (4) impliziert (1):
   Folgt aus den gleichen Überlegungen, welche in Bemerkung 2.) vom Fall $(k1, k2, 0)$ und $(0, k_2, k_3)$ formuliert worden.

(k_1, k_2, k_3) **und** $(k_1', k_2', 0)$



Satz: Seien $k_1, k_2, k_3, k_1', k_2' \in \mathbb{N}_{\geq 2}$ (mit $k_2 \geq k_3$). Dann ist

(1) Sei $k_1' + k_2' - 1 > k_1 + k_2 - 1$ oder $k_1' > k_1 + k_3 - 1$.
Dann existieren $\sigma^0 \in Q$ und $(\sigma_1^0, \sigma_2^0) \in \mathbb{O}(\sigma^0)$, sodass:

(i) $(\sigma_1^0, \sigma_2^0) \notin (k_1, k_2, k_3)(\sigma^0)$ und

(ii) $(\sigma_1^0, \sigma_2^0) \in (k_1', k_2', 0)(\sigma^0)$ gilt.

(2) Sei $k_1' + k_2' - 1 \leq k_1 + k_2 - 1$ und $k_1' \leq k_1 + k_3 - 1$.
Dann ist für ein beliebiges $\sigma \in Q$

$$(k_1, k_2, k_3)(\sigma) \supseteq (k_1', k_2', 0)(\sigma).$$

(3) Sei $k_1' < k_1 + k_3 - 1$ oder $k_1' + k_2' - 1 < \sum_{j=1}^{3} k_j - 2$.
Dann existieren $\sigma^1 \in Q$ und $(\sigma_1^1, \sigma_2^1) \in \mathbb{O}(\sigma^1)$, sodass:

(i) $(\sigma_1^1, \sigma_2^1) \in (k_1, k_2, k_3)(\sigma^1)$ und

(ii) $(\sigma_1^1, \sigma_2^1) \notin (k_1', k_2', 0)(\sigma^1)$ gilt.

(4) Sei $k_1' \geq k_1 + k_3 - 1$ und $k_1' + k_2' - 1 \geq \sum_{j=1}^{3} k_j - 2$.
Dann ist für ein beliebiges $\sigma \in Q$

$$(k_1, k_2, k_3)(\sigma) \subseteq (k_1', k_2', 0)(\sigma).$$

Beweis: (1) Sei $k_1' + k_2' - 1 > k_1 + k_2 - 1$. Dann gilt

$$GP_2((k_1, k_2, k_3)) = k_1 + k_2 - 1 < k_1' + k_2' - 1 = GP_2((k_1', k_2', 0)).$$

Sei $k_1' > k_1 + k_3 - 1$. Dann gilt

$$GP_3((k_1, k_2, k_3)) = k_1 + k_3 - 1 < k_1' = GP_3((k_1', k_2', 0)).$$

Jeweils folgt die Behauptung mit dem Größter-Puffer-Lemma.

(2) Betrachte $\sigma \in Q$, einen zugehörigen, in $(k_1', k_2', 0)$ erzeugten Output

$$(\sigma_1^{(k_1',k_2',0)}, \sigma_2^{(k_1',k_2',0)}) \in (k_1', k_2', 0)(\sigma)$$

und sei $\delta^{(k_1',k_2',0)}$ eine zugehörige Bearbeitungsreihenfolge.

Dieser Output kann in (k_1, k_2, k_3) erzeugt werden:
Nach Split-Lemma ist der Split eindeutig festgelegt. Betrachte $P_1^{(k_1,k_2,k_3)}$ in einem Moment t, in welchem er genau k_1 Jobs geladen hat:

1. *Im Puffer befindet sich ein Job aus $\sigma_1^{(k'1,k_2',0)}$ und $P_2^{(k_1,k_2,k_3)}$ hat noch mehr als einen Slot frei.*
   Lasse diesen Job den Puffer passieren und lade ihn in $P_2^{(k_1,k_2,k_3)}$.
2. *Im Puffer befindet sich ein Job aus $\sigma_2^{(k'1,k_2',0)}$ und $P_3^{(k_1,k_2,k_3)}$ hat noch mehr als einen Slot frei.*
   Lasse diesen Job den Puffer passieren und lade ihn in $P_3^{(k_1,k_2,k_3)}$.
3. *Unter den geladenen Jobs befinden sich nur Aufträge aus $\sigma_2^{(k_1',k_2',0)}$, $P_3^{(k_1,k_2,k_3)}$ hat genau $k_3 - 1$ Jobs geladen und $P_1^{(k_1,k_2,k_3)}$, $P_2^{(k_1,k_2,k_3)}$ und $P_3^{(k_1,k_2,k_3)}$ haben gemeinsam weniger als $k_1' + k_2' - 1$ Jobs geladen.*

   Wegen $k_1 + k_3 - 1 \geq k_1'$ müssen sich weniger als $k_2' - (k_1 + k_3 - k_1') < k_2'$ Jobs in $P_2^{(k_1,k_2,0)}$ befinden (#).
   Bezeichne L_3 die Menge der Jobs in den drei Puffern. Betrachte die Aufträge aus L_3 in $(k_1', k_2', 0)$ in einem Moment t' mit $\gamma^{(k_1',k_2',0)}(t') = \gamma^{(k_1,k_2,k_3)}(t)$ ($*$):

   i) $P_1^{(k_1',k_2',0)}$ hat ausschließlich Jobs aus L_3 geladen:
   Wegen (#) befinden sich weniger als k_2' Jobs aus $\sigma_1^{(k_1',k_2',0)}$ in L_3. Es kann davon ausgegangen werden, dass sich diese Jobs in $P_2^{(k_1',k_2',0)}$ befinden, da ansonsten unnötig freie Pufferplätze ungenutzt bleiben würden.

   ii) $P_1^{(k_1',k_2',0)}$ verfügt über keinen freien Slot mehr:
   Es befinden sich $k_1 + k_3 - 1 > k_1'$ Jobs aus $\sigma_2^{(0,k_2',k_3')}$ in L_3.

   iii) $P_2^{(k_1',k_2',0)}$ hat mindestens einen freien Platz.
   Wegen (#) kann $P_2^{(k_1',k_2',0)}$ nicht ausschließlich mit Jobs aus L_3 befüllt sein. Ein Job $j \notin L_3$ kann erst zu einem späteren Zeitpunkt

geladen werden, da dieser größer (bezüglich seiner Nummerierung) sein muss, als die Jobs in L_3 und $P_1^{(k_1',k_2',0)}$ somit früher einen Job an Maschine 2 ausgeben muss.

Folglich muss $nrj(\delta^{(k_1',k_2',0)}, \gamma^{(k_1,k_2,k_3)}(t)) = nrj(\delta^{(k_1',k_2',0)}, \gamma^{(0,k_2',k_3')}(t')) \in L_3$ sein. Gib diesen Job als nächstes an Maschine 2 aus, was nach Auswurf-Lemma möglich ist.

4. $P_1^{(k_1,k_2,0)}$, $P_2^{(k_1,k_2,0)}$ *und* $P_3^{(k_1,k_2,0)}$ *haben zusammen mindestens* $k_1' + k_2' - 1 = PK((k_1', k_2', 0))$ *Aufträge geladen.*
Wegen $k_1 + k_2 - 1 \geq k_1' + k_2' - 1$ haben $P_1^{(k_1,k_2,k_3)}$, $P_2^{(k_1,k_2,k_3)}$ und $P_3^{(k_1,k_2,k_3)}$ gemeinsam mehr als $k_1' + k_2' - 1$ Jobs geladen. Die Menge dieser Jobs werde mit $J(\sigma^L)$ bezeichnet.
Betrachte diese Jobs in $(k_1', k_2', 0)$ in einem Moment, in dem $\delta^{(k_1',k_2',0)} = \delta^{(k_1,k_2,k_3)}$ gilt $(*)$. Nach Übergangs-Lemma muss $nrj((\sigma_1^{(k_1',k_2',0)}, \sigma_2^{(k_1',k_2',0)})) \in J(\sigma^L)$ gelten, wobei dieser nach Auswurf-Lemma auch in (k_1, k_2, k_3) an Maschine 1 bzw. 2 entsandt werden kann.

Zusammenfassend kann also immer $nrj((\sigma_1^{(k_1',k_2',0)}, \sigma_2^{(k_1',k_2',0)}))$ an Maschine 1 bzw. 2 ausgegeben werden. Durch das γ-Lemma ist die Forderung nach $(*)$ somit also gerechtfertigt. Zusammen mit dem Output-Lemma folgt abschließend $(\sigma_1^{(k_1',k_2',0)}, \sigma_2^{(k_1',k_2',0)}) \in (k_1, k_2, k_3)(\sigma)$.

(3) Sei zunächst $k_1 + k_3 - 1 > k_1'$. Dann folgt wieder mit dem Größter-Puffer-Lemma und

$$GP_3((k_1, k_2, k_3)) = k_1 + k_3 - 1 < k_1' = GP_3((k_1', k_2', 0))$$

die Behauptung.

Sei nun $\sum_{j=1}^{3} k_j - 2 > k_1' + k_2' - 1$. Betrachte

$$\sigma^1 := (1, 2, \ldots, w) \in Q \text{ mit } w := \sum_{j=1}^{3} k_j - 1$$

und den zugehörigen Output

$$(\sigma_1^1, \sigma_2^1) = ((w-1, \ldots, k_2 + k_3 - 1, k_2 - 1, \ldots, 1), (w, k_2 + k_3 - 2, \ldots, k_2)).$$

zu (i): *Angenommen durch* $(k_1', k_2', 0)$ *könnte dieser Output generiert werden*:
In $P_2^{(k_1',k_2',0)}$ können zunächst maximal $k_2' - 1$ Jobs geladen werden, denn ein

Platz muss freibleiben, damit später der Job $w-1$ als erstes ausgegeben werden kann. Danach lädt der Puffer $P_1^{(k_1',k_2',0)}$ weitere k_1' Jobs, wovon einer wieder ausgegeben und an Maschine 2 bzw. $P_2^{(k_1',k_2',0)}$ entsandt werden muss. $P_2^{(k_1',k_2',0)}$ würde über genau einen freien Slot verfügen und müsste folglich auch einen seiner Jobs anschließend an Maschine 1 ausgeben.
Zusammenfassend muss ein Job j mit $j \leq k_1' + k_2' - 1$ als erstes an eine der Maschinen ausgegeben werden. Da $k_1' + k_2' - 1 < w - 1$ gilt, ist $j \neq 1^{st}(\sigma_1^1)$ und $j \neq 1^{st}(\sigma_2^1)$. Widerspruch! (σ_1^1, σ_2^1) kann nicht erzeugt werden.

zu (ii): *In (k_1, k_2, k_3) ist der Output erzeugbar*:
Lasse die ersten $k_2 - 1$ Jobs den ersten Puffer passieren und lade sie in $P_2^{(k_1,k_2,k_3)}$. Analog werden die Jobs k_2 bis $k_2 + k_3 - 2$ in $P_3^{(k_1,k_2,k_3)}$ geladen. Anschließend nimmt $P_1^{(k_1,k_2,k_3)}$ die Jobs $k_2 + k_3 - 2$ bis $w - 1$ auf und kann $w - 1$ nach Split-Lemma an Maschine 1 ausgeben. Abschließend lädt $P_1^{(k_1,k_2,k_3)}$ den Job w. Nun befinden sich alle verbleibenden Jobs in den drei Puffern des Settings und können jeweils nach Auswurf-Lemma in beliebiger Reihenfolge ausgegeben werden. Wählt man die Reihenfolgen aus σ_1^1 bzw. σ_2^1, so entsteht (σ_1^1, σ_2^1).

(4) Betrachte $\sigma \in Q$, einen zugehörigen, in (k_1, k_2, k_3) erzeugten Output

$$(\sigma_1^{(k_1,k_2,k_3)}, \sigma_2^{(k_1,k_2,k_3)}) \in (k_1, k_2, k_3)(\sigma)$$

und eine zugehörige Bearbeitungsreihenfolge $\delta^{(k_1,k_2,k_3)}$.

Dieser Output kann auch durch $(k_1', k_2', 0)$ erzeugt werden:
Der Split ist nach Split-Lemma eindeutig festgelegt. Betrachte $P_1^{(k_1',k_2',0)}$ zu einem Zeitpunkt t', in welchem er gemau k_1' Jobs geladen hat:

1. *Im Puffer befindet sich ein Job aus $\sigma_1^{(k_1,k_2,k_3)}$ und $P_2^{(k_1',k_2',0)}$ hat weniger als $k_2' - 1$ Aufträge geladen.*
   Lasse diesen Job den Puffer passieren und lade ihn in $P_2^{(k_1',k_2',0)}$.
2. *Im Puffer sind nur Jobs aus $\sigma_2^{(k_1,k_2,k_3)}$ und $P_1^{(k_1',k_2',0)}$ und $P_2^{(k_1',k_2',0)}$ haben zusammen weniger als $\sum_{j=1}^{3} k_j - 2$ Jobs geladen.*

   Wegen $k_1' \geq k_1 + k_3 - 1$ müssen sich weniger als $k_2 - (k_1' - (k_1 + k_3 - 1)) - 1 < k_2$ Jobs in $P_2^{(k_1',k_2',0)}$ befinden (#).
   Bezeichne L_3 die Menge der Jobs in den beiden Puffern. Betrachte die Aufträge aus L_3 in (k_1, k_2, k_3) in einem Moment t mit $\gamma^{(k_1,k_2,k_3)}(t) = \gamma^{(k_1',k_2',0)}(t')$ (∗):

i) $P_2^{(k_1,k_2,k_3)}$ hat weniger als k_2 Jobs aus L_3 geladen.
Wegen (#) sind weniger als k_2 Jobs aus $\sigma_1^{(k_1,k_2,k_3)}$ in L_3. Es kann davon ausgegangen werden, dass sich diese Jobs in $P_2^{(k_1,k_2,k_3)}$ befinden, da ansonsten freie Pufferplätze ungenutzt bleiben würden.

ii) Es muss als nächstes ein Job aus L_3 an Maschine 2 ausgegeben werden.
Damit ein Job $j \notin L_3$ in einen der Puffer geladen werden kann, müssen sich alle Jobs aus L_3 in den drei Puffern befinden. Wegen i) sind mindestens k_1+k_3-1 Jobs aus $\sigma^{(k_1,k_2,k_3)}$ in $P_1^{(k_1,k_2,k_3)}$ und $P_3^{(k_1,k_2,k_3)}$. Um solch ein j in $P_1^{(k_1,k_2,k_3)}$ zu laden, muss vorher $P_3^{(k_1,k_2,k_3)}$ k_3 Jobs geladen und einen Job aus L_3 ausgegeben haben.

Folglich muss $nrj(\delta^{(k_1,k_2,k_3)}, \gamma^{(k_1,k_2,k_3)}(t)) = nrj(\delta^{(k_1,k_2,k_3)}, \gamma^{(k_1',k_2')}(t')) \in L_3$ sein. Nach Auswurf-Lemma kann er als nächstes an Maschine 2 gesandt werden.

3. $P_1^{(k_1',k_2',0)}$ *und* $P_2^{(k_1',k_2',0)}$ *haben zusammen mehr als* $\sum_{j=1}^{3} k_j - 2 = PK((k_1,k_2,k_3))$ *Jobs geladen.*
Bezeichne die Menge dieser geladenen Jobs mit $\hat{L}$. Betrachte die Aufträge aus $\hat{L}$ in (k_1,k_2,k_3) in einem Moment t mit $\gamma^{(k_1,k_2,k_3)}(t) = \gamma^{(k_1',k_2',0)}(t')$ $(*)$. Nach Übergangs-Lemma ist $nrj(\delta^{(k_1,k_2,k_3)}, \gamma^{(k_1,k_2,k_3)}(t)) = nrj(\delta^{(k_1,k_2,k_3)}, \gamma^{(k_1',k_2')}(t')) \in L_3$. Gib diesen Job als nächstes an Maschine 1 bzw. 2 aus, was nach Auswurf-Lemma möglich ist.

Habe $P_1^{(k_1',k_2',0)}$ nun zum Zeitpunkt $\bar{t}$ weniger als k_1' Jobs geladen. Es muss sich dabei um die letzten Jobs aus σ handeln, wodurch sich auch $nrj(\delta^{(k_1,k_2,k_3)}, \gamma^{(k_1',k_2',0)}(\bar{t}))$ in einem der Puffer befinden muss.
Wenn auch weiterhin 1. von oben respektiert wird, kann davon ausgegangen werden, dass $P_2^{(k_1',k_2',0)}$ über mindestens einen freien Slot verfügt. Nach Auswurf-Lemma folgt, dass $nrj(\delta^{(k_1,k_2,k_3)}, \gamma^{(k_1',k_2',0)}(\bar{t}))$ als nächstes ausgegeben werden kann.

Zusammenfassend folgt durch das γ-Lemma, dass die Forderung nach $(*)$ legitim ist. Mit dem Output-Lemma folgt $(\sigma_1^{(0,k_2',k_3')}, \sigma_2^{(0,k_2',k_3')}) \in (k_1,k_2,k_3)(\sigma)$. □

Bemerkungen:

1. (2) *impliziert* (3):

Angenommen es gelte (2) und nicht (3). Dann gilt

$$\begin{aligned} k_1' + k_2' - 1 &\leq k_1 + k_2 - 1 \\ k_1' &\leq k_1 + k_3 - 1 \\ k_1' &\geq k_1 + k_3 - 1 \\ k_1' + k_2' - 1 &\geq \sum_{j=1}^{3} k_j - 2. \end{aligned}$$

Aus der ersten und vierten Ungleichung würde $k_1 + k_2 - 1 \geq k_1' + k_2' - 1 \geq \sum_{j=1}^{3} k_j - 2$ folgen. Da $k_3 \in \mathbb{N}_{\geq 2}$ und somit $k_3 - 1 > 0$ ist, liefert dies einen Widerspruch. Aus (2) folgt demnach (3).

2. (4) impliziert (1):
   Folgt aus den gleichen Überlegungen, welche in Bemerkung 2.) vom Fall $(k1, k2, 0)$ und $(0, k_2, k_3)$ formuliert worden.

2.4 Fazit

Betrachte noch einmal die Vergleiche der Settings aus Kapitel 2.3 bezüglich $(k_1, 0, 0)$, allerdings unter der Voraussetzung, dass die jeweilige Summe der Pufferplätze pro Setting übereinstimmt:

1. $(k, 0, 0)$ und $(0, k, 0)$:
   Nach dem Satz zu $(k_1, 0, 0)$ und $(0, k_2', 0)$ gilt für ein beliebiges $\sigma \in Q$

   $$(k, 0, 0)(\sigma) \supseteq (0, k, 0)(\sigma).$$

2. $(k, 0, 0)$ und $(0, k_2, k_3)$ mit $k_2 + k_3 = k$:
   Es gilt somit $k > k_2 + k_3 - 1$ und nach dem Satz zu $(k_1, 0, 0)$ und $(0, k_2', k_3')$ folgt für ein beliebiges $\sigma \in Q$

   $$(k, 0, 0)(\sigma) \supseteq (0, k_2, k_3)(\sigma).$$

3. $(k, 0, 0)$ und $(k_1, k_2, 0)$ mit $k_1 + k_2 = k$:
   Es gilt somit $k > k_1 + k_2 - 1$ und nach dem Satz zu $(k_1, 0, 0)$ und $(k_1', k_2', 0)$ folgt für ein beliebiges $\sigma \in Q$

   $$(k, 0, 0)(\sigma) \supseteq (k_1, k_2, 0)(\sigma).$$

4. $(k, 0, 0)$ und (k_1, k_2, k_3) mit $k_1 + k_2 + k_3 = k$:
   Es gilt somit $k > k_1 + k_2 + k_3 - 2$ und nach dem Satz zu (k_1, k_2, k_3) und $(k_1', 0, 0)$ folgt für ein beliebiges $\sigma \in Q$

   $$(k, 0, 0)(\sigma) \supseteq (k_1, k_2, k_3)(\sigma).$$

Unter dieser Bedingung ist das Setting $(k_1, 0, 0)$ demnach am besten geeignet, um man eine möglichst große Outputmenge zu erzeugen.

Falls ein Unternehmen einen Puffer errichten bzw. vergrößern will und geht man davon aus, dass eine Puffereinheit konstante Kosten besitzt, so empfiehlt es sich folglich den Puffer P_1 zu vergrößern, um eine größere Outputmenge zu erzeugen. An dieser Position kann der Puffer gleichmäßigen Einfluss auf die Queues von Maschine 1 und Maschine 2 nehmen.

Es ist allerdings zu beachten, dass eine größere Outputmenge nicht zwangsläufig bedeutet, dass sich die Lösungqualität in Bezug auf die betrachtete Kos-

tenfunktion (7) verbessert. Eine größere Outputmenge bietet den in Kapitel 3 betrachteten Metaheuristiken zusätzliche Möglichkeiten sich falsch zu entscheiden, das heißt ungünstige Vertauschungen zu wählen, wodurch die Lösungsqualität insgesamt sogar abnehmen könnte.

Weiterhin wurde der Einfluss der Pufferkosten vernachlässigt. Betrachte das Beispiel aus Tabelle 3. Mithilfe der Programme "Optimum_400" und "Optimum_040", welche in Kapitel 4.3.5 vorgestellt werden, wurde gezeigt, dass die minimalen Kosten dieser Queue in $(0, 4, 0)$ 2 und in $(4, 0, 0)$ 3 betragen. Jeweils kann er Output $((4, 1, 3), (2, 5))$ generiert werden, welcher keine Gesamtverspätung verursacht. Allerdings kann dieser Output in $(0, 4, 0)$ mit weniger Vertauschungen erstellt werden, da erst der Split durchgeführt wird und somit eine kleinere Queue in den Puffer geladen wird.

Tabelle 3: Beispiel einer Queue deren optimaler zugehöriger Output in $(0, 4, 0)$ geringere Kosten als in $(4, 0, 0)$ verursacht.

	j_1	j_2	j_3	j_4	j_5
Fertigungsdauer	33	57	22	64	45
Deadline	132	96	186	76	102

Ohne zusätzliche Informationen über die Struktur der Aufträge lässt sich somit kaum eine Aussage darüber treffen, welche Settings besser geeignet sind. Solche Informationen könnten sein:

1. *Existiert für ein $\sigma \in Q$ ein $l \in \mathbb{N}$, sodass eine Vertauschung zweier Jobs $x, y \in J(\sigma)$, mit $|Pos(x, Q) - Pos(y, Q)| > l$ nicht nötig ist, um einen optimalen Output zu erzeugen?*
   In diesem Fall müssten Puffer mit $GP_i((k_1, k_2, k_3)) = l$ nicht weiter ausgebaut werden, wodurch Kosten eingespart werden können.

2. *Muss ein Teil des Outputs eventuell kaum bzw. nicht sortiert werden, da die eingehende Queue σ schon vorsortiert ist?*
   In diesem Fall kann es sinnvoller sein, den Puffer P_2 anstelle von P_1 auszubauen. Nach dem Satz zu $(k_1, 0, 0)$ und $(k_1', k_2', 0)$ können bei gleicher Puffergröße in $(k_1, 0, 0)$ die gleichen Vertauschungen wie in $(0, k_2, 0)$ ausgeführt werden, dabei sind allerdings meist mehr Vertauschungen nötig. Falls Pufferkosten mit einbezogen werden, kann folglich ein zu Tabelle 3 analoger Fall entstehen, sodass ein Output in $(0, k_2, 0)$ kostengünstiger erstellt werden kann.

3 Ameisenalgorithmus und Modifikationen

Bereits für eine kleine Anzahl an Aufträgen können die in Kapitel 2 betrachteten Outputmengen sehr groß werden. Betrachte ein $\sigma \in Q$ in einem Setting $(k_1, 0, 0)$ mit $k_1 \in \mathbb{N}$. Bezüglich der Anzahl der Reihenfolgen, welche $P_1^{(k_1,0,0)}$ aus σ erzeugen kann, ergibt sich:

a) $k_1 \geq |\sigma|$:
Da alle Jobs gleichzeitig geladen werden können, sind alle Umordnungen realisierbar. Das heißt es existieren genau $|\sigma|!$ verschiedene Reihenfolgen, welche durch $P_1^{(k_1,0,0)}$ aus σ erzeugt werden können.

b) $k_1 < |\sigma|$:
Die Queue muss den Puffer sukzessiv durchlaufen. In jedem Schritt können maximal k_1 Jobs in den Puffer geladen werden, wovon anschließend genau einer wieder ausgegeben wird. Nachdem die ersten $|\sigma| - k_1$ Aufträge ausgegeben worden, befinden sich alle restlichen Jobs der Queue im Puffer. Das heißt es folgen $k_1!$ Möglichkeiten diese Jobs anzuordnen. Zusammenfassend existieren

$$\binom{k_1}{1}^{|\sigma|-k_1} \cdot k_1! = k_1^{|\sigma|-k_1} \cdot k_1!$$

verschiedene Reihenfolgen, welche durch $P_1^{(k_1,0,0)}$ erzeugt werden können.

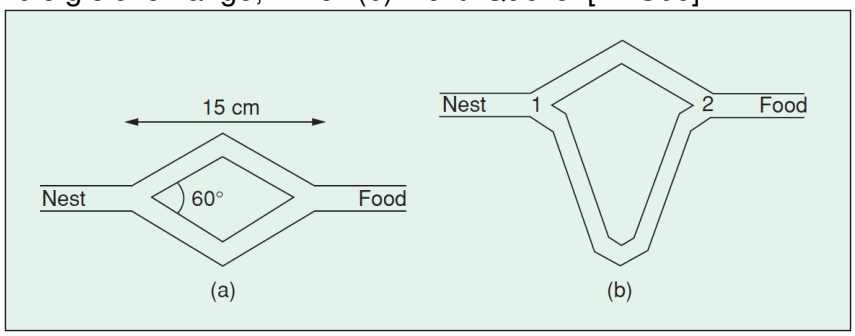
Hinzukommen $2^{|\sigma|}$ mögliche Aufteilungen im Split, da jeder Job entweder zu Maschine 1 oder zu Maschine 2 läuft.
Für $|\sigma| = 30$ und $k_1 = 4$ ergeben sich bereits $4^{26} \cdot 4! \cdot 2^{30} > 10^{26}$ verschiedene Elemente in $(4, 0, 0)(\sigma)$. All diese Lösungen samt Kosten zu bestimmen, stellt einen enormen Rechen- und somit Zeitaufwand dar. Für eine Anwendung in der Realwirtschaft muss daher zur Lösung der Probleme ein Algorithmus verwendet werden, welcher auf geschickte Weise nur einen kleinen Teil aller Lösungen berechnet, aber dennoch sehr gute Ergebnisse liefert. Es empfiehlt sich daher, die Verwendung einer Metaheuristik, welche speziell für die betrachteten Flowshop-Probleme angepasst und erweitert wird.

3.1 Ant Colony Optimization

3.1.1 Zwei-Brücken-Experimente

Neben genetischen Algorithmen oder Tabu-Suche-Verfahren ist ein relativ neuer Ansatz, solche komplexen kombinatorischen Probleme zu lösen, die Verwendung einer Schwarmintelligenz. Einen solchen naturinspirierten Ansatz stellt die Ant Colony Optimization, kurz ACO, dar:
In [DBS06, S. 2] schreiben Dorigo et al., dass Ameisenarten, wie die argentinische Ameise, indirekt durch Stigmergie in Form von Pheromonspuren kommunizieren. Jede Ameise sondert beim Laufen eine gewisse Menge Pheromon ab und jede Ameise neigt dazu, Wegen zu folgen, auf denen eine hohe Pheromonkonzentration vorhanden ist. Weiterhin verdunsten Pheromonspuren nach einer gewissen Zeit. Die Insekten verändern somit ihre lokale Umgebung, um Informationen mit ihren Artgenossen auszutauschen.
Das Verhalten bei der Futtersuche wurde in Zwei-Brücken-Experimenten untersucht. In diesen Experimenten wurde der Bau einer Kolonie argentinischer Ameisen über zwei Brücken mit einer Futterquelle verbunden und anschließend beobachtet, wie sich die Insekten zwischen Bau und Nahrung orientierten. Der Versuchsaufbau ist in Abbildung 4 dargestellt.
Im ersten Fall wurden beide Brücken exakt gleich lang gewählt. Zu Beginn entschieden sich die Ameisen jeweils zufällig und gleichmäßig für einen der beiden Wege, doch nach einiger Zeit führt eine Fluktuation dazu, dass mehr Ameisen sich für den gleichen Weg entscheiden, wodurch die Pheromonkonzentration auf dieser Brücke höher ausfällt. Die stärkere Konzentration führt wiederum dazu, dass mehr Ameisen diesen Weg wählen, wodurch die Pheromonkonzentration weiter ansteigt. Es setzt ein Konvergenzprozess ein, welcher schließlich dazu führt, dass die gesamte Kolonie die gleiche Brücke wählt.
Dieses auf positiver Rückkopplung basierende Gruppenverhalten kann von Ameisen genutzt werden, um jeweils kürzeste Pfade zwischen einer Futterquelle und ihrem Nest zu finden (vergleiche [DBS06, S. 2]).
Um dies zu zeigen, wurde in einem zweiten Experiment eine der beiden Brücken signifikant länger gestaltet als die andere. Nach Dorigo et al. [DBS06, S. 2] stieg die Pheromonkonzentration auf dem kürzeren Weg schnell an, da die Ameisen dort früher zu ihrem Nest zurückkehren und somit früher Pheromon platzieren konnten. Infolgedessen stieg die Wahrscheinlichkeit, dass zukünftige Ameisen wiederum die kürzere Brücke wählen. Dies führt zu einem analogen Konvergenzprozess wie im ersten Fall, allerdings zugunsten der kürzeren Brücke.

Abbildung 4: Darstellung des Aufbaus der Experimente. In Teil (a) haben die Brücken die gleiche Länge; in Teil (b) nicht. Quelle: [DBS06].



3.1.2 Algorithmische Umsetzung

Nach Dorigo et al. [DBS06, S. 2 f.] dienten diese Untersuchungen bezüglich der Futtersuche argentinischer Ameisen als Hauptinspirationsquelle bei der Entwicklung der Ant Colony Optimization. Dabei kreieren eine bestimmte Anzahl künstlicher Ameisen Lösungen für ein Optimierungsproblem und tauschen anschließend Informationen über die Qualität der gefundenen Lösungen aus. Diese Informationen gehen anschließend in Lösungen folgender Generationen künstlicher Ameisen ein.

Ant Colony Optimization wird vorrangig bei kombinatorischen Optimierungsproblemen angewandt, in welchen sich die Konstruktion einer zulässigen Lösung als Pfad in einem Entscheidungsbaum darstellen lässt (vergleiche [GM02, S. 1]). Der Standard-Entwurf eines Ameisenalgorithmus wird in [GM02] wie folgt beschrieben:
Es werden zunächst $m \in \mathbb{N}$ zufällige, zulässige Lösungen des Problems erzeugt und anhand einer Kostenfunktion bewertet. Diese Lösungen werden im weiteren auch als künstliche Ameisen bezeichnet. Im ersten Schritt sind dabei alle Entscheidungen einer Entscheidungsebene, welche während der Lösungskonstruktionen zulässig sind, gleichwahrscheinlich. Eine Entscheidung meint eine Kante (i, j) zwischen den Zuständen oder Knoten i und j. Die relativen Wahrscheinlichkeiten der Entscheidungen, werden durch eine Pheromonmatrix $\mathbb{P} = (\tau_{i,j})$, mit $\tau_{i,i} = 0$, repräsentiert. Zu Beginn hat diese Matrix somit für beliebige $i \neq j$ feste Einträge $\tau_{i,j} = \tau^0 > 0$.
Die Wahrscheinlichkeit einer Entscheidung (i, j) ist gegeben durch

$$p_{i,j} = \frac{\tau_{i,j}}{\sum_{h \in S} \tau_{i,h}}, \tag{20}$$

wobei S die Menge aller Zustände ist, welche von i aus erreichbar, das heißt

durch eine Kante verbunden sind.
Anschließend wird unter den m künstlichen Ameisen diejenige ausgewählt, welche die beste Lösung ihrer Generation gefunden hat. Die Einträge in $\mathbb{P}$, welche die Entscheidungen dieser besten Ameise repräsentieren, werden nun um einen Wert $u \in \mathbb{R}$ erhöht. Weiterhin findet eine Verdunstung statt, das heißt alle Einträge von $\mathbb{P}$ werden um einen Verdunstungs-Faktor $\rho \in (0, 1)$ vermindert:

$$\tau_{i,j} \mapsto \begin{cases} \rho \cdot \tau_{i,j} + u, & \text{falls } (i,j) \text{ wurde in bester Route gewählt} \\ \rho \cdot \tau_{i,j}, & \text{sonst.} \end{cases} \tag{21}$$

Dadurch steigt die Wahrscheinlichkeit, dass in zukünftigen Lösungen Entscheidungen getroffen werden, welche bereits Teil von guten Lösungen sind. Durch den Verdunstungsfaktor sinkt weiterhin die Auswahlwahrscheinlichkeit einer schlechten Lösung. Die Konstruktion von zulässigen Lösungen für das betrachtete Optimierungsproblem ist somit immer durch vorherige, gute Lösungen beeinflusst.
Sobald eine vorher festgelegte Abbruchbedingung erfüllt ist, beispielsweise dass alle Ameisen über mehrere Generationen hinweg die gleiche Lösung wählen, bricht die Berechnung ab. In [DBS06, S. 6] wird angemerkt, dass nicht immer klar ist, ob die Lösungen, welche durch den Ameisenalgorithmus gefunden werden, gegen eine Optimallösung des Problems konvergieren. In den meisten Anwendungsfällen werden allerdings ohnehin NP-schwere Probleme betrachtet. Bei diesen ist in der Regel nicht davon auszugehen, dass eine optimale Lösung gefunden wird, da die exakten Lösungs-Algorithmen exponentielle Laufzeiten besitzen. Auch wenn die beste Lösung, welche durch ACO gefunden wird, nicht zwangsweise optimal ist, werden nach [DBS06, S. 8] hierdurch in der Praxis zumeist sehr gute Ergebnisse erzielt.
Das Verfahren kann weiterhin in Kombination mit anderen Heuristiken bzw. Metaheuristiken verwendet werden. Beispielsweise wird in [Blu05] eine Variante des ACO mit Beam-Suche und in [HL08] eine Kombination aus ACO und Tabu-Suche vorgestellt. In beiden Variationen konnten die Resultate des ACO somit verbessert werden.
Eine weitere Form einer Schwarmintelligenz stellt Bee Colony Optimization dar, welche dem Verhalten von Honigbienen nachempfunden ist. Wie etwa [CSLG06] zeigt, kann auch dieses Verfahren erfolgreich auf Scheduling-Probleme angewandt werden.

3.2 Populationsbasierter Ameisenalgorithmus

3.2.1 Unterschiede zum Standard-Ansatz

Der erste Ameisenalgorithmus wurde 1991 durch M. Dorigo et al. vorgeschlagen und seitdem in einer Vielzahl von Abwandlungen verwendet und weiter verbessert ([DBS06, S. 5]). Eine dieser Variationen ist der sogenannte populationsbasierte Ameisenalgorithmus, kurz pACO, welcher 2002 durch Guntsch und Middendorf in [GM02] vorgestellt wurde.
Im Unterschied zum Standard-Ansatz eines Ameisenalgorithmus werden beim pACO gute Lösungen nicht mehr indirekt über die Pheromonmatrix gespeichert, sondern direkt in einer Lösungs-Population L. Diese Population stellt somit ein Fenster der letzten oder bisher besten k Generationen dar.
Explizit wird auf die Verdunstung des Pheromons verzichtet und die jeweils beste Ameise ihrer Generation wird einer Lösungs-Population L hinzugefügt, welche die Größe $k \in \mathbb{N}$ besitzt.
Nach k Generationen befinden sich demnach genau k Lösungen in L und ab Generation $k+1$ wird zunächst eine Lösung wieder aus der Population gelöscht und die zugehörigen Pheromonwerte wieder aus der Matrix $\mathbb{P}$ entfernt. Dabei ist die gleiche Pheromonmenge aus $\mathbb{P}$ zu entfernen, welche hinzugefügt wurde als die jeweilige Lösung in L aufgenommen wurde. Üblicherweise wird die älteste oder die schlechteste Lösung aus der Population gelöscht.
Als Abbruchkriterium wird meist eine vorher festgelegte Anzahl an Iterationen gewählt, da durch die beschränkte Pheromonmenge kein stationärer Zustand wie im Standard-Ansatz erzeugt wird.

3.2.2 Vorteile

Durch diese Modifikation gilt für eine beliebige Entscheidung (i, j) und unter der Annahme, dass u ein fester Wert ist, dass $\tau_{i,j}$ beschränkt ist:

$$\tau^0 \leq \tau_{i,j} \leq \tau^0 + k \cdot u.$$

Diese Beschränkung bringt den Vorteil mit sich, dass der Einfluss der ersten Lösungen stark reduziert wird. Im Standard-Ansatz ist es möglich, dass falls zu Beginn eine sehr gute Lösung ignoriert wurde, diese später nicht mehr gewählt wird, da der Algorithmus in Richtung einer anderen Lösung konvergiert.
Durch die geringere Menge an Pheromon innerhalb der Matrix ist es außerdem unwahrscheinlicher, dass mehrere Ameisen innerhalb einer Generation die gleichen Lösungen wählen. Es findet folglich, ohne den Rechenaufwand zu erhö-

hen, eine stärkere Lösungsexploration statt, da eine größere Anzahl verschiedener Lösungen getestet wird. Insgesamt wird somit die Wahrscheinlichkeit verringert, dass der Algorithmus in lokalen Minima festsitzt und keine bessere Lösung mehr findet. Zudem reduziert sich die Laufzeitkomplexität im Vergleich zum Standard-Ansatz, da pro Iteration nicht mehr die gesamte Pheromonmatrix aktualisiert, sondern lediglich eine Pheromonspur ersetzt werden muss.

In [GM02] vergleichen Guntsch und Middendorf den pACO mit dem Standard-Ansatz an einer Reihe von Instanzen des „Traveling Salesman Problems“ und des „Quadratic Assignment Problems“. Es stellt sich heraus, dass der populationsbasierte Algorithmus in fast allen Fällen die besten Resultate liefert und mindestens konkurrenzfähig zum Standard-Ansatz ist.

Zusammenfassend stellt der populationsbasierte Ameisenalgorithmus eine sehr erfolgversprechende Variante zum Standard-Ansatz dar, welche vor allem für komplexe, kombinatorische Optimierungprobleme, wie die hier betrachteten Flowshop-Probleme, geeignet ist.

3.3 Alternative Pheromon-Evaluationsverfahren

Die Bestimmung der Auswahlwahrscheinlichkeit einer Entscheidung (i, j) aus den Werten der Pheromonmatrix wird auch als Pheromon-Evaluationsverfahren bezeichnet. In diesem Kapitel werden zwei alternative Verfahren zu (20) vorgestellt, welche sich vor allem für Scheduling-Probleme, wie die in dieser Arbeit betrachteten Flowshop-Probleme, eignen.

3.3.1 Summationsregel

Betrachte eine Folge von Jobs, welche durch einen Puffer umsortiert wird und das standardmäßige Pheromon-Evaluationsverfahren aus (20), wobei $p_{i,j}$ die Wahrscheinlichkeit sei, dass ein Job j auf Position i getauscht werde. Ein hoher Wert $\tau_{i,j}$ symbolisiert in der Regel, dass es vorteilhaft ist, j auf Position i zu setzen. Nehme nun an, dass dennoch ein anderer Job $h \neq j$ auf i mit niedrigem $\tau_{i,h}$ gesetzt wird. Nach [MM00, S. 4 f.] ist es wahrscheinlich, dass in einer guten Reihenfolge der Job j nicht viel später gesetzt werden sollte. Falls es bisher keine oder nur wenige Ameisen gab, die j auf eine Position $l > i$ umgelegt haben, so ist $\tau_{l,j}$ jedoch klein und es kann geschehen, dass j erst viel später gesetzt wird. Auf diese Weise generiert die Ameise also vermutlich eine nutzlose Lösung, welche nicht zur Verbesserung der Resultate beiträgt.
Um dies zu vermeiden, schlagen Merkle und Middendorf in [MM00, S. 4 f.] vor, dass die Pheromonwerte $\tau_{i,j}$ auch auf Positionen $l > i$ Einfluss haben sollen. Realisiert wird dies durch ihre sogenannte Summationsregel:

$$p_{i,j}^{sum} = \frac{(\sum_{v=1}^{i} \tau_{v,j})}{\sum_{h \in S}(\sum_{v=1}^{i} \tau_{v,h})}. \tag{22}$$

Wenn mit Hilfe dieser Regel darüber entschieden wird, ob ein Job j auf Position i gesetzt wird oder nicht, so werden auch die Pheromonwerte für j und alle vorhergehenden Positionen $v < i$ einbezogen und das oben genannte Problem wird gemindert.

3.3.2 Relative Pheromonregel

Betrachte eine Folge von Jobs, welche durch einen Puffer umsortiert wird und das lokale Pheromon-Evaluationsverfahren aus (20), wobei $p_{i,j}$ die Wahrscheinlichkeit sei, dass ein Job j auf Position i getauscht werde. Merkle und Middendorf beschreiben in [MM01, S. 2] die folgende Beobachtung:
Angenommen es existiert ein Job j, welcher bisher nicht auf einem Platz 1 bis

$i-1$ gesetzt wurde, obwohl ein großer Teil des Pheromons der Spalte j auf die Werte $\tau_{1,j}, \ldots, \tau_{(i-1),j}$ verteilt ist. In diesem Fall ist der Wert $\tau_{k,j}$ für ein $k \geq i$ klein und es ist unwahrscheinlich, dass die Ameise den Auftrag j auf Position k sortieren wird, falls ein anderer, noch verfügbarer Job einen großen Wert in Zeile k hat. Dies ist auch richtig, falls $\tau_{k,j}$ der größte Wert aus $\{\tau_{i,j}, \ldots, \tau_{n,j}\}$ ist. Da die Position von j für das Optimierungsproblem trotzdem sehr relevant sein kann, schlagen Merkle und Middendorf vor, statt der absoluten, lieber die relative Pheromonmenge im Rest der Spalte zu betrachten:

$$p_{i,j}^{rel} = \frac{\frac{\tau_{i,j}}{\sum_{v=i}^{n} \tau_{v,j}}}{\sum_{h \in S}\left(\frac{\tau_{i,h}}{\sum_{v=i}^{n} \tau_{v,h}}\right)}. \tag{23}$$

3.4 Puffer-Heuristiken

Ein Ameisenalgorithmus kann mit einer Heuristik η kombiniert werden. Diese geht als Faktor während der Pheromonevaluation ein und dient dem Algorithmus somit als Entscheidungshilfe. Für (20) ergibt sich

$$p_{i,j} = \frac{\tau_{i,j}^a \cdot \eta_{i,j}^b}{\sum_{h \in S}(\tau_{i,h}^a \cdot \eta_{i,h}^b)}, \tag{24}$$

wobei $a, b \in \mathbb{R}$ jeweils der Wichtung zwischen Pheromon und Heuristik entsprechen. Analog ergeben sich (22) und (23) durch

$$p_{i,j}^{sum} = \frac{(\sum_{v=1}^{i} \tau_{v,j}^a \cdot \eta_{v,j}^b)}{\sum_{h \in S}(\sum_{v=1}^{i} \tau_{v,h}^a \cdot \eta_{v,h}^b)} \text{ und } p_{i,j}^{rel} = \frac{\frac{\tau_{i,j}^a \cdot \eta_{i,j}^b}{\sum_{v=i}^{n}(\tau_{v,j}^a \cdot \eta_{v,j}^b)}}{\sum_{h \in S}(\frac{\tau_{i,h}^a \cdot \eta_{i,h}^b}{\sum_{v=i}^{n}(\tau_{v,h}^a \cdot \eta_{v,h}^b)})}. \tag{25}$$

Eine Heuristik generiert für jede Entscheidung (i, j) einen Wert $\eta_{i,j}$ und fungiert somit neben der Pheromonmatrix als zusätzliches Kriterium dafür, welche Entscheidung mit welcher Auswahlwahrscheinlichkeit bewertet wird. Da sich eine Heuristik nur an deterministischen Werten und nicht an den vorhergehenden Lösungen orientiert, kann somit vermieden werden, dass sinnlose bzw. sehr schlechte Lösungen konstruiert werden. Vor allem zu Beginn des Berechnungsprozesses verfügt ein Ameisenalgorithmus über keine bzw. kaum Informationen aus bisher untersuchten Kombinationen, wodurch es geschehen kann, dass sich unter den m Anfangslösungen keine gute Lösung befindet und somit schlechte Kombinationen Einfluss auf zukünftige Lösungen ausüben.
In der Berechnung der Auswahlwahrscheinlichkeiten gehen die $\eta_{i,j}$ positiv in Informationen aus $\mathbb{P}$ ein. Entscheidungen, welche sowohl einen hohen heuristischen Wert haben, als auch Teil vorhergehender guter Lösungen waren, erhalten somit eine sehr hohe Auswahlwahrscheinlichkeit. Die Bewertung von Entscheidungen, welche einen hohen heuristischen Wert, aber einen geringen Pheromonwert (oder andersherum) aufweisen, hängt hingegen stark von den Wichtungen a und b ab.
Im folgenden Kaptitel 3.4.1 werden zunächst einige eigens entwickelte Heuristiken in Form eines Algorithmus vorgestellt und untersucht. Abschließend wird in Kapitel 3.4.2 eine bereits in der Literatur vorgestellte Heuristik betrachtet.

3.4.1 Kriteriums-Heuristiken

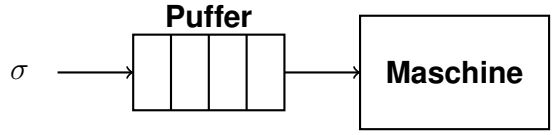
Da alle Aufträge zunächst die erste Fertigungsstufe durchlaufen, kann davon ausgegangen werden, dass die folgenden Informationen über die Jobs der Queue

vorhanden sind, wenn sie einen Puffer durchlaufen:

- Fertigungsdauer
- Deadline
- Länge der Queue
- Position innerhalb der Queue

Die Heuristiken werden im Weiteren als Algorithmen vorgestellt und an einer einstufigen Fertigung (siehe Abbildung 5) getestet. Vor der Maschine befindet sich ein Puffer mit vier Pufferslots, durch welchen die eingehende Queue anhand des Algorithmus umsortiert wird. Pufferkosten werden nicht berücksichtigt. Durch die Vernachlässigung des Ameisenalgorithmus, der Pufferkosten und die Reduktion der Komplexität des Problems, lassen sich leichter Rückschlüsse auf die treibenden Kräfte hinter den Heuristiken führen, da diese weniger Verzerrungen unterliegen. Alle Tests wurden mit Hilfe des Programms "AlgOhneSettings" geschrieben, welches sich im Unterverzeichnis "Puffer-Alg_ohne_ACO_1stufig" auf der CD befindet.

Abbildung 5: Darstellung der einstufigen Fertigung, an welcher die Algorithmen getestet wurden.



Die vorgestellten Algorithmen arbeiten nach folgendem Schema:

1. Es wird eine Kriteriumsfunktion K definiert, welche aus den vorhandenen Informationen eine Kennzahl für ein $\sigma \in Q$ und einen beliebigen Job $j \in J(\sigma)$ berechnet.

2. Anschließend werden die Jobs aufsteigend, anhand dieser Kennzahl sortiert. Sollten für $\sigma \in Q$ und zwei Jobs $j, j' \in J(\sigma)$ die Werte $K(j, \sigma)$, $K(j', \sigma)$ übereinstimmen, so wird keine Vertauschung der beiden Jobs durchgeführt. Auf diese Weise sind die, durch die Algorithmen erzeugten Reihenfolgen eindeutig bestimmt. Falls klar ist, welche Kriteriumsfunktion gemeint ist, bezeichnet $P(\sigma)$ diejenige Queue, welche durch die Umordnungen des Puffers aus σ hervorgeht.

Die spätere Verwendung als Heuristik erfolgt durch das Reziproke dieser Kriteriumsfunktionen:

$$\eta_{i,j} = \frac{1}{K(j,\sigma)}. \tag{26}$$

t-Algorithmus

Für $\sigma \in Q$, $i \in \{1, \ldots, |\sigma|\}$ und $j_i = (t_i, D_i) \in J(\sigma)$ ergibt sich die Kriteriumsfunktion K^t des t-Algorithmus als

$$K^t : J(\sigma) \times Q \to \mathbb{R}, ((t_i, D_i), \sigma) \mapsto t_i. \tag{27}$$

Im t-Algorithmus wird folglich jeder Job nur mit seiner Fertigungsdauer bewertet. Dies vermeidet, dass Jobs mit verhältnismäßig hohen Fertigungszeiten die Maschine blockieren und somit die Verspätung anderer, eventuell sehr kurzer Jobs, unnötig vergrößern.
In Tabelle 4 sieht man, dass dies zu sehr guten Resultaten führen kann. Da die Deadlines aller Jobs von σ^2 recht ähnlich sind, sollte sich eine gute Reihenfolge bezüglich der Gesamtverspätung an den Fertigungszeiten orientieren. Viele andere Algorithmen liefern schlechtere Resultate, obwohl bzw. weil sie die Deadlines einbeziehen.

Tabelle 4: Beispiel 1 des Vergleichs der Puffer-Algorithmen

σ^1	j_1	j_2	j_3	j_4	j_5	j_6	j_7
Fertigungsdauer	13	9	7	11	1	5	3
Deadline	13	14	15	16	17	18	19

Tabelle 5: Resultate der Algorithmen zu Beispiel 1

Algorithmus	**Tardiness**	$P(\sigma^1)$
kein Alg.	128	σ^1
t-Alg.	67	$(j_3, j_5, j_6, j_7, j_2, j_4, j_1)$
D-Alg.	128	σ^1
$(D-t)$-Alg.	136	$(j_1, j_2, j_4, j_3, j_6, j_5, j_7)$
$(\frac{1}{3}, \frac{2}{3})$-Alg.	67	$(j_3, j_5, j_6, j_7, j_2, j_4, j_1)$
$D+t$-Alg.	69	$(j_3, j_5, j_2, j_7, j_6, j_1, j_4)$
$(\frac{2}{3}, \frac{1}{3})$-Alg	88	$(j_2, j_5, j_3, j_1, j_6, j_7, j_4)$
TS-Alg	88	$(j_2, j_5, j_3, j_1, j_6, j_7, j_4)$
$t\uparrow$-Alg.	67	$(j_3, j_5, j_6, j_7, j_2, j_4, j_1)$

Da die Deadline jedoch vollständig vernachlässigt wird, liefert der Algorithmus

vor allem bei Queues, in denen Jobs mit niedriger Fertigungsdauer große Deadlines und Jobs hoher Fertigungsdauer kleine Deadlines haben, schlechte Resultate. Betrachte dazu Tabelle 6: Der t-Algorithmus versucht die Jobs j_5, j_6, j_7 möglichst früh zu setzen, da diese eine geringe Fertigungszeit haben. Es gilt allerdings

$$D(t_5) = D(t_6) = D(t_7) = 150 > 115 = \sum_{i=1}^{7} t(j_i).$$

Unabhängig von ihrer Position in $P(\sigma^1)$ können diese Jobs ihre Deadline somit nicht verletzen. Indem j_5, j_6 und j_7 an die Positionen 2 bis 4 getauscht werden, erhöhen sich Fertigstellungsszeitpunkte von j_2, j_3 und j_4 jeweils unnötig um 15 Zeiteinheiten.

Tabelle 6: Beispiel 2 des Vergleichs der Puffer-Algorithmen

σ^2	j_1	j_2	j_3	j_4	j_5	j_6	j_7
Fertigungsdauer	20	25	25	30	5	5	5
Deadline	60	70	90	80	150	150	150

Tabelle 7: Resultate der Algorithmen zu Beispiel 2

Algorithmus	**Tardiness**	$P(\sigma^2)$
kein Alg.	20	σ^2
t-Alg.	35	$(j_1, j_5, j_6, j_7, j_2, j_3, j_4)$
D-Alg.	10	$(j_1, j_2, j_4, j_3, j_5, j_6, j_7)$
$(D-t)$-Alg.	10	$(j_1, j_2, j_4, j_3, j_5, j_6, j_7)$
$(\frac{1}{3}, \frac{2}{3})$-Alg.	20	σ^2
$D+t$-Alg.	10	$(j_1, j_2, j_4, j_3, j_5, j_6, j_7)$
$(\frac{2}{3}, \frac{1}{3})$-Alg	10	$(j_1, j_2, j_4, j_3, j_5, j_6, j_7)$
TS-Alg	10	$(j_1, j_2, j_4, j_3, j_5, j_6, j_7)$
$t\uparrow$-Alg.	20	σ^2

D-Algorithmus

Für $\sigma \in Q$, $i \in \{1, \ldots, |\sigma|\}$ und $j_i = (t_i, D_i) \in J(\sigma)$ ergibt sich die Kriteriumsfunktion K^D des D-Algorithmus als

$$K^D : J(\sigma) \times Q \to \mathbb{R}, ((t_i, D_i), \sigma) \mapsto D_i. \tag{28}$$

Im D-Algorithmus wird jeder Job folglich nur mit seiner Deadline bewertet. Die Idee dabei ist, Jobs, für welche wenig Zeit eingeplant ist, zuerst zu bearbeiten und damit die Gesamtverspätung zu reduzieren. Weiterhin werden Aufträge, welche sehr hohe Deadlines haben spät gesetzt, wodurch sich sich die Fertigstellungszeitpunkte anderer Jobs verringern und die Gesamtverspätung eventuell gemindert wird. Im Gegensatz zum t-Algorithmus liefert der D-Algorithmus für das Beispiel aus Tabelle 6 sehr gute Resultate, da er sich nicht von den geringen Fertigungsdauern von j_5, j_6 und j_7 beeinflussen lässt.
Andererseits hat dieser Algorithmus bei Queues Probleme, in denen Jobs mit großem t kleine Deadlines haben. Solche Aufträge werden durch den Puffer früh gesetzt, blockieren anschließend die Maschine und führen insgesamt zu einer größeren Gesamtverspätung. Dieses Problem wird gut am Beispiel aus Tabelle 4 verdeutlicht.

$(D - t)$-Algorithmus

Für $\sigma \in Q$, $i \in \{1, \ldots, |\sigma|\}$ und $j_i = (t_i, D_i) \in J(\sigma)$ ergibt sich die Kriteriumsfunktion K^{D-t} des $(D - t)$-Algorithmus als

$$K^{D-t} : J(\sigma) \times Q \to \mathbb{R}, ((t_i, D_i), \sigma) \mapsto D_i - t_i. \tag{29}$$

Im $(D - t)$-Algorithmus wird jeder Job mit der Differenz seiner Deadline und Fertigungszeit bewertet. Diese Differenz entspricht dem spätesten Zeitpunkt, in welchem mit der Bearbeitung des Jobs begonnen werden muss, damit er noch fristgerecht fertiggestellt werden kann. Anhand der Nachteile des t- bzw. D-Algorithmus lässt sich vermuten, dass sowohl Deadline als auch Fertigungsdauer in eine gute Kriteriums-Funktion eingehen sollten. Dies geschieht in (29) durch eine sehr intuitive Kennzahl. Allerdings zeigt ACO:Bsp2, dass dies noch nicht zum Erfolg führt. Das schlechte Ergebnis liegt am negativen Eingang von t innerhalb der Kriteriums-Funktion: Für Jobs mit kleinem D und großem t wird der Ausdruck $D-t$ klein und es stellen sich gerade wieder die gleichen Nachteile wie im D-Algorithmus ein. Im folgenden werden daher Kriteriumsfunktionen betrachtet, bei denen die Parameter t und D jeweils positiv eingehen.

$\alpha \cdot D + \beta \cdot t$-Algorithmen

Das Problem des $(D - t)$-Algorithmus ist, dass die Fertigungszeit negativ eingeht und somit nicht in der Lage ist, die Schwächen des D-Algorithmus auszugleichen. Betrachte daher den $(D + t)$-Algorithmus. Für $\sigma \in Q$, $i \in \{1, \ldots, |\sigma|\}$

und $j_i = (t_i, D_i) \in J(\sigma)$ ergibt sich die Kriteriumsfunktion K^{D+t} als

$$K^{D+t} : J(\sigma) \times Q \to \mathbb{R}, ((t_i, D_i), \sigma) \mapsto D_i + t_i. \tag{30}$$

Allgemeiner wird im Folgenden der $(\alpha D + \beta t)$-Algorithmus mit $\alpha, \beta \in \mathbb{R}_{\geq 0}$ und $\alpha + \beta = 1$ betrachtet. Statt $(\alpha D + \beta t)$ wird auch (α, β) geschrieben. Für $\sigma \in Q$, $i \in \{1, \ldots, |\sigma|\}$ und $j_i = (t_i, D_i) \in J(\sigma)$ ergibt sich die Kriteriumsfunktion $K^{\alpha,\beta}$ als

$$K^{\alpha,\beta} : J(\sigma) \times Q \to \mathbb{R}, ((t_i, D_i), \sigma) \mapsto \alpha \cdot D_i + \beta \cdot t_i. \tag{31}$$

Insbesondere ergeben sich, außer dem $(D - t)$-Algorithmus, alle bisher betrachteten Algorithmen als Spezialfall des (α, β)-Algorithmus. Wähle $(0, 1)$ für K^t aus (27), $(1, 0)$ für K^D aus (28) und $(\frac{1}{2}, \frac{1}{2})$ für K^{D+t} aus (30). Für K^t und K^D ergibt sich dies direkt aus der Definition. Multipliziert man $K^{(\frac{1}{2}, \frac{1}{2})}$ mit dem konstanten Faktor 2, so ändert sich nichts an dessen erzeugter Reihenfolge und es entsteht gerade K^{D+t}.

Wie α und β zu wählen sind, um möglichst gute Ergebnisse zu erzielen, hängt stark von der Beschaffenheit der Jobfolgen ab. Im Weiteren werden neben den bisher vorgestellten Kombinationen $(1, 0), (\frac{1}{2}, \frac{1}{2})$ und $(0, 1)$ auch $(\frac{1}{3}, \frac{2}{3})$ und $(\frac{2}{3}, \frac{1}{3})$ betrachtet.
Da anzunehmen ist, dass eine gute Wichtung von der betrachteten Queue abhängig ist, wird neben den oben genannten, festen Kombinationen, auch folgende, von σ abhängige, Wichtung betrachtet:

$$\alpha : Q \to \mathbb{R}_{\geq 0} \text{ mit } \alpha(\sigma) = \sum_{i=1}^{|\sigma|} D(j_i)$$

$$\beta : Q \to \mathbb{R}_{\geq 0} \text{ mit } \beta(\sigma) = \sum_{i=1}^{|\sigma|} t(j_i).$$

Der zugehörige Algorithmus wird als TotalSum- bzw. TS-Algorithmus bezeichnet. Für $\sigma \in Q$, $i \in \{1, \ldots, |\sigma|\}$ und $j_i = (t_i, D_i) \in J(\sigma)$ gilt

$$K^{TS}(j_i, \sigma) = K^{TS}((t_i, D_i), \sigma) = \frac{\alpha(\sigma)}{\alpha(\sigma) + \beta(\sigma)} \cdot D_i + \frac{\beta(\sigma)}{\alpha(\sigma) + \beta(\sigma)}(\sigma) \cdot t_i. \tag{32}$$

Die Wichtung passt sich an das aggregierte Verhältnis der Deadlines und Fertigungsdauern der Queue an. Falls dieses Verhältnis beispielsweise zu Gunsten der Deadlines ausfällt, ist es eventuell von Vorteil, sich eher an den D-Werten zu orientieren.

Verbesserter t-Algorithmus

Eine andere Herangehensweise, die Nachteile des t-Algorithmus auszugleichen, ist die Verwendung einer stückweise definierten Kriteriumsfunktion. Die in Kapitel 3.4.1 erläuterten Probleme des Algorithmus beruhen darauf, dass Jobs mit sehr hohen Deadlines nicht gesondert betrachtet werden. Dies kann behoben werden, indem beispielsweise das Kriterium solcher Aufträge durch einen geeigneten Multiplikator erhöht wird und somit an eine spätere Stelle getauscht wird.
Betrachte daher den folgenden verbesserten t-Algorithmus bzw. $t\uparrow$-Algorithmus. Für $\sigma \in Q$, $i \in \{1, \ldots, |\sigma|\}$ und $j_i = (t_i, D_i) \in J(\sigma)$ ergibt sich die Kriteriumsfunktion $K^{t\uparrow}$ des $t\uparrow$-Algorithmus als

$$K^{t\uparrow} : J(\sigma) \times Q \to \mathbb{R}, ((t_i, D_i), \sigma) \mapsto \begin{cases} t_i, & D_i \leq \overline{D} \\ \max\{1, D - \overline{D}\} \cdot t_i, & D_i > \overline{D}, \end{cases} \tag{33}$$

wobei $\overline{D}$ die durchschnittliche Deadline der Queue σ ist

$$\overline{D} = \frac{\sum_{l=1}^{|\sigma|} D_l}{|\sigma|}.$$

3.4.2 Improved-Modified-Due-Date-Regel

Zu Vergleichszwecken wird die, in [MM00, S. 5] vorgeschlagene Improved-Modified-Due-Date-Regel, kurz IMDD-Regel, vorgestellt und verwendet. Für Position i und Auftrag j ist

$$\eta_{i,j} = \frac{1}{\max\{T + t(j), D(j)\} - T}, \tag{34}$$

wobei T die Summe der Fertigungsdauern der bereits aus dem Puffer entlassenen Aufträge ist.
Der Ausdruck $\max\{T+t(j), D(j)\}$ bewertet einen Job mit dem Maximum seiner Deadline und seinem potentiellen Fertigstellungszeitpunkt, falls er als nächstes aus dem Puffer ausgegeben wird. Somit werden Jobs, welche in diesem Moment keine Gefahr laufen, ihre Deadline zu verletzen, mit ihrer (verhältnismäßig großen) Deadline bewertet und erhalten damit einen niedrigen heuristischen Wert. Unter Aufträgen, deren Deadline nicht eingehalten werden kann, erhalten diejenigen Jobs einen höheren heuristischen Wert, welche eine kürzere Fertigungsdauer haben. Somit wird vermieden, dass große Jobs die Maschinen blockieren und die Gesamtverspätung unnötig erhöhen.

Wie Merkle und Middendorf erläutern, wirkt der Summand $-T$ im Nenner folgendem Problem entgegen: Der Ausdruck $\max\{T + t(j), D(j)\}$ wird - aufgrund eines großen T's - beim Entscheiden über eine späte Position viel höher als bei einer frühen Position. Infolgedessen sind die heuristischen Unterschiede der Jobs am Ende der Umordnung sehr gering, was den Einfluss der Heuristik mindert.

4 Computerexperimente

4.1 Programmaufbau

An dieser Stelle wird beschrieben, wie der in Kapitel 3.2 vorgestellte, populationsbasierte Ameisenalgorithmus für die Flowshop-Probleme aus Kapitel 1 implementiert wurde.

4.1.1 Vorüberlegungen

Zunächst ist anzumerken, dass im Rahmen dieser Arbeit nicht alle Settings miteinander verglichen werden konnten. Es wird sich daher auf die Betrachtung von $(4, 0, 0)$, $(0, 4, 0)$ und $(0, 2, 2)$ beschränkt. Diese drei Varianten wurden aus folgenden Gründen gewählt:
Unter der Annahme, dass ein Pufferplatz konstante Kosten bei der Errichtung besitzt, sollte zunächst jedes Setting über die gleiche Anzahl an Pufferslots verfügen. Unter dieser Bedingung erzeugt $(4, 0, 0)$ nach Kapitel 2.4 die größte Outputmenge. Da in diesem Fall jegliche Vertauschungen vor dem Split stattfinden, empfiehlt es sich, den konträren Fall $(0, 2, 2)$ ebenfalls zu betrachten. Hier nehmen die Puffer zwar ebenfalls gleichmäßigen Einfluss auf beide Queues eines Outputs, allerdings finden die Vertauschungen erst im Anschluss an den Split statt. Dies bietet zwar den Vorteil, dass (meist) kleinere Queues die Puffer durchlaufen, gleichzeitig halbiert sich jedoch die jeweilige Größe der Puffer. Außerdem wurde $(0, 4, 0)$ in die Untersuchungen aufgenommen. In diesem Setting können keinerlei Outputs generiert werden, welche Umsortierungen in der zweiten Komponente der Outputs verlangen. Zum einen ergibt sich dadurch der Nachteil einer kleineren Lösungsmenge, andererseits wurde bereits in Kapitel 2.4 angesprochen, dass dies nicht unbedingt negative Auswirkungen auf die minimalen Kosten haben muss, da jede Vertauschung Kosten erzeugt. Die künstlichen Ameisen müssen somit einen kleineren Lösungsraum durchsuchen, wodurch eventuell weniger Möglichkeiten auftreten, um sich schlecht zu entscheiden.
Für diese betrachteten Settings durchläuft jeder Job maximal einen Puffer, bevor er an einer Maschine bearbeitet wird. Dadurch kann eine Queue jeden Bearbeitungsschritt vollständig durchlaufen, bevor der nächste berechnet werden muss.

4.1.2 Detaillierte Programmbeschreibung

Nachdem das Programm gestartet wurde und bevor die jeweiligen Problem-Instanzen aus einer externen Datei „Input.txt“ eingelesen werden können, wird der Benutzer aufgefordert, alle benötigten Informationen zur Konfiguration des Algorithmus einzugeben. Tabelle 8 listet diese Werte auf.
Dabei wird der Nutzer aufgefordert, eine vorher festgelegte Anzahl an Neustarts einzugeben, welche festlegt, wie häufig eine Neuberechnung eines Problems durchgeführt werden soll. Dies ist notwendig, um eine durchschnittliche Lösung zu berechnen und somit die Auswirkungen besonders guter bzw. schlechter Durchläufe zu relativieren. Die durchschnittlichen Ergebnisse sind daher besser für eine spätere Analyse der Resultate geeignet.

Tabelle 8: Eingabewerte des Programms zum Lösen der Flowshop-Probleme mittels des pACO. Die Spalte Parameter gibt die Notation im Pseudocode an.

Eingabewert	**Bemerkung**	**Parameter**
Heuristik	Auswahl zwischen allen Heuristiken aus Kapitel 3.4, welche auf die dort beschriebene Weise implementiert wurden. Es ist auch möglich keine Heuristik auszuwählen.	heu
Pheromon-Evaluationsverfahren	Auswahl zwischen lokaler, relativer oder Summationsregel.	$pEval$
Setting	Auswahl zwischen $(4,0,0)$, $(0,4,0)$ und $(0,2,2)$.	$setting$
Iterationszahl	Entspricht der Abbruchbedingung des Algorithmus.	$runMax$
Ameisenzahl	Gibt an, wie viele Lösungen (künstliche Ameisen) pro Iteration erzeugt werden sollen.	$antMax$
Größe der Lösungspopulation	Entspricht dem Wert $\|L\|$.	$popMax$
Länge der Queues in "Input.txt"	-	$jobMax$
Anzahl Neustarts	Anzahl der Berechnungen pro Probleminstanz.	$startMax$

Im Rahmen dieser Arbeit konnte eine Parameteroptimierung nicht durchgeführt werden. Für alle Tests in den weiteren Kapiteln wurden 8.000 Iterationen, 10 künstliche Ameisen und eine Lösungspopulation der Größe 4 gewählt. Diese

Werte haben sich in einigen vorab durchgeführten Tests als gut geeignet herausgestellt und decken sich bezüglich der Populationsgröße und Ameisenanzahl mit der Literatur. Die Iterationszahl ist mit 8.000 recht hoch gewählt. Da die Geschwindigkeit der Berechnungen in dieser Arbeit allerdings eher im Hintergrund steht, bringt dies keine Nachteile mit sich. Für eine Verwendung in der Realwirtschaft müssten eventuell weniger Iterationen gewählt werden.

Nachdem alle relevanten Werte eingegeben worden, wird die Pheromonmatrix $\mathbb{P}$ durch die Kosten eines Standard-Outputs (σ_1^0, σ_2^0) initialisiert (analog zu [MM00, S. 3]). (σ_1^0, σ_2^0) entsteht im jeweils gewählten Setting, wobei jegliche Umordnungen in den Puffern anhand des D-Algorithmus aus Kapitel 3.4.1 durchgeführt werden. Der Split wird so gewählt, dass ein Auftrag $j \in J(\sigma)$ stets zu derjenigen Maschine gesandt wird, welche geringer ausgelastet ist. Gilt $C((\sigma_1^0, \sigma_2^0)) = 0$, so kann die Berechnung abgebrochen werden, da diese Lösung optimal sein muss. Andernfalls gilt $C((\sigma_1^0, \sigma_2^0)) \neq 0$ und

$$\tau^0 := \frac{1}{C((\sigma_1^0, \sigma_2^0))} > 0. \tag{35}$$

Im nächsten Schritt werden in jeder Iteration genau $antMax \in \mathbb{N}$ künstliche Ameisen entsendet, welche eine zulässige Lösung generieren:

1. *Bestimme eine Reihenfolge* $\sigma' \in P_1^{(k_1,k_2,k_3)}(\sigma)$*.*
   Für $(4,0,0)$ wird für jede Position $i \in \{1, \ldots, |\sigma|\}$ ein Auftrag j aus den geladenen Jobs, anhand der Wahrscheinlichkeiten $p_{i,j}$ aus (24) bzw. (25) ausgewählt und gesetzt. $\eta_{i,j}$ entspricht dabei einer Heuristik aus Kapitel 3.4 und $a, b \in \mathbb{R}$ einer Wichtung der Einflüsse aus Heuristik und Pheromon. Falls hingegen $k_1 = 0$ gilt, ist $\sigma' = \sigma$ und es muss keine Berechnung durchgeführt werden.

2. *Split bestimmen:*
   Für jeden Job j wird anschließend entschieden zu welcher Maschine $m \in \{1, 2\}$ er gesandt wird. Die zugehörigen Wahrscheinlichkeiten ergeben sich als
   $$p_{m,j}^{Split} := \frac{\tau_{m,j}}{\tau_{1,j} + \tau_{2,j}}.$$

3. *Bestimme die Umordnungen durch* $P_2^{(k_1,k_2,k_3)}$ *bzw.* $P_3^{(k_1,k_2,k_3)}$*.*
   Wurde das Setting $(4,0,0)$ gewählt, so muss an dieser Stelle keine Berechnung durchgeführt werden, da keine Vertauschungen mehr stattfinden können. Gilt hingegen $k_2 \neq 0$ oder $k_3 \neq 0$, so erfolgen die Berech-

nungen analog zu 1., allerdings jeweils nur für die Jobs, welche im Split zu Maschine 1 bzw. 2 entsandt worden. Die Wahrscheinlichkeit dafür, dass Job j in einer der beiden Queues auf Position i gesetzt wird, ergibt sich jeweils wieder als $p_{i,j}$ analog zu (24) bzw. (25).

4. *Kostenberechnung:*
   Bewerte die generierten Outputs anschließend durch die Kostenfunktion C aus Kapitel 1.5 und speichere die beste der $antMax$ Lösungen dieser Iteration zwischen. Im Folgenden sei diese jeweils mit (σ_1^*, σ_2^*) notiert. Falls eine Lösung mit Kosten in Höhe von 0 generiert wurde, kann die Berechnung abgebrochen werden, da diese optimal sein muss.

Anschließend wird, wie in Kapitel 3.2 beschrieben, die Lösungspopulation, die Pheromonmatrix sowie eventuell die beste bisher gefundene Lösung aktualisiert. Die zur besten Lösung dieser Iteration gehörigen Werte von $\mathbb{P}$ werden jeweils um den Wert

$$\frac{1}{C((\sigma_1^*, \sigma_2^*))}$$

erhöht. An dieser Stelle weicht die Implementation vom Entwurf des pACO aus 3.2 ab, da die betroffenen $\tau_{i,j}$ nicht um einen konstanten Faktor erhöht werden. Dies bietet den Vorteil, dass eine gute Lösung mit geringen Kosten eine stärkere Pheromonspur legen kann als eine Lösung mit zugehörigen hohen Kosten. Für die betrachteten Probleme gilt $C((\sigma_1^*, \sigma_2^*)) \in \mathbb{N}$, da die Berechnung im Falle von $C((\sigma_1^*, \sigma_2^*)) = 0$ bereits abgebrochen worden wäre. Folglich gilt $\frac{1}{C((\sigma_1^*, \sigma_2^*))} \leq 1$ und die Einträge von $\mathbb{P}$ sind durch $k \cdot 1 + \tau^0$ nach oben beschränkt. Befinden sich bereits mehr als $popMax$ Lösungen in der Population, so wird die älteste Lösung aus L ersetzt und die zugehörige Pheromonspur aus $\mathbb{P}$ entfernt. Dabei wird genau die Pheromonmenge entnommen, welche k Generationen zuvor hinzugefügt wurde. Es gilt folglich auch in dieser Variation $\tau_{i,j} \geq \tau^0 > 0$.
Am Ende jeder Iteration wird überprüft, ob es sich bei der jeweiligen Lösung (σ_1^*, σ_2^*) um die bisher beste Lösung dieses Neustarts handelt und das Gesamtminimum wird gegebenenfalls aktualisiert.
Sobald alle Iterationen durchlaufen sind, werden die Konfigurationen des Ameisenalgorithmus, die Problemstellung, die beste gefundene Lösung sowie die minimalen und durchschnittlichen Kosten über alle Neustarts in eine externe Datei ausgegeben.

4.1.3 Pseudocode

Um eine übersichtliche Darstellung des konzipierten Algorithmus zu erzeugen, wird er abschließend in Algorithmus 1 als Pseudocode abgebildet. Neben den Notationen aus Tabelle 8 meint L die Lösungspopulation, X_{ant} eine zu einer Ameise $ant \in \mathbb{N}$ gehörigen künstlichen Ameise sowie X^* die beste (bisherige) Lösung der aktuellen Iteration.

Algorithm 1 Struktur des entworfenen pACO als Pseudocode

```
Bestimme Standard-Output (σ_1^0, σ_2^0) aus (35)
τ^0 ← 1 / C((σ_1^0, σ_2^0))
for start ← 1, startMax do
    ℙ ← τ^0
    for run ← 1, runMax do
        for ant ← 1, antMax do
            Konstruiere eine zulässige Lösung X_ant analog zu Kapitel 4.1.2
            if C(X_ant) = 0 then
                Stop; return X_ant
            end if
            if X_ant ist die bisher kostengünstigste Lösung der Iteration then
                X* ← X_ant
            end if
        end for
        if run > popMax then
            Lösche älteste Lösung aus L
            Lösche zugehörige Pheromonspur aus ℙ
        end if
        Erhöhe die zu X* gehörigen Pheromonwerte um 1 / C(X*)
        Füge X* der Lösungspopulation L hinzu
        if X* ist die bisher kostengünstigste Lösung dieses Neustarts then
            Aktualisiere die minimalen Kosten dieses Neustarts
        end if
    end for
    Aktualisiere die minimalen und durchschnittlichen Kosten des Problems
end for
```

4.2 Benchmark-Sets

Um die Algorithmen zu evaluieren, wurde analog zu [VRM08, S. 11] ein Benchmark-Set an Beispielen erzeugt. Für $n \in \mathbb{N}$ und $\sigma = (j_1, \ldots, j_n) \in Q$ (meist ist $n = 30$) wurden die jeweiligen Fertigungsdauern gleichverteilt aus dem Intervall $[1, \ldots, 100] \cap \mathbb{N}$ ausgewählt. Die zugehörigen, ganzzahligen Deadlines wurden anschließend gleichverteilt aus dem folgenden Intervall gewählt

$$\left[\max \left\{ 0, \sum_{i=1}^{n} t(j_i) \cdot \left(1 - TF - \frac{RDD}{2} \right) \right\}, \sum_{i=1}^{n} t(j_i) \cdot \left(1 - TF + \frac{RDD}{2} \right) \right].$$

Durch RDD (relative range of due dates) wird die Größe des Intervalls bestimmt und TF (tardiness factor) gibt die relative Position der Mitte des Intervalls zwischen 0 und $\sum_{i=1}^{n} t(j_i)$ an.
Pro Kombinationen aus $TF \in \{\frac{2}{5}, \frac{3}{5}, \frac{4}{5}\}$ und $RDD \in \{\frac{1}{5}, \frac{3}{5}, 1\}$ wurden jeweils 5 Queues generiert. Insgesamt entstanden somit $3^2 \cdot 5 = 45$ Beispielinstanzen, an welchen der Algorithmus in seinen verschiedenen Varianten getestet und ausgewertet werden konnte. Die generierten Benchmark-Sets sowie das Programm „CreateInput“, welches zum Erzeugen der Instanzen verwendet wurde, befindet sich im Verzeichnis „Benchmarksets“ auf der CD.

4.3 Auswertung

In diesem Abschnitt werden die Computerexperimente im Speziellen erläutert. Es wird zunächst darauf eingegangen, wie die einzelnen Untersuchungen aufgebaut sind, anhand welcher Kennzahlen die Auswertung stattgefunden hat und welche Schlussfolgerungen sich daraus ergeben. Die Versuchsdaten und Auswertungen befinden sich im Verzeichnis „Computerexperimente“ auf der CD im Anhang.

4.3.1 Wichtungen der Heuristiken

Bevor im Kapitel 4.3.2 die Einflüsse der Heuristiken betrachtet werden können, wird an dieser Stelle die Wichtung der Einflüsse von Heuristik und Pheromonmatrix bei der Lösungskonstruktion analysiert. Die Entscheidung, welcher Job auf welche Position gesetzt wird, fällt anhand der Wahrscheinlichkeiten aus (24). Gilt beispielsweise $a = 1$ und $b = 0$, so sind die $p_{i,j}$ nur abhängig von den Informationen aus der Pheromonmatrix. Gilt hingegen $a = 0$ und $b = 1$, fallen die Entscheidungen ausschließlich anhand der Heuristik. In der Literatur wird üblicherweise eine Wichtung mit $a = 1$ und $b \in \{1, 3, 5\}$ verwendet (etwa in [MM00] oder [GM02]).
Dabei wurden die 45 Benchmark-Instanzen im Setting $(4, 0, 0)$ durch den pACO zusammen mit dem lokalen Pheromon-Evaluationsverfahren und den verschiedenen Puffer-Heuristiken aus Kapitel 3.4 betrachtet. Nach Kapitel 2.4 besitzt $(4, 0, 0)$ die größte Outputmenge unter den Settings mit genau vier Pufferslots, wodurch sich die unterschiedlichen Qualitäten der Variationen des Algorithmus am besten herausbilden können. Die weiteren Einstellungen decken sich mit denen aus 4.1. Aus oben genannten Gründen wurden alle Wichtungen mit $a = 1$ und $b \in \{1, 3, 5\}$ getestet. Für jede Kombination aus Wichtung und Heuristik wurden pro Beispiel die minimalen und die durchschnittlichen Kosten über jeweils fünf Neustarts ermittelt. Zu jeder Heuristik wurde anschließend betrachtet, welche Wichtung die besten Ergebnisse erzielte. Zur Auswertung wurde herangezogen, wie häufig die jeweilige Wichtung gegenüber den anderen Kombinationen aus a und b das beste Ergebnis erreichte (siehe Abbildung 6) und wie stark sich die Kosten durchschnittlich (Schnitt über alle Beispiele) zu $b = 1$ unterscheiden (siehe Abbildung 7). Die vollständige Auswertung der Tests befindet sich in der Datei „Wichtung-Vergleich.ods“ im Unterverzeichnis „Test_Wichtung“.

Die Abbildungen 6 und 7 stellen die Ergebnisse der Auswertungen in Form

Abbildung 6: Verteilung der Ergebnisse bezüglich der Häufigkeit der besten Kombination bei variierender Heuristik. "$b = 1$", "$b = 3$" und "$b = 5$" repräsentieren die jeweilige Wichtung und "min" beziehungsweise "avg" geben an, ob die Betrachtung bezüglich der minimalen oder der durchschnittlichen gefundenen Lösungen stattfindet.

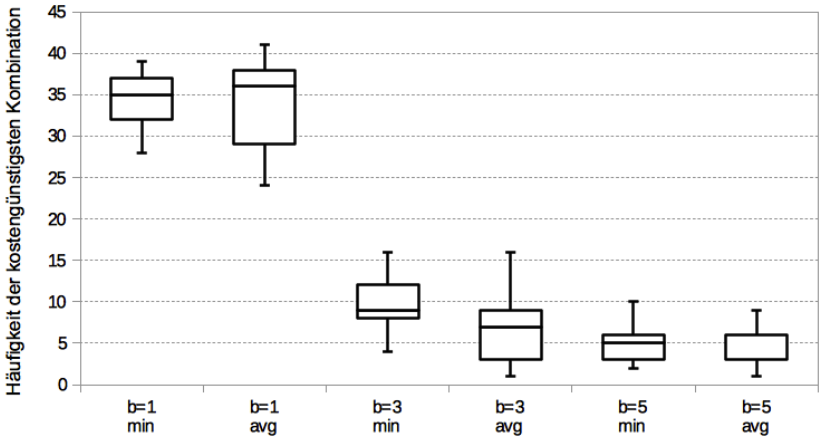
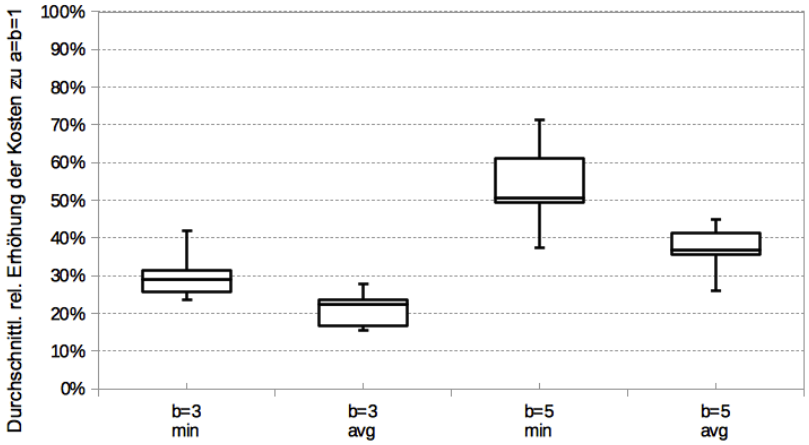


Abbildung 7: Verteilung der durchschnittlichen Erhöhung der Kosten im Vergleich zur Wichtung $a = b = 1$ bei variierender Heuristik. Die Beschriftung der Abszissenachse wurde analog zu Abbildung 6 gewählt. Da beide Verteilungen über 0 liegen, liefern diese Wichtungen durchschnittlich höhere Kosten als $a = b = 1$.



von Boxplots dar. Anhand dieser Diagramme ist ersichtlich, dass die Wichtung $a = b = 1$ unabhängig vom verwendeten Pheromon-Evaluationsverfahren die besten Ergebnisse liefert. Durch diese Kombination von a und b konnten sowohl die Gesamtkosten im Durchschnitt am stärksten reduziert als auch das beste Ergebnis am häufigsten erzielt werden.

Basierend auf dieser Analyse, gehen bei den weiteren Untersuchungen die heuristischen Einflüsse gleich stark mit den Informationen aus der Pheromonmatrix ein, das heißt es gilt $a = b = 1$.

4.3.2 Vergleich der Heuristiken

Es werden nun die Effekte der unterschiedlichen Puffer-Heuristiken betrachtet. Hierfür wurde der pACO samt der Heuristiken in der Wichtung $a = b = 1$ im Setting $(4, 0, 0)$ mit den drei vorgestellten Pheromon-Evaluationsverfahren an den 45 Benchmark-Instanzen getestet. Die weiteren Einstellungen decken sich mit denen aus 4.1.
Für jede Kombination aus Pheromon-Evaluationsverfahren und Heuristik wurden pro Beispiel die minimalen und die durchschnittlichen Kosten über jeweils 25 Neustarts ermittelt. Zu jedem Evaluationsverfahren wurde anschließend jeweils betrachtet, welche Heuristiken die besten Ergebnisse erzielten. Zur Auswertung wurde der Anteil der verbesserten Ergebnisse und die durchschnittliche (Schnitt über alle Beispiele), relative Veränderung der Kosten pro Beispiel im Vergleich zum pACO ohne heuristische Information betrachtet. Die vollständige Auswertung der Tests befindet sich in der Datei „Heu-pEval-Vergleich.ods“ im Verzeichnis „Test_Heu-pEval“.

Da die Ergebnisse abhängig von den Parametern TF und RDD aus Kapitel 4.2 sind, wird zunächst eine gesonderte Betrachtung der ersten 15 Instanzen vorgenommen. Hier lieferten die Varianten mit heuristischem Einfluss besonders schlechte Resultate in Bezug auf die relative Reduktion der Gesamtkosten. Im Reiter „Einstellungen“ kann nachvollzogen werden, dass für diese Queues $TF = 0,4$ gilt, wodurch die Deadlines dieser Jobs relativ hoch ausfallen. Da die Fertigungszeiten analog zu den restlichen Instanzen generiert worden, die Deadlines jedoch höher ausfallen, steigt die Wahrscheinlichkeit, dass die Gesamtkosten einer gefundenen Lösung dieser Beispiele niedrig ausfallen. Dies spiegelt sich auch in den Ergebnissen des pACO ohne Heuristik wieder, der für diese Instanzen Outputs mit verhältnismäßig geringen Gesamtkosten gefunden hat. Kleine absolute Abweichungen können somit gleichzeitig starken, relativen Abweichungen entsprechen. Da für die Auswertung die durchschnittliche prozentuale Veränderung der Kosten verwendet wird, verfälschen diese Beispiele somit die Auswertung.
Für die ersten zehn Instanzen mit $RDD = 0,2$ bzw. $RDD = 0,6$ fällt zudem das Intervall, aus welchem die Deadlines ausgewählt worden, relativ klein aus. Insgesamt variieren die Jobs bezüglich ihrer Deadlines demzufolge nicht sehr stark, wodurch eine Vielzahl von Outputs existiert, welche eine geringe Gesamtverspätung verursachen. Demzufolge existieren Outputs mit geringer Verspätung und verhältnismäßig wenigen Vertauschungen der Aufträge. Die Heuristiken berücksichtigen dieses Phenomen nicht, wodurch hohe Werte $\eta_{i,j}$ unab-

hängig davon entstehen, ob eine Vertauschung stattfindet oder nicht. Das heißt, dass durch die Verwendung einer Heuristik für die künstlichen Ameisen häufig ein starker Anreiz entsteht, Vertauschungen der Aufträge durchzuführen, wodurch jeweils Pufferkosten entstehen. Um diesbezüglich eine genauere Untersuchung durchführen zu können, wurden die Daten der ersten 15 Instanzen erneut berechnet, wobei diesmal zusätzlich die zugehörigen Pufferkosten ausgegeben worden. Im Reiter „Result_15“ der gleichen Datei befindet sich die zugehörige Auswertung. Wie in Tabelle 9 beispielhaft dargestellt ist, liegt der Anteil der Pufferkosten, unabhängig vom Pheromon-Evaluationsverfahren oder davon welche Heuristik im Speziellen verwendet wurde, in den ersten zehn Beispielen bei fast 100%. Die Verwendung einer Heuristik erzeugt hier demzufolge, wie oben beschrieben, hauptsächlich mehr Vertauschungen, erhöht allerdings nicht die Gesamtverspätung.
Aus diesen Gründen wird im Folgenden oft die Untersuchung der ersten 15 Probleme gesondert von den Instanzen 16 bis 45 durchgeführt.
Weiterhin ist anzumerken, dass die Modifikation der Pheromonaktualisierung, welche in Kapitel 4.1.2 beschrieben wurde, sich möglicherweise negativ auf diese Probleme ausgewirkt hat. Die obere Grenze der Einträge von $\mathbb{P}$ ist mit $k + \tau^0$ im Vergleich zu $\tau^0 \in (0, 1]$ relativ hoch. Es kann somit geschehen, dass die $\tau_{i,j}$, welche zu einer bestimmten Lösung gehören, sehr groß werden und die künstlichen Ameisen folglich häufig die gleiche Lösung konstruieren. Insgesamt mindert dies die Lösungsexploration und somit die Qualität des Algorithmus. Um dies zu vermeiden, hätte es sich empfohlen die $\tau_{i,j}$ zusätzlich zu beschränken, beispielsweise durch $\min\{1, k + \tau^0\}$.

Tabelle 9: Anteil der Pufferkosten an den Gesamtkosten für die ersten 15 Beispielinstanzen. Gezeigt werden pACO ohne Heuristik sowie in Kombination mit der t-Heuristik und der IMDD-Regel. Es wurde jeweils das lokale Pheromon-Evaluationsverfahren verwendet.

	pACO	pACO+t-Heu	pACO+IMDD-Regel
Instanzen 1-10	100%	100%	100%
Instanzen 11-15	$54,6\%$	$23,3\%$	$60,0\%$

Unter den Kriteriums-Heuristiken bewirkten lediglich unter Verwendung der Summationsregel und unter Einschränkung auf die ersten 15 Beispiele die $(D + t)$-, die $(\frac{2}{3}, \frac{1}{3})-$ und die TS-Heuristik eine leichte Reduktion der Kosten um durchschnittlich ca. $0,5 - 2,5\%$. Wie in den Abbildungen 8 und 9 ersichtlich ist, liegen die Verteilungen ansonsten oberhalb der 0%. Dies ist gleichbedeutend damit, dass unabhängig vom betrachteten Pheromon-Evaluationsverfahren, keine dieser Puffer-Heuristiken die Resultate des pACO verbessern konnte. Gemittelt auf

alle Beispiele wurden die Kosten sogar (teilweise sehr stark) erhöht.
Dies lässt sich vor allem dadurch erklären, dass die verwendeten Heuristiken zu statisch konzipiert sind und zu keinem Zeitpunkt die bisherige Lösungskonstruktion miteinbeziehen. Würden die Kriteriumsfunktionen die Positionen der Jobs in der resultierenden Queue berücksichtigen, so würden sich die Werte der Heuristiken deutlicher unterscheiden und die Heuristik wäre insgesamt dynamischer.
Bezüglich der TS-Heuristik leitet sich zudem aus der Konstruktionsvorschrift der Benchmark-Instanzen ab, dass die Deadlines zumeist um ein Vielfaches größer als die zugehörigen Fertigungsdauern sind. Somit ist der Ausdruck $\sum_{i=1}^{|\sigma|} D(j_i)$ stets viel größer als $\sum_{i=1}^{|\sigma|} t(j_i)$, wodurch der TS-Algorithmus seine Wichtung immer stark auf die Deadlines ausrichtet. Es stellte sich heraus, dass es somit nicht mehr möglich ist, die Nachteile der D- bzw. $(\frac{2}{3}, \frac{1}{3})$-Heuristik auszugleichen. Der Ansatz, die Wichtung lediglich am Verhältnis zwischen Gesamtfertigungsdauern und -deadlines festzumachen, scheint demnach nicht zielführend zu sein.
Ein ähnliches Problem tritt bei der $t\uparrow$-Heuristik auf, denn der Faktor $D - \overline{D}$ skaliert ebenfalls nicht gut für die hier betrachteten Beispiele. Für Beispiel 35 ergibt sich beispielsweise $\overline{D} \approx 802$ und für $j_6 = (65, 943)$ somit $D - \overline{D} = 141$. Anstelle eines Faktors, der sich aus der absoluten Abweichung zur mittleren Deadline ergibt, hätte es sich beispielsweise empfohlen einen Faktor zu wählen, welcher sich aus dem relativen Unterschied zwischen Deadline und $\overline{D}$ ergibt.
Verbesserungen an den vorgestellten Algorithmen konnten im Rahmen dieser Arbeit nicht mehr vollzogen werden.

Wie in Abbildung 9 ersichtlich ist, verbesserte die IMDD-Regel aus Kapitel 3.4.2 die Ergebnisse des pACO für die Instanzen 16 bis 45 hingegen. Eingeschränkt auf diese Beispiele, trat unabhängig vom Pheromon-Evaluationsverfahren in $80 - 90\%$ der Fällen eine Reduktion der Gesamtverspätung ein (in der Datei "Test_Heu-pEval" ersichtlich). Für die Summationsregel, bewirkte die IMDD-Regel sogar in den ersten 15 Beispielen eine Verbesserung der Ergebnisse im Vergleich zum pACO ohne heuristischen Einfluss (siehe Abbildung 8).

Zusammenfassend sollte für Probleme, welche den Beispielen 16 bis 45 ähneln, der pACO kombiniert mit der IMDD-Regel verwendet werden. Falls hingegen die betrachteten Probleme eher den ersten 15 Beispielen ähneln und die lokale oder die relative Pheromon-Regel angewandt wird, ist von der Verwendung der in dieser Arbeit vorgestellten Heuristiken abzuraten. Falls hingegen die

Abbildung 8: Verteilung der durchschnittlichen Erhöhung der Kosten im Vergleich zum pACO ohne heuristischen Einfluss bei variierender Heuristik anhand der Instanzen 1 bis 15. Die Abszisseneinträge entsprechen den verschiedenen Pheromon-Evaluationsverfahren aus Kapitel 3.3. Die nicht dargestellten Maxima der Verteilungen befinden sich bei $164,5\%$ (lokal) und $169,4\%$ (relativ). Die Boxplots beschreiben alle untersuchten Heuristiken aus Kapitel 3.4. Das Minimum bezüglich der Summationsregel wird durch die IMDD-Regel angenommen.

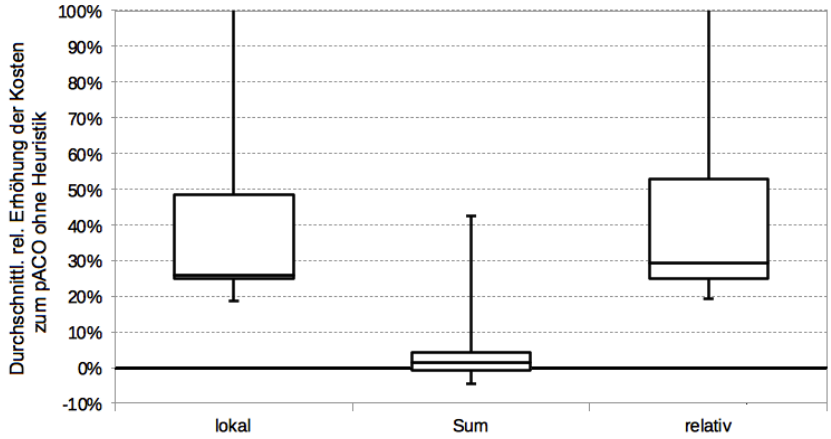
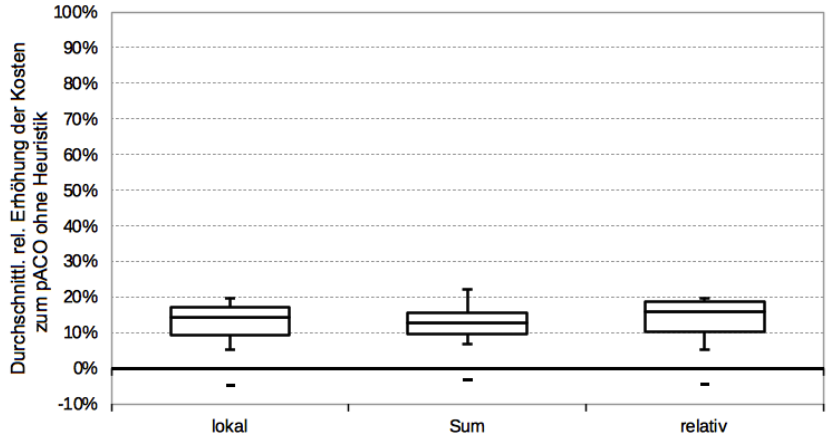


Abbildung 9: Eine zu Abbildung 8 analoge Darstellung der Ergebnisse bezüglich der Instanzen 16 bis 45. Die Werte des pACO mit der IMDD-Regel sind hier jedoch einzeln eingetragen und entsprechen den Markierungen im negativen Bereich. Die Boxplots beschreiben die Ergebnisse der sonstigen Heuristiken.



Summationsregel verwendet wird, verbessert ebenfalls die IMDD-Heuristik die Ergebnisse.

4.3.3 Vergleich der Pheromon-Evaluationsverfahren

Um die Betrachtung der Heuristiken abzuschließen, müssen noch die Pheromon-Evaluationsverfahren untereinander verglichen werden. Resultierend aus den Ergebnissen der Kapitel 4.3.1 und 4.3.2 werden an dieser Stelle nur die

Pheromon-Evaluationsverfahren für den pACO ohne heuristischen Einfluss und den pACO kombiniert mit der IMDD-Regel und der Wichtung $a = b = 1$ betrachtet. Diese Variationen des pACO wurden an den 45 Benchmark-Instanzen im Setting $(4, 0, 0)$ und den Standard-Einstellungen aus 4.1 gestestet.
Die Auswertung erfolgte anhand der gleichen Daten, welche in Kapitel 4.3.2 verwendet worden und stützt sich auf die gleichen Kennzahlen. Aus den selben Gründen wie in 4.3.2 wurde weiterhin eine gesonderte Betrachtung der ersten 15 Beispiele und der Instanzen 16 bis 45 durchgeführt. Die vollständige Auswertung der Tests befindet sich in der Datei „Heu-pEval-Vergleich.ods“, im Reiter „Results_pEval=var“ aus dem Verzeichnis „Test_Heu-pEval“.

Betrachte Tabelle 10: Bezüglich des pACO ohne Heuristik ergab sich für die Beispiele 16 bis 45 unter Verwendung der Summationsregel in ca. $86,7\%$ der Fälle eine Reduktion der durchschnittlichen Kosten im Vergleich zur lokalen Pheromon-Regel. Über all diese 30 Beispiele gemittelt, wurden die Resultate um ca. 8% verbessert. Die relative Pheromonregel bewirkte hingegen für alle 45 Beispiele eine gleichmäßige, geringe Reduktion der Kosten um ca. 1%. Eine Verbesserung der durchschnittlichen Kosten trat dabei in knapp über der Hälfte der betrachteten Fälle ein.
Für den pACO kombiniert mit der IMDD-Heuristik ist an Tabelle 11 ersichtlich, dass sich die Summationsregel klar gegenüber den anderen beiden Evaluationsverfahren durchsetzte. Im Vergleich zur lokalen Pheromon-Regel bewirkte die Summationsregel eine durchschnittliche Reduktion der Kosten um 10% pro Beispiel. Insgesamt trat in über 90% der Fälle eine Verbesserung der Ergebnisse ein.

Tabelle 10: Evaluation des Einflusses der verschiedenen Pheromon-Evaluationsverfahren auf den pACO ohne Heuristik. Im oberen Teil der Tabelle wird die durchschnittliche, relative Reduktion der Gesamtkosten dargestellt; im unteren Teil der Anteil an verbesserten Instanzen (jeweils im Vergleich zum lokalen Evaluationsverfahren). Die Auswertung ist unterteilt in die ersten 15 und die letzten 30 Beispiele.

pACO	relative Pheromonregel	Summationsregel
durchschnittliche relative Erhöhung der Kosten		
Instanzen 1 bis 15	$-1,2\%$	$25,5\%$
Instanzen 16 bis 45	$-0,8\%$	$-8,0\%$
Anteil der verbesserten Instanzen		
Instanzen 1 bis 15	$53,3\%$	$26,7\%$
Instanzen 16 bis 45	$50,0\%$	$86,7\%$

Im Reiter „Alg_vgl“ in der gleichen Datei befindet sich ein abschließender Ver-

Tabelle 11: Eine zu Abbildung 10 analoge Darstellung der Ergebnisse bezüglich des pACO kombiniert mit der IMDD-Regel.

pACO + IMDD-Regel	relative Pheromonregel	Summationsregel
durchschnittliche relative Erhöhung der Kosten		
Instanzen 1 bis 15	$1,3\%$	$-19,6\%$
Instanzen 16 bis 45	$-0,4\%$	$-6,5\%$
Anteil der verbesserten Instanzen		
Instanzen 1 bis 15	$26,7\%$	100%
Instanzen 16 bis 45	$63,3\%$	90%

gleich zwischen pACO mit relativer Pheromon- bzw. Summationsregel und dem pACO kombiniert mit der IMDD- und Summationsregel. Es stellte sich heraus, dass für Probleme, welche den ersten 15 Beispielen ähneln, keine Heuristik und die relative Pheromon-Regel verwendet werden sollte. Für Probleme, welche den Beispielen 16 bis 45 nahekommen, erzielt die IMDD-Heuristik kombiniert mit der Summationsregel die besten Ergebnisse.

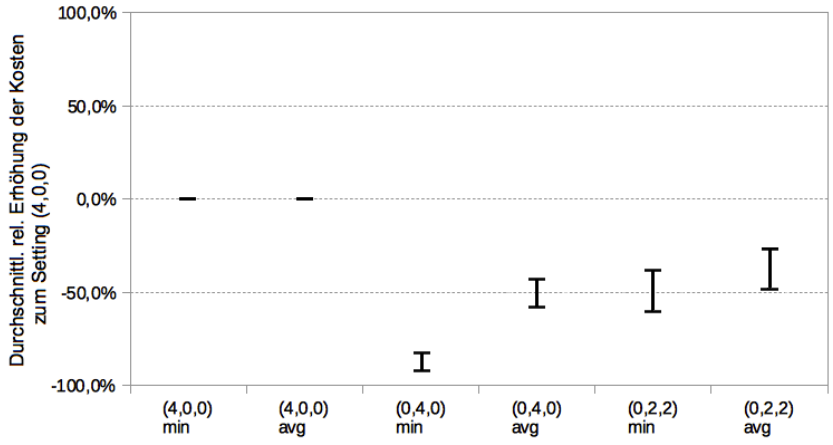
4.3.4 Vergleich der Settings

Als Ergänzung zu den theoretischen Betrachtungen der Settings aus Kapitel 2 werden an dieser Stelle beispielhaft die Settings $(4,0,0)$, $(0,4,0)$ und $(0,2,2)$ zusammen mit den Ergebnissen aus 4.3.1- 4.3.3 an den 45 Benchmark-Instanzen getestet. Als Algorithmen wurden folglich der pACO ohne Heuristik mit der relativen Pheromonregel und der pACO mit der Summations- und IMDD-Regel in der Wichtung $a = b = 1$ verwendet. $(4,0,0)$ besitzt in diesem Fall zwar jeweils die größte Outputmenge, nach den Überlegungen aus Kapitel 2.4 heißt dies allerdings nicht zwangsweise, dass in diesem Setting jeweils die besten Ergebnisse erzielt werden.
Für jede Kombination aus den drei Settings und den zwei Variationen des pACO wurden pro Beispiel die minimalen und durchschnittlichen Kosten über jeweils 25 Neustarts ermittelt. Zu jedem Algorithmus wurde anschließend betrachtet, wie sich die jeweiligen Settings auf die Gesamtkosten der einzelnen Benchmark-Instanzen auswirken. Zur Auswertung wurde, analog zu den vorhergegangenen Unterkapiteln 4.3.1 bis 4.3.3, der Anteil der verbesserten Ergebnisse und die durchschnittliche, relative Veränderung der Kosten pro Beispiel im Vergleich zum Setting $(4,0,0)$ betrachtet. Die vollständige Auswertung der Tests befindet sich in der Datei „Setting-Vergleich.ods“ im Verzeichnis „Test_Settings“.

Beim Betrachten der Ergebnisse musste wieder eine Fallunterscheidung durchgeführt werden. In Abbildung 10 liegen alle Werte bezüglich $(0,4,0)$ und $(0,2,2)$ unterhalb von 0%. Mit Hilfe dieser Settings konnten demnach Lösungen mit ge-

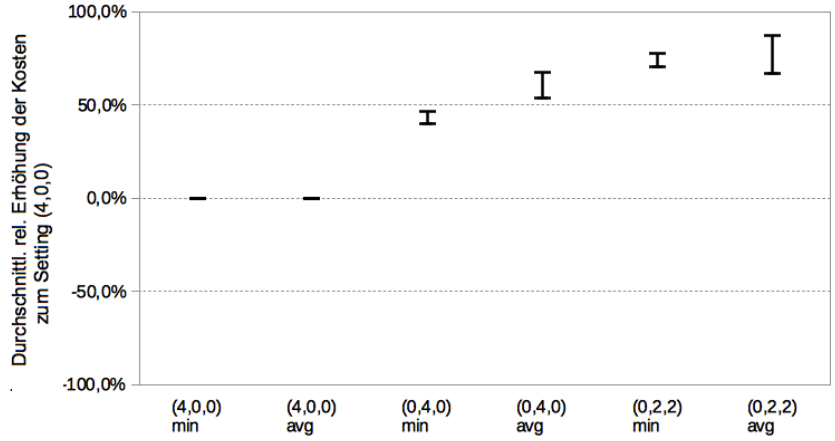
ringeren Kosten erzeugt werden. Folglich sind für die ersten zehn Beispiele vor allem $(0, 4, 0)$, aber auch das Setting $(0, 2, 2)$ in Bezug auf die durchschnittlichen und die minimalen Kosten besser geeignet als $(4, 0, 0)$. Wie in Kapitel 4.3.2 erläutert, sind die minimalen Kosten dieser Beispiele sehr gering und es existieren optimale bzw. gute Lösungen, welche keine oder kaum Vertauschungen der Aufträge verlangen. Falls für eine Instanz ein Output existiert, dessen Kosten sich als 0 ergeben, so ist dieser in der Outputmenge eines jeden Settings enthalten, da dieser keine Vertauschungen benötigt. Die kleineren Outputmengen der Settings $(0, 4, 0)$ und $(0, 2, 2)$ erwiesen sich somit als vorteilhaft. In einer kleineren Lösungsmenge existieren weniger zulässige Entscheidungen für die künstlichen Ameisen, um schlechte bzw. teure Lösungen zu erstellen.
Ebenfalls unabhängig vom verwendeten Algorithmus stellte sich bezüglich der Instanzen 11 bis 45 hingegen $(4, 0, 0)$ als am besten geeignet heraus (siehe Abbildung 11). In „Setting-Vergleich.ods“ kann weiterhin nachvollzogen werden, dass dieses Setting, unabhängig davon, ob die minimalen oder durchschnittlichen Kosten betrachtet worden, in über 80% der Fällen das beste Resultat liefert.

Abbildung 10: Verteilung der durchschnittlichen relativen Erhöhung der Kosten im Vergleich zum Setting $(4, 0, 0)$ bezüglich der ersten 10 Instanzen. Ein jeder Graph setzt sich aus den Ergebnissen der beiden betrachteten Algorithmen zusammen. Je untersuchtem Setting sind die minimalen und die durchschnittlichen gefundenen Kosten dargestellt.



Zusammenfassend sollte für Probleme, welche den ersten zehn Instanzen ähneln, welche also sehr gute Lösungen mit sehr wenigen Vertauschungen besitzen, das Setting $(0, 4, 0)$ gewählt werden. Sollten die zu bearbeitenden Queues eher der Struktur der Beispiele 11 bis 45 entsprechen, sollte $(4, 0, 0)$ gewählt werden, da die Gesamtkosten durch die größere Outputmenge teils sehr stark

Abbildung 11: Eine zu Abbildung 10 analoge Darstellung der Ergebnisse bezüglich der Instanzen 11 bis 45.



reduziert werden können.

4.3.5 Lösungsqualität

Abschließend soll in diesem Kapitel die Lösungsqualität der Algorithmen an kleineren Instanzen untersucht werden. Hierfür wurden die Programme „Optimum_400“ und „Optimum_040“ geschrieben, welche durch die Berechnung aller zulässigen Lösungen jeweils den optimalen zugehörigen Output einer Queue im Setting $(4, 0, 0)$ bzw. $(0, 4, 0)$ bestimmen. Um die Berechnung zu beschleunigen, wurden zusätzlich folgende zwei Aspekte berücksichtigt:

1. Falls eine Lösung mit Kosten in Höhe von 0 gefunden wurde, kann die Berechnung abgebrochen werden, da dieser optimal sein muss.

2. In „Optimum_400“ muss nur die Hälfte aller $2^{|\sigma|}$ Splits berechnet werden, da die Maschinen 1 und 2 identisch sind und der Puffer vor der Aufteilung plaziert ist.

Beide Programme samt Quellcodes befinden sich im Verzeichnis "Optima_400_040” auf der CD.
Durch die in der Einleitung des Kapitels 3 angesprochenen, sehr großen Outputmengen, ist es nicht möglich, die Optima der in Kapitel 4.3.1 bis 4.3.4 verwendeten Benchmark-Sets zu berechnen. Stattdessen werden an dieser Stelle 27 neue Beispiel-Queues generiert, wobei die gleichen Kombinationen aus $TF \in \{\frac{2}{5}, \frac{3}{5}, \frac{4}{5}\}$ und $RDD \in \{\frac{1}{5}, \frac{3}{5}, 1\}$ gewählt werden, allerdings nur jeweils drei Instanzen pro Kombination generiert worden. Diese neuen Queues bestehen aus je zehn Aufträgen, sodass ihr optimaler zugehöriger Output in einem vertretbaren Zeitaufwand berechnet werden kann.

Anschließend wurden pro Beispiel in den Settings $(4, 0, 0)$ und $(0, 4, 0)$ jeweils die minimalen und durchschnittlichen Kosten des pACO über 25 Neustarts berechnet. Dabei wurde für die Instanzen 1 bis 9, welche die ersten 15 Benchmark-Beispiele repräsentieren, der pACO ohne Heuristik mit der relativen Pheromon-Regel und für die restlichen Beispiele der pACO zusammen mit der Summations- und IMDD-Regel verwendet. Zur Auswertung wurde untersucht, wie oft die gefundene beste Lösung dem Minimum entspricht und wie groß die durchschnittliche relative Abweichung zwischen bester bzw. durchschnittlicher Lösung und dem Minimum ist.

Tabelle 12: Evaluation der Lösungsqualität des pACO im Vergleich zur Optimallösung bei Queues der Länge 10. Dargestellt wird, wie häufig das jeweilige Minimum gefunden wurde und wie stark die gefundenen minimalen bzw. durchschnittlichen Lösungen im Durchschnitt vom Optimum abweichen.

	$(4, 0, 0)$	$(0, 4, 0)$
Minimum gefunden	$63,0\%$	$77,8\%$
Abweichung zw. bester gefundener Lösung und Minimum	$0,46\%$	$0,31\%$
Abweichung zw. durchschnittlicher gefundener Lösung und Minimum	$3,80\%$	$6,69\%$

Wie in Tabelle 12 ersichtlich ist, wurde die optimale Lösung in 63% der Fälle in $(4, 0, 0)$ und in $77,8\%$ der Fälle in $(0, 4, 0)$ gefunden. Die relativen Abweichungen zum Minimum sind ebenfalls sehr gering; die Variationen des pACO liefern folglich sehr gute Resultate bezüglich dieser kurzen Queues. Daraus lässt sich jedoch im Allgemeinen keine Aussage darüber ableiten, wie nah die gefundenen Lösungen aus Kapitel 4.3.1 bis 4.3.4 an den zugehörigen Optima liegen.

4.3.6 Zusammenfassung

Der konzipierte Ameisenalgorithmus samt seiner Variationen wurde an einer Reihe von 45 Benchmark-Beispielen getestet. Dabei stellte sich heraus, dass eine gleichstarke Wichtung zwischen Pheromon und der Heuristik, einer schwächeren Wichtung der heuristischen Informationen vorzuziehen ist.
Die Resultate bezüglich der verwendeten Heuristiken sind abhängig von den Parametern RDD und TF der Benchmark-Instanzen. Es wurde festgestellt, dass für die ersten zehn Beispiele eine Vielzahl von Outputs ohne Gesamtverspätung existieren, sodass es hauptsächlich darauf ankommt, möglichst wenig Umsortierungen vorzunehmen. Die Heuristiken berücksichtigen diese Tatsache allerdings nicht, wodurch hohe Werte $\eta_{i,j}$ unabhängig davon entstehen, ob dabei eine Vertauschung stattfindet oder nicht. Das heißt, dass durch die Verwendung einer Heuristik für die künstlichen Ameisen häufig ein starker Anreiz entsteht,

Vertauschungen der Aufträge durchzuführen, wodurch jeweils Pufferkosten resultieren. Dieses Problem hätte eventuell umgangen werden können, wären die Pufferkosten nicht extern hinzugefügt, sondern durch zusätzliche Zeiteinheiten in der Bearbeitungszeit modelliert worden.
Für die ersten 15 Beispiele lieferte der pACO ohne Heuristik die geringsten Kosten und für die restlichen 30 erzielte der pACO mit der IMDD-Regel die besten Resultate. Die Heuristiken auf Basis einer Kriteriumsfunktion verschlechtern hingegen die Ergebnisse in fast allen Fällen, da sie zu keinem Zeitpunkt die bisherige Lösungskonstruktion miteinbeziehen und somit zu statisch konzipiert sind. Für die TS- und die $t\uparrow$-Heuristik skalieren zusätzlich einzelne Parameter nicht gut für die betrachteten Instanzen.
Das relative Pheromon-Evaluationsverfahren ergab in den Tests für den pACO ohne Heuristik die besten Resultate. Für die Variante des pACO mit der IMDD-Regel setzte sich hingegen die Summationsregel gegenüber den anderen Evaluationsverfahren durch.
Darauffolgend wurden die Resultate beispielhaft in Abhängigkeit der Settings $(4, 0, 0)$, $(0, 4, 0)$ und $(0, 2, 2)$ untersucht. Es stellte sich heraus, dass die Algorithmen in den Beispielen 11 bis 45 stark von der größeren Outputmenge des Settings $(4, 0, 0)$ profitieren, für die ersten zehn Beispiele jedoch vor allem $(0, 4, 0)$ zu bevorzugen ist. Grund hierfür ist, dass in diesen Instanzen sehr gute Lösungen existieren, welche keine oder kaum Pufferung benötigen. Aus der größeren Lösungsmenge des Settings $(4, 0, 0)$ resultiert somit lediglich eine größere Anzahl an schlechten Entscheidungsmöglichkeiten für die künstlichen Ameisen.
Abschließend wurden die besten Varianten des pACO an kleineren Beispiel-Instanzen erneut getestet und mit deren Optimallösungen verglichen, wobei nur geringe Abweichungen festzustellen waren. Allerdings lassen sich diese Ergebnisse nicht ohne Weiteres auf größere Probleme übertragen, da die Lösungsräume mit der Länge der Queues exponentiell an Größe zunehmen.
Weiterhin ist anzumerken, dass die obere Grenze der Einträge von $\mathbb{P}$ mit $k + \tau^0$ im Vergleich zu $\tau^0 \in (0, 1]$ zu groß ist. Die $\tau_{i,j}$, welche zu einer bestimmten Lösung gehören, können hierdurch sehr groß werden, was die Lösungsexploration und somit die Qualität des Algorithmus mindert. Um dies zu vermeiden, hätte es sich empfohlen, die $\tau_{i,j}$ zusätzlich zu beschränken, beispielsweise durch $\min\{1, k + \tau^0\}$.

5 Zusammenfassung und Ausblick

Das Ziel der vorliegenden Arbeit war es, einen Ameisenalgorithmus für eine spezielle Klasse von Scheduling-Problemen zu konzipieren, umzusetzen und durch eine Reihe von verschiedenen Heuristiken und Pheromon-Evaluationsverfahren zu verbessern. Neben der Entwicklung des Lösungsalgorithmus sollte weiterhin eine theoretische Untersuchung der Problemstruktur stattfinden. Die entworfenen Algorithmen sowie die gewonnenen theoretischen Erkenntnisse sollten abschließend in einigen Computerexperimenten verglichen und ausgewertet werden.
Die Untersuchung aus Kapitel 2 ergaben tiefe Einblicke in die Auswirkungen der Platzierung der Puffer auf die jeweilige Outputmenge eines Settings. Falls eine große Output- bzw. Lösungsmenge erzielt werden soll, so sollte vor allem der Puffer $P_1^{(k_1,k_2,k_3)}$ ausgebaut werden.
In den Experimenten zeigte sich, dass die selbst entwickelten Heuristiken nicht zur Steigerung der Lösungsqualität beitragen, da sie die Lösungskonstruktion nicht miteinbeziehen und teilweise nicht gut für die betrachteten Benchmark-Instanzen skalieren. Im Gegensatz dazu wirkte sich der Einsatz der IMDD-Regel auf viele der getesteten Beispiele sehr gut aus. Insgesamt setzten sich die Kombinationen aus IMDD-Heuristik und Summationsregel sowie der pACO ohne heuristische Informationen kombiniert mit der relativen Pheromon-Regel durch. Weiterhin konnte festgestellt werden, dass der Ameisenalgorithmus für Probleme, welche optimale Lösungen mit wenig Vertauschungen besitzen, von Settings mit kleinen Outputmengen profitiert, da er einen kleineren Lösungsraum durchsuchen muss. Andernfalls wirkt sich die größere Outputmenge sehr positiv auf die Ergebnisse aus. Die Untersuchungen bezüglich der Lösungsqualität bestätigen, dass der entwickelte Algorithmus (zumindest für kleinere Beispiele) sehr gute Lösungen findet, die nicht stark vom Optimum abweichen.

Neben den Puffer-Heuristiken konnte nicht auf Entscheidungshilfen bezüglich des Splits eingegangen werden. Ein solche Split-Heuristik könnte wie folgt lauten: Sende einen Job jeweils zu der Maschine, welche in diesem Moment die geringere Auslastung hat.
Weiterhin könnten die vorgestellten Kriteriumsheuristiken noch einmal überarbeitet und analog zur IMDD-Regel dynamischer gestaltet werden. Hierfür könnte jeweils anstelle der Fertigungszeit der (potentielle) Fertigstellungszeitpunkt gewählt werden.
Diese Modifikationen könnten sehr positive Auswirkungen auf die Lösungsqua-

lität des populationsbasierten Ameisenalgorithmus haben, konnten Im Rahmen dieser Arbeit allerdings nicht mehr vollzogen werden.

Literatur

[BDG06] BIANCO, L. ; DELL'OLMO, P. ; GIORDANI, S.: Scheduling models for air traffic control in terminal areas. In: *Journal of Scheduling* (2006), S. 223–253

[Blu05] BLUM, C.: Beam-ACO—hybridizing ant colony optimization with beam search: an application to open shop scheduling. In: *Computers and Operations Research* (2005), S. 1565–1591

[CSLG06] CHONG, C. S. ; SIVAKUMAR, A. I. ; LOW, M. Y. H. ; GAY, K. L.: A bee colony optimization algorithm to job shop scheduling. In: *Winter Simulation Conference*, 2006, S. 1954–1961

[DBS06] DORIGO, M. ; BIRATTARI, M. ; STÜTZLE, T.: Ant Colony Optimization. In: *IEEE Computational Intelligence Magazine* (2006), S. 28–39

[GM02] GUNTSCH, M. ; MIDDENDORF, M.: A Population Based Approach for ACO. In: *Applications of Evolutionary Computing 2279* (2002), S. 72–81

[Gon11] GONDEK, V.: *Hybrid Flow-Shop Scheduling mit verschiedenen Restriktionen: Heuristische Lösung und LP-basierte untere Schranken*, Universität Duisburg-Essen, Diss., 2011

[Her13] HERRMANN, M.: *Reihenfolgeprobleme bei zweistufiger Fertigung: mathematische Analyse und Entwurf von Lösungsheuristiken*, Universität Leipzig, Diplomarbeit, 2013

[HL08] HUANG, K. ; LIAO, C.: Ant colony optimization combined with taboo search for the job shop scheduling problem. In: *Computers and Operations Research* (2008), S. 1030–1046

[Len15] LENZEN, M.: *Das zweite Buch der Natur*. http://bit.ly/1NkDWbf, 15.08.2015. – Online; aufgerufen am 12.10.2015. Ein Screenshot des Artikels befindet sich auf der CD.

[MM00] MERKLE, D. ; MIDDENDORF, M.: An ant algorithm with a new pheromone evaluation rule for total tardiness problems. In: *Proceedings of the EvoWorkshops 2000 Lecture Notes in Computer Science*, 2000, S. 287–296

[MM01] MERKLE, D. ; MIDDENDORF, M.: On the behaviour of ant algorithms: Studies on simple problems. In: *Proceedings 4th Metaheuristics International Conference*, 2001, S. 573–577

[VRM08] VALLADA, E. ; RUIZ, R. ; MINELLA, G.: Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. In: *Computers and Operations Research 35* (2008), S. 1350–1373

Abbildungsverzeichnis

Tabellenverzeichnis

CD-Inhalt

Quellcodes-und-Programme

Alle verwendeten Programme samt zugehöriger Quellcodes befinden sich im Verzeichnis "Quellcodes-und-Programme".

Computerexperimente

Im Verzeichnis "Computerexperimente" befinden sich sowohl die jeweiligen Versuchsdaten, als auch die zugehörigen Analysen und Auswertungen.

Literatur

Die im Literaturverzeichnis angegebenen Quellen sowie die Diplomarbeit im PDF-Format befinden sich im Verzeichnis "Literatur".

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Ort, Datum	Martin Winkel