An efficient and general approach for the joint order batching and picker routing problem

Olivier Briant, Hadrien Cambazard, Diego Cattaruzza, Nicolas Catusse, Anne-Laure Ladier, Maxime Ogier

An efficient and general approach for the joint order batching and picker routing problem

Olivier Briant[1], Hadrien Cambazard[1], Diego Cattaruzza[2], Nicolas Catusse[1], Anne-Laure Ladier[3], and Maxime Ogier[1]

[1]Univ. Grenoble Alpes, CNRS, Grenoble INP, G-SCOP, 38000 Grenoble, France
[2]Univ. Lille, CNRS, Centrale Lille, Inria, UMR 9189 - CRIStAL, F-59000 Lille, France
[3]Univ. Lyon, INSA Lyon, DISP laboratory EA 4570, F-69100 Villeurbanne, France

Abstract

Picking is the process of retrieving products from inventory. It is mostly done manually by dedicated employees called pickers and is considered the most expensive of warehouse operations. To reduce the picking cost, customer orders can be grouped into batches that are then collected by traveling the shortest possible distance.

This work presents an exponential linear programming formulation to tackle the joint order batching and picker routing problem. Variables, or columns, are related to the picking routes in the warehouse. Computing such routes is generally an intractable routing problem and relates to the well known traveling salesman problem (TSP). Nonetheless, the rectangular warehouse's layouts can be used to efficiently solve the corresponding TSP and take into account in the development of an efficient subroutine, called oracle. We therefore investigate whether such an oracle allows for an effective exponential formulation.

Experimented on a publicly available benchmark, the algorithm proves to be very effective. It improves many of the best known solutions and provides very strong lower bounds. Finally, this approach is applied to another industrial case to demonstrate its interest for this field of application.

Keywords: order batching, picker routing, column generation.

1 Introduction

Order picking is often considered the most important warehousing process since it estimatedly accounts for the majority of the total operational warehouse costs (Tompkins et al., 2010). Pickers follow routes in the warehouse

pushing a trolley. They collect items to form customer orders. To save time, several orders might be *batched*, i.e., put together to be collected in one single route. This is possible as long as they respect capacity constraints, namely they fit in a trolley. A key decision is therefore how to group customer's orders to minimize the total walked distance.

Once the batch is computed, the problem of identifying the shortest route in the warehouse to visit all the locations required by a given batch is called the picker routing problem (PRP). It has been widely studied in regular rectangular warehouses (Ratliff & Rosenthal, 1983; Hall, 1993; Petersen, 1997; de Koster & Van der Poort, 1998; Vaughan, 1999; Roodbergen & de Koster, 2001a,b; Pansart et al., 2018) and to the best of our knowledge, it can not be solved in polynomial time. However, the quality of the picking routes strongly depends on the batching decisions that are made prior to the routing decisions. It is quite clear that both sets of decisions should be taken jointly to ensure an efficient picking policy: this problem is known as the joint order batching and picker routing problem (JOBPRP, Valle et al. (2017)).

In this work we propose an exponential linear programming (LP) formulation of the JOBPRP, where variables (or columns) are related to single picking routes in the warehouse. More precisely, a column refers to a route involving a set of picking operations and satisfying the side constraints required at the trolley's level, such as the capacity or the possibility to mix orders. Computing such a picking route is an intractable routing problem in general and, depending on the warehouse layout, can closely relate to the traveling salesman problem (TSP). The rationale of our approach is however to consider that the PRP alone is easy enough in practice, due to nowadays warehouses structure, to be solved exactly. This assumption is a cornerstone of the proposed algorithm and is supported by the existing literature on the PRP as well as the previous work and industrial experience of the authors (Bué et al., 2018; Pansart et al., 2018; Cambazard & Catusse, 2018). In the HappyChic industrial case presented in Section 4.2, the graph that represents the warehouse is acyclic and therefore computing picking routes boils down to an easy path problem. This is a common situation when the traffic in the warehouse is unidirectional for safety reasons. Most often, warehouses have a regular rectangular structure made of aisles and cross-aisles with a bidirectional traffic. This layout is used in the benchmark proposed by Valle et al. (2017) and coming from the Foodmart database. In such a layout, Ratliff & Rosenthal (1983) and Roodbergen & de Koster (2001b) have shown that dynamic programming algorithms can take advantage of the rectangular structure to efficiently solve the corresponding TSP when the warehouse has two or three cross-aisles. Moreover this approach has been shown in Cambazard & Catusse (2018) to scale to rectangular warehouses with up to eight cross-aisles, which is beyond most real-life warehouse's sizes.

Based on these previous works, our approach assumes that an efficient

oracle is available to provide optimal picking routes in the warehouse. We show that such an oracle allows for a very efficient exponential LP formulation of the JOBPRP. The pricing problem can be seen as a prize-collecting TSP with a capacity constraint, and the pricing algorithm heavily relies on the picking oracle to generate cutting planes. A number of improvements are proposed to speed up the pricing. A feasible solution is obtained by solving a mixed integer program (MIP) with the known columns once the LP relaxation has been computed. The reported lower and upper bounds provide an optimality guarantee and considerably improve the existing results of Valle et al. (2017). Finally, the proposed methodology is applied on two distinct industrial benchmarks, Foodmart and HappyChic, that have slightly different warehouse structures. This contributes in asserting the interest and generality of this approach.

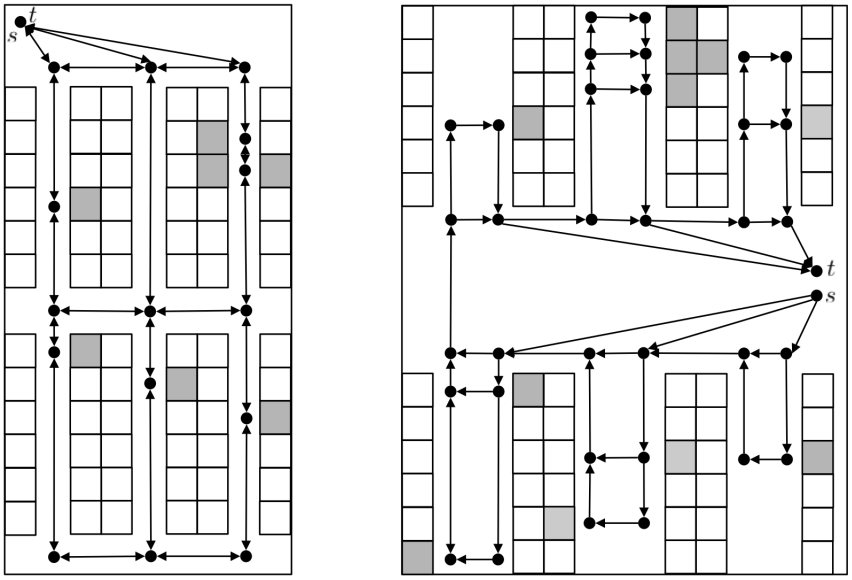
The remainder of the paper is organised as follows. Section 2 states the JOBPRP by slightly generalizing the specifications proposed in Valle et al. (2017). A literature review is then given in Section 3. Section 4 presents the approach on the two industrial cases, namely Foodmart (Section 4.1) and HappyChic (Section 4.2). The main focus is on Foodmart since it allows a comparison to previous academic work. The presentation on the HappyChic case highlights the differences to Foodmart without going into details since the same methodology is applied. Numerical results are reported in Section 5 while Section 6 concludes the paper.

2 Problem specifications and industrial applications

The warehouse layout is modeled as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ where $\mathcal{V}$ contains two types of vertices, locations and intersections as well as two depots: $\mathcal{V} = \mathcal{V}_L \cup \mathcal{V}_I \cup \{s\} \cup \{t\}$. Each location in the location set $\mathcal{V}_L$ contains one or more products to be picked, whereas the intersections in $\mathcal{V}_I$ are used to encode the warehouse structure and allow the picker to change direction. Additionally, s denotes the depot where picking routes start and t is the depot where routes end and items are dropped off. s and t can be located at the same spot. Moreover d_{ij} denotes the distance associated with each arc $(i, j) \in \mathcal{A}$ while D_{ij} is the shortest path distance for each pair $(i, j) \in (\mathcal{V}_L \cup \{s\} \cup \{t\}) \times (\mathcal{V}_L \cup \{s\} \cup \{t\})$. Note that if $(i, j) \in \mathcal{A}$, then $d_{ij} = D_{ij}$.

Figure 1 shows two typical examples of warehouse layouts used as benchmarks in the present paper. A regular rectangular layout made of three vertical aisles and three horizontal cross-aisles is shown on Figure 1a. Products are located on both sides of vertical aisles; cross-aisles do not contain any products but enable the order picker to navigate in the warehouse. Such a layout has been used by numerous authors in the past to define the PRP (Roodbergen & de Koster, 2001a). It is the setup of the Foodmart

benchmark. Figure 1b shows an acyclic layout where pickers are not allowed to backtrack. It is another typical industrial setup where the flow is constrained in a single direction and an aisle must be entered and exited on the same side. It is the setup of the HappyChic benchmark.



(a) Foodmart benchmark (b) HappyChic benchmark

Figure 1: Examples of two warehouse layouts with the corresponding directed graph encoding the possible moves of the picker, to pick products at the locations in grey.

The set of products is denoted by $\mathcal{P}$. A product may have several dimensions such as weight and volume and we refer to the set of dimensions as $\mathcal{W}$. A *product* $p \in \mathcal{P}$ is characterized by its location $v_p \in \mathcal{V}_L$ and its size V_p^w in each dimension $w \in \mathcal{W}$.

The pickers need to prepare a set of orders that is referred to as $\mathcal{O}$ ($|\mathcal{O}| = m$); an order $o \in \mathcal{O}$ is defined as a set of products with an associated quantity to pick. We define an *order line* as a product to pick with the associated quantity and we denote by $\mathcal{L}$ the set of all order lines to be prepared ($|\mathcal{L}| = n$). As a consequence an order $o \in \mathcal{O}$ is defined as a set of order lines that will be indicated by $\mathcal{L}_o \subseteq \mathcal{L}$. Then, an order line $l \in \mathcal{L}_o$ related to an order o is defined as a pair (p_l, Q_l) where $p_l \in \mathcal{P}$ defines the product and Q_l is the number of such products to pick.

Note that the size of order o in dimension w is simply computed as $V_o^w = \sum_{(p_l,Q_l)\in\mathcal{L}_o} V_{p_l}^w Q_l$. Moreover, an order can be split in at most B_o boxes.

The picking operations in the warehouse consist in the collection of order

lines by pickers that push a trolley with B boxes. A box has a capacity V^w in dimension $w \in \mathcal{W}$. The quantity Q_l of an order line l can be split among several boxes: a box is therefore filled with what we call *partial order lines*. A partial order line is a pair (l, q) that determines the quantity q of order line l collected in a specific box ($q \leq Q_l$). A box can only contain partial order lines from a single order.

A solution is a collection of routes $\bar{\mathcal{K}}$ from s to t in the warehouse layout $\mathcal{G}$. Each route k is performed by a picker that pushes a trolley and collects partial order lines into its boxes $\mathcal{B}_k$ ($|\mathcal{B}_k| = B$). Each box $b \in \mathcal{B}_k$ contains a set of partial order lines $\tilde{\mathcal{L}}_{kb}$ of a single order. The capacities of the boxes must be satisfied in each dimension $w \in \mathcal{W}$ *i.e.* $\sum_{(l,q)\in\tilde{\mathcal{L}}_{kb}} qV^w_{p_l} \leq V^w, \forall w \in \mathcal{W}$. Finally, all order lines must be picked. The objective is to minimize $\sum_{k\in\bar{\mathcal{K}}} d_k$ where d_k is the optimal distance to pick all the partial order lines of route k.

The specifications given above are general enough to encompass the two industrial cases used as benchmark. They are slightly more general than the ones used in Valle et al. (2017), in order to address the HappyChic case as well. The industrial context of each application is summarized below.

Foodmart case. This first case originates from online grocery shopping, where orders may be composed of dozens of items. The datasets have been designed by Valle et al. (2017) from the publicly available database (Thia (2008)) composed of anonymised customer purchases over two years for a chain of supermarkets. We list below the particularities of this application regarding the general problem description. First, there is a single dimension for products and boxes ($|\mathcal{W}| = 1$). The box capacity V^1 expresses the maximum number of items it can contain. Therefore, each product p has a size $V^1_p = 1$. In this case, $B_o = \lceil \frac{V^1_o}{V^1} \rceil$ denotes the exact number of boxes required by order o. Second, an order must be picked entirely by a single trolley. Thus, it is assumed that $B_o \leq B$ in the benchmark proposed by Valle et al. (2017).

HappyChic case. HappyChic is a French company specialized in men clothes, that runs a warehouse located in the North of France, dedicated to clothing products for the brand Jules. This warehouse supplies shop located all over France. An order o represents the demand of one shop and thus counts many order lines. As a consequence, an order does not usually fit into one box, but may be split into several boxes. The maximum number of boxes B_o to pick an order o is used to guarantee a good filling rate of the boxes and consequently reduces the transportation costs. Two dimensions are considered for the box capacity:

weight and volume. Hence, a box has a maximum weight V^1 and a maximum volume V^2 and each product $p \in \mathcal{P}$ has a product weight V_p^1 and a product volume V_p^2.

To ease the pickers' task and for safety reasons, a *walking direction policy* is imposed in the warehouse. Aisles are ordered following a clockwise direction and pickers must visit them in this same order. Moreover, visiting an aisle follows a *backtracking policy*: when a picker gets into the aisle, he/she walks until the position of the furthest product to pick, then he/she crosses the aisle and walks back until the beginning of the aisle, possibly picking other products on the way. Thus, the associated warehouse layout graph $\mathcal{G}$ is acyclic (see Figure 1b).

Pickers push a trolley with up to $B = 6$ boxes, that can be used to prepare different orders. Moreover, boxes associated to the same order can be assigned to different trolleys. A route is thus a batch of up to $B = 6$ boxes possibly associated with different orders. When a picking route is completed, the picker drops off the boxes on a conveyor belt located in the middle of the picking zone, that brings the boxes to the delivery zone. As a result, the picking zone contains multiple depots (one for each aisle in the warehouse). Then in this case, s and t represent fictive depots, with a fixed distance with all actual depots.

3 Literature review

Warehouse management includes several optimization problems such as conceiving picking routes, batching orders, storage assignment, layout design and zoning (see, for example de Koster et al. (2007)). To optimize productivity into a warehouse different decisions may be taken. These decisions can be associated with different time horizons. Strategic decisions determine the layout of the warehouse and the positioning of each zone (receiving, storage, picking and delivery) with respect to the others. The decisions that involve the storage and picking policy can be viewed as strategic decisions as well. Tactical decisions can involve determining the location of products based on their forecast demand in the storage and picking zone. Finally, at the operational level batches of orders and order picking routes need to be efficiently computed (de Koster et al., 2007; Marchet et al., 2015).

Based on the available technology of the warehouse, picking operation can be classified in five classes (Marchet et al., 2015):

Picker-to-parts where pickers move around the warehouse to pick items;

Parts-to-picker where automated devices bring loads to pickers that are in charge of picking the right quantity required by the order under consideration;

Pick-to-box where pickers are assigned to different zones and boxes containing customer orders move by means of a conveyor through different zones in order to be filled;

Pick-and-sort where orders are batched and pickers collect items of a certain product to satisfy orders in the whole batch – then a conveyor brings all the items to the sorting area where orders are formed;

Completely automated picking where humans do not intervene.

The majority of warehouses implement picking systems where the human presence is necessary for operations (de Koster et al., 2007). Recent studies (Ecommerce-Europe, 2017; Marchet et al., 2015) show the cost of automation is too high to be profitable in short or mid term horizons. Thus the travel time to pick products that constitute orders, which is the largest component of labour in classical warehouses (Bartholdi & Hackman, 2017), can represent up to 60-70% of the whole operating cost of the warehouse (Chen et al., 2015). For these reasons and since this paper deals with human picking operations, we concentrate our review on papers related to picker-to-parts systems.

The picker routing problem (PRP) can be seen as the problem of picking a list of products located in the warehouses, minimizing the traveled distance or time. More formally, the picking zone of the warehouse is composed of v vertical aisles and h horizontal cross-aisles. The horizontal cross-aisles usually do not store products and are used by pickers to move from one vertical aisle to another. We note the set of products to be picked as $\bar{\mathcal{V}}_L \subset \mathcal{V}_L$. Thus, the relevant set of location for the order picking problem is $\bar{\mathcal{V}}_L \cup \mathcal{V}_I$. The PRP answers the question: given the set $\bar{\mathcal{V}}_L$, what is the shortest (with respect to distance or time) route to pick all the products in $\bar{\mathcal{V}}_L$? This problem is a particular case of the classical travelling salesman problem (TSP, Dantzig et al. (1954), Orman & Williams (2006)) in which the picker is the salesman. The TSP is known to be NP-hard in the general case (Karp, 1972). The PRP can also be seen as a Steiner TSP (Fleischmann, 1985; Cornuéjols et al., 1985). The Steiner TSP distinguishes between locations that can be visited (here the intersections, $\mathcal{V}_I$) and locations that must be visited (here $\bar{\mathcal{V}}_L$).

Due to the particular structure of the warehouse layout graph, efficient exact approaches can be obtained using the dynamic programming paradigm. The first attempt has been proposed in Ratliff & Rosenthal (1983) for the case of a single block (i.e., 2 cross-aisles) in 1983. Recently, Cambazard & Catusse (2018) developed a dynamic programming approach which can solve any rectilinear TSP and can therefore solve the PRP for any rectangular warehouse with h cross-aisles. However its complexity is exponential in h. The reader interested in exact algorithms for the PRP is referred to the recent survey by Pansart et al. (2018).

The PRP is solved when the set of products to be picked, called the batch, has already been determined (*i.e.* the batch is an input of the PRP). The batch constitution rules can differ based on the picking policy of the warehouse. A batch can correspond to a single order or be the union of different orders. In the latter case, the picker picks items of several orders at once, thus reducing the total walking distance. How orders are batched together is an optimization problem in itself, called the order batching problem (OBP). The OBP answers the question: how to batch orders together such that the travelled distance or time to collect all the products is minimized? The OBP is known to be NP-hard in the strong sense in the general case and polynomially solvable if batches contain at most two orders (Gademann & Van de Velde, 2005).

Constructive heuristics to batch orders can be classified in three classes (de Koster et al., 1999). The first considers simple sequential strategies, such as the First-Come First-Served batching heuristic, where orders are sorted based on their arrival at the warehouse and batches are created accordingly respecting capacity constraints (Gibson & Sharp, 1992). A second class consists of seed algorithms which first identify a seed order, then associate the remaining orders with a seed order to construct a batch. The third class of batching algorithms make use of the saving paradigm and is inspired from the Clarke and Wright heuristic (Clarke & Wright, 1964): initially each order constitutes a batch, then the savings in distance or time of merging two batches is calculated. The merge leading to the largest distance or time reduction is implemented.

The different batching strategies can be evaluated by fixing a routing algorithm that calculates the total travelled distance or time (de Koster et al., 1999). This is usually performed with the so-called S-shape algorithm or the Largest-gap strategy (de Koster et al., 1999; Henn et al., 2010), or a combination of these two algorithms (Roodbergen & de Koster, 2001a; Menéndez et al., 2017). The main difference between the two strategies is that in the former the picker that enters one aisle goes across the entire aisle, while in the latter it gets in and out from the same side of the aisle.

Recently, trajectory-based and population-based approaches relying on local search procedures have been developed for the OBP. As an example, Henn et al. (2010) propose an iterated local search as well as a ant colony algorithm, while Menéndez et al. (2017) use a variable neighbourhood search. The interested reader is referred to the survey by Cergibozan & Tasan (2016).

As the reader may notice, the PRP and OBP are strongly linked, since the OBP needs to be solved to provide an input for the PRP while the PRP is solved to evaluate the efficiency of procedures developed for the OBP. As a consequence the scientific community started considering the integrated problem where the OBP and the PRP are simultaneously considered and solved. The problem that arises is the joint order batching and picker routing problem (JOBPRP). Won & Olafsson (2005) consider the JOBPRP se-

quentially by first batching orders and second by solving a TSP for each batch. Kulak et al. (2012) propose a tabu search heuristic in which the initial batches are created with a clustering method and the routing is calculated using TSP-based heuristics such as savings and nearest neighbour heuristics for routes construction and Or-opt and 2-opt heuristics for routes improvement. New solutions are created by relocating and swapping orders among batches. Cheng et al. (2015) use a particle swarm optimization approach to batch orders and an ant colony optimization algorithm to optimize the order picking. To the best of our knowledge, the only exact approach for the JOBPRP so far has been proposed by Valle et al. (2017). They develop a branch-and-cut algorithm, and the initial solution is computed using a saving heuristic.

Whenever due dates are associated with each order, the sequence on which orders are processed have an impact on the total tardiness of the operations. As a consequence, scholars started considering the integrated problem that simultaneously consider order batching, order sequencing and picker routing. The interested reader is referred to the recent survey on these problems by Cano et al. (2018).

4 Proposed approach

We propose an exponential LP formulation of the JOBPRP where variables or columns represent picking routes in the warehouse. A column refers to a route involving a set of picking operations and satisfying the side constraints required at the trolley's level. The exact definition of a column strongly depends on whether an order can be split in several trolleys or not. Thus details of the proposed methodology are presented separately for the two problems encountered by Foodmart (Valle et al., 2017) (Section 4.1) and HappyChic (Section 4.2). However the same heuristic framework is applied in both cases.

Let us introduce some notations. The exponential LP formulation with integer variables is denoted (M). The set of all feasible picking routes (columns) is denoted $\mathcal{K}$. The linear relaxation of (M) is the so-called master problem and denoted (LM).

Since the set $\mathcal{K}$ of the columns defining (M) and (LM) is of exponential size, we define programs $(M(\mathcal{K}'))$ and $(LM(\mathcal{K}'))$ obtained from (M) and (LM) on a subset of columns $\mathcal{K}' \subset \mathcal{K}$. $(LM(\mathcal{K}'))$ will be called the restricted master problem.

In order to tackle the exponential number of variables, we develop a column generation procedure to add negative reduced cost columns into the restricted master problem.

The proposed algorithm is a heuristic that provides performance guarantee with upper and lower bounds on the optimal value of the master prob-

lem. At each iteration of the column generation, a lower bound on the master problem is computed, namely the Lagrangian bound. Once the master problem has been solved to optimality (or after a time limit), an integer solution is obtained by solving the integer version of the restricted master problem, containing all the columns generated during the procedure. This step provides a feasible solution and thus an upper bound on the optimal value.

Columns of negative reduced cost or a proof that no such column exists is obtained by solving the so-called pricing problem, denoted $(Pr(\mathcal{K}'))$. It is a mixed integer linear programming (MIP) model given for each application. It encompasses the constraints related to a single trolley to the exception of the routing part. The exact route distance is not initially modeled in $(Pr(\mathcal{K}'))$. It is increasingly approximated into $(Pr(\mathcal{K}'))$ with the use of cutting planes computed with the oracle for the picker routing problem. Two key elements are then used to solve the pricing problem efficiently:

Tour constraints are iteratively added as cutting planes to $(Pr(\mathcal{K}'))$. These constraints are based on the known picking routes $\mathcal{K}'$. They allow to iteratively improve the estimation of the distance associated with the column that $(Pr(\mathcal{K}'))$ should provide.

A relaxation of the routing part added to $(Pr(\mathcal{K}'))$. To strengthen the formulation of $(Pr(\mathcal{K}'))$, a set of constraints that model the routing are added to the formulation of $(Pr(\mathcal{K}'))$. However to not increase the difficulty of solving $(Pr(\mathcal{K}'))$, the variables involved in these new constraints are kept real.

The outline of the proposed column generation procedure is presented in Algorithm 1.

Note that the pricing problem $(Pr(\mathcal{K}'))$ is solved twice, on lines 6 and 9 of Algorithm 1. The routing relaxation is added to the model at the second call, when the first call has failed. The addition of the routing relaxation helps to derive a lower bound of the pricing problem, which in turn might help to improve the Lagrangian bound.

This approach with the pricing problem differs from what is usually done to solve routing problems with branch-and-price paradigms. In that case the pricing problem consists in solving an elementary shortest path problem with resources constraints (ESPPRC), the solution of which corresponds to the route (the column) to add into the restricted master problem (Feillet, 2010). This kind of pricing algorithms rely on the representation of partial paths (that represent partial solutions) by labels that are extended during the exploration of the graph on which the shortest path with resources constraints is computed. The extension of a label means including a new customer visit into the partial solution. The cost of the new label can be calculated in $O(1)$ and is the sum of the cost of the extended label plus the cost of the arc

Algorithm 1 Column generation heuristic.

1: $\mathcal{K}' \leftarrow$ a set of initial columns (picking routes)
2: $LB \leftarrow 0$
3: Initialize stabilization (artificial variables, initial target, penalization function)
4: **while** (LM) is not solved to optimality and time limit is not reached **do**
5: Solve $(LM(\mathcal{K}'))$
6: Solve $(Pr(\mathcal{K}'))$
7: $LagB \leftarrow$ compute Lagrangian bound
8: **if** $(Pr(\mathcal{K}'))$ has not found a negative reduced cost column **then**
9: Solve $(Pr(\mathcal{K}'))$ with routing relaxation
10: $LagB \leftarrow$ update Lagrangian bound
11: **end if**
12: $LB \leftarrow \max\{LB; LagB\}$
13: Compute $\mathcal{K}_{rich}$, a rich column set from the columns found by $(Pr(\mathcal{K}'))$
14: $\mathcal{K}' \leftarrow \mathcal{K}' \cup \mathcal{K}_{rich}$
15: **if** lower bound of $(Pr(\mathcal{K}')) \geq 0$ and all artificial variables are zero **then**
16: (LM) has been solved to optimality by $(LM(\mathcal{K}'))$
17: **else**
18: Update stabilization (target, penalization function)
19: **end if**
20: **end while**
21: $UB \leftarrow$ value obtained by solving $(M(\mathcal{K}'))$
22: Post-optimization of the best solution
23: $UB \leftarrow$ value after post-optimization

linking the last customer in the partial path and the new added customer. In our case, adding an order to a picking route consists in adding a set of picking locations to the partial picking route. As a consequence, the cost update cannot be made in constant time and labelling-based approaches would not easily extend to this more general case.

The performances of the proposed column generation algorithm are strengthened using the following key elements:

A rich column set $\mathcal{K}_{rich}$ is added into the restricted master problem at each iteration of the column generation (line 13 of Algorithm 1). For each column generated by the pricing problem, two different strategies are considered to fill $\mathcal{K}_{rich}$: (i) the column is completed with a set of columns that constitute a feasible solution of (M), and (ii) similar columns with fewer order lines are generated and also added into the

master problem. Each column generated when solving the pricing thus gives rise to a number of additional columns enriching the pool of the master.

The aim of the rich column set is to speed up the solving of the master problem (LM) and to improve the quality of the bounds.

A stabilization technique is introduced into the column generation in order to reduce the stability issues usually encountered at the beginning of column generation procedure. It is performed in lines 3 and 18 of Algorithm 1. Among the different methods for stabilizing dual variables (see for example Briant et al. (2008)), the one chosen in this paper consists in adding bounded artificial variables whose cost corresponds to a target that is regularly updated.

The master problem, the pricing problem and the features of the algorithm are presented in details in Sections 4.1 and 4.2 for Foodmart and HappyChic cases respectively.

4.1 Case of Foodmart

4.1.1 Master problem

The formulation is presented here for the case studied by Valle et al. (2017) and specified in Section 2, where an order must be entirely collected in a single trolley. Let us denote by $\mathcal{K}$ the set of all subsets of orders in $\mathcal{O}$ that satisfy the trolley's capacity: $\mathcal{K} = \{\mathcal{S} \subseteq \mathcal{O} \mid \sum_{o \in \mathcal{S}} B_o \leq B\}$. An element $k \in \mathcal{K}$ can be represented as a column $e^k = (e^k_1, \ldots, e^k_o, \ldots, e^k_m)$ where each $e^k_o \in \{0, 1\}$ indicates whether all products of order o are included or not in the element k. In the following, a column k can be written as a subset of orders $\{i \in \mathcal{O} | e^k_i = 1\}$ for sake of simplicity. We denote by d_k the distance of an optimal route to collect all the products of the subset of orders k. With an abuse of terminology we will call $\mathcal{K}$ the set of all feasible routes and an element $k \in \mathcal{K}$ a route. In this case we refer to an optimal route of length d_k needed to collect all products in k.

The formulation is based on variables $\rho_k \in \{0, 1\}$ expressing whether a route k is chosen. An exponential integer formulation (M) can be written as follows:

$$(M) \begin{cases} \min & \sum_{k \in \mathcal{K}} d_k \rho_k & & & & \\ \text{s.t.} & \sum_{k \in \mathcal{K}} e^k_o \rho_k & \geq & 1 & \forall o \in \mathcal{O} & (1) \\ & \rho_k & \in & \{0, 1\} & \forall k \in \mathcal{K} & (2) \end{cases}$$

Constraints (1) make sure that each order is collected and the objective is to minimize the total distance required to collect all orders. We focus on the

linear relaxation (LM) of this formulation. Moreover, since $\mathcal{K}$ is exponential in size, the formulation is defined in practice on a subset of known routes $\mathcal{K}' \subset \mathcal{K}$ and the restricted master problem $(LM(\mathcal{K}'))$ is:

$$(LM(\mathcal{K}')) \begin{cases} \min & \sum_{k \in \mathcal{K}'} d_k \rho_k & & & & & \\ \text{s.t.} & \sum_{k \in \mathcal{K}'} e_o^k \rho_k & \geq & 1 & \forall o \in \mathcal{O} & (3) & (\beta_o) \\ & \rho_k & \geq & 0 & \forall k \in \mathcal{K}' & (4) & \end{cases}$$

Note that it is not necessary to enforce $\rho_k \leq 1$ since it is satisfied in all optimal solutions (assuming $d_k > 0, \ \forall k \in \mathcal{K}$, which is always verified in practice).

The optimal value of such an exponential formulation can be computed by a technique known as column generation. The simplex algorithm does not need all variables to be explicitly included in the model, but only requires an algorithm, referred to as the pricing algorithm, to provide at each iteration a negative reduced cost variable or to prove that none exists. Let us denote by β_o the value of the dual variable related to Constraint (3). By definition, the reduced cost r_k of a route k is defined as:

$$r_k = d_k - \sum_{o \in \mathcal{O}} \beta_o e_o^k$$

The pricing step must identify a feasible route k (satisfying the capacity constraint) so that r_k is negative; or it must prove that no such route exists to terminate the simplex algorithm.

4.1.2 Pricing problem

The PRP consists in computing the optimal distance d_k of a route k to collect a given set of orders. As explained before, it is considered a hard problem in itself for rectangular warehouses but can be addressed efficiently using dynamic programming. Recall that we assume that an oracle to compute d_k is available. In practice we use the algorithm proposed by Cambazard & Catusse (2018) for the rectilinear TSP and experimented in Pansart et al. (2018) for the PRP.

The pricing step is solved by a cutting plane algorithm based on the following model:

$$(Pr(\mathcal{K}'))\begin{cases} \min & d - \sum_{o\in\mathcal{O}} \beta_o e_o & & & & \\ \text{s.t.} & \sum_{o\in\mathcal{O}} B_o e_o & \leq & B & & (5) \\ & d_k\left(\sum_{o\in k} e_o - |k| + 1\right) & \leq & d & \forall k \in \mathcal{K}' & (6) \\ & e_o & \in & \{0,1\} & \forall o \in \mathcal{O} & (7) \\ & d & \geq & 0 & & (8) \end{cases}$$

Variable e_o indicates whether order o is selected and Constraint (5) enforces the maximum number of boxes in a trolley. Variable d encodes a lower bound of the distance of a route. It is strengthened by the dynamic generation of Constraints (6) that are called the *tour constraints*. Typically, any selection of orders that is a superset of a known set of orders k must require a distance longer than d_k so that $d \geq d_k$. Constraints (6) simply states this relation for all sets of orders $\mathcal{K}'$ known in the restricted master problem and generated during the pricing algorithm.

Note that when route k visits a pair of orders i and j, *i.e.* $k = \{i, j\}$, Constraints (6) writes as follow:

$$d_{\{i,j\}}(e_i + e_j - 1) \leq d \quad \forall k \in \{\mathcal{K}' | k = \{i, j\}\}$$

In this specific case, the constraint can be improved using the geometric fact that $d_{\{i,j\}} \leq d_{\{i\}} + d_{\{j\}}$. We use instead the following constraint for pairs of orders:

$$d_{\{i\}}e_i + (d_{\{i,j\}} - d_{\{i\}})e_j \quad \leq \quad d \quad \forall k \in \{\mathcal{K}' | k = \{i, j\}\} \quad (6 \text{ bis})$$

In the following, the optimal value of $(Pr(\mathcal{K}'))$ is denoted $\phi^*(\beta, \mathcal{K})$ so that $\phi^*(\beta, \mathcal{K}) = \min_{k\in\mathcal{K}}(d_k - \sum_{o\in\mathcal{O}} \beta_o e_o^k)$.

Notice first that for any set $\mathcal{K}'$ of known routes, the optimal value $\phi^*(\beta, \mathcal{K}')$ of $(Pr(\mathcal{K}'))$ gives a lower bound of the best possible reduced cost $\phi^*(\beta, \mathcal{K})$ so that

$$\phi^*(\beta, \mathcal{K}') \leq \phi^*(\beta, \mathcal{K})$$

The pricing algorithm works by iteratively solving $(Pr(\mathcal{K}'))$, re-evaluating the exact distance and adding the corresponding tour constraint. Its principles are given in Algorithm 2.

A limit of this approach is that the number of tour constraints generated by $(Pr(\mathcal{K}'))$ can grow considerably. To reduce this number of iterations in practice, we first limit the number of iterations in the pricing algorithm. If the pricing algorithm has not proved that no negative reduced cost route exists after the given number of iterations, it is solved again (line 9 of Algorithm 1) adding a *relaxation of the routing*, explained hereafter, into the

Algorithm 2 Pricing algorithm.

1: Solve $(Pr(\mathcal{K}'))$ to get $\phi^*(\beta, \mathcal{K}')$ and a corresponding optimal solution e^*. Solution e^* defines a route denoted k^*.
2: **if** $\phi^*(\beta, \mathcal{K}') \geq 0$ **then**
3: Exit: no route with a negative reduced cost can exist because $\phi^*(\beta, \mathcal{K}') \leq \phi^*(\beta, \mathcal{K})$
4: **else**
5: Evaluate d_{k^*} with the picking oracle *i.e.* compute an optimal route to collect all products of the orders o such that $e^*_o = 1$.
6: **if** $d_{k^*} - \sum_{o \in \mathcal{O}} \beta_o e^*_o < 0$ **then**
7: A negative reduced cost route has been found: exit by returning k^*
8: **else**
9: add k^* to $\mathcal{K}'$
10: return to line 1
11: **end if**
12: **end if**

formulation of $(Pr(\mathcal{K}'))$. In that case, solving $(Pr(\mathcal{K}'))$ becomes more time consuming, so only one iteration is performed and a computation time limit is set.

Adding a relaxation of the routing The formulation of $(Pr(\mathcal{K}'))$ given in the previous section relies on known routes to estimate the distance d. It is possible to strengthen $(Pr(\mathcal{K}'))$ by adding constraints that model a relaxation of the TSP. The usual LP formulation for TSP by Dantzig et al. (1954) is based on sub-tour elimination and relies on an exponential number of constraints. We chose a relaxation recently proposed in Pansart et al. (2018) that takes advantage of the warehouse structure. The TSP is modeled as a Steiner TSP (Fleischmann, 1985) in the graph representing the warehouse, the LP model is a compact typical flow model (Orman & Williams, 2006; Letchford et al., 2013) enriched with a number of inequalities described in Pansart et al. (2018) and partly used by Valle et al. (2017). This TSP formulation is added to $(Pr(\mathcal{K}'))$ but the variables are kept fractional to avoid solving the TSP exactly which is often too expensive. In practice, the bound obtained for d considerably improves to the expense of the time needed to solve $(Pr(\mathcal{K}'))$.

4.1.3 Rich column set

In order to speed up the solving of the master problem (LM) and to improve the quality of the bounds, it is beneficial to add a set of columns into the restricted master problem $(LM(\mathcal{K}'))$ at each iteration of the column generation heuristic (line 13 and 14 of Algorithm 1). Indeed, several simplex

pivots can then be performed while solving the restricted master problem if the routes are interesting, *i.e.,* have or may have a negative reduced cost.

We thus add to $\mathcal{K}'$ a rich set of columns $\mathcal{K}_{rich}$, besides the routes generated by the pricing algorithm ($Pr(\mathcal{K}')$). For each column found by ($Pr(\mathcal{K}')$), a set of promising routes is generated and added to $\mathcal{K}_{rich}$. Two different and already mentioned strategies are considered and detailed afterwards.

Feasible primal solution Given a route k^* found by the pricing algorithm ($Pr(\mathcal{K}')$), this strategy aims at generating a set $\mathcal{K}_{rich1}(k^*)$ of routes to cover all the orders with efficient routes, *i.e.* such that $\{k^*\} \cup \mathcal{K}_{rich1}(k^*)$ constitutes a feasible integer solution for the master problem.

Additional routes are greedily generated one by one from the remaining unpicked orders. Once a route is generated, only the remaining unpicked orders are considered to generate the next route. When all orders have been included, the set of generated routes defines a feasible primal solution. A route is generated by filling the trolley with available orders ranked by the potential decrease in the reduced cost that the inclusion would produce. The order leading to the maximum decrease of reduced cost is chosen first.

To precisely evaluate the reduced cost of a route the exact distance must be computed with the oracle. Doing it for every remaining order would be very costly. Thus a fast approximation of the distance is used to select the few promising orders for which an exact evaluation is performed: the distance increase when adding an order o to a set of orders $\mathcal{S}$ is estimated by:

$$score(o, \mathcal{S}) = \max_{l \in \mathcal{L}_o} \min_{o_i \in \mathcal{S}, l_j \in \mathcal{L}_{o_i}} D_{p_l, p_{l_j}}$$

Note finally that routes in $\mathcal{K}'$ can also be considered to complete route k^* in order to provide an integer feasible solution. Instead of greedily generate routes, the ones in $\mathcal{K}'$ can be used as a starting point. The procedure to generate a feasible primal solution is presented in Algorithm 3.

Generate sub-tours Given a route k^* provided by the pricing algorithm ($Pr(\mathcal{K}')$), this strategy aims at generating a set $\mathcal{K}_{rich2}(k^*)$ of sub-tours of k^*, by removing one or more orders from k^*.

First, all sub-tours k' of k^* that use no more than 75% of the capacity of the trolley are generated.

Then, from route k^*, a set of routes is generated by iteratively removing one order. The key idea here is to sort the orders to be removed according to the complexity of picking them up. This complexity is estimated by averaging the reduced costs of the known routes that contain the order. This favors the generation of negative reduced cost routes. This procedure is presented in Algorithm 4.

Algorithm 3 Generate a feasible primal solution.

Require: k^* a route found by $(Pr(\mathcal{K}'))$ and $\mathcal{K}'$ the set of all known routes

$\mathcal{K}_{rich1}(k^*) \leftarrow \emptyset$
$\mathcal{O}' \leftarrow \mathcal{O} \backslash k^*$, the set of unpicked orders (k^* also represents a set of orders)
$\mathcal{K}_{init} \leftarrow$ select routes from $\mathcal{K}'$
while $\mathcal{O}'$ is not empty **do**
  Remove from K_{init} the routes k_{init} not compatible with $\mathcal{O}'$ ($k_{init} \not\subset \mathcal{O}'$)
  $k' \leftarrow$ route in K_{init} with lowest reduced cost (or $k' \leftarrow \emptyset$ if $K_{init} = \emptyset$)
  while an order in $\mathcal{O}'$ can be added into k' **do**
    $o \leftarrow$ order in $\mathcal{O}'$ that can be added into k' with lowest $score(o, k')$
    $k \leftarrow k' \cup \{o\}$
    $\mathcal{O}' \leftarrow \mathcal{O}' \setminus \{o\}$
  end while
  $\mathcal{K}_{rich1}(k^*) \leftarrow \mathcal{K}_{rich1}(k^*) \cup \{k'\}$
end while
return $\mathcal{K}_{rich1}(k^*)$

Algorithm 4 Generate successive sub-tours

Require: k^* a route found by $(Pr(\mathcal{K}'))$ and $\mathcal{K}'$ the set of all known routes

$k' \leftarrow k^*$, the initial route
$\mathcal{K}_{rich2}(k^*) \leftarrow$, the set of routes generated from k^*
$\mathcal{O}^{k^*} = \{o \in \mathcal{O} | e_o^{k^*} = 1\}$, the set of orders picked in route k^*
Sort $\mathcal{O}^{k^*}$ by increasing value of γ_o, the average reduced cost of routes in $\mathcal{K}'$ containing o
for each order $o \in O^{k^*}$ **do**
  $k' \leftarrow k' \setminus \{o\}$
  $\mathcal{K}_{rich2} \leftarrow \mathcal{K}_{rich2} \cup \{k'\}$
end for
return $\mathcal{K}_{rich2}(k^*)$

4.1.4 Stabilization

Column generation is well-known to suffer from stability issues at the beginning of the process: the dual values typically jump from one extreme to another until enough columns are known. Several methods exist to tackle this problem (see for example Briant et al. (2008)).

To reduce this phenomenon, we initially provide for each dual variable β_o a target t_o, which is an estimation of the optimal value of β_o in (LM), and a gap width δ_o. If the dual value is greater than $t_o + \delta_o$, a linear penalty s_o is applied. The target t_o, the gap width δ_o and the penalty s_o are updated along the column generation procedure.

To prove the optimality of the restricted master problem, the dual variables need to be unconstrained at the end of the resolution, i.e. $\delta_o = +\infty$. To translate this in $(LM(\mathcal{K}'))$, we add non-negative artificial variables v_o for each $o \in \mathcal{O}$. The stabilized restricted master problem $(SLM(\mathcal{K}'))$ is thus written as follows:

$$
(SLM(\mathcal{K}')) \begin{cases}
\min & \displaystyle\sum_{k \in \mathcal{K}'} d_k \rho_k + \sum_{o \in \mathcal{O}} (t_o + \delta_o) v_o & & & & & \\
\text{s.t.} & \displaystyle\sum_{k \in \mathcal{K}'} e_o^k \rho_k + v_o & \geq & 1 & \forall o \in \mathcal{O} & (9) & (\beta_o) \\
& v_o & \leq & s_o & \forall o \in \mathcal{O} & (10) & (\varepsilon_o) \\
& \rho_k & \geq & 0 & \forall k \in \mathcal{K}' & (11) & \\
& v_o & \geq & 0 & \forall o \in \mathcal{O} & (12) &
\end{cases}
$$

Note that Constraints (10) enforce the penalty if $\beta_o > t_o + \delta_o$, as the dual constraint associated to v_o is $\beta_o \leq t_o + \delta_o + \varepsilon_o$, with $\varepsilon_o \geq 0$.

For each order o, let $d(o)$ be the optimal distance to pick all the items of order o. After some experiments, initial values are defined as (line 3 in Algorithm 1):

$$
\begin{cases}
t_o^{\text{init}} = 0.7 \times d(o) \frac{B_o}{B} \\
\delta_o^{\text{init}} = 0.2 \times t_o \\
s_o^{\text{init}} = 1
\end{cases}
$$

At each iteration of the column generation, after solving the stabilized restricted master problem $(SLM(\mathcal{K}'))$, solving the pricing problem $(Pr(\mathcal{K}'))$ and computing the Lagrangian lower bound described in Section 4.1.5, the target t_o and the gap width δ_o are updated (line 18 of Algorithm 1) as follows.

If the lower bound LB is improved by the Lagrangian lower bound, the target t_o is reinitialized to the current value of the dual variable β_o, and the gap width δ_o is divided by 2. If the lower bound LB is not improved, the target t_o is kept unchanged, and the gap width δ_o is multiplied by 1.1 if the current value of v_o is positive, and by 0.8 otherwise. Moreover, if the pricing algorithm proves that there are noroutes with negative reduce cost, but some artificial variables v_o are positive, the optimality of the restricted master problem is not proved. In this case, all gap widths are increased

by 2.5, with the same target value. Then the column generation procedure executes again with a new iteration.

In all cases, the gap width δ_o is kept greater or equal to $0.1 \times \delta_o^{init}$ where δ_o^{init} is the initial value of δ_o, and the cost s_o is updated by the formula $s_o = \delta_o^{init}/\delta_o$. This update of the target and the gap are detailed in Algorithm 5.

Algorithm 5 Update stabilization (for an order o).

if the lower bound LB is improved by the Lagrangian lower bound **then**
  $t_o \leftarrow \beta_o$ (current value of the dual variable)
  $\delta_o \leftarrow \max\{0.5 \times \delta_o; 0.1 \times \delta_o^{\text{init}}\}$
else
  if $v_o > 0$ **then**
    $\delta_o \leftarrow 1.1 \times \delta_o$
  else
    $\delta_o \leftarrow \max\{0.8 \times \delta_o; 0.1 \times \delta_o^{\text{init}}\}$
  end if
end if
if the pricing algorithm proves that no routes with negative reduce cost exist, **and** some artificial variables v_o are positive **then**
  the optimality of the restricted master problem is not proven
  $\delta_o \leftarrow 2.5 \times \delta_o$
end if
$s_o \leftarrow \frac{\delta_o^{\text{init}}}{\delta_o}$

4.1.5 Computation of a lower bound

A typical lower bound based on Lagrangian relaxation is traditionally computed at each iteration of the column generation procedure. Let us define $K^{\max}$ as an upper bound on the maximum number of routes in an optimal solution. Consider the redundant constraint $\sum_{k \in \mathcal{K}} \rho_k \leq K^{\max}$ in the unrestricted and integer problem (M) and relax Constraints (1) in a Lagrangian manner with lagrangian multipliers $\beta_o \geq 0$. Denote also by z_M^* the optimal value of (M). The Lagrangian bound of z_M^* , denoted $\Theta(\beta)$ is defined as follow:

$$\Theta(\beta) = \sum_{o \in \mathcal{O}} \beta_o + \min \left\{ \sum_{k \in \mathcal{K}} (d_k - \sum_{o \in \mathcal{O}} \beta_o e_o^k) \rho_k \mid \sum_{k \in \mathcal{K}} \rho_k \leq K^{\max}, \rho^k \in \{0,1\}, \forall k \in \mathcal{K} \right\}$$

Let us define the lagrangian multipliers β_o as the optimal dual values of Constraints (3) of problem $(LM(\mathcal{K}'))$ (or Constraints (9) of problem $(SLM(\mathcal{K}'))$ if the master problem is stabilized). Remember that $\phi^*(\beta, \mathcal{K}) =$

$\min_{k \in \mathcal{K}}(d_k - \sum_{o \in \mathcal{O}} \beta_o e_o^k)$ and the optimal value of $(Pr(\mathcal{K}'))$ is $\phi^*(\beta, \mathcal{K}') \leq \phi^*(\beta, \mathcal{K})$. Thus, we have

$$\xi(\beta, \mathcal{K}') = \sum_{o \in \mathcal{O}} \beta_o \; + \; K^{\max} \cdot \phi^*(\beta, \mathcal{K}') \; \leq \; \sum_{o \in \mathcal{O}} \beta_o \; + \; K^{\max} \cdot \phi^*(\beta, \mathcal{K}) \; \leq \; \Theta(\beta)$$

The quantity $\xi(\beta, \mathcal{K}')$ therefore provides a lower bound of z_M^* at any iteration of the column generation procedure and can be returned as an optimality guarantee if the procedure fails to terminate within the time limit. Notice that, if the column generation procedure terminates by solving (LM) to optimality, *i.e.* $\phi^*(\beta, \mathcal{K}') = 0$ and $v_o = 0$ for all $o \in \mathcal{O}$, then $\xi(\beta, \mathcal{K}') = z_{LM}^*$ the optimal value of (LM).

An obvious value for $K^{\max}$ is $|\mathcal{O}|$, *i.e.* the number of orders. In order to strengthen the value of $K^{\max}$, observe that, in an optimal solution:

- all routes (except perhaps one) have a load of at least $1 + \lfloor B/2 \rfloor$, where B is the capacity of a trolley. Otherwise, two routes can be merged in a single one. Therefore:

$$K^{\max} = \left\lfloor \frac{\sum_{o \in \mathcal{O}} B_o - 1}{1 + \lfloor B/2 \rfloor} \right\rfloor + 1$$

- each route has a distance greater or equal to the distance needed to collect any of its orders alone. In addition, the number of orders picked in a route is at most the capacity B of the trolley. Hence from any upper bound UB of z_M^* we can thus compute a value of $K^{\max}$ with Algorithm 6. The value of UB used is the best one obtained when generating a rich column set (see the computation of $\mathcal{K}_{rich1}(k^*)$ in Section 4.1.3).

4.1.6 Master problem implementation details

Initial columns in the master problem The master problem is initially populated with the following columns. For each order o, a route is generated where only order o is picked. For each pair of orders (o, o'), a route is generated where only these two orders are picked, if it is possible within the trolley capacity B. Note that when the number of orders is large (≥ 200), it is too time-consuming to compute all pairs of orders and add them into the master problem. Hence a score (already mentioned Section 4.1.3) is computed for each pair of orders: $score(o, o') = \max_{l \in \mathcal{L}_o} \min_{l' \in \mathcal{L}_{o'}} D_{p_l, p_{l'}}$. Then, only the $|\mathcal{O}|$ smallest scoring pairs are considered to be part of the initial columns of the master problem.

Algorithm 6 Computation of $K^{\max}$ based on an upper bound

Require: UB an upper bound of z^*_M
$d\,[1..\,|\mathcal{O}|] \leftarrow$ distances to collect each individual order; in increasing order

$K^{\max} \leftarrow 0$, $ub \leftarrow 0$, $k \leftarrow |\mathcal{O}|$
while $k \geq 1$ **do**
  $ub \leftarrow ub + d[k]$
  $K^{\max} \leftarrow K^{\max} + 1$
  $k \leftarrow k - B$
end while
$k \leftarrow 1$
while $ub + d[k] < UB$ **do**
  $ub \leftarrow ub + d[k]$
  $K^{\max} \leftarrow K^{\max} + 1$
  $k \leftarrow k + 1$
end while
return $K^{\max}$

Strengthening the master problem The minimum number of trolleys required to pick all orders can be found by solving a bin-packing problem. Enforcing a simple lower bound $T = \left\lceil \frac{\sum_{o\in\mathcal{O}} B_o}{B} \right\rceil$ of this number of trolleys improves the overall linear relaxation. Thus we add the following constraint in the restricted master problem ($LM(\mathcal{K}')$):

$$\sum_{k\in\mathcal{K}'} \rho_k \geq T \qquad (13)$$

A similar constraint would also be valid for any subset of orders $\mathcal{S} \subseteq \mathcal{O}$. We consider the subsets $\mathcal{S}$ that require the same minimum number T of trolleys and define $\mathcal{N} = \left\{\mathcal{S} \subseteq \mathcal{O} \middle| \left\lceil \frac{\sum_{o\in\mathcal{S}} B_o}{B} \right\rceil = T \right\}$. Let us index $\mathcal{N}$ by i and note $\mathcal{S}_i$ the i^{th} element of $\mathcal{N}$. The constraints added to the master are thus the following:

$$\sum_{k\in\mathcal{K}'|k\cap\mathcal{S}_i\neq\emptyset} \rho_k \geq T \qquad \forall \mathcal{S}_i \in \mathcal{N} \qquad (14)$$

Note that only the $\mathcal{S}_i$ that are minimum regarding inclusion can be considered in Constraints 14.

These new constraints lead to a new calculation of the reduced costs of the routes. Let μ_i be the dual variable associated with $\mathcal{S}_i \in \mathcal{N}$. The reduced cost of a route k is then

$$r_k = d_k - \sum_{o\in k} \beta_o - \sum_{\mathcal{S}_i\in\mathcal{N}|k\cap\mathcal{S}_i\neq\emptyset} \mu_i$$

The pricing problem must therefore be updated accordingly. In particular, binary variables must be introduced to know whether the route intersects each $\mathcal{S}_i \in \mathcal{N}$. We do not give the details here since the methodology remains the same.

4.1.7 Feasible integer solutions and post-optimization.

Primal heuristics Once (LM) has been solved optimally or the time limit is reached, an integer solution is obtained by solving $(M(\mathcal{K}'))$, *i.e.* by forcing each ρ_k generated in the course of the algorithm to have a $\{0,1\}$ domain. In other words, a MIP model is solved with the existing columns. The column set must therefore be rich enough to provide enough opportunities for covering all the orders with efficient routes.

Post-optimization A post-optimization procedure using a simple *hill-climbing* is applied from the best feasible solution computed by the MIP. The neighborhood is made of two moves, (1) transferring an order from one route to another and (2) swapping two orders belonging to distinct routes. The improvement of the objective function is evaluated with the picking oracle *i.e.,* a call to the dynamic programming algorithm. Only feasible moves are considered and the first improving move found is performed. When a local minimum is found (all possible transfers or swaps are non improving from the current solution) the algorithm restarts the search from a random initial feasible solution. It stops after 20 non improving restarts have been performed or a time limit has been reached.

4.2 Case of HappyChic

Let us now turn our attention to the second industrial case encountered in the HappyChic company. In this case, an order can be split into several trolleys. Hence a route can no longer be defined by a set of orders. Therefore the major difference in the HappyChic case is that a route is defined by a set of partial order lines. In this section, we mainly focus on the differences induced by this column definition. More precisely, a column $k \in \mathcal{K}$ represents a route in the warehouse defined by:

- the **amount of products** picked for each order line and denoted by $a^k = (a^k_1, \ldots, a^k_l, \ldots, a^k_n)$. Each $a^k_l \in \mathbb{N}$ gives the amount of product p_l collected for order line l. We recall that an order line l is defined as a pair (p_l, Q_l) where $p_l \in \mathcal{P}$ defines the product and Q_l is the number of items to pick. To easily state the pricing problem we use variable $e^k_l \in \{0,1\}$ to state whether order line l is picked or not by route k so that $e^k_l = 1 \Leftrightarrow a^k_l > 0$. In order to ease notations, similarly to the Foodmart case, we will write $l \in k$ to express that $e^k_l = 1$, and $|k| = \sum_{l=1}^{n} e^k_l$ represents the number of order lines picked in route k.

- the **number of boxes** used and denoted $q^k = (q_1^k, \ldots, q_o^k, \ldots, q_m^k)$ where each $q_o^k \in \mathbb{N}$ indicates the number of boxes used for order o in the trolley performing the route k.

The restricted master problem ($LM(\mathcal{K}')$) is now defined as follows:

$$(LM(\mathcal{K}')) \begin{cases} \min & \sum_{k \in \mathcal{K}'} d_k \rho_k & & & & & \\ \text{s.t.} & \sum_{k \in \mathcal{K}'} a_l^k \rho_k & \geq & Q_l & \forall l \in \mathcal{L} & (15) & (\beta_l) \\ & \sum_{k \in \mathcal{K}'} q_o^k \rho_k & \leq & B_o & \forall o \in \mathcal{O} & (16) & (\gamma_o) \\ & \rho_k & \geq & 0 & \forall k \in \mathcal{K}' & (17) & \end{cases}$$

Constraints (15) state that the Q_l items of each order line l are picked in the warehouse so that all orders are satisfied. Constraints (16) make sure that each order o is not assigned to more than B_o boxes.

The reduced cost r_k of route k is defined as

$$r_k = d_k - \sum_{l \in \mathcal{L}} a_l^k \beta_l - \sum_{o \in \mathcal{O}} q_o^k \gamma_o$$

where β_l and γ_o are respectively the dual values of Constraints (15) and (16).

4.2.1 Pricing problem.

The HappyChic case requires to handle an additional decision compared to the Foodmart case. The exact content of each box must be decided. In other words, we must decide the number of products of each order line assigned to each box of the trolley. Let us introduce $\mathcal{B}= \{1, \ldots, B\}$ as the set of boxes in a picking route. The decision variables are now the following:

- $e_l = 1$ if order line l is picked, 0 otherwise.
- a_l is the amount of product p_l picked for order line l.
- a_{lb} is the amount of product p_l put in box b.
- $y_{ob} = 1$ if order o is picked in box b, 0 otherwise.
- q_o is the number of boxes used to pick products of order o.
- d is the estimated value (lower bound) of the distance to travel to pick all the products.

The pricing problem $(Pr(\mathcal{K}'))$ is thus:

$$
(Pr(\mathcal{K}')) \begin{cases}
\min \quad d - \sum_{l \in \mathcal{L}} \beta_l a_l - \sum_{o \in \mathcal{O}} \gamma_o q_o & & & & \\
\text{s.t.} \quad \sum_{b \in \mathcal{B}} a_{lb} & = & a_l & \forall l \in \mathcal{L} & (18) \\
\sum_{l \in \mathcal{L}_o} V_{p_l}^w a_{lb} & \leq & V_o^w y_{ob} & \forall b \in \mathcal{B}, w \in \mathcal{W}, o \in \mathcal{O} & (19) \\
a_{lb} & \leq & Q_l y_{ob} & \forall o \in \mathcal{O}, \forall l \in \mathcal{L}_o, \forall b \in \mathcal{B} & (20) \\
\sum_{b \in \mathcal{B}} y_{ob} & = & q_o & \forall o \in \mathcal{O} & (21) \\
a_l & \leq & Q_l e_l & \forall l \in \mathcal{L} & (22) \\
d_k \left(\sum_{l \in k} e_l - |k| + 1 \right) & \leq & d & \forall k \in \mathcal{K}' & (23) \\
\sum_{o \in \mathcal{O}} y_{ob} & \leq & 1 & \forall b \in \mathcal{B} & (24) \\
q_o & \leq & B_o & \forall o \in \mathcal{O} & (25) \\
a_l, a_{lb} & \in & \mathbb{N} & \forall l \in \mathcal{L} & (26) \\
e_l, y_{ob} & \in & \{0,1\} & \forall l \in \mathcal{L} & (27)
\end{cases}
$$

Constraints (18) ensure the amount of product picked in each box is equal to the total amount of products picked. Constraints (19) check that products in each box respect each dimension $w \in \mathcal{W}$ of the box capacity, V_o^w being the size of order o in dimension w. Constraints (20) link together a_{lb} and y_{ob}, by making sure that if order line l from order o is picked in box b, then order o is picked in box b. Constraints (21) permit to count the number of boxes used for each order. Constraints (22) link together a_l the quantity of product p_l picked and e_l. Constraints (23) are the tour constraints giving a relaxation on the total distance d to pick the products. If a route k previously computed (in set $\mathcal{K}'$) contains a subset of the order lines to pick, then distance d is greater than or equal to d_k, the distance to pick order lines in route k. Constraints (24) ensure that each box contains no more than one order. Constraints (25) are added to avoid generating infeasible columns regarding Constraints (16) of the master problem and ensure that the limit on the number of boxes for each order is respected by the generated route.

Since all boxes are identical, constraints are added to break some of the symmetries between boxes. These constraints assume there is a natural ordering of boxes and orders.

$$
\begin{aligned}
\sum_{o' \in \mathcal{O}; o' \leq o} y_{o'b} + \sum_{o' \in \mathcal{O}; o' > o} y_{o'b'} &\leq 1 && \forall b, b' \in \mathcal{B}, b' < b, o \in \mathcal{O} && (28) \\
\sum_{o \in \mathcal{O}} y_{ob} &\leq \sum_{o \in \mathcal{O}} y_{ob'} && \forall b, b' \in \mathcal{B}, b' < b && (29)
\end{aligned}
$$

Constraints (28) ensure that the boxes with smallest indices contain orders with smallest indices. Constraints (29) ensure that if a box b is used, boxes with a smaller index are also used.

In this industrial case, the graph representing the warehouse is acyclic (see Figure 1b for an example). Hence, it is possible to give a topological order of the locations $\mathcal{V}_L$. If a route has to visit locations $\mathcal{V}' \subseteq \mathcal{V}_L$, these locations can be ordered and numbered from 1 to $|\mathcal{V}'|$. The total distance to perform the route is thus $d = D_{s1} + \sum_{i=1}^{|\mathcal{V}'|-1} D_{i(i+1)} + D_{|\mathcal{V}'|t}$. The picker routing problem is therefore easy and the oracle computing the distance d_k for a new column k runs in linear time.

Adding a relaxation of the routing A simple linear model encoding the path of the picker can be added to $(Pr(\mathcal{K}'))$ in order to strengthen the formulation. It is the linear relaxation of a typical IP model for shortest path where a binary variable x_{ij} indicates whether arc (i, j) is included into the shortest path or not. The following constraints are thus added to $(Pr(\mathcal{K}'))$:

$$\sum_{(j,i)\in\delta^-(i)} x_{ji} = \sum_{(i,j)\in\delta^+(i)} x_{ij} \quad \forall i \in \mathcal{V}_L \cup \mathcal{V}_I \quad (30)$$

$$\sum_{(s,i)\in\delta^+(s)} x_{si} = 1 \quad (31)$$

$$\sum_{(v_{p_l},j)\in\delta^+(v_{p_l})} x_{v_{p_l}j} \geq e_l \quad \forall l \in \mathcal{L} \quad (32)$$

$$d = \sum_{i,j\in\mathcal{V}} D_{ij}x_{ij} \quad (33)$$

$$x_{ij} \leq 1 \quad \forall (i,j) \in \mathcal{A} \quad (34)$$

Constraints (30) ensure the flow or path conservation for all vertices except the initial and final depots. Constraint (31) ensures that one unit of flow is sent from the initial depot. Constraints (32) link x_{ij} variables with e_l variables: if order line l is picked, then location v_{p_l} has to be visited (at least one unit of flow goes out from this location). Constraint (33) assigns to d the exact distance to perform the route. Once the e_l variables are fixed, the linear relaxation of $(Pr(\mathcal{K}'))$ provides an integer solution of the x_{ij} variables. As a result this relaxation actually gives an optimal route with its exact distance so that the tour constraints are not needed anymore. In other words, Constraint (33) can replace Constraints (23).

Since computing the exact distance of the routing can be done by adding Constraints (30)-(34), these constraints are always added in the pricing problem. So contrary to the Foodmart case, no cutting plane is added to solve the pricing problem.

4.2.2 Rich column set.

Feasible primal solution Each time a column k is found by the pricing algorithm, the corresponding order lines are excluded from the demand, and the number of available boxes for the orders are updated according to the boxes used in column k. A set of routes to collect the remaining order lines

is generated using a dynamic programming procedure, based on the one proposed in Bué et al. (2018) where the constraint on the minimum volume of boxes is removed. Each route corresponds to a column which is added into the restricted master problem.

Generate sub-tours The key idea is the same as in the Foodmart case, but instead of dealing with orders, we deal with the locations visited in a route.

A column k can be associated to the set of the visited locations $\mathcal{V}^k = \{v \in \mathcal{V}_L | v \text{ is visited in route } k\}$. First, all columns k' that correspond to a set of locations $\mathcal{V}^{k'} \subseteq \mathcal{V}^k$ such that $|\mathcal{V}^{k'}| = |\mathcal{V}^k| - 1$ are generated. This corresponds to all the columns where all the order lines in one location have been removed. Then, from a column k, a set of columns is generated by iteratively removing one location (and its associated order lines), as long as it permits to decrease the distance of the corresponding route. At each iteration, the location that provides the best decrease of the distance is chosen.

4.2.3 Stabilization and lower bound.

Stabilization Stabilization is performed like in the Foodmart case.

Lagrangian bound The Lagrangian bound is defined similarly to the Foodmart case. The only difference is the computation of $K^{\max}$. Let us note UB an upper bound of z_M^*, $d^{\min}$ and $d^{\max}$ are respectively the minimum and maximum distance to collect an individual order line. Then the value of $K^{\max}$ can be defined as:

$$K^{\max} = 1 + \left\lfloor \frac{UB - d^{\max}}{d^{\min}} \right\rfloor$$

4.2.4 Master problem implementation details.

Initial columns in the master problem Initially, the master problem is populated with the following columns. For each order line l, the items are assigned to one or more boxes. Note that an order line may require more than one box. Then a column is generated for each box, where items picked are from the same order line. Because of the maximum number of boxes per order, these columns do not provide a feasible solution. Hence, we have adapted the dynamic programming procedure proposed in Bué et al. (2018) in order to generate a feasible solution. The adaptation consists in removing the penalty for the minimum volume of boxes, and adding a high penalty for boxes if the proposed solution for an order has more boxes than

the maximum number allowed. The columns corresponding to this solution are added into the master problem.

Strengthening the master Similarly to the Foodmart case, the lower bound given by the restricted master problem can be improved by adding a constraint enforcing a minimum number of trolleys T, which can be computed as follows:

$$T = \left\lceil \frac{\sum_{o \in \mathcal{O}} \left(\max_{w \in \mathcal{W}} \frac{V_o^w}{V^w} \right)}{B} \right\rceil$$

In addition, the cuts defined in Constraints (14) are added in the same way for the HappyChic case, with the value of T previously defined.

4.2.5 Feasible integer solution.

The primal heuristic is the same as in the Foodmart case. No post-optimization procedure is applied.

5 Numerical experiments

The experiments were performed on an Intel Xeon E5-2440 v2 @ 1.9 GHz processor and 32 GB of RAM and each algorithm ran with a memory limit of 4 GB of RAM on a single thread.

5.1 Benchmark

The benchmark originated from Foodmart is described in details in Valle et al. (2017) and is publicly available. It comes from a database of anonymized customer purchases over two years for a chain of supermarkets. Orders are made by combining the purchases of customers over the first Δ days and instances are generated with $\Delta \in \{5, 10, 20\}$ leading to instances with larger orders (more products and number of items) as Δ increases. The number of orders m varies from 5 to 30. A trolley is made of 8 boxes, each able to contain 40 items. A single warehouse layout is used containing 8 aisles and 3 cross-aisles where the warehouse can store a total of 1584 products.

The benchmark originated from HappyChic is described in details in Bué et al. (2018). It is based on the warehouse activity of HappyChic for 17 working days between January and November 2017. All instances with less than 2000 items to pick are used to assess the algorithm in the present work. Overall there is a total of 279 instances. The warehouse layout is presented in Figure 1b and differs from the typical rectangular warehouse as explained before.

All instances from the two benchmarks are publicly available[1] in the same text format, matching the problem specification used in this paper. In particular, the warehouse is abstracted away as a graph to encompass very general warehouse layouts.

5.2 Comparison with Valle et al. (2017)

In this section, we report a direct comparison, on the Foodmart benchmark, of the branch and cut algorithm (BC) presented in Valle et al. (2017) and the Column Generation Heuristic (CGH) proposed in this work. A maximum of 2 hours is given to CGH. An additional 10% of time (12 minutes in this case) is given to the primal heuristic and the post-optimization mentioned in Section 4.1.7. Note that BC was run with a 6 hours time limit on a faster machine.

Table 1 reports the results. The CPU(s) column denotes the total time elapsed in seconds. TL means the time limit has been reached. UB and LB respectively denotes the best upper and lower bounds obtained at the end of the algorithm. The algorithm terminates either because the linear relaxation of the master is computed (it is proven that no negative reduced cost columns exist) or the time limit was reached. GAP(%) is computed as $\frac{100(\text{UB}-\text{LB})}{\text{LB}}$. Note that the GAP(%) reported in the work of Valle et al. (2017) is computed as $\frac{100(\text{UB}-\text{LB})}{\mathbf{UB}}$. The columns ITER and NCOL respectively give the number of iterations performed (number of times the master is solved) and the total number of columns added to the master. Note that the column FLB, reported for BC, denotes the lower bound at the root node of their model showing the quality of the linear relaxation. It is therefore directly comparable to the LB column of our approach which is the linear relaxation of the master (or the best Lagrangian bound found when the time limit is reached).

CGH improves all lower bounds and all upper bounds of the 7 instances left unsolved to optimality by BC. CHG is very competitive in running times and runs faster than BC. Note that although CGH is not an exact algorithm since no branching is performed, the optimality gap provided remains very small and in 20 cases it is able to prove optimality. On one side, for the 42 instances solved optimally by BC, the gap achieved by CGH is of 0.3 % in average and 1.9 % in the worst case. On the other side, the lower bounds of CGH significantly improves the ones of BC for the 7 unsolved instances. Finally note that CGH is able to get 40 optimal solutions on the 42 instances solved optimally by BC.

Notice that the superiority of CGH over BC regarding running times increases with Δ. In particular for $\Delta = 20$, CGH runs much faster for

[1] All datasets as well as detailed computational results are available on this page: https://pagesperso.g-scop.grenoble-inp.fr/~cambazah/batching/

		BC					CGH					
Δ	m	CPU(s)	UB	LB	FLB	GAP(%)	CPU(s)	UB	LB	GAP(%)	ITER	NCOL
5	5	3.9	348.6	348.6	346.5	0.0	**1.2**	348.6	348.6	0.0	1	31
	6	2.9	364.8	364.8	364.8	0.0	**1.1**	364.8	364.8	0.0	1	63
	7	8.9	374.8	374.8	372.8	0.0	**1.8**	374.8	374.8	0.0	1	123
	8	95.2	503.8	503.8	462.1	0.0	**3.2**	503.8	503.8	0.0	7	218
	9	151.8	539.6	539.6	479.2	0.0	**4.3**	539.6	539.6	0.0	2	351
	10	111.0	581.4	581.4	497.8	0.0	**10.9**	581.4	581.4	0.0	14	631
	11	97.1	613.5	**613.5**	509.9	**0.0**	53.7	613.5	611.6	0.3	24	946
	12	256.7	621.4	**621.4**	527.9	**0.0**	45.3	621.4	613.4	1.3	28	1 143
	13	168.9	623.4	623.4	528.2	0.0	**75.9**	623.4	623.4	0.0	31	2 266
	14	263.8	639.3	**639.3**	525.5	**0.0**	116.2	639.3	637.7	0.2	31	3 054
	15	348.9	653.4	653.4	522.4	0.0	**166.5**	653.4	653.4	0.0	19	4 308
	20	2 990.9	870.4	**870.4**	598.4	**0.0**	1 596.8	870.4	862.1	1.0	58	11 953
	25	21 600.0	1 127.4	927.0	640.8	21.6	TL	**1 105.9**	**1 083.9**	**2.0**	87	24 815
	30	21 600.0	1 221.9	894.3	647.5	36.6	TL	**1 207.7**	**1 000.5**	**20.7**	95	30 988
10	5	1.5	371.1	371.1	371.1	0.0	**0.9**	371.1	371.1	0.0	1	31
	6	6.6	377.1	377.1	374.7	0.0	**1.2**	377.1	377.1	0.0	2	61
	7	197.5	549.8	549.8	503.4	0.0	**1.8**	549.8	549.8	0.0	5	91
	8	286.8	584.2	584.2	508.4	0.0	**1.4**	584.2	584.2	0.0	1	76
	9	356.1	637.4	637.4	528.6	0.0	**7.2**	637.4	637.4	0.0	6	222
	10	393.0	661.8	661.8	553.4	0.0	**16.2**	661.8	661.8	0.0	13	319
	11	383.7	699.8	**699.8**	553.8	**0.0**	114.8	699.8	692.4	1.1	33	752
	12	217.7	707.7	**707.7**	553.6	**0.0**	183.0	707.7	703.4	0.6	47	1 189
	13	385.4	725.7	**725.7**	560.2	**0.0**	124.1	725.7	712.2	1.9	33	2 129
	14	483.8	727.8	**727.8**	562.4	**0.0**	182.6	731.7	723.8	1.1	42	3 073
	15	1 832.6	882.6	**882.6**	593.1	**0.0**	189.2	882.6	881.7	0.1	40	3 364
	20	11 015.4	992.4	**992.4**	672.0	**0.0**	955.8	994.6	976.3	1.9	46	8 820
	25	21 600.0	1 213.4	1 011.3	707.8	20.0	2 677.6	**1 191.7**	**1 187.3**	**0.4**	70	15 090
	30	21 600.0	1 330.0	963.1	706.3	38.1	TL	**1 274.0**	**1 159.6**	**9.9**	130	32 383
20	5	110.9	573.8	573.8	535.3	0.0	**0.7**	573.8	573.8	0.0	3	24
	6	278.9	656.2	656.2	580.9	0.0	**1.2**	656.2	656.2	0.0	5	47
	7	263.2	689.8	689.8	571.4	0.0	**8.0**	689.8	689.8	0.0	20	59
	8	189.0	697.8	697.8	568.7	0.0	**10.5**	697.8	697.8	0.0	17	119
	9	305.5	727.7	**727.7**	578.0	**0.0**	41.2	727.7	726.8	0.1	35	205
	10	584.2	920.5	**920.5**	634.2	**0.0**	37.9	920.5	905.3	1.7	28	304
	11	703.8	980.5	980.5	672.9	0.0	**11.7**	980.5	980.5	0.0	13	369
	12	830.1	1 004.3	**1 004.3**	666.0	**0.0**	41.9	1 004.3	999.3	0.5	27	557
	13	1 177.8	1 009.1	**1 009.1**	696.5	**0.0**	82.7	1 009.1	1 008.1	0.1	24	618
	14	1 306.0	1 011.1	1 011.1	679.2	0.0	**70.6**	1 011.1	1 011.1	0.0	43	795
	15	3 793.7	1 028.7	**1 028.7**	671.5	**0.0**	87.8	1 028.7	1 025.5	0.3	34	1 191
	20	21 600.0	1 335.2	1 124.6	729.0	18.7	619.6	**1 333.7**	**1 332.3**	**0.1**	62	3 011
	25	21 600.0	1 694.9	1 166.8	824.5	45.3	3 288.5	**1 621.2**	**1 602.9**	**1.1**	89	6 272
	30	21 600.0	1 966.9	1 071.9	853.4	83.5	TL	**1 879.2**	**1 843.1**	**2.0**	112	11 722

Table 1: Comparison with Valle et al. (2017) on the Foodmart benchmark with a 2 hours time limit.

the instances with few orders and considerably improves the gap for 20, 25 and 30 orders. Parameter Δ of Valle et al. (2017) somehow relates to the size/volume of each order and instances for $\Delta = 20$ tend to have more routes in their optimal solution. It seems that the ability to compute optimal routes efficiently in the warehouse is increasingly critical as Δ increases.

5.3 Analysis of the column generation algorithm

In this section, we analyse the robustness and scalability of CGH when it is run with smaller time limits of 300 seconds, 600 seconds and 1 hour. Table 2 reports the results over all instances where CGH does not terminate within 300 seconds (i.e the algorithm is interrupted because the time limit is reached). The table gives the upper and lower bounds obtained as well as the corresponding optimality gap.

		Time limit 300s				Time limit 600s				Time limit 3600s			
Δ	m	CPU(s)	UB	LB	GAP(%)	CPU(s)	UB	LB	GAP(%)	CPU(s)	UB	LB	GAP(%)
5	20	TL	884.8	829.5	6.7	TL	870.4	846.9	2.8	1 504.3	870.4	862.1	1.0
	25	TL	1 107.7	915.3	21.0	TL	1 107.7	911.5	21.5	TL	1 105.9	1 064.9	3.9
	30	TL	1 208.0	880.2	37.2	TL	1 199.9	880.2	36.3	TL	1 199.9	940.2	27.6
10	20	TL	994.6	962.0	3.4	TL	994.6	974.8	2.0	987.6	994.6	976.3	1.9
	25	TL	1 199.7	1 139.6	5.3	TL	1 199.7	1 158.6	3.5	2 780.5	1 189.7	1 187.3	0.2
	30	TL	1 280.0	986.7	29.7	TL	1 280.0	995.0	28.6	TL	1 278.0	1 018.1	25.5
20	20	TL	1 345.7	1 303.3	3.3	TL	1 333.7	1 331.9	0.1	660.7	1 333.7	1 332.3	0.1
	25	TL	1 629.2	1 522.0	7.0	TL	1 629.2	1 542.4	5.6	3 397.9	1 621.2	1 602.9	1.1
	30	TL	1 889.5	1 674.6	12.8	TL	1 888.8	1 721.1	9.7	TL	1 877.5	1 784.7	5.2

Table 2: Results for various time limits.

Although the results improve as the limit limit is increased, the bounds computed within 300 seconds are already of high quality.

We now analyse the behaviour of the algorithm and the time spent at each stage. Table 3 reports the time (in seconds) spent when solving the pricing problem ($Pr(\mathcal{K}')$) (CPU-PRICING), the master problem (CPU-MASTER), the final MIP when variables are converted to binary variables, the post-optimization process on the solution provided by the MIP (CPU-POST) as well as the total time (CPU). We also give the value of the upper bound obtained after solving the MIP (UB-MIP) along with the final upper bound (UB) that is computed at the end of the post-optimization process. The column IMPR provides the improvement of the upper bound due to the post-optimisation process (IMPR(%) $= \frac{100(\text{UB}-\text{MIP}-\text{UB})}{\text{UB}-\text{MIP}}$).

It can be seen that the time is mostly used for solving the pricing problem and the master is always solved instantly. Even the final MIP over the known columns remains easy to solve optimally in less than one minute. The post-optimization process always requires less than two minutes although the time limit for the post-optimization is set to 10% of the time limit, i.e. 12 minutes. Notice that the upper bounds found by the column generation process alone (UB-MIP) are excellent and the improvement (IMPR(%)) done by the post-optimization to reach the final upper bound (UB) is not really significant. This improvement however grows with the number of orders.

5.4 Feasible heuristics and interest of optimal routing

Valle et al. (2017) argue that when batching and routing problems are sep-

Δ	m	UB	UB-MIP	IMPR(%)	CPU-PRICING	CPU-MASTER	CPU-MIP	CPU-POST	CPU(s)
5	5	348.6	348.6	0.0	0.3	0.0	0.0	0.0	1.2
	6	364.8	364.8	0.0	0.7	0.0	0.0	0.0	1.1
	7	374.8	374.8	0.0	1.2	0.0	0.0	0.0	1.8
	8	503.8	503.8	0.0	2.6	0.0	0.0	0.0	3.2
	9	539.6	539.6	0.0	3.6	0.0	0.0	0.0	4.3
	10	581.4	581.4	0.0	9.9	0.0	0.0	0.0	10.9
	11	613.5	613.5	0.0	19.4	0.1	0.1	8.7	53.7
	12	621.4	621.4	0.0	22.8	0.9	0.3	9.9	45.3
	13	623.4	623.4	0.0	65.0	2.5	2.1	0.0	75.9
	14	639.3	639.4	0.0	84.5	3.7	6.0	13.8	116.2
	15	653.4	653.4	0.0	151.4	2.8	4.3	0.0	166.5
	20	870.4	886.0	1.8	1 508.2	13.8	6.3	26.9	1 596.8
	25	1 105.9	1 108.4	0.2	7 144.0	1.6	1.9	43.6	7 246.5
	30	1 207.7	1 240.4	2.6	6 755.0	122.7	49.8	48.7	7 304.0
10	5	371.1	371.1	0.0	0.3	0.0	0.0	0.0	0.9
	6	377.1	377.1	0.0	0.7	0.0	0.0	0.0	1.2
	7	549.8	549.8	0.0	1.2	0.0	0.0	0.0	1.8
	8	584.2	584.2	0.0	0.8	0.0	0.0	0.0	1.4
	9	637.4	637.4	0.0	3.4	0.0	0.0	0.0	7.2
	10	661.8	661.8	0.0	6.2	0.0	0.0	0.0	16.2
	11	699.8	705.8	0.9	21.7	0.5	0.1	20.3	114.8
	12	707.7	707.7	0.0	49.8	2.0	0.3	16.3	183.0
	13	725.7	731.8	0.8	86.3	2.5	1.5	27.6	124.1
	14	731.7	731.7	0.0	147.3	3.7	2.6	18.5	182.6
	15	882.6	884.5	0.2	113.0	0.0	0.0	51.5	189.2
	20	994.6	1 004.9	1.0	851.1	12.2	10.4	42.6	955.8
	25	1 191.7	1 203.7	1.0	2 459.5	2.6	1.9	105.5	2 677.6
	30	1 274.0	1 274.0	0.0	6 555.6	163.1	32.7	70.3	7 305.8
20	5	573.8	573.8	0.0	0.2	0.0	0.0	0.0	0.7
	6	656.2	656.2	0.0	0.6	0.0	0.0	0.0	1.2
	7	689.8	689.8	0.0	0.9	0.0	0.0	0.0	8.0
	8	697.8	697.8	0.0	2.4	0.0	0.0	0.0	10.5
	9	727.7	729.4	0.2	5.4	0.0	0.0	11.8	41.2
	10	920.5	924.4	0.4	7.3	0.0	0.0	27.5	37.9
	11	980.5	980.5	0.0	8.0	0.0	0.0	0.0	11.7
	12	1 004.3	1 004.3	0.0	15.9	0.0	0.0	14.7	41.9
	13	1 009.1	1 010.2	0.1	23.8	0.1	0.0	30.5	82.7
	14	1 011.1	1 011.1	0.0	39.0	0.4	0.1	0.0	70.6
	15	1 028.7	1 028.7	0.0	58.8	0.6	0.3	19.6	87.8
	20	1 333.7	1 336.1	0.2	535.9	3.2	1.0	51.5	619.6
	25	1 621.2	1 621.2	0.0	3 191.4	4.4	1.2	50.0	3 288.5
	30	1 879.2	1 879.2	0.0	7 116.5	1.8	1.7	68.2	7 271.4

Table 3: Detailed results of the column generation algorithm with a 2 hours time limit. All CPU times are in seconds.

arated, their "*proposed method is able to solve the routing problem to optimality very quickly*" (see Section 6.5 of Valle et al. (2017)). We believe that this ability is also a key feature of CGH and we report the results obtained by the dynamic programming approach on the same experiments presented in Valle et al. (2017).

Valle et al. (2017) propose to compute feasible solutions (upper bounds) on very large instances (with up to 5000 orders). In that case, their model is used to compute the optimal route of a single trolley. Typically, the

algorithm denoted *Valle et al. (ii)* is the implementation of the Clarke and Wright algorithm in the context of picking where the savings are estimated using typical picking heuristics (S-Shape, Largerst-gap, Combined) and where routes are solved optimally in the final assignment of orders to trolleys (see Section 4.2.2 of Valle et al. (2017)). The algorithm denoted *Valle et al. (iii)* refers to a version where each saving is calculated exactly by computing the optimal route. The reader can refer to Section 6.5.2 of Valle et al. (2017) for more details.

Results are presented in Table 4. The algorithm denoted Clarke-Pdyn is our implementation of the algorithm *Valle et al. (iii)* where the savings are computed exactly by calling the dynamic programming algorithm.

It performs the $O(2m + m^2)$ computations of *savings* (recall that m is the number of orders). The CPU times of the two approaches are therefore directly comparable since the exact same number of computations of optimal routes are made. We also apply CGH algorithm with a time limit of 2 hours plus an additional 10% of time for the primal heuristic and post-optimization. Note that in this case we seek for a negative reduced cost column by applying a local search procedure, before solving the pricing problem.

	Valle et al. (ii)		Valle et al. (iii)		Clarke-Pdyn		CGH 7 200 s			
m	UB	CPU(s)	UB	CPU(s)	UB	CPU(s)	UB	UB-MIP	LB	CPU(s)
50	2 231,2	3,2	2 182,7	61,2	**1 866,0**	5,1	**1 593,3**	**1 593,3**	**913,9**	TL
100	4 012,1	5,5	3 830,9	226,0	**3 270,5**	18,4	**2 882,5**	**3 080,9**	**479,6**	TL
200	6 653,7	6,5	6 395,3	689,4	**5 972,8**	63,5	**5 608,8**	6 716,3	0,0	TL
500	13 899,5	16,1	13 794,5	4 486,6	14 041,6	401,9	**13 237,0**	15 972,0	0,0	TL
1000	26 160,7	53,7	-	-	26 203,8	1 682,8	**25 504,2**	30 719,3	0,0	TL
2000	49 740,0	179,0	-	-	-	-	**48 173,0**	57 763,5	0,0	TL
5000	**118 269,7**	1 048,0	-	-	-	-	128 811,7	137 165,4	0,0	TL

Table 4: Quality and speed of heuristics taking advantage of optimal routing for large instances.

The Clarke-Pdyn algorithm does not give the exact same upper bounds as *Valle et al (iii)* and probably differs slightly from the implementation of *Valle et al (iii)* in the ways ties are broken. In any case it is roughly 10 times faster to compute the $O(2m + m^2)$ picking routes demonstrating the efficiency of dynamic programming for picking as opposed to the MIP model.

Moreover, it can be noticed that CGH is able to run on these large instances, and even to provide lower bounds for instances with 50 and 100 orders. And when solving these large instances, very good upper bounds are obtained after applying the post-optimization process as described in Section 4.1.7. A maximum of 10% of the running time (here 12 minutes) is used for this post-optimization. Hence, post-optimization is really effective on these large instances. Results for 5000 orders are not really good with CHG. This is mainly due to the time limit: with a time limit of 6 hours, we are able to obtain a feasible solution with value of 115 629.25. From these

results, the improvement of CGH for these large instances seems to represent an interesting prospect.

A few experiments are also performed in Valle et al. (2017) with more aisles and cross-aisles (at most 5 cross-aisles) which makes the routing problem significantly harder for the linear model used requiring up to 90 seconds to compute six optimal routes. Note that the dynamic programming approach used in the present paper scales to 9 cross-aisles, solving any instances up to 6 cross-aisles (with up to 60 aisles and 240 products) in less than one second (Pansart et al., 2018).

5.5 Impact of the key features of CGH

We evaluate the effect of three features of CGH presented in Section 4: the addition of a rich set of columns at each iteration, the stabilization, and the addition of Constraints (14) on the minumum number of trolleys to strengthen the master problem. Table 5 shows the overall results averaged by class of instances for Foodmart benchmark with a short time limit of 300 seconds. Each column represents a class of instances for the three values of Δ (5, 10, 20). *CHG* is the algorithm with all its features. *CGH - no rcs* does not use a rich column set. *CGH - no stab* does not consider stabilization of the dual values. *CGH - no cuts* does not include Constraints (14) in the master problem. Finally the last case does not consider any of this three features.

Δ		5	10	20
CGH	UB	647.5	743.1	1 011.7
	UB-MIP	659.2	758.9	1 020.4
	LB	605.5	713.1	983.8
CGH - no rcs	UB	646.9	744.7	1 010.8
	UB-MIP	697.8	827.7	1 062.4
	LB	517.8	642.9	923.6
CGH - no stab	UB	646.1	743.8	1 013.5
	UB-MIP	657.3	758.7	1 019.9
	LB	575.9	682.6	969.3
CGH - no cuts	UB	648.3	745.1	1 014.5
	UB-MIP	657.7	765.7	1 024.2
	LB	593.0	693.9	959.0
CGH - no rcs no stab no cuts	UB	648.1	745.3	1 013.7
	UB-MIP	677.2	785.2	1 033.5
	LB	514.1	605.4	883.6

Table 5: Overall results averraged by class of instances of the algorithm with a 300s time limit with and without three key features: rich column set (rcs), stabilization (stab), and cuts to strengthen the master (cuts).

We can note that the rich column set strategy is the key to the efficiency of CGH. Stabilization turns out to be less essential although it does help to quickly identify good lower bounds. Constraints (14) are really helpful to provide better lower bounds.

5.6 Results of HappyChic

In this section, we report average results for the 279 instances for the HappyChic case. Each instance is run with a 600 seconds time limit. Detailed results are reported in the project web page[2]. Table 6 reports average results. Each line corresponds to a set of instances with the same number of routes in the solution. This number of routes is reported in the column NbRoutes. The column NbInstances reports the number of instances for each set. The columns for GAP(%) respectively report the average, the minimum and the maximum gap obtained with the column generation approach over the set of instances. UB and LB are the upper and lower bounds at the end of the algorithm. The gap is computed as $\frac{100(\text{UB}-\text{LB})}{\text{LB}}$. Note that for two instances the algorithm provided no lower bound, hence the gap is infinite. These two instances are not included for the average and maximum gaps reported in Table 6. The IMPR(%) columns respectively report the average, the minimum and the maximum deviation in percentage with the results obtained with the dynamic programming procedure proposed in Bué et al. (2018) (where only the minimum volume constraint on boxes is removed). Let UBH be the value of the solution obtained with the dynamic programming procedure; the improvement is computed as $\frac{100(\text{UB}-\text{UBH})}{\text{UBH}}$. Hence a positive value means that CGH founds a better solution.

		CGH					
		GAP(%)			IMPR(%)		
NbRoutes	NbInstances	avg	min	max	avg	min	max
1	16	0.0	0.0	0.0	0.0	0.0	0.0
2	27	13.0	0.0	62.8	4.0	0.0	23.5
3	37	35.7	8.2	127.9	4.1	0.0	19.2
4	41	68.6	21.8	276.1	3.2	-0.5	10.4
5	34	85.7	3.8	296.2	2.4	0.0	16.5
6	38	98.1	19.8	297.7	2.4	0.0	11.0
{7;8}	30	155.4	24.0	1032.3	3.1	0.0	12.0
{9;10}	25	124.6	20.5	354.4	1.5	0.0	4.9
≥ 11	31	413.5	31.9	4111.9	0.6	-0.5	23.5

Table 6: Global results of the algorithm with a 600s time limit on the HappyChic benchmark.

CGH provides optimality proof for all instances with one route, and for some instances with two routes. Some good lower bounds can be found even for larger instances, e.g. with 5 routes. Moreover, the quality of the upper bound is very good since it permits to improve the results provided with the algorithm proposed by Bué et al. (2018). For some instances, the improvement reaches up to 23.5%. These results over the HappyChic benchmark prove the ability of CGH to provide very good upper bounds and some

[2] https://pagesperso.g-scop.grenoble-inp.fr/~cambazah/batching/

lower bounds in a reasonable amount of time for industrial instances where the exact content of each box also has to be decided.

6 Conclusion

We proposed an exponential linear programming formulation of the joint order batching and picker routing problem, where variables (or columns) are related to single picking routes in the warehouse. The rationale of our approach is to consider that the picker routing alone, in nowadays warehouses, is easy enough in practice to be solved exactly. This work therefore builds upon previous work of the authors (Cambazard & Catusse, 2018; Pansart et al., 2018; Bué et al., 2018) and existing literature related to efficient optimal solving of the picker routing. The proposed approach is designed to provide accurate lower and upper bounds without necessarily focusing on the optimality proof. This design allows for a good scalability while providing quality guarantees on the solutions found.

It is directly compared on a publicly available benchmark with a recent branch and cut algorithm (Valle et al., 2017) improving a number of best known lower and upper bounds within short time limits. The algorithm is also assessed on another industrial benchmark coming from the HappyChic company with real-life datasets containing up to 2000 items to pick. In this latter case, the problem specifications are more general than the one considered in Valle et al. (2017). In particular, orders can be split among trolleys which requires to deal with quantities (number of items) as opposed to full orders. This new benchmark is made publicly available along with the detailed results given by our algorithm[3].

We believe that the optimal picker routing could be leveraged further, in particular to provide a fast and heuristic pricing. It might also lead to tackle more integrated problems where additional levels of decisions in the warehouse (e.g. storage location assignment) can be made jointly with batching and picking.

References

Bartholdi, J. J., & Hackman, S. T. (2017). *Warehouse & Distribution Science, Release 0.98*. Atlanta: Georgia Institute of Technology, School of Industrial and Systems Engineering, The Supply Chain and Logistics Institute.

Briant, O., Lemaréchal, C., Meurdesoif, P., Michel, S., Perrot, N., & Vanderbeck, F. (2008). Comparison of bundle and classical column generation. *Mathematical Programming*, *113*, 299–344.

[3] https://pagesperso.g-scop.grenoble-inp.fr/~cambazah/batching/

Bué, M., Cattaruzza, D., Ogier, M., & Semet, F. (2018). An integrated order batching and picker routing problem. URL: https://hal.archives-ouvertes.fr/hal-01849980 working paper or preprint.

Cambazard, H., & Catusse, N. (2018). Fixed-parameter algorithms for rectilinear steiner tree and rectilinear traveling salesman problem in the plane. *European Journal of Operational Research*, *270*, 419–429.

Cano, J. A., Correa-Espinal, A. A., & Gómez-Montoya, R. A. (2018). A review of research trends in order batching, sequencing and picker routing problems. *Rivista Espacios*, *39*.

Cergibozan, c., & Tasan, A. S. (2016). Order batching operations: an overview of classification, solution techniques, and future research. *Journal of Intelligent Manufacturing*, (pp. 1–15).

Chen, T.-L., Cheng, C.-Y., Chen, Y.-Y., & Chan, L.-k. (2015). An efficient hybrid algorithm for integrated order batching, sequencing and routing problem. *International Journal of Production Economics*, *159*, 158–167.

Cheng, C.-Y., Chen, Y.-Y., Chen, T.-L., & Yoo, J. J.-W. (2015). Using a hybrid approach based on the particle swarm optimization and ant colony optimization to solve the joint order batching and picker routing problem. *International Journal of Production Economics*, *170*, 805–814.

Clarke, G., & Wright, J. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, *12*, 565–581.

Cornuéjols, G., Fonlupt, J., & Naddef, D. (1985). The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical programming*, *33*, 1–27.

Dantzig, G., Fulkerson, R., & Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, *2*, 393–410.

Ecommerce-Europe (2017). Ecommerce europe. https://www.ecommerce-europe.eu/. Online; accessed 24-October-2017.

Feillet, D. (2010). A tutorial on column generation and branch-and-price for vehicle routing problems. *4OR A Quarterly Journal of Operations Research*, *8*, 407–424.

Fleischmann, B. (1985). A cutting plane procedure for the travelling salesman problem on road networks. *European Journal of Operational Research*, *21*, 307–317.

Gademann, N., & Van de Velde, S. (2005). Order batching to minimize total travel time in a parallel-aisle warehouse. *Computers & Operations Research*, *37*, 63–75.

Gibson, D. R., & Sharp, G. P. (1992). Order batching procedures. *European Journal of Operational Research*, *58*, 57–67.

Hall, R. W. (1993). Distance approximations for routing manual pickers in a warehouse. *IIE transactions*, *25*, 76–87.

Henn, S., Koch, S., Doerner, K., Strauss, C., & W ascher, G. (2010). Metaheuristics for the order batching problem in manual order picking systems. *Business Research*, *3*, 82–105.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations* (pp. 85–103). Springer.

de Koster, M., Van der Poort, E., & Wolters, M. (1999). Efficient orderbatching methods in warehouses. *International Journal of Production Research*, *37*, 1479–1504.

de Koster, R., Le-Duc, T., & Roodbergen, K. J. (2007). Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, *182*, 481–501.

de Koster, R., & Van der Poort, E. (1998). Routing orderpickers in a warehouse: a comparison between optimal and heuristic solutions. *IIE transactions*, *30*, 469–480.

Kulak, O., Sahin, Y., & Egemen Taner, M. (2012). Joint order batching and picker routing in single and multiple-cross-aisle warehouses using cluster-based tabu search algorithms. *Flexible Services and Manufacturing Journal*, *24*, 52–80.

Letchford, A. N., Nasiri, S. D., & Theis, D. O. (2013). Compact formulations of the steiner traveling salesman problem and related problems. *European Journal of Operational Research*, *228*, 83–92.

Marchet, G., Melacini, M., & Perotti, S. (2015). Investigating order picking system adoption: a case-study-based approach. *International Journal of Logistics: Research and Applications*, *18*, 82–98.

Menéndez, B., Pardo, E. G., Alonso-Ayuso, A., Molina, E., & Duarte, A. (2017). Variable neighborhood search strategies for the order batching problem. *Computers & Operations Research*, *78*, 500–512.

Orman, A., & Williams, H. P. (2006). A survey of different integer programming formulations of the travelling salesman problem. *Optimisation,*

economics and financial analysis. Advances in computational management science, *9*, 93–106.

Pansart, L., Catusse, N., & Cambazard, H. (2018). Exact algorithms for the order picking problem. *Computers & Operations Research*, *100*, 117–127.

Petersen, C. G. (1997). An evaluation of order picking routeing policies. *International Journal of Operations & Production Management*, *17*, 1098–1111.

Ratliff, H. D., & Rosenthal, A. S. (1983). Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. *Operations Research*, *31*, 507–521.

Roodbergen, K. J., & de Koster, R. (2001a). Routing methods for warehouses with multiple cross aisles. *International Journal of Production Research*, *39*, 1865–1883.

Roodbergen, K. J., & de Koster, R. (2001b). Routing order pickers in a warehouse with a middle aisle. *European Journal of Operational Research*, *133*, 32–43.

Thia, F. (2008). Mysql foodmart database. URL: http://pentaho-en.phi-integration.com/mondrian/mysql-foodmart-database.

Tompkins, J. A., White, J. A., Bozer, Y. A., & Tanchoco, J. M. A. (2010). *Facilities planning*. John Wiley & Sons.

Valle, C. A., Beasley, J. E., & da Cunha, A. S. (2017). Optimally solving the joint order batching and picker routing problem. *European Journal of Operational Research*, *262*, 817–834.

Vaughan, T. (1999). The effect of warehouse cross aisles on order picking efficiency. *International Journal of Production Research*, *37*, 881–897.

Won, J., & Olafsson, S. (2005). Joint order batching and order picking in warehouse operations. *International Journal of Production Research*, *43*, 1427–1442.