Compositional pareto-algebraic heuristic for packing problems

Montoya Aguirre, M.I.

Award date:
2016

Eindhoven University of Technology

Department of Mathematics and Computer Science

Compositional Pareto-algebraic Heuristic for Packing Problems

M.I. Montoya Aguirre

supervised at TU/e by

Marc Geilen
Joost van Pinxten
supervised at Vanderlande Industries by

Gaston Weijenberg
Kostas Alogariastos

May 23, 2016

Abstract

This study investigates the applicability of Compositional Pareto-algebraic Heuristic (CPH) to packing problems at Vanderlande. The two problems under study are Luggage Batch Selection (LBS) and Orderline Allocation (OA). These problems belong to the domain of multi-objective combinatorial optimization and are currently solved using Meta-heuristics such as Simulated Annealing and Genetic Algorithms. We show that these problems are compatible with CPH and explain the required modifications to the heuristic for each of the problems. Two individual CPH frameworks are presented, one for each of the case studies. The implementations of the CPH frameworks obtained valid solutions for both problems with solution quality comparable to the meta-heuristic methods used by Vanderlande. As other heuristics, CPH can be adjusted to favor solution quality or execution time. For Orderline Allocation (OA), the CPH framework allows a reduction of the execution time of two orders of magnitude by compromising on average 2% in solution quality. Even when adjusting CPH to aim for solution quality, the solutions provided for both problems were inferior to those obtained by meta-heuristics.

Dedication

To my family, for their unconditional support, love and encouragement.

Preface

This thesis is the result of my graduation project at Vanderlande Industries. It builds on work done by Hamid Shojaei, Marc Geilen, Twan Basten and Joost van Pinxten on Pareto Algebra, multi-objective optimization and Compositional Pareto-algebraic Heuristic (CPH). The graduation project was conducted under the supervision and guidance of Gaston Weijenberg and Kostas Alogariastos at Vanderlande and mentored by Marc Geilen and Joost van Pinxten at TU/e.

Contents

Chapter 1

Introduction

This thesis presents efforts to adapt and develop heuristics for combinatorial optimization problems. These are practical problems whithin real processes in key areas of Vanderlande. The analysis contains problem descriptions and formalization. A literature review was conducted in order to find feasible methods for solving these problems. A short summary of the different methods is presented. The method chosen for these problems, Compositional Pareto-algebraic Heuristic (CPH), is explained in a more detailed fashion. Consequently, CPH is adapted to tackle both of these problems. Lastly the applicability of this method is evaluated for each of the problems.

1.1 Problem descriptions

In this section, we will discuss the practical problems under study. These problems are from two different areas: luggage handling and warehouse automation. Both problems are combinatorial optimization problems with similar characteristics. In particular, both problems require selection and packing of items in containers of finite size.

1.1.1 Luggage Batch Selection

The first case study is set in the context of baggage handling. The case focuses on improving the loading process in large commercial airplanes. In this type of aircraft, luggage items are loaded into containers. Then these containers are loaded into the plane. During this process it is important to reduce the packing time and maximize the usage of these containers. Thus, the objective is having full containers that are easy to pack.

The automated airport baggage system is in charge of selecting the luggage items for each container. The luggage items are received in a streaming fashion (as groups of bags are checked-in by passengers). The system first evaluates the dimensions and weight of luggage items as they enter the system. Then, when

enough items are available, the system selects a subset of items in order to fill a container. The main objective when selecting these items is to use most of the container's capacity. Once the items are selected, they are sent out of the system to pack the container. The process continues by repeating the previous steps until all the luggage of a flight is packed.

While the main objective is to maximize container utilization, there are other secondary objectives. It is desirable to pack all items of a passenger in a single container. For security reasons, if a passenger misses his flight, all luggage items from this passenger are removed from the plane. If the luggage of that passenger was loaded in different containers, more than one container would have to be removed from the plane. Thus, it is better to avoid this situation by ensuring that only one container will be removed in case of a passenger absence.

The other secondary objective is balancing the proportion of small and big luggage items within a container. Usually when packing a container, the big items are stacked first to form a stable foundation for the smaller items. There are other reasons for balancing the size distribution of bags in the container. For instance, having a container with mostly large items may render the packing task more difficult since there are no small items to pack in the tight spots between large items. On the other hand, packing a container full of small items may take longer due to the large number of luggage items. Finally, large bags are handled with machines and workers handle small items by themselves. If the containers have mainly one type of items, the result is having the unwanted scenario of idling workers or idling machines.

1.1.2 Orderline Allocation for Load Forming Logic

The second problem of this assignment is set in the context of warehousing for retail. Warehousing handles the fulfillment of store items. During this process, large product lots are received from the manufacturers or producers. Then these lots are unpacked and stored. Afterwards, some of these products are used to fulfill individual store orders. The challenge is to fulfill orders using as few carriers (containers) as possible while packing the items in a store-friendly way. A store friendly carrier will contain mostly order lines (products and quantities) that share the same shelf or are located closely. When the carriers are received by the store, the employees re-stock the shelves with these products. It is easier for workers if the carriers contain items whose locations are close to each other and which belong to the same item families.

Because of store policies, there are some items that should never share a carrier. For example, fruits and cleaning products should not be packed in the same carrier. It is also detrimental to split an order line between carriers. However, sometimes the order line exceeds the carrier capacity. In this case, the only way to pack the order line is to fill one carrier with as many of those items as possible and pack the remaining items in another carrier. As a result, that order line will be split in multiple carriers.

This is a Bin Packing Problem because we must pack all items from a set in as few containers as possible. The full set of items is known before any of the packing is done. Once the bin configurations are calculated, a second test is

required. This test checks if the selected configuration is stackable in 3D space. In some cases, configurations that comply with volume and weight constraints can't be arranged in a way that fits the carrier. In this case a new configuration must be selected. When the selected configuration passes the stacking test, the order is fulfilled by the automated warehouse system.

1.2 Combinatorial optimization problems

Many combinatorial optimization problems have been widely studied in the available literature. Some of these problems share many characteristics with the ones tackled in this project. The literature problems with similarities include the knapsack problem and the bin packing problem. Thus, it is beneficial to evaluate the approaches used in the literature when solving these problems. From this review, some methods and approaches for solving packing problems will be selected for further evaluations and eventually matching them to the problems under study in this project.

1.2.1 Knapsack problem

The knapsack problem is about selecting a group of items and packing them into a knapsack. Each item has a value and a weight. The knapsack has a maximum capacity. The objective is to select the items that maximize the knapsack value without exceeding the weight capacity.

It is formally defined, given:

- A knapsack with capacity $W \in \mathbb{N}$
- A set of items $I = \{i_1, \ldots, i_n\}$ available for packing in the knapsack
- Each item i_k has a weight w_k, and a value v_k

$$x_k = \begin{cases} 1, & \text{if item } i_k \text{ is packed} \\ 0, & \text{if item } i_k \text{ is NOT packed} \end{cases}$$

$$maximize \sum_{i_k \in I} v_k x_k \tag{1.1}$$

such that:

$$\sum_{i_k \in I}^{n} w_k x_k \leq W \tag{1.2}$$

1.2.2 Bin packing problem

The bin packing problem is about packing a group of items into a series of bins. Items have a weight and bins have a maximum capacity. The objective is to minimize the number of bins used. It is required to pack all items. The capacities of the bins cannot be exceeded.

Given:

- A set of items $I = \{i_1 ... i_n\}$ must be packed into a set of bins $B = \{b_1 ... b_k\}$
- Each item has a weight w_i
- Each bin has a capacity c_b

$$minimize \sum_{b=1}^{k} y_b \tag{1.3}$$

$$y_b = \begin{cases} 1, & \text{if } \sum_{b=1}^{k} x_{ib} > 0 \\ 0, & \text{if } \sum_{b=1}^{k} x_{ib} = 0 \end{cases}$$

$$x_{ib} = \begin{cases} 1, & \text{if item } i \text{ is assigned to bin } b \\ 0, & \text{if item } i \text{ is NOT assigned to bin } b \end{cases}$$

$$\sum_{i=1}^{n} w_i x_{ib} \leq c_b \tag{1.4}$$

The weight of the items cannot exceed the capacity of the bin.

1.2.3 Computational complexity

It is important to emphasize that most combinatorial optimization problems become too computationally expensive as they scale. Increasing the number of items in a bin packing problems generates millions of possible combinations. The knapsack and bin packing problems (Optimization) are of NP-Hard Complexity [4]. Performing exhaustive search on all combinations becomes unfeasible as the problem scale in a context with limited computational or timing budgets [7]. There are some clever approaches that manage to obtain optimal solutions for similar optimization problems of small and medium size. The problems under investigation have multiple optimization objectives. This particularity turns them into more difficult problems. There are no efficient algorithms for optimally computing large scale multi-objective combinatorial problems. Most of the approaches that produce good results in acceptable execution times make use of some sort of heuristics.

1.3 Multi-objective Optimization and Pareto Algebra

In section 1.2, combinatorial optimization problems were introduced. The case studies are similar to those problems but are more complex because multiple criteria are optimized. In other words, the problems under study are multi-objective combinatorial optimization problems. The interesting aspect of these problems is that objectives for optimization are potentially dependent on each other and of conflicting nature. In this case judging the quality of the solutions

is not as trivial as having the greater or lower scalar value for maximization and minimization problems respectively.

For this type of problems, the concept of Pareto optimality is used to determine which solutions are better than others. Ideally, in multi-objective optimization problems the goal is to obtain a set of solutions that are better than the rest and represent the trade-offs between objectives. A solution dominates another solution when it is better in at least one of the objectives and at least equal in the rest of the objectives. The solutions that are not dominated form a set called the Pareto front. Heuristics for multi-objective combinatorial problems aim to find a set of solutions that approximates the Pareto front.

Chapter 2

Related work

2.1 Approximation Algorithms

Several algorithms have been explored to approximate a solution for the bin packing problem. Approximation algorithms focus on providing a solution in linear or log-linear time. There are many popular algorithms like "Best fit", "First fit", "Best fit decreasing" and "First fit decreasing". "First fit" places items in the first bin that can allocate the item. When there are many bins that could allocate the item, "Best fit" place the item in the bin that will become the tightest after item insertion. That is, the bin will be the one with the least available space when compared to the other options. There are variations on these algorithms that re-arrange the items in decreasing order according to their weights.

The ordering algorithms produce, in the worst case scenario, a $\frac{11}{9}OPT$ solution in at most $O(n \log_n)$. The non-sorting versions produces, in the worst case scenario, $\frac{17}{10}OPT$ solution in at most $O(n)$ operations.[6]

2.2 Exact Algorithms

Many exact algorithms for packing problems take advantage of upper and lower bounds computations to steer the heuristics into the right direction and discard partial solutions that will not lead to optimal solutions[3]. A key part of this methodology is the use of Linear programming to solve a relaxation of the problem. Then using this solution to construct a feasible solution for the problem. This process is called rounding. After rounding, the process continues with heuristics that explore the new reduced solution space.This process is followed by pruning of the solution space tree. This method was designed for solving the cutting stock and knapsack problems.

2.3 Compositional Pareto-algebraic Heuristics

Compositional Pareto-algebraic Heuristic (CPH) provides a fast and scalable method of solving combinatorial optimization problems [9][10] It is based on Pareto algebra principles. It was originally designed to solve instances of the Multi-dimensional multi-objective knapsack problem. CPH computes solutions in incremental fashion while getting rid of partial solutions that will not lead to optimal solutions.
The key elements of CPH are the configurations. Items of the knapsack problem are converted into configurations. These configurations are composed of value dimensions and resource dimensions. These configurations are then combined to create new configurations. The combining process is called Product Sum. This process is what takes most of the heuristic time, therefore, it is desirable to have configurations that are inexpensive to merge. This process will produce more configurations each time. In a large problem, the number of configurations might grow too large, making it too computationally expensive to find a solution. CPH deals with this problem by using Pareto minimization, a simple technique to eliminate inferior configurations. By removing inferior configurations, Pareto algebra shows that the optimal solutions are never removed and that configurations that cannot lead to Pareto-optimal solutions are removed as early as possible.

2.4 Bin Packing Problem Under Multiple Objectives: a Heuristic Approximation Approach

This section introduces a heuristic approximation approach to the Bin Packing Problem (BPP) under multiple objectives.[5] It deals with at least two conflicting objectives such as minimizing the number of bins used and minimizing bin heterogeneity. It produces a solution space that shows the trade-off between these objectives. The algorithm is based on the "best fit" approximation algorithm with the addition of a method for controlling the secondary objective (bin heterogeneity). A variable called U_{max} set the maximum heterogeneity allowed for bins. The algorithm is run several times while increasing U_{max}, and the best results, considering a single objective optimization, are saved and aggregated. Finally the set of results is minimized using Pareto minimization.

Chapter 3

Formalization

In previous chapters we defined the case studies. We also introduced literature problems similar to the ones in the case studies. This chapters makes the connection between the practical problems and the formalization necessary to define the case studies as multi-objective combinatorial optimization problems.

3.1 Luggage Batch Selection

In the Luggage Batch Selection scenario we start with a set of bags. We must select a subset of these bags for packing in a container of a fixed size. This container is easier to pack if the corresponding batch is composed by bags that comprise the whole range of weights and volumes. It is acceptable to have bags of similar characteristics but overall, weights and volumes should vary. Besides optimizing the container utilization, it is important that the batch of bags has similar statistic indicators as the ones from the initial bag set. The purpose of having a similar distribution is to have subsequent batches of quality comparable to the batch currently being created.

In other words, we have a knapsack problem with weight and volume restrictions. However, the items don't have an intrinsic value. Instead, the value of the items in the knapsack is determined by characteristics that are affected by the rest of the items of the knapsack. The value is determined by the total volume of the knapsack and how similar are the bag distributions of the knapsack and the set of remaining bags (the ones that were not selected to be in the Knapsack).

Given:

- A knapsack with capacities $W \in \mathbb{N}$ for weight and $V \in \mathbb{N}$ for volume.
- A set E_{bs} containing items $\{i_1 ... i_n\}$ available for packing the knapsack
- Each item i has a weight w_i, and a volume v_i. Each item represents a group of 1 or more bags. Normally it represents all bags from a passenger. Note: this group is treated as a single knapsack item to turn one of the

objectives into a hard constraint (It is preferred that all the bags from a passenger go in the same knapsack).

As defined before, the objective is to maximize the knapsack value while meeting the capacity constraints.

The weight of the knapsack cannot exceed its capacity:

$$\sum_{i=1}^{n} w_i x_i \leq W \tag{3.1}$$

The total volume cannot exceed the volume capacity of the knapsack:

$$\sum_{i=1}^{n} v_i x_i \leq V \tag{3.2}$$

The LBS problem has the goals of maximizing the similarity between statistical properties of the group of items in the knapsack and the items in the E_{bs}, and maximizing the knapsack utilization. To achieve these goals, the knapsack value is composed by 4 value dimensions. At the beginning of the batch, target values for these dimensions are calculated from the characteristics of the E_{bs}, such as standard deviation, average volume and estimated number of bags per batch. Additionally, the target volume is set as the maximum volume of the knapsack. The value dimensions are defined as the distance between the target quantities and the current knapsack quantities on:

- Number of bags contained in the selected items. 3.3
- Average weight of the bags contained in the selected items. 3.4
- Volume of the items in the knapsack. 3.5
- Standard deviation on volume of the bags contained in the selected items. 3.6

$$x_i = \begin{cases} 0, & \text{if item } i \text{ is NOT in the knapsack} \\ 1, & \text{if item } i \text{ is in the knapsack} \end{cases}$$

$$\text{Number of bags} = \sum_{i=1}^{n} x_i \tag{3.3}$$

$$\text{Average weight in the knapsack} = \frac{\sum_{i=1}^{n} w_i x_i}{Numberofbags} \tag{3.4}$$

$$\text{Volume of the items in the knapsack} = \sum_{i=1}^{n} v_i x_i \tag{3.5}$$

$$\text{Standard Deviation on Volume} = \sqrt{\sum_{i=1}^{n} (v_i x_i - \mu)^2}, \mu = \frac{\sum_{i=1}^{n} v_i x_i}{n} \tag{3.6}$$

Since the actual value is in the comparison of these values with the goal values (obtained from the E_{bs} set), a difference and absolute value computation is in order.

$$\text{Value} = |\text{Knapsack Property} - \text{Goal Property}| \tag{3.7}$$

3.2 Orderline Allocation for Load Forming Logic

The Orderline allocation problem is a version of BPP. For this reason, mapping it to CPH requires more creativity than the batch selection problem. Instances of CPH must have a fixed solution space, this is, the space shouldn't be infinite. For this reason we restrict the number of carriers/bins at the start of CPH.When mapping LFL into CPH, some of the objectives are transformed into monotonic properties.

Given:

- A set of items $I = \{i_1, \ldots, i_n\}$ available for packing in a set of bins $J = \{j_1, \ldots, j_m\}$.
- Each item i_k has a weight $w_k \in \mathbb{Z}^+$ and belongs to a partitioning(group) $q_k \in Q = \{q_1, \ldots, q_r\}$
- Each item has a volume $v_k \in \mathbb{Z}^+$
- Each bin has a fixed capacity $c \in \mathbb{Z}^+$. All bins have equal capacity.

The objectives of the optimization problem are the following:

$$\textbf{Maximize: } \text{Volume Utilization } = \frac{\text{Theoretical Minimum Volume}}{\text{Actual Volume}} \tag{3.8}$$

For the case study, all bins have equal capacity. Therefore this would be the same as minimizing the number of bins in use.

$$\text{Theoretical Minimum Volume} = \left\lceil \frac{\sum_1^n v_k}{c} \right\rceil c$$

It is theoretical because items are seen as fluids and exactly fit bins without wasting additional volume. Theoretical Minimum Volume is only achievable in some cases.

$$\text{Actual Volume } = \sum_{k=1}^{n} y_k c$$

$$y_k = \begin{cases} 0, & \text{if bin } j_k \text{ is NOT used} \\ 1, & \text{if bin } j_k \text{ is used} \end{cases}$$

$$\textbf{Maximize: } \text{partitioning score } = \frac{\sum_{q_k \in Q} \text{Volume Utilization of partitioning } q_k}{N} \tag{3.9}$$

Constraints of the problem:

$$\sum_{1}^{n} w_i x_{ij} \leq c_j y_j \tag{3.10}$$

The weight of the items cannot exceed the capacity of the bin.

$$x_{kl} = \begin{cases} 0, & \text{if item } i_k \text{ is NOT assigned to bin } j_l \\ 1, & \text{if item } i_k \text{ is assigned to bin } j_l \end{cases}$$

$$\forall i_k \left[i_k \in I : \sum_{1}^{m} x_{kl} = 1 \right] \tag{3.11}$$

Every item must be packed in a bin. An item cannot be packed in more than one bin.

Chapter 4

CPH for Packing Problems

Compositional Pareto algebraic heuristics provides a fast and scalable method of solving combinatorial optimization problems. The problems under study are currently solved by using meta-heuristics such as genetic algorithms and simulated annealing. In both cases there are time and scale constraints that make it unfeasible to compute an optimal solution. Thus, a good enough approximation is the only option. It is worth investigating whether CPH is a better approach in terms of computational complexity, solution quality or both. Additionally, there are some particularities within the problems that could benefit from the compositionality of CPH.

4.1 The CPH Framework

The CPH framework consists of several steps used to formulate an optimization problem and find solutions. The first step is to convert an optimization problem into a CPH representation, called CPH instance. Configurations are the atomic units of CPH. They are very flexible because they can be used to represent single items, partial solutions and complete solutions of the CPH instance. Configurations hold information about the resources used by the solution and the corresponding value.

The options available in the problem, whether it is choosing items for the knapsack or the bin to allocate an item, are used to create a set of configurations which will then be used by CPH to search the solution space. This search is done using the Product-sum, bound, minimize operations. These three operations are the core of CPH.

The product-sum operation is used to combine sets of configurations. This is how we explore the solution space in an incremental fashion, by combining small parts of the problem rather than using complete solutions. The bound operation checks whether the solutions available violate any of the resource constrains. If they do, then those configurations are discarded by the operation. The minimize operation uses Pareto algebra and the dominance relations defined in the CPH instance to discard configurations that are dominated by others. Pareto

minimization works great in problems that are monotonic because configurations that are dominated will never reach Pareto-optimal solutions. Both case studies lack monotonicity, this will be an interesting challenge for CPH.

The key part of applying the framework to an optimization problem is constructing an appropriate CPH model that captures enough information about the choices and combines configurations in a efficient way. The model should be accurate enough so that a CPH solution is easily mapped back to a problem solution. The model must contain all the key elements that are evaluated in a problem solution. And lastly, the model should be small and light enough so that the operations such as Product-sum and Pareto minimization are efficient.

4.2 CPH for Luggage Batch Selection

Mapping the Luggage Batch Selection (LBS) problem to CPH appears to be a straight forward task because of the similarity with the knapsack problem. However there are some characteristics of the batch selection problem that require extra caution. The non-monotonic objectives and the fact that volume increases value and resource usage reduce the effectiveness of CPH. Using objectives that are non-monotonic will increase the chances of discarding too many partial solutions, early in the process. Despite those complications, the problem has multiple value and resources dimensions. This characteristic will reduce the chances of discarding promising configurations.

The configurations from this problem will contain four value dimensions:

- Distance to maximum volume: Less is better.
- Distance to average weight: Less is better.
- Distance to standard deviation on volume: Less is better.
- Distance to the preferred Number of bags: Less is better.

Besides the value dimensions, resource dimensions will keep track of the allowance of weight and volume in the containers:

- Volume.
- Weight.

Configurations should be as small and lightweight as possible in order to achieve efficient computation of partial solutions. They contain data for value dimensions, resource dimensions and special attributes to keep track of the state of the solution. For example to know which groups of bags have been selected for one of the final solutions.

- Number of bags in the knapsack.
- Average weight in the knapsack.
- Standard deviation on volume of the knapsack.

- Configurations (Groups of bags) used to produce the current configuration.

4.2.1 Pre-processing and creation of configuration sets

Pre-processing is required to transform a list of luggage into a CPH instance. In this process, two types of configurations are produced. These types are single bag configuration and multiple bag configurations. This distinction is done because in some cases, passengers travel with multiple bags. The only difference between these types of bags is that the single bags items have a default standard deviation on volume of zero. All the bags from a single passenger are grouped into a CPH configuration.

Table 4.1: Initial bag list

PassengerID	Volume [L]	Weight [kg]
1	83.30	9.50
1	68.10	9.50
2	113.60	22.00
3	36.10	6.00
3	81.00	16.50
3	104.60	19.00

To create those configurations, the value and resource dimensions must be filled using the characteristics of the bags in the configurations. Utilization, volume and weight are very straight forward because they are set by the sum of volumes and weights of the bags in the configurations and the size of the containers. However, for the rest of the value dimensions, we need target quantities. These quantities are calculated from the set of items available in the electronic bag storage at the moment of start of the batch creation process. These calculations include standard deviation, average weight and the preferred number of bags in the knapsack.

Table 4.2: Goals obtained from Table 4.1 using maximum container Volume $250l$

STDEV(V)	AVG(W)	Bags
27.59	13.75	3.08

Once these quantities are known, they are then compared to the corresponding statistics in the configurations. For single bag configurations, the standard deviation on volume is zero, so instead of combining them directly, they are calculated using the other attributes of the new configuration.

Table 4.3: CPH configuration list

PassengerID	Bags	Volume	Weight	σ(v)	AVG(w)	Δ σ(v)	ΔAVG(w)	ΔBags
1	2	151.40	19.00	10.75	9.50	16.85	4.25	1
2	1	113.60	22.00	0.00	22.00	27.59	8.25	2
3	3	221.70	41.50	34.80	13.83	7.20	0.08	0

4.2.2 Combining configurations

Combining configurations is the core process of the product sum operation. The process of combining two configurations should be computationally inexpensive, otherwise the benefits of CPH are diluted. In the context of luggage batch selection, combining configurations represents the action of adding a bag or set of bags to the container. It is important to emphasize that a configuration is an entity by itself and not a list of bags. It is this distinction that makes it efficient to compute the characteristics (value dimensions, resource dimensions and auxiliary dimensions) of a resulting con figuration.

Continuing with the example shown in section 4.2.1, we illustrate how the process of combining configurations works. From now on, we will identify configurations from 4.3 by their PassengerID column. When combining configurations 1 and 3, the outcome will be a new configuration with the characteristics obtained from the union of bags contained in configuration 1 and configuration 3. However, characteristics of individual bags are not available in a configuration, they were lost when we transformed the bag list into a configuration list. Thus, the only way of computing a new configuration is using the data available in configurations 1 and 3. For some dimensions, such as volume, bags and weight, it is as simple as adding the volume of configurations 1 and 3. For some other dimensions the process is not as simple but remains computationally inexpensive. This is the case for the calculation of average weight which is shown in equation 4.1

$$AVG(w)_{C3} = \frac{Weight_{C1} + Weight_{C2}}{Bags_{C1} + Bags_{C2}} \tag{4.1}$$

While most of the dimensions are very simple and inexpensive to compute, standard deviation is not as efficient because it requires the computation of square roots of real numbers. Another difficulty is that the formula for standard deviation requires the values for all elements in a sample. In this case those values would be the volumes of all bags in the configuration. Values, which are not available anymore. However, it is possible to calculate the standard deviation of a new group when the mean, number of elements and standard deviation of its partitions are know.

$$\sigma = \sqrt{E[(X - \mu)^2]} \tag{4.2}$$

Standard Deviation

$$\mu_{1:n} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{4.3}$$

Mean

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_{1:n})^2 = \frac{n-1}{n}\left(\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_{1:n})^2\right) \tag{4.4}$$

Variance

$$\begin{aligned}(m+n)(\sigma^2_{1:n+m} + \mu^2_{1:m+n}) &= \sum_{i=1}^{n+m} x_i^2 \\ &= \sum_{i=1}^{n} x_i^2 + \sum_{i=1+n}^{n+m} x_i^2 \\ &= n(\sigma^2_{1:n} + \mu^2_{1:n}) + m(\sigma^2_{1+n:m+n} + \mu^2_{1+n:m+n})\end{aligned}$$

$$\sigma^2_{1:m+n} = \frac{n(\sigma^2_{1:n} + \mu^2_{1:n}) + m(\sigma^2_{1+n:m+n} + \mu^2_{1+n:m+n})}{m+n} - \mu^2_{1:m+n} \tag{4.5}$$

$$STDEV(v)^2_{C_3} = \frac{Bags_{C_2}\left(STDEV(v)^2_{C_2} + (\frac{Volume_{C_2}}{Bags_{C_2}})^2\right) + Bags_{C_1}\left(STDEV(v)^2_{C_1} + (\frac{Volume_{C_1}}{Bags_{C_1}})^2\right)}{Bags_{C_1} + Bags_{C_2}} - \mu^2_{C_3} \tag{4.6}$$

$$\mu_{C_3} = \frac{Volume_{C_1} + Volume_{C_2}}{Bags_{C_1} + Bags_{C_2}} \tag{4.7}$$

With this formulation for the standard deviation, all the data required is available within the two configurations to be combined.

4.2.3 CPH Algorithm

The CPH algorithm uses as input the configuration set created from the flight bag list. As other heuristics, the purpose of the algorithm is to do a search of the solution space in a way that it finds close to optimal solutions while avoiding the sections of the solution space that do not provide valuable solutions. CPH maintains a set of partial solutions. The set grows as new configurations (from the configuration set) are combined with the ones in the Partial solution set. The algorithm takes one configuration from the initial configuration set and combines it with all the configurations that at that moment reside in the partial solution set. In this way, CPH explores the solution space.

Algorithm 1: CPH for LBS

Input : S a set of configurations, Rb resource bounds
Output: Result, a configuration set, of which any configuration is a valid LBS solution
Step 1 : Initialize set of partial solutions S_p and the set of configurations S
Step 2 : Perform incremental computations
forall the $S_i \in S$ **do**
    //Combine a configuration from the configuration set with the one in the partial solution set.
    S_p = ProductSum(S_p, S_i);
    //Discard the dominated configurations
    S_p = Pareto-minimize(S_p);
Step 3 : Return results: S_p

After each Product-sum operation (Combining a new configuration with all the partial solution set elements), there is a bound check. The algorithm discards all configurations that violate resource constrains. For example, exceeding the maximum volume of the container. So far we have a way of exploring the solution space. However, that search space is not reduced in any way. To reduce the search space and guide the solution creation towards superior configurations, Pareto Minimization is used. The value dimensions enumerated in section 4.2 are used to evaluate the dominance relations between configurations. The configurations that are strongly dominated by other configurations are discarded.

After all the configurations of the configuration set are combined, the partial solution set becomes a solution set. All the configurations in the set now represent a solution for the luggage batch selection problem.

4.3 CPH for Orderline Allocation

We present the modifications required in order to map the problem under study to CPH. CPH performs better when the decision variables have a monotonic effect on the results. To ensure that this is the case, utilization and partitioning will be redefined.

$$\text{Minimize: Utilization Volume} = \sum_{i=1}^{n} y_j c \tag{4.8}$$

$$y_j = \begin{cases} 0, & \text{if bin } j \text{ does NOT contain items} \\ 1, & \text{if bin } j \text{ contains items} \end{cases}$$

$$\text{Minimize: Partitioning Volume} = \sum_{k=1}^{N} \sum_{j=1}^{n} x_{kj} c \tag{4.9}$$

$$x_{kj} = \begin{cases} 0, & \text{if category } k \text{ is NOT assigned to bin } j \\ 1, & \text{if category } k \text{ is assigned to bin } j \end{cases}$$

With these new definitions of utilization and partitioning, adding items to the bins can only increase those values. Thus there are no instances in which utilization or partitioning can shift abruptly.

The configurations from this problem will contain three value dimensions:

- Utilization volume: Less is better
- Partitioning volume: Less is better
- Heterogeneity: Less is better.

Configurations will also have special attributes to keep track of the state of the solution. For example, to know how full the carriers are, which groups of items are in which bin and to provide a complete orderline allocation as a result of the CPH algorithm.

- Weight and Volume used in each bin.
- Bins where item groups(partitionings) are present.
- Heterogeniety of bins.
- Identifiers of configurations used when building the current configuration.

4.3.1 Pre-processing and creation of configuration sets

The data for the Load forming logic problem is derived from orders. The orders contain a list of Order lines. The Order is evaluated to estimate the lower bounds on utilization and partitioning. These estimations are used to assess the solution quality. In addition to evaluation the solutions, the bound on utilization is used to determine the number of carriers available for CPH. The number of carriers is calculated by adding an extra 25% of carriers to the number of carriers in the lower bound. This percentage was chosen because utilization bellow 75% is not within the accepted performance metrics.

An additional pre-processing step is to remove those order lines that exceed carrier capacity. Entire carriers are filled with a portion of these order lines and removed from the problem context. The remaining portion of the order line remains in the problem context. For example, a hypothetical order line has a weight of 500kg and the allowed weight per carrier is 400kg. Then 400kg (Or as close to 400kg as the box weights allow) will be removed from that order line (and packaged in a carrier out of the scope of the current optimization problem). The remaining part of the order line (100kg) will still be part of the optimization problem.

After those steps are completed, the order is transformed into a CPH instance. The decision variables in the problem indicate the location of the order lines. In other words, which order lines go in which carriers. Each order line will be represented by a set of CPH configurations. The number of configurations will

depend on the number of carriers designated for the bin packing problem. This number, as previously explained, is determined by the volume and weight of the entire order.

In addition to the previous steps, the index of the order lines are added to the configurations so that at the end of the CPH process a Load Forming Logic solution is generated from the CPH result.

4.3.2 Combining configurations

Combining configurations is the core process of the product sum operation. The process of combining two configurations should be computationally inexpensive and straight forward. In the context of Load Forming Logic, the weight and volume of the resulting configuration are the sum of the weights and volumes of the configurations being combined. Two data structures (dictionaries) are used to keep track of partitioning and heterogeneity. A dictionary contains the information of which bins contain a certain category. The dictionaries are cross-referenced and updated in the new configuration using the data from its parent configurations. It is the same case with another dictionary which holds the information necessary to compute the heterogeneity of the carriers. Updating the utilization volume consists of checking which carriers are used. This check is positive if the volume of a carrier is greater than zero in any of the configurations being combined.

4.3.3 CPH algorithm

The CPH algorithm uses as input the configuration sets created from the Order. Besides the configuration set, a partial solution set is created. This set is initially empty. The algorithm is composed by the product-sum operation, bound checking, pareto minimization and a reduction of the partial solution set. Each of these steps is performed every time a configuration set is processed.

The product-sum operation is used to explore the solution space. It combines configurations from two different configuration sets. In the case of Load Forming Logic (LFL), this operation is performed on the set partial solutions with a set of configurations from the configuration sets.

Algorithm 2: CPH for LFL Orderline Allocation

Input : S a set of configurations sets, Rb resource bounds, max number k of configurations in the partial solution set
Output: Result, a configuration set, of which any configuration is a valid OA solution
Step 1 : Initialize set of partial solutions S_p and the set of configurations S
Step 2 : Perform incremental computations
forall the *sets* $CS \in S$ **do**
  forall the *configurations* $c \in CS$ **do**
    //Reduce the number of intermediate configurations
    S_p = Reduce(S_p, k);
    //Combine the configuration from the configuration set with the ones in the partial solution set.
    S_p = ProductSum(S_p, c);
    //Discard the dominated configurations
    S_p = Pareto-minimize(S_p);
Step 3 : Return results: S_p

The set of partial solutions grows as a result of performing the product sum operation. This could also be visualized as a tree growing with the addition of nodes representing a placement for new order lines.

The next step in the algorithm is the Bounding process. The process inspects all the configurations in the partial solution set. It uses the resource bounds defined with the problem. The bounding process checks the volume and weight of all the bins in each configuration. If a configuration contains bins which weight or volume exceed the bounds, the configuration is removed from the partial solution set.

In the previous steps, CPH is exploring the space and removing invalid configurations. The following step is to guide the search towards the better results by discarding the least promising configurations. This is done by means of Pareto Minimization of the partial solution set. CPH evaluates the configuration's value dimensions: Partitioning Volume, Utilization Volume and Heterogeneity. Configurations in the partial solution set are discarded if they are dominated by other configurations in the set.

For LFL, strong dominance was used to determine which configurations would be discarded. This decision was based on the fact that LFL configurations may have the same values for utilization volume, partitioning volume and heterogeneity but represent a completely different orderline distribution that may lead to very different solutions. Weak dominance would discard a large portion of the configurations in the partial solution set, this will limit the search at an extreme extent and render the algorithm ineffective. While strong dominance makes it possible to reach a wider range of solutions, the partial solution set grows very large very fast. In most of our experiments, during CPH, the partial solution set grew to tens of thousands of configurations. At this point, the Pareto minimization step becomes too computationally expensive. For this reason, with each CPH step, the partial solution set is reduced to an arbitrary number of configurations.This number, the maximum number of configurations in the par-

tial solution set, is defined before starting CPH, and remains constant through the entire process.

Restricting the number of configurations that the set of partial solutions can hold is necessary for CPH in the LFL context. The selection is done by a geometric selection algorithm that uses the 2-d value-aggregate resource usage of the configurations[1], in this case utilization volume and partitioning volume to simplify the selection of configurations. This algorithm selects solutions that represent the trade-off between Partitioning volume and Utilization Volume by using the ratio between those objectives in the configurations. The maximum number of configurations is seen as a parameter to manage the trade-offs between solution quality and delay of the heuristic. As shown in [9] the larger the number of configurations allowed, the better solutions CPH will produce at the cost of larger delay.

Chapter 5

Performance evaluation

This chapter presents an analysis of the performance of both CPH implementations. A detailed description of the experiments is included. These experiments were conducted in order to measure the execution time and solution quality of CPH when processing real problems in LFL and LBS context. A short comparison of CPH and the current methods methods is provided at the end of sections 5.1 and 5.2.

5.1 Luggage Batch Selection

In this section we present a series of experiments conducted using CPH and the current solution for the LBS problem. We explain the flight data used for testing, experiments, and the methods using for judging the quality of the solutions provided by each approach.

5.1.1 Test Environment

The tests were conducted using a personal computer running Windows 7 Professional Edition (64 Bits). The system contained an i5-3340M CPU with two cores, each operating at 2.7GHz, and 12GB of RAM. CPH was implemented using C# and Visual Studio 2013. The current solution was implemented using Microsoft Visual Basic.

5.1.2 Data

The data used to experiment with the different aspects of CPH originated at one of the airports under Vanderlande commission. Three real data sets from actual flights were used in the experiments. Some preliminary tests were conducted using an artificial set of data that was obtained by merging information from two different flights and randomly assigning passenger IDs to some of the bags in the list.

5.1.3 Execution time and number of bags in the E_{bs}

The execution time and solution quality determine the effectiveness of an approach for solving an optimization problem. In the case of LBS the problem size scales with the number of bags available in the electronic bag storage. Usually the number of bags ranges between 20 and 450. When converted to CPH configurations we obtained at most 300 configurations for any of the flights in the flight data-set. With a larger problem, the number of options available increases and the execution time for the heuristic increases as well.

To test the ratio at which the execution time increases compared to the number of items processed, we ran 1000 instances of CPH a fixed number of randomly selected configurations. We conducted this experiment using 50, 100, 200 and 300 configurations. Table 5.1 shows the average execution time, the duration of the longest and shortest runs and the standard deviation on execution time. The baseline, at 50 configurations, had an average running time of 16 milliseconds. The number of configurations were double in each of the two following columns. The increase in execution time was only 12.5% and 11.1% respectively. This would indicate that the execution time increases by at a much lower rate, almost by an order of magnitude, than the problem size. The exception is the the last column at 300 configurations where we get an increase of 40% in execution time with a 50% increase of problem size.

Table 5.1: Execution times of CPH for problems containing 50, 100, 200 and 300 configurations. Statistics of 1000 runs at each number of configurations with shuffled input.

# Configurations	**50**	**100**	**200**	**300**
AVG [ms]	16	18	20	28
MAX [ms]	622	849	671	988
MIN [ms]	<1	<1	<1	1
STDEV [ms]	51	65	55	65

The fact that average execution time grows at slower rate than the problem size is a desirable quality. The problem with CPH is the high variability of the execution time. The slowest runs took between 33 and 47 times the average time. Since the average execution time is skewed because of the high variation and disproportionately large outliers, a deeper look into execution time is shown in figure 5.2. The column on the left shows the probability for CPH to finish in each time interval and the column on the right shows the cumulative distribution function of the probability of completion over time. In most cases, CPH will complete before the average running time.

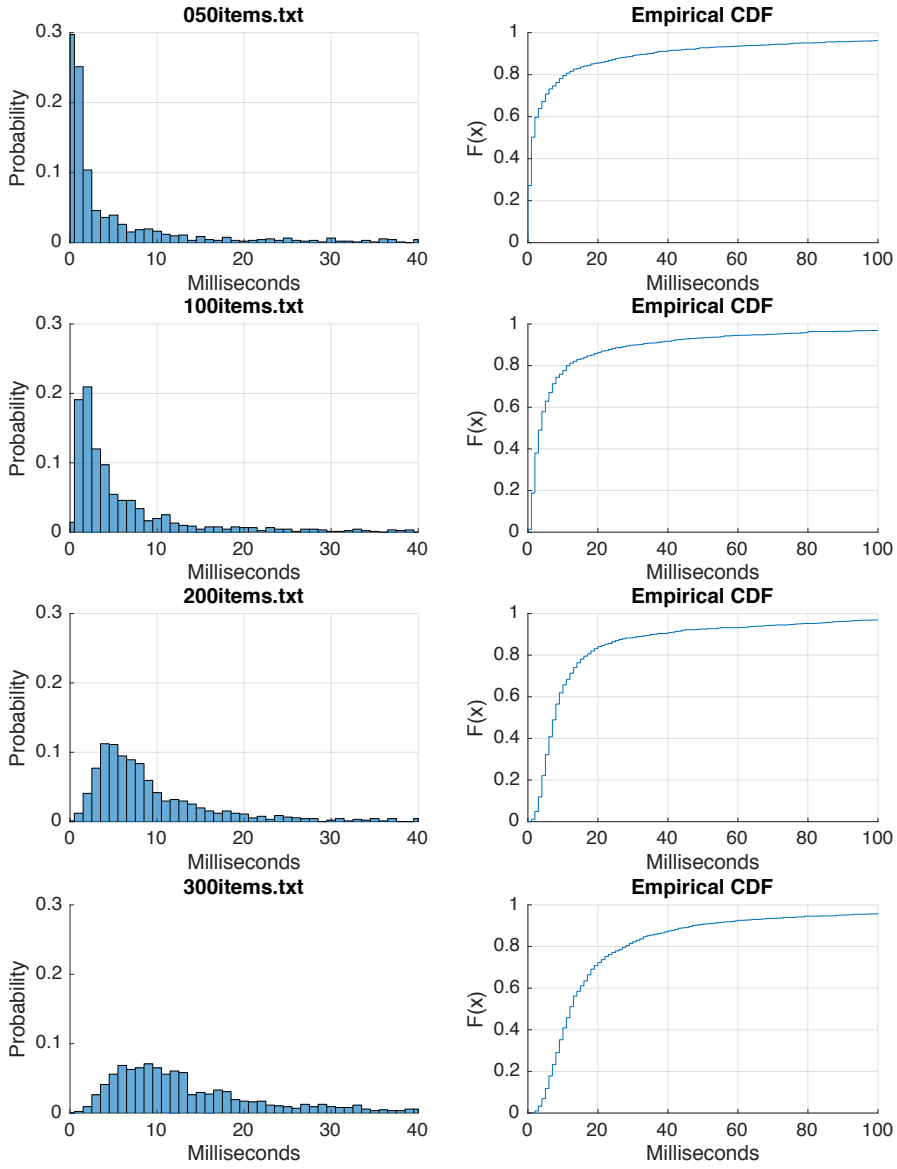


Figure 5.1: Left: Probability distribution, on the y axis, for CPH completion for a given time-frame in milliseconds shown on the x axis. Right: CDF of CPH completion before time x.

5.1.4 Solution quality sensitivity when processing bags in different order

As stated in previous sections, measuring the quality of solutions in multi-objective optimization is not a trivial mater. For the LBS problem, we decided to use the Hyper-volume indicator when measuring solution quality. It has proven to be a meaningful indicator [13] for these types of problems and it's

relatively efficient to compute [11].

LBS is a non monotonic problem. In other words, dominated configurations could lead to better solutions than the dominating configurations. For this reason, the set of final solutions depends on the order in which the configurations are processed by CPH. Figure 5.2 shows two runs of CPH using the same configurations as input but in different order. These instances evolve differently, produce very different solutions and have different execution times.

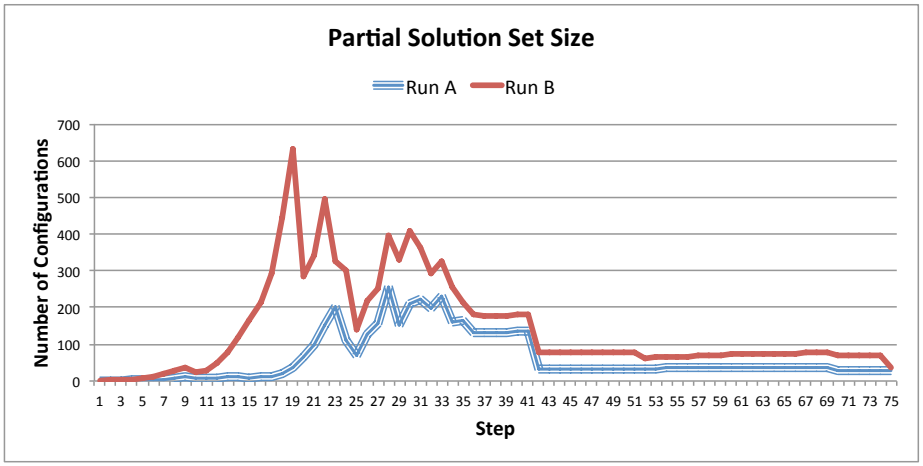


Figure 5.2: The evolution of the number of configurations in the partial solution set. Two separate runs are illustrated.

The Hyper-volume indicator shows the quality of the solution set produced by CPH.The indicator requires a reference point in the multi-dimensional space for the calculations. In this case we set the reference to 5% of the target in all dimensions. In other words, the Hyper-volume ranges from 0, for a solution set that does not dominate a single point located 5% off of targets in all dimensions, and 1, for a set that dominates the whole solution space. The Hyper-volume score will be expressed as a percentage between 0% and 100%. Table 5.2 that there is a huge variation on solution quality, given the parameters set for the Hyper-volume calculation. While it is likely to obtain solutions within 5% of the targets in all objective dimensions, it is also very likely to obtain solutions that miss the 5% tolerance in at least one dimension. Another interesting observation is that CPH finds better solution sets when it has more configurations to work with.

Table 5.2: Hyper-volume indicator values. Statistic from 10 runs for each problem size.

# Configurations	**50**	**100**	**200**	**300**
AVG	30%	37%	43%	45%
MAX	98%	96%	99%	94%
MIN	0%	0%	0%	0%

5.1.5 Solution quality with multiple CPH runs for a single batch

In previous sections, results show that it is common that CPH instances fail to provide consistent solution quality. Also, some instances may run for hundreds of milliseconds more than the average case. An implementation of CPH with uniformity in solution quality and execution time would be much more useful. In this section, we try an approach that runs multiple instances of CPH for the same problem while shuffling the input configurations and limiting the running time for each run.

Table 5.3: Results for multiple runs of CPH per problem, each limited to 2ms.

# Runs	**5**	**7**	**10**
AVG Hyper-volume	76%	83%	89%
AVG [ms]	10.82	15.08	21.45
MAX [ms]	11.08	15.62	23.08
MIN [ms]	10.54	14.54	20.46

Results from table 5.3 show that this implementation generates solutions with much better average Hyper-volume and more uniform running times. With 5 runs, only 1.5% of the solutions have 0% Hyper-volume. With 7 runs, that percentage decreases to 0.76% and with 10 runs there were no solutions with Hyper-volume of 0%.

5.1.6 Comparison of CPH and Simulated Annealing

The current solution for LBS at Vanderlande uses Simulated Annealing. To investigate the relative effectiveness of CPH, we compared solution quality and execution time of both methods. An experiment was conducted using the sample flight data. Results shown on table 5.4 indicate that in average, Simulated Annealing finds solutions with higher Hyper-volume than CPH at similar execution times. The only version of CPH that obtained better solutions, CPH10, had considerably higher running time than Simulated annealing.

Table 5.4: Comparison of CPH and SA for the sample bag flight data. Averages over 20 runs

Method	CPH 5	CPH 7	CPH 10	SA
Hypervolume (5% ref)	76.17%	82.94%	89.41%	84.36%
Solutions with objectives outside 1% target front	32.02%	22.30%	15.00%	20.87%
Solutions with objectives outside 5% target front	1.53%	0.76%	0.00%	2.19%
Average running time	140ms	196ms	278ms	216ms
Min running time	137ms	189ms	266ms	156ms
Maximum running time	144ms	203ms	300ms	265ms

5.2 Load Forming Logic

In this section we show the results of several experiments conducted on CPH for the Orderline Allocation (OA) problem. The data used for the experiments is presented. The experiments were designed to evaluate the execution time and solution quality of CPH.

5.2.1 Test Environment

The tests were conducted using a personal computer running Windows 7 Professional Edition (64 Bits). The system contained an i5-3340M CPU with two cores, each operating at 2.7GHz, and 12GB of RAM. Both CPH and the current solution were implemented using C# and Visual Studio 2013.

5.2.2 Data

Vanderlande provided several data-sets containing real orders from their main clients. These orders belong to representative sets that according to Vanderlande depict the challenges of Load Forming Logic (LFL) and OA. The data-set contains orders that are representative of the normal characteristics of orders such as size, items distribution and variability.

5.2.3 Execution time sensitivity to order size

The execution time of a CPH instance depends on two factors: The size of the order and the number of partial solutions allowed in the partial solution set. The size is an important factor because the number of CPH configuration sets is equal to the number of orderlines in the order. Additional the size of these configuration sets is directly related to the number of carriers required by the order. Usually larger orders require more carriers.

Table 5.5: Comparison of the smallest and largest orders in the Data-set.

Order #	8653	5840
Volume	18.4 m^3	1 m^3
Weight	7,866 kg	367 kg
Orderlines	702	40
Carriers	22	2
CPH time	10.953s	0.015s

The extreme comparison of order sizes is shown in table 5.5. For order 8653 configurations are 10 times larger, because of the number of carriers, and it contains 17 times the orderlines of order 5840. As result, the execution time of order 8653 is almost three orders of magnitude greater than order 5840.

5.2.4 Execution time sensitivity to partial solution set size

The other factor that affects execution time of CPH is the maximum number of configurations allowed in the partial solution set. Figure 5.3 shows the relation between

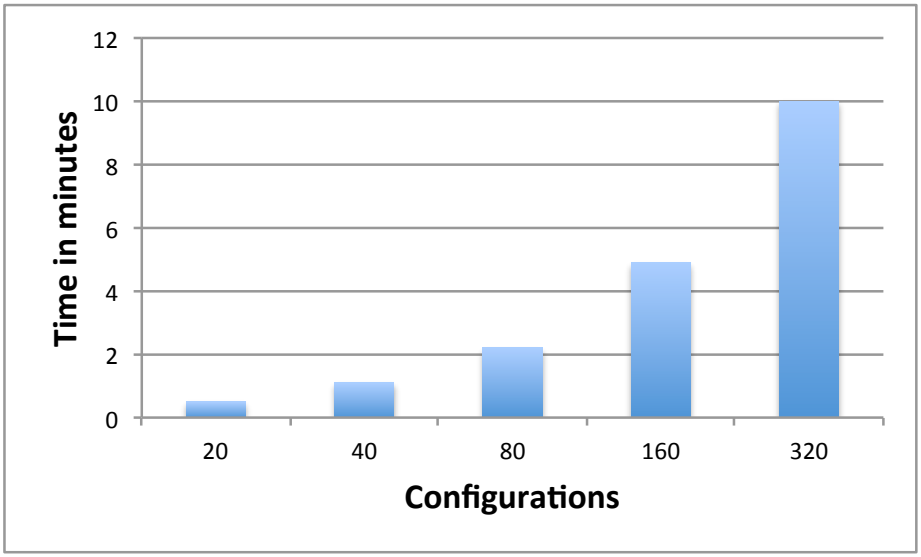
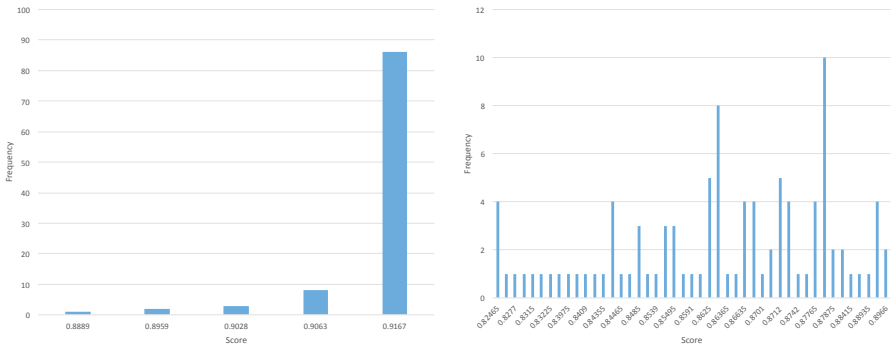


Figure 5.3: Relation between execution time and the maximum number of configurations allowed in the partial solution set. Total running time for a subset of 24 orders from the "Detailed Design" data-set

Time increases linearly when increasing the maximum number of configurations allowed in the partial solution set. This is expected considering that scalability is one of the design principles of CPH.

5.2.5 Solution quality sensitivity to sequential variation of orderlines

Solution quality for Orderline Allocation (OA) is measured with the indicators set by Vanderlande: Utilization and Partitioning. For OA, the order in which CPH processes items is vital. After trying different sequences of items, we found that the most beneficial way to process the orderlines is to group the orderlines by partitioning. Then within these partitions, orderlines are ordered by decreasing volume. Yet, the order in which the partitionings are processed had very particular results for each order. Thus, we couldn't find an order that would overall benefit solution quality across the range of orders from the order-set used for experimenting.



(a) Order 1061

(b) Order 8515

Figure 5.4: Frequency on the y axis, and solution score on the x axis for 100 runs of CPH when shuffling the order in which partitionings are processed.

To illustrate that shuffling the partitionings has a more significant effect larger orders, 5.4 compares a smaller order, sub-figure b, to a larger order on the right. The solutions with the highest scores were saved for 100 runs with shuffled partitionings. Solution quality on the smaller order varies 2.2% and the highest scoring solution has a larger frequency than the others, at 86% frequency vs 8%. On the larger order, sub-figure b, not not only do we have more solutions but their frequency is distributed more uniformly than in the other order. Also, in the large order there is a higher difference in the scores for best and worst solutions provided at 7%.

5.2.6 Solution quality sensitivity to partial solution set size

Increasing the number of configurations allowed in the partial solution set parameter increases the solution quality. Table 5.6 shows how the solution score is affected by doubling the P_s capacity, and consequently doubling the execution time. In most cases we observe an increment in solution score but it grows much slower than the execution time, for example when increasing capacity from 80 to 160, execution time grows by 100% (as explained in reftimeps) but solution score increases only by 0.11%.

Table 5.6: Solution quality score at different partial solution set capacities

P_s max size	20	40	80	160	320	640
Score	89.85%	90.58%	90.15%	90.26%	90.28%	90.33%

5.2.7 Comparison of CPH and OLAA

The Orderline Allocation Algorithm (OLAA) is the name of the current solution that Vanderlande uses in LFL. It is a genetic algorithm that optimizes the

Utilization and Partitioning of the Orders. The implementation details of this algorithm are beyond the scope of this project. The relevant part about OLAA is how it compares to CPH in terms of solution quality and execution time. CPH, even when using a large capacity partial solution set, is much faster than OLAA single thread implementation. It is important to point out that in practice, OLAA is partially parallel and pipelined with other processes within LFL. For this reason, a multi-threaded run of OLAA was included. Figure 5.5 shows that there is a difference of almost two orders of magnitude between OLAA and CPH execution times.

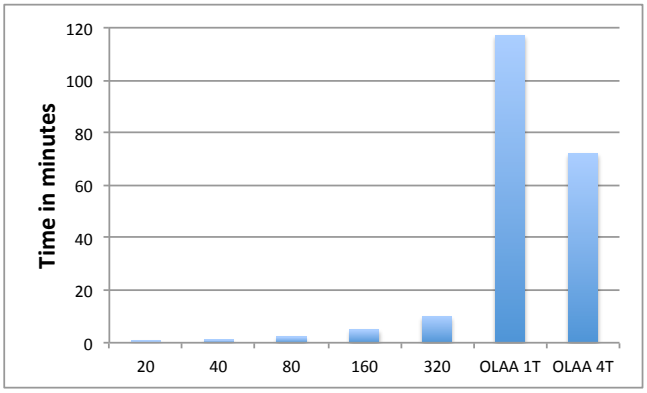


Figure 5.5: Execution time for CPH at different P_s capacities and OLAA using 1 and 4 threads.

When comparing solution quality OLAA has the upper hand. Although CPH provides comparable results, in average with 1.8% lower score than OLAA, OLAA solutions usually dominate the solution set from CPH. Table 5.7 shows detailed results, broken down by order.

Table 5.7: Utilization and Partitioning scores of OLAA and CPH using 24 orders from the "AH40Trendsets" data-set.

	OLAA Benchmark 160		CPH maxConfigs=40		CPH - OLAA	
Order	U	P	U	P	U	P
8659.dat	86.67%	98.33%	81.25%	97.92%	-5.42%	-0.41%
1329.dat	90.91%	100.00%	90.91%	95.45%	0.00%	-4.55%
1232.dat	88.24%	90.19%	78.95%	95.83%	-9.29%	5.64%
1288.dat	87.50%	97.73%	77.78%	97.73%	-9.72%	0.00%
1067.dat	90.00%	95.45%	81.82%	95.45%	-8.18%	0.00%
8635.dat	93.33%	90.12%	87.50%	92.36%	-5.83%	2.24%
8592.dat	88.90%	98.18%	80.00%	100.00%	-8.90%	1.82%
1006.dat	100.00%	100.00%	100.00%	96.43%	0.00%	-3.57%
1467.dat	90.91%	95.67%	83.33%	96.97%	-7.58%	1.30%
1135.dat	90.00%	89.20%	81.82%	94.55%	-8.18%	5.35%
1117.dat	84.62%	96.03%	84.62%	96.03%	0.00%	0.00%
8501.dat	90.00%	86.97%	85.71%	89.31%	-4.29%	2.34%
1434.dat	86.67%	83.43%	81.25%	84.50%	-5.42%	1.07%
8694.dat	81.82%	98.70%	81.82%	97.73%	0.00%	-0.97%
1061.dat	87.50%	95.83%	87.50%	95.83%	0.00%	0.00%
5864.dat	100.00%	100.00%	100.00%	100.00%	0.00%	0.00%
5840.dat	100.00%	100.00%	100.00%	100.00%	0.00%	0.00%
1395.dat	100.00%	100.00%	100.00%	95.83%	0.00%	-4.17%
1133.dat	87.50%	98.15%	82.35%	95.83%	-5.15%	-2.32%
5609.dat	66.67%	100.00%	66.67%	100.00%	0.00%	0.00%
1105.dat	88.89%	98.33%	80.00%	100.00%	-8.89%	1.67%
8653.dat	88.00%	88.27%	81.48%	95.45%	-6.52%	7.18%
8515.dat	91.67%	89.18%	84.62%	89.39%	-7.05%	0.21%
1567.dat	85.71%	100.00%	85.71%	100.00%	0.00%	0.00%
Average	89.40%	95.41%	85.21%	95.94%	-4.18%	0.53%
Weighted Score	92.40%		90.58%		-1.82%	

Chapter 6

Conclusions & Future work

CPH frameworks for the case studies were presented in this report. In both cases, CPH produced solutions of comparable quality and execution time with the current solutions. The lack of monotonicity in these problems was the main obstacle for the presented heuristics. In the area of packing problems CPH con be positioned as an alternative between Meta-heuristics and approximation algorithms because out experiments it provided solutions of inferior quality th Meta-heuristics but with much lower execution times.

6.1 Luggage Batch Selection

The current solution, Simulated Annealing, outperformed CPH in both execution time and solution quality metrics. The only cases in which CPH provided better batches was when processing the last batches of a flight. This due to the flexibility to optimize in multiple dimensions, including the number of bags per batch. Despite advantage, we believe CPH is not the right direction for improving the LBS process.

6.2 Load Forming Logic: Orderline Allocation

From the results of chapter 5 CPH seems like an interesting alternative for solving multi-objective bin packing problems, specially in processes where solutions are required in swiftly. The reduction in execution time from using CPH compared to the current methods would be of at least an order of magnitude. Nevertheless the LFL process cannot justify decreasing solution quality. The only way for CPH to be part of the Vanderlande process would be to match the solution quality of the current method. We believe it would be worth investigating further CPH for the OA problem. Specifically balancing weight and volume usage in the carriers and implementing greedy optimization on the partial solution set between CPH steps.

Bibliography

[1] Md Mostofa Akbar, M Sohel Rahman, Mohammad Kaykobad, Eric G Manning, and Gholamali C Shoja. Solving the multidimensional multiple-choice knapsack problem by constructing convex hulls. *Computers & operations research*, 33(5):1259–1273, 2006.

[2] G Belov and G Scheithauer. A branch-and-cut-and-price algorithm for one-and two-dimensional two-staged cutting (stock) problems. 2003.

[3] Maxence Delorme, Manuel Iori, and Silvano Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. In *Decision models for smarter cities*, 2014.

[4] Emanuel Falkenauer. *Genetic Algorithms and Grouping Problems*. John Wiley & Sons, Inc., New York, NY, USA, 1998.

[5] Martin Josef Geiger. Bin packing under multiple objectives - a heuristic approximation approach. 2008.

[6] David S. Johnson, Alan Demers, Jeffrey D. Ullman, Michael R Garey, and Ronald L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.

[7] Richard E Korf. A new algorithm for optimal bin packing. In *AAAI/IAAI*, pages 731–736, 2002.

[8] Mark W Krentel. The complexity of optimization problems. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 69–76. ACM, 1986.

[9] Hamid Shojaei, Twan Basten, Marc Geilen, and Azadeh Davoodi. A fast and scalable multidimensional multiple-choice knapsack heuristic. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 18(4):51, 2013.

[10] J. van Pinxten, M.C.W. Geilen, T. Basten, U. Waqas, and L. Somers. Online heuristic for the multi-objective generalized traveling salesman problem. In *In Design, Automation and Test in Europe, DATE 2016*. EEDA, March 2016.

[11] Lyndon While, Phil Hingston, Luigi Barone, and Simon Huband. A faster algorithm for calculating hypervolume. *Evolutionary Computation, IEEE Transactions on*, 10(1):29–38, 2006.

[12] Zhenyu Yan, Linghai Zhang, Lishan Kang, and Guangming Lin. A new moea for multi-objective tsp and its convergence property analysis. In *Evolutionary Multi-criterion Optimization*, pages 342–354. Springer, 2003.

[13] Eckart Zitzler, Dimo Brockhoff, and Lothar Thiele. The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration. In *Evolutionary multi-criterion optimization*, pages 862–876. Springer, 2007.

Acronyms

BPP Bin Packing Problem. 12

CPH Compositional Pareto-algebraic Heuristic. 1, 3, 6, 12, 17, 18, 20, 22, 24–37

LBS Luggage Batch Selection. 1, 4, 6, 14, 18, 27–31

LFL Load Forming Logic. 4, 7, 24–27, 32, 34, 35, 37

OA Orderline Allocation. 1, 4, 7, 15, 22, 32, 33, 37

OLAA Orderline Allocation Algorithm. 5, 34–36